Iterative Distillation for Feature-Consistent Sparse Autoencoders

Cristina P. Martin-Linares [1 2] **Jonathan P. Ling** [3]

Abstract

Sparse autoencoders (SAEs) aim to separate model activations into monosemantic, human-interpretable features. In practice, learned features are often redundant and differ across training runs and sparsity levels. These differences make interpretations difficult to transfer and reuse. We introduce Distilled Matryoshka Sparse Autoencoders (DMSAEs), a training pipeline that distills a compact core of consistently useful features and reuses this core to train new SAEs. DMSAEs use an iterative distillation cycle. First, we train a Matryoshka SAE with a shared core. We then use gradient-based scores on SAE activations before BatchTopK to rank candidate features by sensitivity to next-token loss. Finally, we retain the smallest subset whose cumulative score reaches a fixed fraction of the total score. Across cycles, we transfer only the core encoder weight vectors and then reinitialize the core decoder and all non-core latents for the next training cycle. On residual stream activations from layer 12 of Gemma-2-2B, seven distillation cycles produced a core of 197 features that were repeatedly selected. Training using this distilled core improves several SAEBench metrics and demonstrates that consistent sets of latent features can be transferred across sparsity levels.

1. Introduction

Sparse autoencoders (SAEs) are widely used tools for probing neural network activations that map dense internal representations to sparse latent codes. These codes often correspond to concepts that humans can understand. However, SAEs can have persistent problems with interpretability, such as feature splitting, feature absorption, and redundancy. For example, an SAE can split a broad concept such as animal into specific features such as "dog" and "cat," or it can split the concept only partly and create systematic blind spots (Chanin et al., 2024). These problems make SAE features more difficult to interpret and can reduce their usefulness in downstream analyses and interventions (Bricken et al., 2023; Gao et al., 2024).

Matryoshka SAEs address some of these problems by training several nested dictionaries at the same time, with smaller prefixes that learn broad features and larger prefixes that learn more specific features (Bussmann et al., 2025). This hierarchical structure can reduce feature absorption and preserve broad concepts at different dictionary sizes. However, this nested structure does not ensure that the learned features are distinct or stable. Matryoshka SAEs can still learn redundant or ambiguous latent features, and different training runs or sparsity levels can produce different features (Leask et al., 2025). These differences make it difficult to compare, transfer, and reuse feature interpretations. Therefore, recent work identifies feature consistency as an important requirement for reproducible mechanistic interpretability (Song et al., 2025; Paulo & Belrose, 2025).

Ordered SAEs (OSAEs) (Wang et al., 2025) extend Matryoshka SAEs with a nested-dropout objective that trains latent prefixes and gives the latents a global order. This order reduces permutation non-identifiability and improves feature consistency across training runs. In contrast, Matching Pursuit SAEs (MP-SAEs) (Costa et al., 2025) use a residual-guided sequential encoder. At each inference step, an MP-SAE selects the feature with the largest projection onto the current residual and adds its projected contribution to the reconstruction. As a result, the squared reconstruction error cannot increase and decreases when the selected projection is nonzero. However, the encoder makes locally optimal selections in sequence and does not jointly optimize all active features. Thus, it does not generally guarantee the optimal feature support and coefficients for a fixed sparsity budget. Each residual also depends on the preceding selections, so early selections can affect the features that the encoder selects later. OSAEs and MP-SAEs therefore add different forms of structure: OSAEs improve feature order and consistency across runs, whereas MP-SAEs improve

[1]Department of Biomedical Engineering, Johns Hopkins University, Baltimore, MD, United States [2]Center for Computational Biology, Whiting School of Engineering, Johns Hopkins University, Baltimore, MD, United States [3]Department of Pathology, Johns Hopkins University, Baltimore, MD, United States. Correspondence to: Cristina P. Martin-Linares <cristina.ml@jhu.edu>.

Mechanistic Interpretability Workshop at the 43rd International Conference on Machine Learning, Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

residual-guided feature extraction. However, neither method directly orders features by their effect on model behavior. Their feature priorities depend mainly on reconstruction objectives, which can favor features that reconstruct activations accurately but have little effect on model behavior.

In this work, we introduce *Distilled Matryoshka Sparse Autoencoders* (DMSAEs), an attribution-guided extension of Matryoshka SAEs that distills a compact, reusable set of consistently high-value latent features. DMSAEs proceed in a sequence of distillation cycles. In each cycle, we train a Matryoshka SAE with a designated shared core of latents that contributes to every reconstruction. After training, we calculate a gradient $\times$ activation score for each candidate latent using the unmasked ReLU activation and the next-token loss gradient. We select the smallest set whose cumulative score reaches a fixed fraction of the total score. DMSAEs use repeated selection and retraining to identify encoder directions that remain useful across runs and sparsity levels.

Contributions

(1) *Attribution-guided feature distillation.* We introduce DMSAEs, which use an iterative train-and-select procedure and gradient $\times$ activation attribution to distill a compact latent core that is relevant to model behavior.

(2) *Feature consistency via transferable encoder directions.* Across cycles, DMSAEs transfer and freeze only the core encoder weight vectors, which keeps the encoder detection directions stable across checkpoints and Matryoshka prefixes, while thresholds and decoder directions are re-learned.

(3) *Improved SAEBench performance.* For layer 12 activations from Gemma-2-2B, we use seven cycles to distill a core of 197 latents and improve several SAEBench metrics compared with Matryoshka SAE baselines.

2. Background

2.1. Sparse Autoencoders can disentangle polysemantic features

Early mechanistic interpretability work focused on inspecting individual latents, attention heads, and MLP units. However, polysemanticity complicated these approaches because a single unit could activate for several unrelated features depending on the context (Olah et al., 2020; Elhage et al., 2022). Sparse autoencoders (SAEs) were proposed as a way to move from neuron-level to feature-level analysis by learning a sparse, overcomplete basis over model activations (Bricken et al., 2023; Cunningham et al., 2023).

Given an activation vector $x \in \mathbb{R}^d$ taken from a layer of a language model, an SAE learns an encoder and decoder

$$f(x) = \mathrm{ReLU}(W_{\mathrm{enc}}x + b_{\mathrm{enc}}), \tag{1}$$

$$\hat{x} = W_{\mathrm{dec}}f(x) + b_{\mathrm{dec}}. \tag{2}$$

where $f(x) \in \mathbb{R}^m$ is constrained to be sparse and nonnegative, and each dimension of f is interpreted as a candidate feature. SAE variants differ primarily in how sparsity is enforced, for example ℓ_1 penalties, TopK-style ℓ_0 constraints, or learned thresholds (Gao et al., 2024; Bussmann et al., 2024).

This framework supports many interpretability applications, including manual feature analysis (Bricken et al., 2023), automated feature labelling (Gao et al., 2024), circuit discovery (Marks et al., 2024), attention-head analysis (Kissane et al., 2024), and model comparison with sparse crosscoders (Lindsey et al., 2024). Ideally, SAEs map model activations to sparse latents that represent distinct features that humans can interpret.

2.2. Limitations of Sparse Autoencoders

In practice, however, current SAEs do not always achieve this ideal. One key consideration is that SAE training minimizes a loss of the form

$$\mathcal{L}(x) = \|x - \hat{x}\|_2^2 + \lambda\,\mathrm{sparsity}(f(x))$$

However, neither term directly targets model behavior. The reconstruction term preserves activation information, while the sparsity term promotes sparse representations as a proxy for interpretability.

Feature splitting, absorption, and composition. When the dictionary is large relative to the intrinsic dimensionality of the activations, sparsity pressure can fragment broad concepts into many narrowly specialized latents (Bricken et al., 2023). For example, a single "punctuation marks" feature can split into separate latents for commas, periods, and question marks, while no latent represents the more abstract category that is relevant to model behavior. As the dictionary size increases, this fragmentation often produces latents that are more selective versions of existing features, rather than genuinely new concepts (Leask et al., 2025; Karvonen et al., 2025).

Feature absorption is a related failure mode that is more difficult to measure (Chanin et al., 2024). Instead of splitting fully, a general latent develops systematic gaps when more specific latents take over. For example, a numerical latent can stop firing on dates or percentages after more specialized latents emerge. As a result, the latent's concept coverage decreases over time, and the feature becomes less reliable and more difficult to interpret.

Sparsity can also encourage composite features when factors frequently occur together, e.g. "red triangle" instead of separate "red" and "triangle" features (Wattenberg & Viégas, 2024). Meta-SAE analyses suggest that many latents are not fully independent and can often be approximated as sparse mixtures of more basic "meta-latents" (Leask et al., 2025). These results indicate that SAE dictionaries can contain overlapping latent structure, which is also consistent with high cosine similarity among decoder vectors (Gao et al., 2024; Bussmann et al., 2024).

Scaling pathologies and downstream performance. As dictionary size increases, the problems become more severe. Larger SAEs consistently achieve lower reconstruction loss (Gao et al., 2024; Templeton et al., 2024), but their performance on downstream tasks often plateaus or decreases (Karvonen et al., 2025). One possible explanation is that the additional capacity lets the model reorganize information to improve the reconstruction and sparsity objective without improving disentanglement into features that humans can interpret.

Recent benchmarks reinforce a cautious view of current SAE methods. SAEBench compares many SAE architectures with simple baselines and finds that, when labeled data is available, linear probes and shallow MLPs trained directly on raw activations often match or exceed the performance of SAE-derived features (Karvonen et al., 2025). Thus, current SAEs do not consistently outperform other methods on supervised downstream tasks and introduce trade-offs between reconstruction, sparsity, and interpretability. Their main advantage is unsupervised feature discovery, as they can identify structure in model activations without curated labels (Bricken et al., 2023; Templeton et al., 2024).

In summary, SAEs still require architectural improvements to discover stable, monosemantic features reliably at scale.

2.3. Matryoshka SAEs can address some SAE limitations

Matryoshka Representation Learning (MRL) (Kusupati et al., 2022) learns a single representation with a coarse-to-fine structure by training on nested prefixes of the embedding: early coordinates capture broad patterns, and later coordinates add detail.

Matryoshka Sparse Autoencoders (MSAEs) apply the same idea to SAEs by learning multiple nested dictionaries inside one autoencoder (Bussmann et al., 2025). Given prefix sizes $M = \{m_1 < m_2 < \cdots < m_n\}$, MSAEs minimize reconstruction losses using only the first m_i latents for each $m_i \in M$. This forces early latents to reconstruct the input on their own, encouraging them to capture common, general features, while later latents can focus on more specific refinements without overwriting the earlier ones.

On SAEBench with Gemma-2-2B, MSAEs achieve strong reconstruction while reducing absorption, splitting, and composition compared to other SAE variants (Karvonen et al., 2025). They also perform well on sparse probing, targeted concept removal, and spurious correlation removal (Karvonen et al., 2025). However, Matryoshka structure does not fully solve redundancy or ensure that the same useful features reappear across runs or sparsity levels, and training many prefixes adds overhead. This motivates work that goes beyond Matryoshka SAEs to further reduce redundancy and stabilize feature meanings.

3. Distilled Matryoshka Sparse Autoencoders

DMSAEs build on Matryoshka SAEs by reserving a small set of latent features, termed the "core", that is used in every prefix reconstruction during a training run. After each run, we rank candidate latents by their loss-sensitivity scores and select a new core from the highest-scoring latents. We use the term distillation to refer to selecting a compact set of encoder directions that stay useful across training cycles. Unless otherwise stated, each DMSAE training cycle is trained on 500 million tokens with a fixed dictionary size of $K = 65{,}536$.

Core placement and prefix structure. Let the total dictionary size be K (in our experiments we set $K = 65536$, matching SAEBench). We index cycles by $t = 0, 1, \ldots, T$, where $t = 0$ denotes the released SAEBench Matryoshka SAE checkpoint (i.e. trained externally) and cycles $t \geq 1$ are the train-and-select training runs in our distillation procedure. We denote $C^{(t)}$ as the set of latents selected after cycle t by the attribution rule below (for $t = 0$ this selection is computed directly from the SAEBench checkpoint) and then used as the frozen core in cycle $t{+}1$. In cycle $t{+}1$, we place the core latents as the first $|C^{(t)}|$ indices in the dictionary, followed by non-core latents partitioned into Matryoshka groups. Let $G = (g_0, \ldots, g_L)$ denote the widths of the non-core Matryoshka groups. For a cycle with core size c, we keep the total dictionary width fixed by using

$$G_c = (g_0, \ldots, g_{L-1}, g_L - c),$$

so that

$$c + \sum_{\ell=0}^{L} (G_c)_\ell = K.$$

We then define the cumulative non-core prefix sizes by

$$m_\ell(c) = \sum_{i=0}^{\ell} (G_c)_i, \qquad \ell = 0, \ldots, L.$$

The reconstruction losses begin at *core* + *prefix* 0, which contains the core and the first $m_0(c)$ non-core latents, then *core* + *prefix* 1, and so on. There is no reconstruction term

for the core alone. An overview of the train-and-select workflow is shown in Figure 1. Unless otherwise stated, $C^{(t)}$ contains indices local to checkpoint t. When selected encoder rows are transferred into the next checkpoint, their identities are tracked through this transfer; reinitialized non-core rows receive new identities.

An overview of the train-and-select workflow is shown in Figure 1. Unless otherwise stated, $C^{(t)}$ contains indices local to checkpoint t. When selected encoder rows are transferred into the next checkpoint, their identities are tracked through this transfer; reinitialized non-core rows receive new identities.

Core and calibrated global sparsity. DMSAEs split the latent dictionary into a *core* and a *non-core* set. Although there is no reconstruction loss for the core alone, the core contributes to every Matryoshka reconstruction during each training cycle, and its encoder weight vectors are frozen. The non-core latents remain fully trainable.

During distillation training cycles, we use a calibrated global BatchTopK rule. BatchTopK is applied across both core and non-core latents, so the core still participates in the global competition. However, the global BatchTopK level is dynamically adjusted using only the measured non-core activity, so that the expected non-core L_0 remains approximately $k_{\text{non-core}}$ active non-core latents per token. Thus, the core is not counted toward the target activity budget, even though it is still affected by the global BatchTopK mask. Fixing this non-core activity budget keeps non-core representations approximately consistent across distillation cycles despite changes in core size.

$$\begin{aligned} f(x;\theta) &= \mathrm{ReLU}(W_{\text{enc}}x + b_{\text{enc}}), \\ \tilde{f}(x;\theta,c,k_{\text{glob}}) &\coloneqq \mathrm{BatchTopK}_{k_{\text{glob}}}\big(f(x;\theta)\big), \end{aligned} \tag{3}$$

$$\begin{aligned} &k_{\text{glob}} \text{ is adjusted online so that} \\ &L_{0,\text{non-core}} = \mathbb{E}_x\left[\left\|\tilde{f}_{c:K}(x)\right\|_0\right] \approx k_{\text{non-core}}. \end{aligned} \tag{4}$$

We refer to this as the dense-core variant because core activity is measured rather than directly targeted: core latents compete in the global BatchTopK mask, but they are not included in the target activity budget used to calibrate that mask. We tune the global BatchTopK level so that non-core activity stays near $k_{\text{non-core}}$, while measuring the resulting core activity empirically. Each reconstruction loss uses the core plus a growing prefix of the non-core latents (of length m). When benchmarking at a nominal SAEBench target sparsity k for a total dictionary size K, we scale the non-core activity target to

$$k_{\text{non-core}} \;=\; \mathrm{round}\left(k \cdot \frac{K - |C^{\star}|}{K}\right), \tag{5}$$

so that the fraction of non-core latents permitted to be active matches the baseline setting. In the dense-core variant, the measured global activity decomposes as in Eq. (10).

Algorithm 1 (Appendix F) provides end-to-end pseudocode for the training and selection distillation procedure, including (i) cycle indexing and core placement, (ii) within cycle Matryoshka training with dense core, and (iii) attribution-based core replacement. The attribution ranking and coverage-based selection rule used to form $C^{(t)}$ from the candidate pool $P^{(t)}$ are given explicitly in Algorithm 2 (Appendix F).

Training objective. Let $\hat{x}_{m_\ell(c)}(x)$ denote the reconstruction using the core latents together with the first $m_\ell(c)$ non-core latents, and let $w_\ell \geq 0$ denote the configurable weight assigned to reconstruction level ℓ. Each cycle minimizes the Matryoshka reconstruction loss

$$\mathcal{L}_{\text{DMSAE}}(x) = \frac{1}{L+1}\sum_{\ell=0}^{L} w_\ell \left\|x - \hat{x}_{m_\ell(c)}(x)\right\|_2^2 + \alpha\, \mathcal{L}_{\text{aux}}(x). \tag{6}$$

where $\mathcal{L}_{\text{aux}}$ denotes the standard auxiliary loss used in BatchTopK-style SAEs.

Transfer and reinitialization across cycles. After cycle t, we select a core set $C^{(t)}$ (by attribution below). To start cycle $t{+}1$, we restart by transferring the encoder rows for these latents into the first $c = |C^{(t)}|$ rows of W_{enc} and freezing rows $0{:}c$ during cycle $t{+}1$. Importantly, the encoder biases are not transferred. We reinitialize (i) all non-core encoder weights, (ii) all encoder biases (core and non-core), and (iii) the entire decoder $(W_{\text{dec}}, b_{\text{dec}})$. Thus, only the frozen encoder directions transfer between cycles. A transferred latent preserves the same projection direction $w^\top x$ during the next cycle and continues to later cycles only if it is selected again.

Feature consistency. We use the term feature consistency to mean invariance of the encoder weight vector for a latent. When a latent is promoted into the core, we copy its encoder row w_j^{enc} and freeze it during the next cycle. If it is selected again, it is copied and frozen again, preserving the projection direction $(w_j^{\text{enc}})^\top x$ across each restart that it survives. We reinitialize encoder biases and the decoder each cycle and compute attribution using decoder directions (Appendix G).

Attribution-guided core selection. After each cycle t, we score each candidate latent in the *core + prefix* 0 pool. For

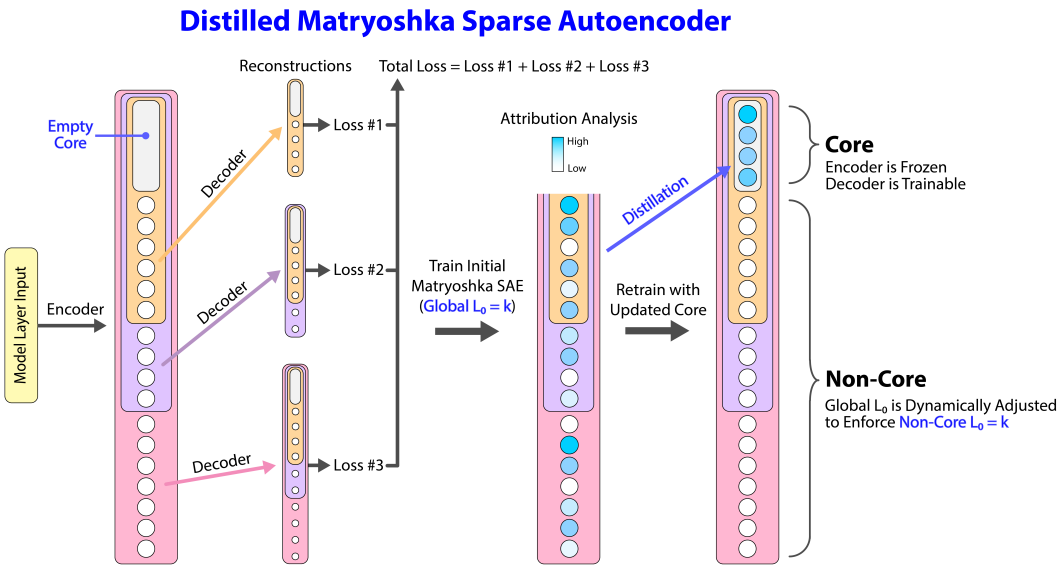

Figure 1. **DMSAE distillation schematic.** Each cycle trains a Matryoshka SAE where the non-core latents are divided into groups of specified widths, the smallest reconstruction uses *core* + *prefix* 0, and larger reconstructions cumulatively add subsequent non-core groups. After training, we score latents in *core* + *prefix* 0 using a gradient $\times$ activation score for next-token loss, and then choose the smallest set that accounts for a target fraction (τ) of the total attribution. This set becomes the core for the next cycle. Across cycles, we transfer and freeze only the core encoder weight vectors, while core encoder biases, core decoder weights and biases, and all non-core latents are randomly reinitialized. Dictionary size $K = 65536$

token position u, let $x_u \in \mathbb{R}^d$ be the captured activation vector (e.g. a residual-stream vector) and let

$$g_u \;=\; \frac{\partial L_{\mathrm{NT}}}{\partial x_u} \in \mathbb{R}^d$$

be the gradient of next-token cross-entropy loss with respect to that activation. Let $a_{u,j} = f_j(x_u)$ be the post-ReLU SAE activation of candidate latent j before BatchTopK. The candidate pool contains the core and prefix 0. Let w_j^{dec} be the decoder direction of latent j. Define its unit direction as $\bar{w}_j^{\mathrm{dec}} = w_j^{\mathrm{dec}}/\|w_j^{\mathrm{dec}}\|_2$. We define the per-token gradient projection

$$s_{u,j} \;=\; g_u^\top \bar{w}_j^{\mathrm{dec}},$$

and the per-token attribution magnitude (gradient $\times$ activation)

$$\mathrm{G} \times \mathrm{A}_{u,j} \;=\; \big|a_{u,j}\, s_{u,j}\big|.$$

We take the absolute value to include positive and negative loss sensitivities.

To aggregate over sampled token positions, we use a high quantile (e.g. q=0.999) because per-token $|a_{u,j}s_{u,j}|$ is typically heavy-tailed; a high quantile emphasizes upper-tail attribution magnitude while being less sensitive than a max to single outliers:

$$A_j \;=\; \mathrm{Quantile}_u\big(\mathrm{G} \times \mathrm{A}_{u,j}; q\big). \tag{7}$$

Gradient $\times$ activation arises as a first-order proxy for the loss change under a perturbation that removes latent j's unmasked decoded vector (Appendix G):

$$\Delta L \approx -a_{u,j}\, g_u^\top w_j^{\mathrm{dec}}.$$

We use the unit decoder direction $\bar{w}_j^{\mathrm{dec}}$ so the score measures directional sensitivity, rather than decoder norms. Thus, A_j should be interpreted as a ranking statistic for loss-sensitive directions rather than an exact prediction of the loss change.

$$C^{(t)} \;=\; \arg\min_{S \subseteq P^{(t)}} \left\{ |S| \;:\; \sum_{j \in S} A_j \geq \tau \sum_{j \in P^{(t)}} A_j \right\}. \tag{8}$$

In practice we compute these scores over sampled token positions and then form $C^{(t)}$ by sorting and taking the smallest prefix achieving attribution coverage τ; see Algorithm 2 (Appendix F) for pseudocode. Here, $P^{(t)}$ is the candidate pool, defined as exactly the latents decoded in the smallest reconstruction loss term in cycle t (the current core union prefix 0). In our experiments we use τ=0.9.

Although the attribution candidate pool is restricted to the latents decoded in the smallest reconstruction term, attribution uses their unmasked post-ReLU activations. Training uses the calibrated global BatchTopK mask.

Two-stage training procedure: core distillation and distilled-core transfer. The full train-and-select core distillation stage corresponds to running Algorithm 1 for T cycles at a fixed non-core activity target (e.g. $k_{\text{non-core}}$=320), while the distilled-core transfer stage (training new DMSAEs at multiple target sparsities using the fixed $C^\star$) is summarized in Algorithm 3 (Appendix F).

Core distillation. We run T train-and-select cycles at a fixed non-core activity target (here $k_{\text{non-core}}$=320). In each cycle, we train a DMSAE, compute attribution scores for latents in the smallest reconstruction prefix (core + prefix 0), and define the next cycle's core $C^{(t)}$ as the smallest subset whose cumulative attribution reaches the target coverage. Repeating this yields a sequence of candidate cores $\{C^{(t)}\}$ (Fig. 2).

Distilled-core selection and transfer.

We then extract a compact set of latents from cycle 7 that persisted across the final restart. Because the core selected after cycle $T-1$ is transferred into the first $|C^{(T-1)}|$ rows of checkpoint T, we define the distilled core as the cycle T indices that belong to this copied range and are selected again after cycle T:

$$C^\star := C^{(T)} \cap \left\{0, \ldots, |C^{(T-1)}| - 1\right\}, \tag{9}$$

yielding 197 latents. We train new DMSAEs from scratch at multiple sparsity targets, freezing only the encoder weight vectors indexed by $C^\star$ and reinitializing all remaining parameters; these models are the ones evaluated by SAEBench.

Core/non-core sparsity variants. Our main results use the dense core DMSAE variant; Appendix E reports a sparse core ablation for direct sparsity-matched comparison to SAEBench.

Dense core variant. In the dense-core variant, BatchTopK is applied globally across both core and non-core latents. However, the BatchTopK level is dynamically calibrated using only the measured non-core activity, so that $L_{0,\text{non}-\text{core}} \approx k_{\text{non}-\text{core}}$. Thus, the core participates in the global competition, but it is not included in the target activity budget. For SAEBench plots indexed by a nominal target sparsity k and total dictionary size K, we use the scaled non-core target $k_{\text{non-core}}$ defined in Eq. (5).

The effective expected global sparsity is

$$L_{0,\text{global}} = L_{0,\text{core}} + L_{0,\text{non-core}}. \tag{10}$$

In the dense core variant, BatchTopK calibration keeps $L_{0,\text{non-core}} \approx k_{\text{non-core}}$ by construction, and $L_{0,\text{core}}$ is measured empirically.

Sparse core variant. We apply the standard global BatchTopK constraint with fixed target k across the full dictionary, so both core and non-core activity count toward the same global activity budget. The only architectural difference from vanilla MSAE is that the core encoder weight vectors are frozen.

4. Experiments

We evaluate Distilled Matryoshka SAEs (DMSAEs) on the SAEBench interpretability benchmark suite using activations from layer 12 of Gemma-2-2B. All SAEs use a dictionary size of $K = 65,536$, matching SAEBench. Our experiments proceed in three stages: (1) selecting a sparsity level and promotion threshold that yield stable cores under distillation, (2) running a longer distillation schedule at the chosen hyperparameters to obtain a compact "distilled core", and (3) benchmarking DMSAEs built from this distilled core against the Matryoshka SAEs in SAEBench. Throughout, we follow the SAEBench evaluation protocol and report the same set of metrics.

Reproducibility details. We follow the SAEBench training and evaluation protocol for the released Matryoshka SAE checkpoint and for all newly trained Matryoshka SAE baselines, BatchTopK baselines, and DMSAE runs. The only differences from the SAEBench protocol are the freezing and masking rules described in Algorithms 1 and 3.

Code availability. DMSAE is available at https://github.com/cristinamartinlinares/distilled_matryoshka_sae.

Selecting sparsity and promotion threshold. We sweep over target sparsities $k \in \{20, 40, 80, 120, \ldots, 640\}$ and run four train-and-select training cycles per setting (cycles 1-4), starting from the SAEBench Matryoshka SAE checkpoint (cycle 0). In each cycle, we compute gradient $\times$ activation attribution scores as defined in Eq. (7) and form an attribution coverage curve: the cumulative fraction of cumulative attribution score captured by the top-ranked latents.

We first sweep the attribution coverage threshold (τ) from τ=0.1 to τ=1.0 and find that promoting more latents steadily reduces reconstruction loss and increases fraction of variance explained, with τ=0.9 close to τ=1.0 (Figure 5).

Fixing the coverage threshold at τ=0.9, we then compare the carryover across cycles of selected core latents as a function of sparsity. As shown in Figure 6, very low sparsities (k = 20-80) yield little carryover, while very large k (a weak sparsity constraint / high activity, e.g. k=640) can lead to degenerate optimization behavior (e.g. collapse at k=640). Intermediate sparsities exhibit substantially higher

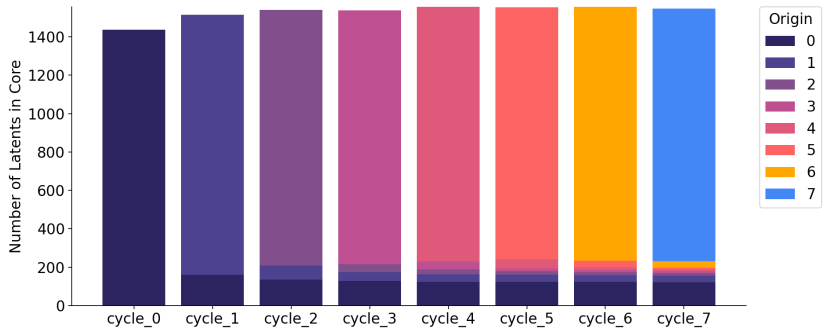

Figure 2. Evolution of core latents across distillation cycles (non-core activity target $k_{\text{non-core}}$=320, selection threshold τ=0.9). After each training cycle t (including $t = 0$, the SAEBench checkpoint), we rank candidate latents by their unmasked directional-sensitivity scores for the language model's next-token loss and select the smallest set $C^{(t)}$ whose cumulative score reaches 90% of the total attribution-score mass. The y-axis shows the number of latents selected into the core at the end of cycle t. Colors track encoder-row identities through explicit copies and indicate the cycle in which each direction first entered a selected core, showing how later cycles combine previously discovered directions with newly identified ones. For benchmarking, we retained the cycle 7 latents that were copied from the cycle 6 core and selected again after cycle 7, yielding 197 latents.

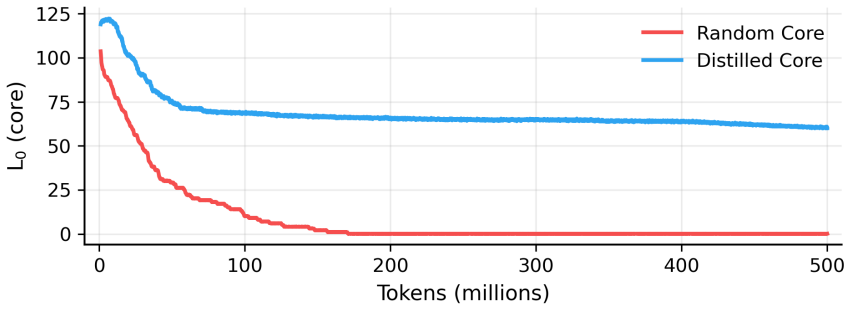

Figure 3. Distillation uncovers useful latents. We trained two DMSAEs with non-core activity target $k_{\text{non-core}} = 320$, one that uses a core derived from our distillation procedure (blue; Fig. 2, cycle 7) and another that uses a randomly initialized core of equivalent size (red). The y-axis shows the mean number of core latents active per sample during training ($L_{0,\text{core}}$). The model with a randomly initialized core drives $L_{0,\text{core}}$ towards zero during training, while the model with the distilled core continues to rely on the core throughout training. This suggests that distillation identifies core latents that are systematically useful for reconstruction.

reuse across cycles; we therefore focus subsequent experiments on this intermediate regime and use $k_{\text{non-core}}$=320 in all dense-core distillation runs reported below.

Multi-cycle distillation and the distilled core. Having identified $k_{\text{non-core}}$=320 and an attribution coverage threshold τ=0.9 as our working hyperparameters, we run T=7 train-and-select cycles, training for 500M tokens from The Pile each cycle. We denote by cycle 0 the released SAEBench Matryoshka SAE checkpoint (not trained by us) and compute an initial promoted core $C^{(0)}$ from this checkpoint using Eq. (7) to seed cycle 1. Each subsequent cycle is a restart: we transfer and freeze only the selected encoder weight vectors for the next cycle's core, and we reinitialize all non-core parameters and the entire decoder.

At the end of each cycle t, we score latents in the smallest reconstruction prefix (core + prefix 0) using the gradient $\times$ activation attribution in Eq. (7), sort latents by attribution, and select the smallest set $C^{(t)}$ whose cumulative attribution reaches 90% of the total attribution score. Figure 2 summarizes how the selected sets evolve across cycles and shows that, while many latents are newly selected in each cycle, a subset persists across multiple restarts.

After completing T=7 cycles, we extract a compact distilled core by retaining only latents that persist through successive training cycles. We define the distilled core $C^\star$ as in Eq. (9).

However, to first test whether $C^\star$ reflects useful encoder directions, we repeat the same procedure with a randomly initialized core of the same size. While the non-core activity target is fixed by construction (Eq. (5)), the random-core model drives $L_{0,\text{core}}$ toward zero during training, whereas the distilled-core model continues to rely on the core for reconstruction (Figure 3). This provides evidence that the distilled encoder directions remain useful under retraining, whereas randomly initialized directions do not.

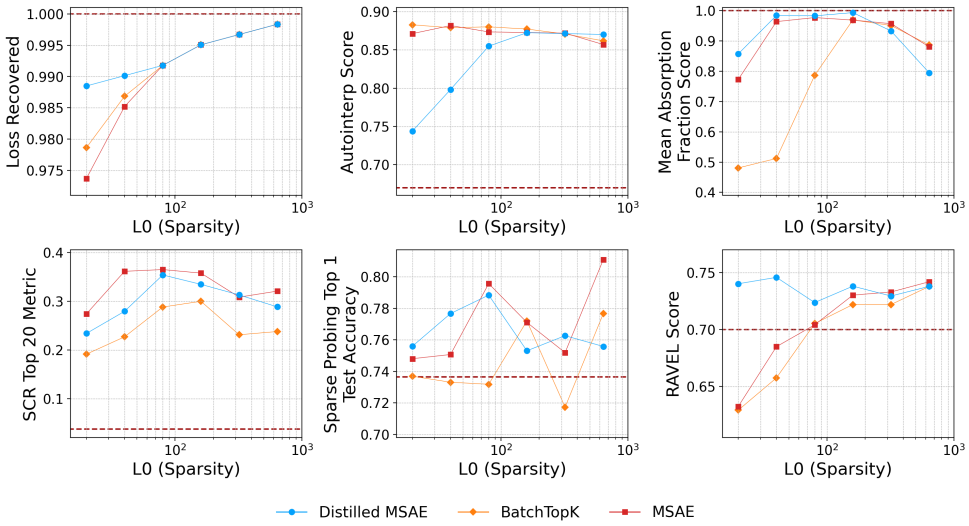

Figure 4. SAEBench performance across sparsities (dense core DMSAE). SAEBench metrics versus measured sparsity (mean number of active latents) for Distilled MSAE (DMSAE) with a dense core, compared with Matryoshka SAE (MSAE) and BatchTopK, evaluated on Gemma-2-2B layer 12 activations (all SAEs have a dictionary size of 65k). Here, the DMSAE core (197 latents) is taken from the distilled core produced by the $k_{\text{non-core}}$=320 distillation run (Figure 2). We then used this core to train new DMSAEs from scratch at each target sparsity ($k \in \{20, 40, 80, 160, 320, 640\}$). Core encoder weights are frozen while the global BatchTopK level is calibrated to match the target non-core activity. DMSAE matches or improves several metrics, particularly absorption and RAVEL, but AutoInterp drops sharply at the lowest sparsities. Appendix E provides a sparse core variant that matches the vanilla Matryoshka SAE setup used in SAEBench by applying a fixed global BatchTopK target across the full dictionary, including the core (Figure 7). The resulting trends are qualitatively consistent with the dense-core setting.

Building DMSAEs from the distilled core. Using the distilled core $C^\star$ (197 latents), we train new models at each target sparsity by randomly initializing all non-core parameters, initializing the core encoder rows from $C^\star$, and freezing only those rows. Thus, each benchmark model costs the same as a SAEBench baseline to train; the multi-cycle procedure is used only to identify $C^\star$ at $k_{\text{non-core}} = 320$.

For main results we use the dense-core regime. We apply global BatchTopK to the full dictionary, but its level is calibrated so that non-core activity matches Eq. (5). Figure 4 shows that dense-core DMSAEs match or improve several SAEBench metrics relative to MSAEs, especially absorption and RAVEL, while AutoInterp drops at very low sparsities ($k = 20$-80). We think that this decrease occurs because the sparsity regimes do not match. A core distilled at an intermediate activity level might not be suitable for an extreme bottleneck. A core distilled specifically for low-k models, or the sparse-core variant, could possibly reduce this problem. Appendix E reports the sparse-core curves with the standard fixed global BatchTopK budget. Algorithm 3 summarizes the dense-core transfer setting; in the sparse-core setting, k_{glob} is fixed to k and the non-core calibration is omitted.

5. Limitations

The DMSAE method uses multiple distillation cycles to identify a stable core, so its computational cost is much higher than the cost of training one Matryoshka SAE. In our setting, core discovery uses $T = 7$ full train-and-select cycles, with 500M tokens in each cycle, while deployment of the distilled core uses only one standard training cycle at each target sparsity. We treat core discovery as a one-time construction cost because a core distilled for a given model/layer and training distribution can be reused across many downstream SAEs and analyses, which distributes the initial cost across these uses. After a high-value core is identified, training and evaluation of a DMSAE with a fixed core cost the same as training a standard Matryoshka SAE. We expect that useful latents can be identified more efficiently than with distillation across multiple training cycles.

Core selection depends on the attribution method. For example, gradient $\times$ activation is a fast first-order approximation, but may be suboptimal. Latent ablation offers a more direct measure of importance, but at a higher computational cost.

Freezing encoder weight vectors stabilizes each core latent's projection direction. However, because the core encoder biases and decoder weights and biases are reinitialized, a latent's activation region, firing rate, and contribution to reconstruction can still change across cycles. Therefore, the latent function and decoder direction are not invariant across cycles.

Finally, in the dense-core transfer setting, the distilled core

is included in the global BatchTopK mask but is not included in the target activity budget used to calibrate that mask. Because core activity is measured rather than fixed, this can increase the total number of active latents, so comparisons can depend on how sparsity is defined. Therefore, Figure 7 reports a sparsity-matched control using a fixed global BatchTopK target that includes the core for direct comparison to baseline Matryoshka SAEs, and these results show the same qualitative trends as dense core DMSAEs.

6. Conclusion

We introduce Distilled Matryoshka Sparse Autoencoders (DMSAEs), which use attribution-guided training to distill a compact core of reusable encoder directions from Matryoshka SAEs. By transferring and freezing only these encoder rows across cycles, DMSAEs enforce cross-run consistency at the level of encoder directions. On Gemma-2-2B layer 12 activations, distillation yields a 197-latent core that can be reused across sparsity targets and improves multiple SAEBench metrics compared to Matryoshka SAE baselines. Future work focuses on reducing the computational cost of distillation and extending the DMSAE framework to other models and scientific domains.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

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

Bussmann, B., Leask, P., and Nanda, N. Batchtopk sparse autoencoders. *arXiv preprint arXiv:2412.06410*, 2024.

Bussmann, B., Nabeshima, N., Karvonen, A., and Nanda, N. Learning multi-level features with matryoshka sparse autoencoders. *arXiv preprint arXiv:2503.17547*, 2025.

Candès, E. J., Li, X., Ma, Y., and Wright, J. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3): 1–37, 2011.

Chanin, D., Wilken-Smith, J., Dulka, T., Bhatnagar, H., Golechha, S., and Bloom, J. A is for absorption: Studying feature splitting and absorption in sparse autoencoders. *arXiv preprint arXiv:2409.14507*, 2024.

Costa, V., Fel, T., Lubana, E. S., Tolooshams, B., and Ba, D. Evaluating sparse autoencoders: From shallow design to matching pursuit. *arXiv preprint arXiv:2506.05239*, 2025.

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

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

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

Jalali, A., Sanghavi, S., Ruan, C., and Xing, E. P. A dirty model for multi-task learning. In *Advances in Neural Information Processing Systems*, 2010.

Jenatton, R., Mairal, J., Obozinski, G., and Bach, F. Proximal methods for hierarchical sparse coding. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

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 preprint arXiv:2503.09532*, 2025.

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

Kusupati, A., Bhatt, G., Rege, A., Wallingford, M., Sinha, A., Ramanujan, V., Howard-Snyder, W., Chen, K., Kakade, S., Jain, P., et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

Leask, P., Bussmann, B., Pearce, M., Bloom, J., Tigges, C., Moubayed, N. A., Sharkey, L., and Nanda, N. Sparse autoencoders do not find canonical units of analysis. *arXiv preprint arXiv:2502.04878*, 2025.

Lindsey, J., Templeton, A., Marcus, J., Conerly, T., Batson, J., and Olah, C. Sparse crosscoders for cross-layer features and model diffing. https://transformer-circuits.pub/2024/crosscoders/index.html, 2024. Accessed: 2025-12-26.

Marks, S., Rager, C., Michaud, E. J., Belinkov, Y., Bau, D., and Mueller, A. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. *arXiv preprint arXiv:2403.19647*, 2024.

Obozinski, G., Wainwright, M. J., and Jordan, M. I. Support union recovery in high-dimensional multivariate regression. In *Advances in Neural Information Processing Systems*, 2010.

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

Paulo, G. and Belrose, N. Sparse autoencoders trained on the same data learn different features. *arXiv preprint arXiv:2501.16615*, 2025.

Song, X., Muhamed, A., Zheng, Y., Kong, L., Tang, Z., Diab, M. T., Smith, V., and Zhang, K. Position: Mechanistic interpretability should prioritize feature consistency in saes. *arXiv preprint arXiv:2505.20254*, 2025.

Templeton, A., Conerly, T., Marcus, J., Lindsey, J., Bricken, T., Chen, B., Pearce, A., Citro, C., Ameisen, E., Jones, A., Cunningham, H., Turner, N. L., McDougall, C., MacDiarmid, M., Tamkin, A., Durmus, E., Hume, T., Mosconi, F., Freeman, C. D., Sumers, T. R., Rees, E., Batson, J., Jermyn, A., Carter, S., Olah, C., and Henighan, T. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/.

Wang, S. L., Quach, A., Parsan, N., and Yang, J. J. Enforcing orderedness to improve feature consistency. *arXiv preprint arXiv:2512.02194*, 2025.

Wattenberg, M. and Viégas, F. B. Relational composition in neural networks: A survey and call to action. *arXiv preprint arXiv:2407.14662*, 2024.

Yang, E. and Ravikumar, P. Dirty statistical models. In *Advances in Neural Information Processing Systems*, 2013.

Zhao, P., Rocha, G., and Yu, B. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics*, 37(6A):3468–3497, 2009.

A. Justification for sparsity decomposition

This appendix outlines the sparsity mechanisms used by DMSAEs. Let C and N denote the index sets of core and non-core latents. For an input x, let $f(x) \in \mathbb{R}^K$ be the latent vector (after applying ReLU but before any BatchTopK masking). Let $\tilde{f}(x)$ denote the latent vector actually used for reconstruction after applying the relevant sparsity mechanism. In dense-core variants, BatchTopK is applied globally across all K latents, but its level is calibrated using only the measured non-core activity. In sparse-core variants, BatchTopK is also applied globally across all K latents, but with a fixed global target k. We define the active sets using $\tilde{f}$:

$$S_C(x) = \{j \in C : \tilde{f}_j(x) > 0\}, \qquad S_N(x) = \{j \in N : \tilde{f}_j(x) > 0\}.$$

We denote the corresponding average sparsities by

$$L_{0,\text{core}} = \mathbb{E}_x |S_C(x)|, \qquad L_{0,\text{non-core}} = \mathbb{E}_x |S_N(x)|.$$

Remark (Non-core budget calibration). In the dense-core variant, BatchTopK is applied globally across core and non-core coordinates. The global BatchTopK level is calibrated to a fixed non-core target $\bar{L}_{0,\text{non-core}}$ (i.e., a fixed $k_{\text{non-core}}$). Thus, the expected non-core activity $L_{0,\text{non-core}}$ is approximately invariant across distillation cycles even as the core size changes. However, the global activity $L_{0,\text{global}}$ can still vary due to changes in $L_{0,\text{core}}$.

Proposition 1 (Global L_0 decomposition). Under the definitions above, the expected global sparsity decomposes as in Eq. (10).

Proof. $S_C(x)$ and $S_N(x)$ are disjoint by construction, so $|S_C(x) \cup S_N(x)| = |S_C(x)| + |S_N(x)|$. Taking expectations on both sides yields Eq. (10). □

This identity justifies Eq. (10). In the dense-core variant, the effective global sparsity increases by the observed $L_{0,\text{core}}$, while the non-core activity budget remains approximately fixed.

B. Connections to structured sparsity and multi-task feature learning

This appendix compares DMSAEs with several standard methods in representation learning and structured regularization. These comparisons show similarities only, they do not establish formal equivalence. DMSAEs use a nonconvex training objective and a BatchTopK-style activation constraint.

Shared-plus-sparse decomposition (multi-task / "dirty model"). A DMSAE learns a set of latent features and uses them for several related reconstruction tasks. Different Matryoshka prefix lengths and different distillation cycles produce these related tasks. The DMSAE uses a common set of *core* latent features for all tasks, while other latent features can adapt to one task and become task-specific. This structure is similar to multi-task "dirty model" methods. In these methods, the parameters of each task contain a shared component and a sparse task-specific component (Jalali et al., 2010; Yang & Ravikumar, 2013). These models can identify the active features and keep the error within a specified limit when two conditions apply. First, the shared structure and the task-specific structure have little overlap. Second, each task uses only a small number of task-specific features. These results support the idea that a small shared basis is useful when related tasks have the same underlying structure.

Hierarchical and structured sparsity. Since the core latent features are available in each Matryoshka prefix, a DMSAE has an implicit hierarchy. The core latent features provide a baseline representation. Additional non-core latent features add detail as the prefix length increases. This structure is similar to hierarchical or tree-structured sparsity. In these methods, a variable can become active only when its parent variables are active (Zhao et al., 2009; Jenatton et al., 2011). This structure often produces a coarse-to-fine representation. Higher-level groups represent broad patterns. Lower-level groups represent more specific details. A DMSAE produces a similar effect through its core and expanding prefixes. It does not use an explicit tree-structured penalty.

Backbone and residual perspective (low-rank plus sparse). Another analogy is to a low-rank and sparse decomposition, such as robust PCA. These methods separate the data into two components, one component representing a shared global

structure and the other component representing sparse, local differences (Candès et al., 2011). From this perspective, the DMSAE core is a dense backbone of features that are useful for many inputs. The remaining latent features form a sparse residual. This residual corrects the remaining reconstruction errors and represents more specialized patterns. This interpretation explains why frequent activation of the core can be useful. It also explains why the remaining latent features can use a fixed and limited activation budget.

Group-specific ℓ_0 constraints and the dynamic global expected ℓ_0 (L_0) schedule. The dynamic global L_0 rule in Eq. (10) is a practical substitute for the separate sparsity budgets for the core and non-core groups. In principle, one can set a target only for the non-core activity budget. The learned activations and the global BatchTopK mask can then determine the core activity:

$$\min_{\theta} \ \mathbb{E}_x\big[\mathcal{L}_{\text{recon}}(x;\theta)\big] \quad \text{with} \quad \mathbb{E}_x\big[\|\tilde{f}_N(x;\theta)\|_0\big] \approx \bar{L}_{0,\text{non-core}},$$

where $\tilde{f}_N(x)$ denotes the masked non-core activations. The method measures the corresponding core activity instead of setting a fixed target for it. This approach is similar to convex formulations that assign different weights to shared and task-specific groups, such as sparse group regularization (Obozinski et al., 2010).

In practice, our main dense-core DMSAE variant applies BatchTopK globally across both core and non-core latents. However, it calibrates the global BatchTopK level using only the measured non-core activity. This method keeps the non-core budget approximately constant across cycles and allows us to measure the core activity during training. The resulting total activity decomposes as $L_{0,\text{global}} = L_{0,\text{core}} + L_{0,\text{non-core}}$ (Eq. (10)). For a direct comparison with SAEBench, Appendix E also includes a sparse-core ablation. This ablation applies standard global BatchTopK with a fixed target k across all latents, including the core.

C. Choosing the attribution coverage threshold for core promotion

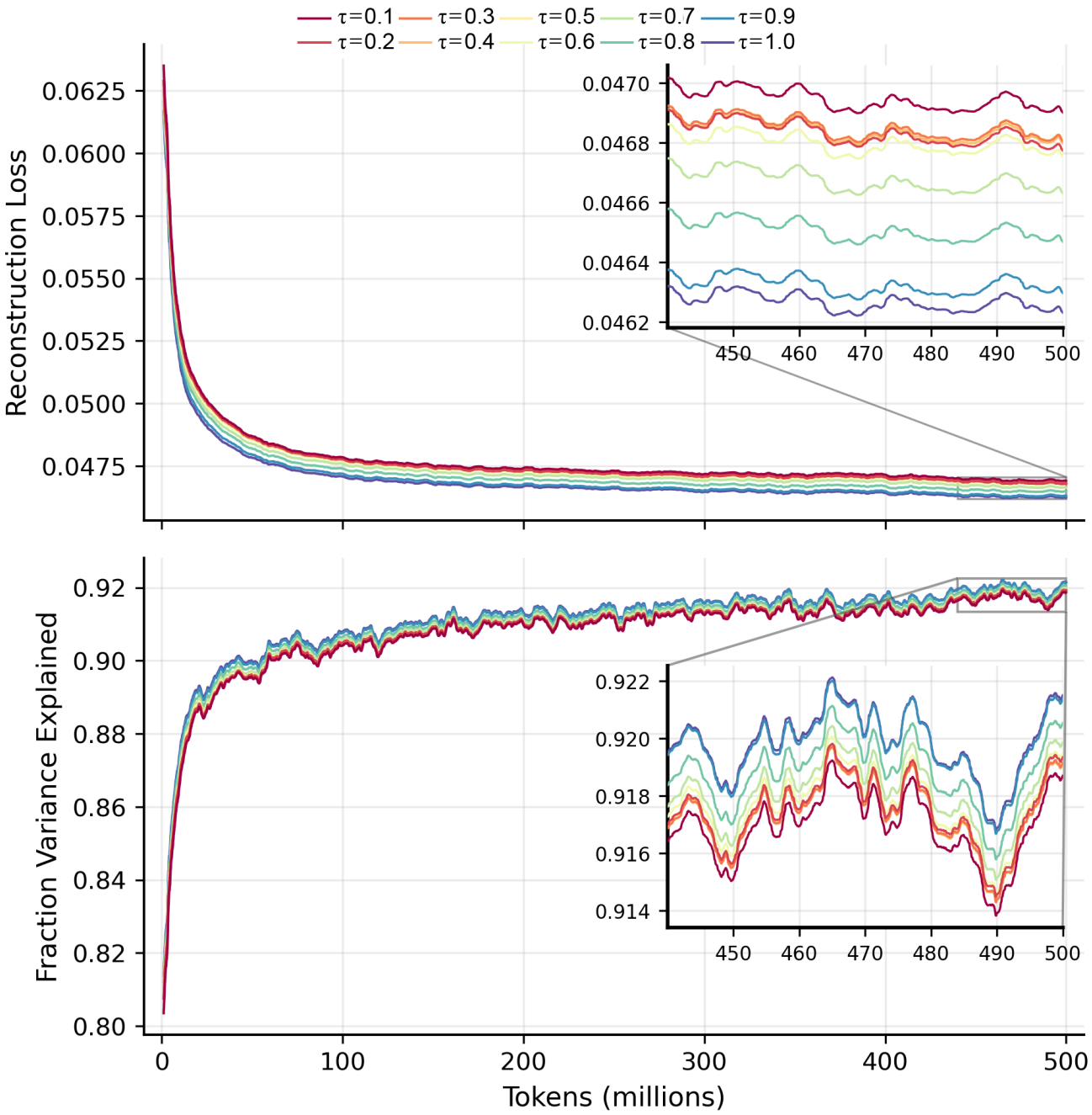

Figure 5. Choosing the attribution coverage threshold for core promotion. We trained dense-core DMSAEs ($k_{\text{non-core}}$=320) using different attribution coverage thresholds (τ). Promoting more latents steadily reduces reconstruction loss (top) and increases fraction of variance explained (bottom) throughout training, with no clear saturation up to τ=1.0. The τ=0.9 setting nearly overlaps τ=1.0 in fraction of variance explained and achieves similar reconstruction loss. These results motivate using τ=0.9 in our dense-core setting. Because core activity is not fixed in this setting, the curves do not isolate the effect of τ at matched global sparsity.

D. Core stability depends strongly on the sparsity target.

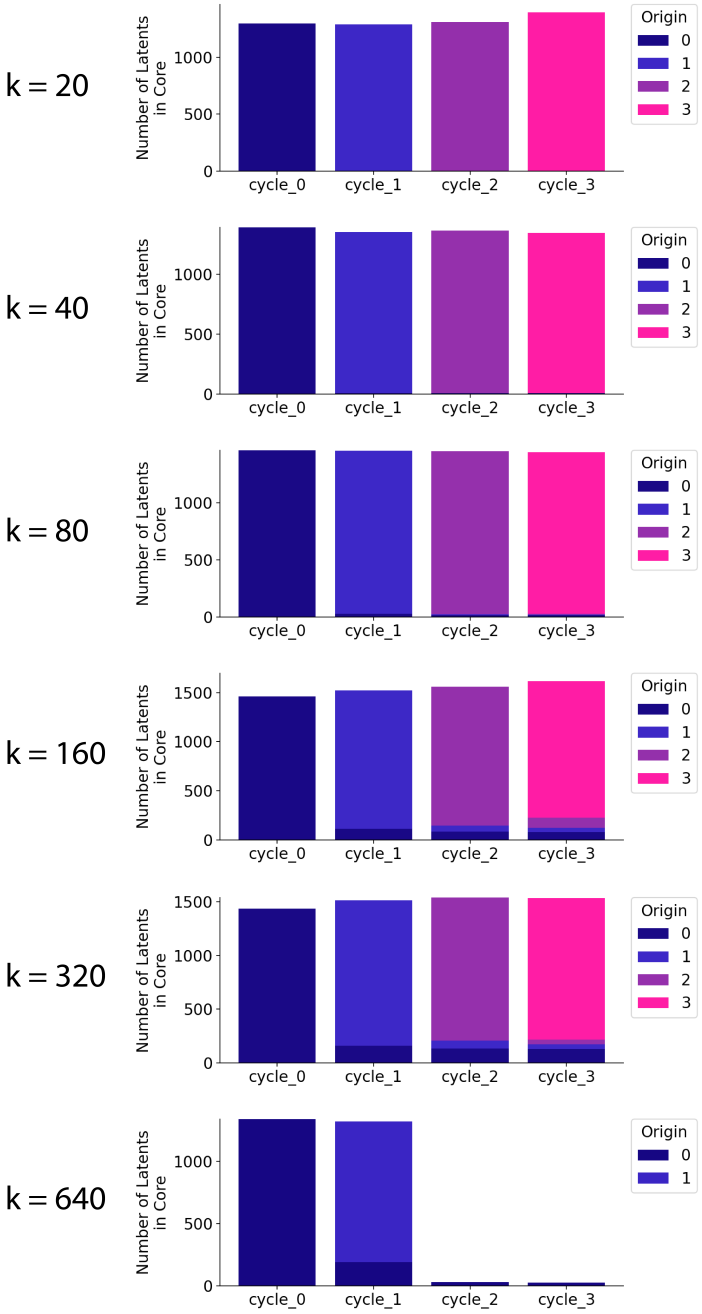

Figure 6. We ran four dense-core DMSAE distillation cycles for each non-core activity target $k_{\text{non-core}}$. After each cycle t, we score latents in the smallest reconstruction prefix (core + prefix 0) by gradient $\times$ activation attribution to next-token loss and select the smallest set whose cumulative score reaches 90% of the total attribution-score mass (τ=0.9) as the next cycle's core. Bars show the number of selected core latents per cycle and colors track encoder-row identities through explicit copies and indicate the cycle in which each direction first entered a selected core, visualizing carryover across cycles. Only the intermediate values ($k_{\text{non-core}}$=160 and $k_{\text{non-core}}$=320) exhibit substantial carryover into later cores. At $k_{\text{non-core}}$=640 (largest k, i.e., weakest sparsity constraint / highest activity), optimization degenerated after cycle 1 and reconstruction failed, thus we omitted subsequent cycles for k=640.

E. Core sparsity ablation on SAEBench.

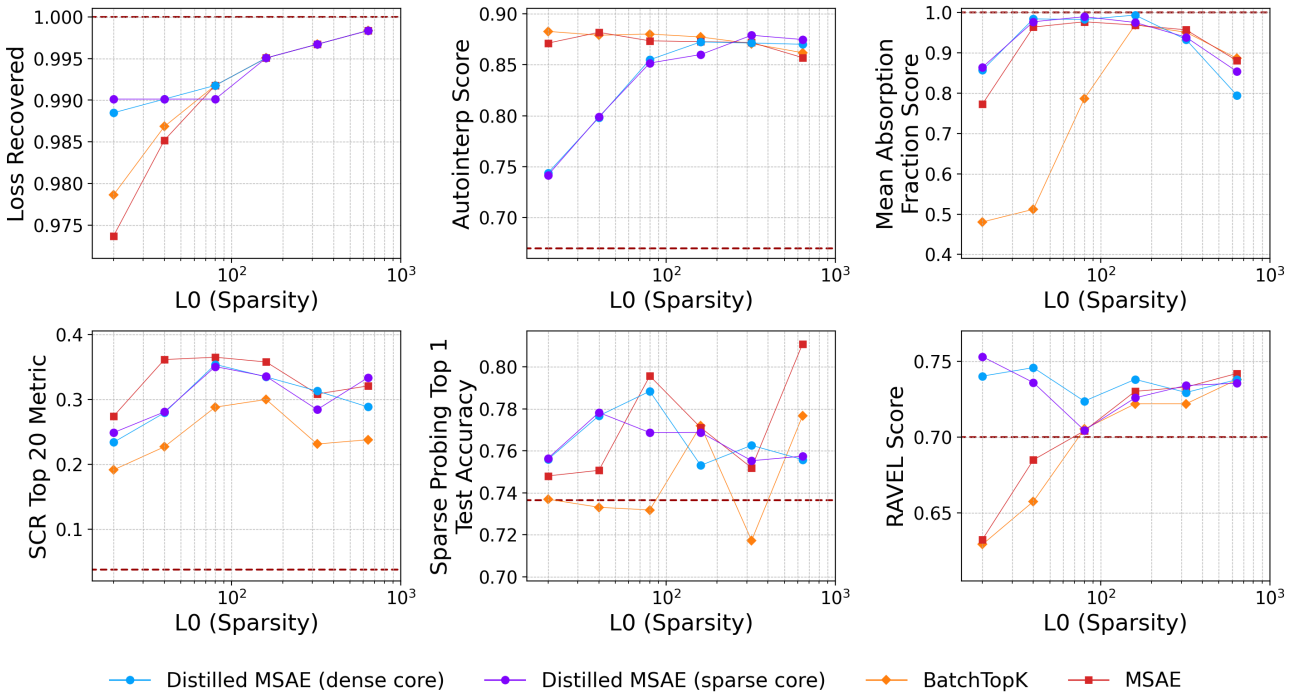

Figure 7. Performance across top k targets in the SAEBench (as in Fig. 4), but comparing two DMSAE sparsity rules using the same distilled core (197 latents). Dense core (blue) applies BatchTopK globally across core and non-core latents, but dynamically calibrates the global BatchTopK level using only the measured non-core activity. Sparse core (purple) applies the standard global BatchTopK constraint with fixed target k to the entire dictionary, including the core, matching the vanilla Matryoshka SAE setup for a more direct comparison. The sparse core DMSAE follows the same overall trends as dense core DMSAE. BatchTopK and Matryoshka SAE baselines from SAEBench are included for reference, and all SAEs were trained using a dictionary of size $K = 65536$.

F. Pseudocode for DMSAE training and core selection

This appendix provides pseudocode for: (i) the full train-and-select distillation loop (Algorithm 1), (ii) core selection by attribution from the candidate pool (Algorithm 2), and (iii) training a DMSAE from a fixed distilled core $C^\star$ (Algorithm 3).

Between cycles, we only transfer the encoder weight vectors for the selected core latents. We transfer them into the first c rows of W_{enc} and freeze those rows during the next cycle. Every cycle, we reset all encoder biases b_{enc}, all non-core encoder rows $W_{\text{enc}}[c{:}K, :]$, and the full decoder $(W_{\text{dec}}, b_{\text{dec}})$.

In the dense-core setting, $k_{\text{non-core}}$ is the target non-core activity. We apply a global $\mathrm{BatchTopK}_{k_{\text{glob}}}$ to $f(x)$ and update k_{glob} online so that $L_{0,\text{non-core}} \approx k_{\text{non-core}}$. For attribution, we use the unmasked post-ReLU activations of candidates in *core* + *prefix* 0. We do not apply BatchTopK during attribution.

Algorithm 1 DMSAE train-and-select distillation (dense core)

Require: SAEBench Matryoshka SAE checkpoint (cycle t=0); dataset $\mathcal{D}$; total width K; non-core group widths $G = (g_0, \ldots, g_L)$ satisfying $\sum_{\ell=0}^{L} g_\ell = K$; reconstruction weights $\{w_\ell\}_{\ell=0}^{L}$; auxiliary coefficient α; cycles T; non-core activity target $k_{\text{non-core}}$; quantile q; attribution coverage τ.

Ensure: Selected cores $\{C^{(t)}\}_{t=0}^{T}$ and distilled core $C^\star$.

Cycle 0 (no training): set $c \leftarrow 0$, $G_c \leftarrow G$, $m_\ell(c) \leftarrow \sum_{i=0}^{\ell} (G_c)_i$ for $\ell = 0, \ldots, L$, $P^{(0)} \leftarrow \{0, \ldots, m_0(c) - 1\}$, and compute $C^{(0)} \leftarrow \textsc{SelectCoreByAttribution}(\theta^{(0)}, P^{(0)}, q, \tau)$.

for $t = 1$ to T **do**

  $c \leftarrow |C^{(t-1)}|$.

  Set $G_c \leftarrow (g_0, \ldots, g_{L-1}, g_L - c)$ and $m_\ell(c) \leftarrow \sum_{i=0}^{\ell} (G_c)_i$ for $\ell = 0, \ldots, L$.

  Restart: initialize $\theta_0^{(t)}$**.**

  Set $W_{\text{enc}}[0{:}c, :] \leftarrow W_{\text{enc}}^{(t-1)}[C^{(t-1)}, :]$ and freeze rows $0{:}c$.

  Reinitialize: (i) all encoder biases b_{enc}, (ii) all non-core encoder rows $c{:}K$, and (iii) the entire decoder $(W_{\text{dec}}, b_{\text{dec}})$.

  $$\theta^{(t)} \leftarrow \arg\min_{\theta \in \Theta_t} \mathbb{E}_{x \sim \mathcal{D}} \left[\frac{1}{L+1} \sum_{\ell=0}^{L} w_\ell \| x - \hat{x}_{m_\ell(c)}(x; \theta, \tilde{f}) \|_2^2 + \alpha \, \mathcal{L}_{\text{aux}}(x; \theta, \tilde{f}) \right],$$

  where $f(x; \theta) = \mathrm{ReLU}(W_{\text{enc}} x + b_{\text{enc}})$,

  $\tilde{f}(x; \theta, c, k_{\text{glob}}) = \mathrm{BatchTopK}_{k_{\text{glob}}}(f(x; \theta))$,

  and k_{glob} is updated online so that $L_{0,\text{non-core}} \approx k_{\text{non-core}}$.

  Reselect core: define candidate pool $P^{(t)} = \{0, \ldots, c + m_0(c) - 1\}$ (core+prefix 0).

  $C^{(t)} \leftarrow \textsc{SelectCoreByAttribution}(\theta^{(t)}, P^{(t)}, q, \tau)$.

end for

Distilled core: $C^\star \leftarrow C^{(T)} \cap \{0, \ldots, |C^{(T-1)}| - 1\}$.

Algorithm 2 SELECTCOREBYATTRIBUTION(θ, P, q, τ)

Require: SAE parameters $\theta = \{W_{\text{enc}}, b_{\text{enc}}, W_{\text{dec}}, b_{\text{dec}}\}$; candidate pool P (core+prefix 0); quantile q; coverage τ.
Ensure: Ordered selected set $C \subseteq P$ for the next cycle.

Sample token positions U; obtain (x_u, g_u), where $g_u = \partial L_{\text{NT}}/\partial x_u$.
for each attribution batch $\{(x_u, g_u)\}$ **do**
  Compute $f_u \leftarrow \text{ReLU}(W_{\text{enc}} x_u + b_{\text{enc}}) \in \mathbb{R}^K$ for each u in the batch.
  for each latent $j \in P$ **do**
    $\bar{w}_j^{\text{dec}} \leftarrow W_{\text{dec}}[:, j]/\|W_{\text{dec}}[:, j]\|_2$.
    For each u in the batch: set $a_{u,j} \leftarrow f_{u,j}$, compute $s_{u,j} \leftarrow g_u^\top \bar{w}_j^{\text{dec}}$, and set $\text{G} \times \text{A}_{u,j} \leftarrow |a_{u,j} s_{u,j}|$.
  end for
end for
For each $j \in P$, set $A_j \leftarrow \text{Quantile}_q(\{\text{G} \times \text{A}_{u,j}\}_u)$.
Sort P by descending A_j: $(j_1, \ldots, j_{|P|})$; let $A_{\text{tot}} = \sum_{j \in P} A_j$.
If $A_{\text{tot}} = 0$, return $C = \varnothing$.
Return the smallest prefix $C = (j_1, \ldots, j_r)$ s.t. $\sum_{\ell=1}^{r} A_{j_\ell} \geq \tau\, A_{\text{tot}}$.

Algorithm 3 Training a DMSAE from a fixed distilled core $C^\star$ (core-transfer stage)

Require: Distilled core encoder directions $\{w_1^\star, \ldots, w_c^\star\}$ where $c = |C^\star|$; total width K;
non-core group widths $G = (g_0, \ldots, g_L)$ satisfying $\sum_{\ell=0}^{L} g_\ell = K$; target sparsity k; reconstruction weights $\{w_\ell\}_{\ell=0}^{L}$; auxiliary coefficient α; dataset $\mathcal{D}$; step size η.
Ensure: Trained DMSAE parameters $\theta^\star \in \Theta^\star$.

Initialize parameters $\theta = \{W_{\text{enc}}, b_{\text{enc}}, W_{\text{dec}}, b_{\text{dec}}\}$.
Set $W_{\text{enc}}[0{:}c, :] \leftarrow [w_1^\star; \ldots; w_c^\star]$ and freeze rows $0{:}c$.
Randomly initialize $W_{\text{enc}}[c{:}K, :]$, all b_{enc}, all W_{dec}, and b_{dec}.
Define feasible set $\Theta^\star := \{\theta :\ W_{\text{enc}}[0{:}c, :] = [w_1^\star; \ldots; w_c^\star]\}$.
Set $G_c \leftarrow (g_0, \ldots, g_{L-1}, g_L - c)$ and $m_\ell(c) \leftarrow \sum_{i=0}^{\ell} (G_c)_i$ for $\ell = 0, \ldots, L$.
Set $k_{\text{non-core}} \leftarrow \text{round}\big(k \cdot \frac{K-c}{K}\big)$.
for each mini-batch $X = \{x_b\}_{b=1}^{B}$ from $\mathcal{D}$ **do**
  Compute $F \leftarrow f(X; \theta) \ = \ \text{ReLU}(W_{\text{enc}} X^\top + b_{\text{enc}})^\top$.
  Set $\tilde{F} \leftarrow \text{BatchTopK}_{k_{\text{glob}}}(F)$, where k_{glob} is updated so that $L_{0,\text{non-core}} \approx k_{\text{non-core}}$.
  $\hat{\mathcal{J}}(X; \theta) \leftarrow \dfrac{1}{B(L+1)} \displaystyle\sum_{\ell=0}^{L} w_\ell \|X - \hat{X}_{m_\ell(c)}(X; \theta, \tilde{F})\|_F^2 + \alpha\, \mathcal{L}_{\text{aux}}(X; \theta, \tilde{F})$.
  $\theta \leftarrow \Pi_{\Theta^\star}\Big(\theta - \eta\, \nabla_\theta \hat{\mathcal{J}}(X; \theta)\Big)$.
end for
Return: $\theta^\star \leftarrow \theta$.

G. Additional derivations for gradient × activation attribution and coverage-based promotion

This appendix derives the gradient × activation (G × A) method that we use to rank latents. It also explains the attribution-coverage rule that we use to promote latents. These details complement the main text definitions of G × A and attribution coverage.

Let $x_u \in \mathbb{R}^d$ denote the activation vector at token position u, and let $L_{\text{NT}}(x_u)$ be the language model next-token loss viewed as a differentiable function of x_u (holding the remainder of the computation fixed). Define

$$g_u \ := \ \nabla_{x_u} L_{\text{NT}}(x_u) \in \mathbb{R}^d.$$

Let the SAE produce post-nonlinearity activations $f(x_u) \in \mathbb{R}^K$ and let $\tilde{f}(x_u)$ denote the latent vector used for reconstruction, after applying BatchTopK. For latent index j, denote its unmasked activation by

$$a_{u,j} \ := \ f_j(x_u) \ \geq 0.$$

Let $w_j^{\mathrm{dec}} \in \mathbb{R}^d$ be the decoder vector for latent j, and define its unit direction

$$\bar{w}_j^{\mathrm{dec}} \;:=\; \frac{w_j^{\mathrm{dec}}}{\|w_j^{\mathrm{dec}}\|_2}.$$

The per-token projection and attribution magnitude are

$$s_{u,j} \;:=\; g_u^\top \bar{w}_j^{\mathrm{dec}}, \qquad \mathrm{G} \times \mathrm{A}_{u,j} \;:=\; |a_{u,j}\, s_{u,j}|.$$

An aggregate score is formed via a high quantile over token positions:

$$A_j \;:=\; \mathrm{Quantile}_u(\mathrm{G} \times \mathrm{A}_{u,j}; q).$$

We compute attribution in activation space, so we score each latent using its decoder column. The gradient $g_u = \nabla_{x_u} L_{\mathrm{NT}}(x_u)$ and w_j^{dec} both live in $\mathbb{R}^d$, and the projection $g_u^\top \bar{w}_j^{\mathrm{dec}}$ measures how sensitive the loss is to moving along the direction that latent j adds back to the activation.

Across cycles, we keep the detector direction of each core latent fixed by transferring and freezing its encoder row w_j^{enc}. In the preactivation $z_{u,j} = (w_j^{\mathrm{enc}})^\top x_u + b_{\mathrm{enc},j}$, this row sets the detector direction. We reinitialize the encoder biases and the full decoder in each cycle.

In the dense-core setting, we apply BatchTopK globally to the core and non-core latents. We calibrate the global BatchTopK level using only the measured non-core activity (Eqs. (3)–(4)). Attribution does not apply this mask. The following derivation shows how G $\times$ A estimates the effect of a perturbation on the loss.

First-order derivation: G $\times$ A as a proxy for loss change. By first-order Taylor expansion with $\Delta x_u \in \mathbb{R}^d$,

$$L_{\mathrm{NT}}(x_u + \Delta x_u) = L_{\mathrm{NT}}(x_u) + g_u^\top \Delta x_u + R(\Delta x_u), \tag{11}$$

with $R(\Delta x_u) = O(\|\Delta x_u\|_2^2)$ provided L_{NT} is twice differentiable in a neighborhood of x_u. In particular,

$$|R(\Delta x_u)| \le \tfrac{1}{2}\Big(\sup_{t\in[0,1]} \|\nabla^2 L_{\mathrm{NT}}(x_u + t\Delta x_u)\|_{\mathrm{op}}\Big)\, \|\Delta x_u\|_2^2.$$

Latent contribution as a perturbation. For clarity, we omit the biases because this does not change the directional argument for each latent. We can then define an unmasked decoded vector:

$$\hat{x}_u^{\mathrm{unmasked}} \approx \sum_{j=1}^{K} a_{u,j}\, w_j^{\mathrm{dec}}.$$

The unmasked decoded vector associated with latent j is

$$\Delta x_{u,j} \;:=\; a_{u,j}\, w_j^{\mathrm{dec}}.$$

Consider the perturbation that removes only this vector:

$$x_u^{(-j)} \;:=\; x_u - \Delta x_{u,j}.$$

Then applying (11) with $\Delta x_u = -\Delta x_{u,j}$ yields

$$L_{\mathrm{NT}}(x_u^{(-j)}) - L_{\mathrm{NT}}(x_u) \approx -\, g_u^\top \Delta x_{u,j} \;=\; -\, a_{u,j}\, g_u^\top w_j^{\mathrm{dec}}. \tag{12}$$

Therefore, the magnitude of the first-order predicted change is

$$\big|L_{\mathrm{NT}}(x_u^{(-j)}) - L_{\mathrm{NT}}(x_u)\big| \approx \big|a_{u,j}\, g_u^\top w_j^{\mathrm{dec}}\big|. \tag{13}$$

This establishes the basic "activation $\times$ gradient-projection" structure.

Directional derivative derivation: G × A as instantaneous sensitivity. We can also interpret G × A by using a scalar amplitude parameter. Define a one-dimensional path that moves along latent j's decoded unit direction:

$$x_u(\alpha) \;:=\; x_u + \alpha\, a_{u,j}\, \bar{w}_j^{\mathrm{dec}}.$$

By the chain rule,

$$\begin{aligned}\frac{d}{d\alpha} L_{\mathrm{NT}}(x_u(\alpha))\Big|_{\alpha=0} &= \nabla_{x_u} L_{\mathrm{NT}}(x_u)^\top \frac{d}{d\alpha} x_u(\alpha)\Big|_{\alpha=0} \\ &= g_u^\top \big(a_{u,j}\, \bar{w}_j^{\mathrm{dec}}\big) \;=\; a_{u,j}\, (g_u^\top \bar{w}_j^{\mathrm{dec}}) \;=\; a_{u,j}\, s_{u,j}. \end{aligned} \tag{14}$$

Thus, $|a_{u,j} s_{u,j}|$ measures how sensitive the next-token loss is to a very small change in the direction of latent j. The latent activation at token position u scales this value.

Relation to explicit latent ablation. Latent ablation from the BatchTopK reconstruction can be defined as the realized loss difference

$$\Delta L_{u,j}^{\mathrm{abl}} \;:=\; L_{\mathrm{NT}}(x_u^{(-j)}) - L_{\mathrm{NT}}(x_u), \qquad x_u^{(-j)} = x_u - \tilde{f}_j(x_u) w_j^{\mathrm{dec}}.$$

Under a first-order approximation,

$$\Delta L_{u,j}^{\mathrm{abl}} \;\approx\; -\tilde{f}_j(x_u)\, g_u^\top w_j^{\mathrm{dec}}.$$

By contrast, the core-selection score uses the unmasked activation $a_{u,j} = f_j(x_u)$ and the unit decoder direction:

$$\mathrm{G} \times \mathrm{A}_{u,j} = \left| a_{u,j}\, g_u^\top \bar{w}_j^{\mathrm{dec}} \right|.$$

Therefore, the G × A score measures sensitivity to the unmasked perturbation defined above.

Normalization of decoder vectors. When nonlinear effects are ignored, SAEs have a basic rescaling property for each latent. If a latent activation is multiplied by a constant and its decoder vector is divided by the same constant, the reconstruction can remain approximately unchanged. Concretely, if for a given latent j we apply the transformation

$$a'_{u,j} \;:=\; c\, a_{u,j}, \qquad w_j^{\mathrm{dec}\,\prime} \;:=\; \frac{1}{c}\, w_j^{\mathrm{dec}},$$

then the contribution $a_{u,j} w_j^{\mathrm{dec}}$ is invariant:

$$a'_{u,j} w_j^{\mathrm{dec}\,\prime} \;=\; (c a_{u,j}) \Big(\frac{1}{c} w_j^{\mathrm{dec}} \Big) \;=\; a_{u,j} w_j^{\mathrm{dec}}.$$

However, the first-order proxy $\left| a_{u,j} g_u^\top w_j^{\mathrm{dec}} \right|$ is also invariant under this idealized transformation:

$$|a'_{u,j} g_u^\top w_j^{\mathrm{dec}\,\prime}| = |c a_{u,j} \cdot g_u^\top (w_j^{\mathrm{dec}}/c)| = |a_{u,j} g_u^\top w_j^{\mathrm{dec}}|.$$

Thus, the unnormalized first-order proxy is invariant under this idealized rescaling. Our normalized score serves a different purpose: it removes explicit dependence on the decoder norm. It is not invariant under the same rescaling because, for $c > 0$,

$$\bar{w}_j^{\mathrm{dec}\,\prime} = \bar{w}_j^{\mathrm{dec}}, \qquad |a'_{u,j} g_u^\top \bar{w}_j^{\mathrm{dec}\,\prime}| = c\, |a_{u,j} g_u^\top \bar{w}_j^{\mathrm{dec}}|.$$

The normalized score therefore assumes that latent activation scales are sufficiently comparable across candidates. We use it as a directional-sensitivity ranking statistic, not as a scale-invariant estimate of the loss change caused by ablation.

Quantile aggregation: robustness to heavy tails. Let $Z_{u,j} := \mathrm{G} \times \mathrm{A}_{u,j} = |a_{u,j} s_{u,j}|$ be the per-token score. In many settings, $Z_{u,j}$ is heavy-tailed across tokens: most tokens contribute near-zero while a small subset produce large values. Aggregating by a high quantile $A_j = \mathrm{Quantile}(Z_{u,j}; q)$ has two useful properties:

1. **Outlier robustness vs. max:** A_j is less sensitive than $\max_u Z_{u,j}$ to a single extreme token.
2. **Upper-tail emphasis vs. mean:** Compared with $\mathbb{E}[Z_{u,j}]$, a high quantile emphasizes upper-tail magnitude rather than average attribution mass. It does not by itself measure firing frequency or persistence across contexts.

These properties make the quantile useful for ranking latents by upper-tail directional sensitivity, but do not by themselves establish that a latent is consistently useful across many token contexts.

Justification for attribution coverage. The promotion rule selects the smallest subset of candidate latents whose cumulative score reaches a target fraction τ of the total. Suppose we have nonnegative scores $\{A_j\}_{j\in P}$, and define the optimization problem

$$\min_{S\subseteq P} |S| \quad \text{s.t.} \quad \sum_{j\in S} A_j \;\geq\; \tau \sum_{j\in P} A_j. \tag{15}$$

Lemma G.1 (Greedy prefix is optimal)**.** *Let $(j_{(1)}, \ldots, j_{(|P|)})$ be an ordering of P such that*

$$A_{j_{(1)}} \geq A_{j_{(2)}} \geq \cdots \geq A_{j_{(|P|)}} \geq 0.$$

Define $S_0 := 0$, define partial sums $S_r := \sum_{\ell=1}^{r} A_{j_{(\ell)}}$ for $r \geq 1$, and let

$$r^\star := \min\left\{ r \in \{0, \ldots, |P|\} : S_r \geq \tau \sum_{j\in P} A_j \right\}.$$

Then the set $C := \{j_{(1)}, \ldots, j_{(r^\star)}\}$ is an optimal solution to (15).

Proof. Fix r and consider any subset $S \subseteq P$ with $|S| = r$. The maximum possible value of $\sum_{j\in S} A_j$ over all such subsets is obtained by choosing the r largest scores:

$$\sum_{j\in S} A_j \;\leq\; \sum_{\ell=1}^{r} A_{j_{(\ell)}} \;=\; S_r,$$

This result follows from a standard exchange argument. Suppose that S omits some $j_{(\ell)}$ with a large score and includes an element with a smaller score. Replace the element with the smaller score by $j_{(\ell)}$. This replacement increases the sum. Repeat this step until S is the top-r set.

By definition of $r^\star$, for all $r < r^\star$ we have $S_r < \tau \sum_{j\in P} A_j$, hence no subset of size r can satisfy the constraint in (15). Meanwhile the top-$r^\star$ subset achieves $S_{r^\star} \geq \tau \sum_{j\in P} A_j$, so it is feasible. Therefore, $r^\star$ is the minimum feasible cardinality and the top-$r^\star$ prefix is optimal. □

What G $\times$ A does not guarantee. The first-order arguments above justify G $\times$ A as a computationally efficient proxy, but do not imply:

1. The true effect of ablating a latent can differ due to higher-order terms $R(\Delta x_u)$ in (11) and due to nonlinear dependence of downstream computations on x_u.
2. Even if reconstructions decompose linearly, the total loss change from multiple ablations can differ from the sum of the separate loss changes. This difference can occur when higher-order terms are large.
3. Normalization helps compare decoder directions, but the effective semantics can still shift because the activation gate depends on encoder bias and BatchTopK masking, which can change the set of token positions where a latent is nonzero.