# Efficient Training of Sparse Autoencoders for Large Language Models via Layer Groups

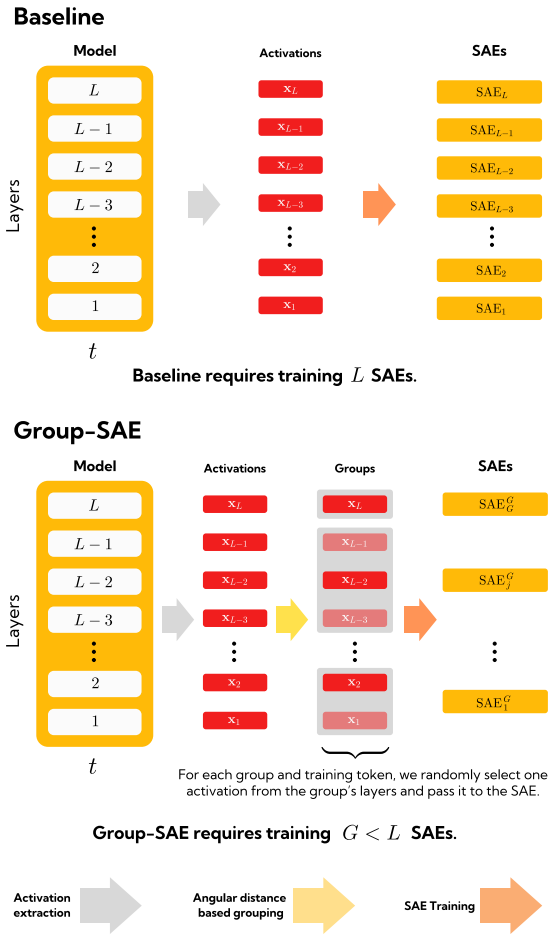
**Anonymous ACL submission**

## Abstract

Sparse Autoencoders (SAEs) have recently been employed as a promising unsupervised approach for understanding the representations of layers of Large Language Models (LLMs). However, with the growth in model size and complexity, training SAEs is computationally intensive, as typically one SAE is trained for each model layer. To address such limitation, we propose *Group-SAE*, a novel strategy to train SAEs. Our method considers the similarity of the residual stream representations between contiguous layers to group similar layers and train a single SAE per group. To balance the trade-off between efficiency and performance, we further introduce *AMAD* (Average Maximum Angular Distance), an empirical metric that guides the selection of an optimal number of groups based on representational similarity across layers. Experiments on models from the Pythia family show that our approach significantly accelerates training with minimal impact on reconstruction quality and comparable downstream task performance and interpretability over baseline SAEs trained layer by layer. This method provides an efficient and scalable strategy for training SAEs in modern LLMs.

## 1 Introduction

Sparse Autoencoders (SAEs) (Makhzani and Frey, 2014) have recently emerged (Huben et al., 2024; Bricken et al., 2023) as a promising technique to tackle the polysemanticity of neurons in the activations of Large Language Models (LLMs) (Olah et al., 2020). SAEs decompose models' activations into a sparse combination of human-interpretable directions, also called *features*. Despite the strengths in interpretability, SAEs face challenges that hinder their large-scale adoption (Sharkey et al., 2025). One of them is the high training and evaluation costs, which increase as model sizes and parameter counts grow. Notably, a separate SAE is typically trained for each component (e.g., the output of the attention, the MLP, or a full transformer block) at every layer of an LLM, with a number of features that is a multiple of the dimensionality of the activation space of the model. For instance, a single SAE trained on the activations of a layer of Llama-3.1 8B (Grattafiori et al., 2024), with an expansion factor of 32, involves approximately $4096^2 \times 32 \times 2 \approx 1.073$ billion parameters. Such high computational demand increases




The illustration of our method. While standard training of SAEs requires training one per layer, our method first groups layers by angular similarity and then trains a single SAE for each group.

training time and requires substantial hardware resources and energy consumption, making the approach increasingly impractical as models scale. Moreover, often to make SAEs useful in practice, all their features have to be manually annotated. Even when using auto-interpretability techniques, this process can become very costly (Paulo et al., 2024b).

Facing such challenges, in this work we introduce **Group-SAE**, a method to reduce the computational overhead of training, evaluating, and interpreting SAEs. Our method leverages the similarity of the representations shared by close layers to reduce the total number of trained SAEs and uses a single SAE to reconstruct activations from different layers. The proposed technique follows primary observations that nearby neural network layers tend to learn similar levels of representations (Szegedy et al., 2014; Zeiler and Fergus, 2014; Jawahar et al., 2019). Shallow layers typically focus on capturing low-level features, while deeper layers are believed to learn high-level abstractions. In addition, Gromov et al. (2024) empirically shows that adjacent layers in LLMs could encode similar information.

Additionally, we introduce **AMAD** (Average Maximum Angular Distance), a novel empirical metric for selecting the optimal number of groups to partition a model's layers–an important choice that balances performance and computational efficiency: more groups tend to improve performance but reduce computational savings, while fewer groups offer greater efficiency at the cost of decreasing performance.

After thoroughly evaluating reconstruction, downstream, and interpretability performance of our methods on three models of varying sizes from the Pythia family (Biderman et al., 2023)–Pythia-160M, Pythia-410M, and Pythia-1B–we show that our method has several advantages compared to baselines. In particular, Group-SAE (with AMAD) finds an optimal tradeoff between training costs and performance of the SAE. It significantly reduces the number of trained SAEs reducing training costs up to 50%. Moreover, such a novel approach only incurs a slight decrease in reconstruction quality and achieves comparable downstream performance. Finally, from an interpretability point of view, Group-SAEs offers the same, or even slightly better, level of interpretability when compared with their baseline counterparts.

Our **contributions** can be summarized as follows:

- We propose a novel method named **Group-SAE**, which partitions the layers of a model into groups and trains a single SAE for each group, thus significantly reducing the total number of SAEs to train.
- We introduce **AMAD** (Average Maximum Angular Distance), a new empirical metric for selecting the optimal number of groups, enabling an effective trade-off between computational efficiency and performance.
- To advance research on interpretability and Sparse Autoencoders, we will openly release all our trained SAEs and Group-SAEs.

## 2 Background and Related Work

### 2.1 Sparse Autoencoders

SAEs (Bricken et al., 2023) are a promising interpretability technique that decomposes dense LLM activations into a sparse combination of human-interpretable features. SAEs are based on two key intuitions. The first is the Linear Representation Hypothesis (LRH), which, supported by substantial empirical evidence (Mikolov et al., 2013; Nanda et al., 2023; Park et al., 2023), posits that Neural Networks (NNs) exhibit interpretable linear directions in their activation space. The second is the Superposition Hypothesis (SH), which assumes that observed NNs are dense compressions of a larger sparse model where each neuron corresponds to a specific feature (Elhage et al., 2022).

Within this framework, SAEs disentangle the effects of superposition, enabling the learning of interpretable linear directions in the model's activations. Formally, given an activation $\mathbf{x} \in \mathbb{R}^n$, a SAE reconstructs it through two steps. First, it encodes the activation into the feature space as:

$$\mathbf{f}(\mathbf{x}) = \sigma\left(\mathbf{b}_e + \mathbf{W}_e\left(\mathbf{x} - \mathbf{b}_d\right)\right) \quad (1)$$

where $\mathbf{f}(\mathbf{x})$ represents feature activations, $\mathbf{b}_e, \mathbf{b}_d \in \mathbb{R}^m$ are bias terms, $\mathbf{W}_e \in \mathbb{R}^{m \times n}$ is the encoder matrix, and $\sigma$ is an activation function. Typically, $m = c \cdot n$, with the expansion factor $c \in \{2^k \mid k \in \mathbb{N}_+\}$. ReLU was initially proposed (Bricken et al., 2023), and then its limitations led to the development of two notable alternatives: TopK (Gao et al., 2024) and JumpReLU (Rajamanoharan et al., 2024).

Then, the feature vector is projected back into the model's activation space using:

$$\widehat{\mathbf{x}} = \mathbf{b}_d + \mathbf{W}_d\,\mathbf{f}(\mathbf{x}) \quad (2)$$

where $\mathbf{W}_d \in \mathbb{R}^{n \times m}$ is the decoder matrix, with each column corresponding to a learned feature vector.

SAEs are trained to minimize the MSE between original activations and SAE reconstruction. To enforce feature sparsity, an additional penalty is usually included in the loss function, either as the $L_1$ norm (Bricken et al., 2023) or the $L_0$ norm (Rajamanoharan et al., 2024) of $\mathbf{f}(\mathbf{x})$, scaled by a positive factor $\lambda$, termed the *sparsity coefficient*. Formally, the loss function can be written as:

$$\mathcal{L}(\mathbf{x}) = \|\mathbf{x} - \widehat{\mathbf{x}}\|_2^2 + \lambda \|\mathbf{f}(\mathbf{x})\|_s \tag{3}$$

with $s \in \{0, 1\}$. On the other hand, when using TopK (Gao et al., 2024), no additional loss components are needed, as the activation function inherently enforces sparsity.

### 2.2 Shared SAEs

While SAEs were originally designed to reconstruct activations from a single model component (e.g., the output of a specific layer, MLP, or Attention), subsequent approaches have explored their application to activations from multiple layers. For instance, Yun et al. (2023) and Lawson et al. (2024) employed a single SAE to reconstruct activations from all residual stream layers of a model, aiming to analyze how features evolve across layers. More recently, Lindsey et al. (2024) extended this concept by introducing *Crosscoder*, a modified SAE architecture that creates a unified representation of computations across multiple layers.

These methods are driven by empirical evidence suggesting that information in LLMs is often shared and rather redundant across nearby layers (Phang et al., 2021; Gromov et al., 2024). In this work, we leverage this principle to explore the optimal balance between performance and computational efficiency when applying SAEs to multiple layers.

### 2.3 Improving SAE efficiency

As highlighted by Sharkey et al. (2025), one of the major challenges of SAEs is their high training and evaluation costs. As previously mentioned, SAEs scale alongside model size, making them impractical for low-resource settings. Furthermore, interpreting the meaning of SAE features presents an additional challenge. Even with automated techniques, interpretation costs can reach thousands of dollars (Paulo et al., 2024b).

To mitigate training costs, Gao et al. (2024) investigated the scaling laws of SAEs to determine the optimal balance between model size and sparsity. Recent work has also explored transfer learning as a means to enhance SAE training efficiency. For instance, Kissane et al. (2024) and Lieberum et al. (2024) demonstrated that SAE weights can be transferred between base and instruction-tuned versions of Gemma-1 (Team et al., 2024a) and Gemma-2 (Team et al., 2024b), respectively. Additionally, Ghilardi et al. (2024) showed that transferability also occurs within different layers of a single model, both in forward and backward directions.

## 3 Method

### 3.1 Group-SAEs

In our approach, a **Group-SAE** is defined as a sparse autoencoder that is trained to reconstruct the activations from multiple layers that have been grouped together, rather than training an individual SAE for each layer. This grouping leverages the observation that nearby layers tend to exhibit similar activation patterns (cf. Figure 5, 6, and Figure 7 in Appendix C).

For a model with $L$ layers, there are theoretically $G! \cdot S(L, G)$ ways to partition the layers into $G$ groups—where $S(L, G)$ denotes the Stirling number of the second kind. Because this number grows rapidly with model depth, we instead employ an agglomerative clustering strategy based on angular distances between layers to efficiently determine a suitable grouping.

Specifically, we compute the mean angular distance between the residual activations of each layer using 10M tokens from our training set (see Appendix C for detailed measurements), following the formulation in (Gromov et al., 2024). We then apply a bottom-up hierarchical clustering method with complete linkage (Nielsen and Nielsen, 2016). At each step, the two groups with the smallest inter-group distance[1] are merged. This merging continues until exactly $G$ groups remain, ensuring that within each group the maximum angular distance is minimized.

### 3.2 Choice of $G$

The choice of $G$, the number of groups of layers, is an important choice to make in our method as

[1] In complete linkage, the inter-group distance is defined as $D(X, Y) = \max_{\mathbf{x} \in X,\, \mathbf{y} \in Y} d_{\text{angular}}(\mathbf{x}, \mathbf{y})$ for groups $X$ and $Y$

it influences both computational savings and SAE performance. To select the optimal value of $G$ for a given model, we propose an empirical metric called the **Average Maximum Angular Distance** (AMAD), defined as:

$$\text{AMAD}_\theta(G) = \frac{1}{G}\sum_{g=1}^{G} D_g, \tag{4}$$

where $D_g$ is the maximum angular distance between any pair of activations within group $g$. AMAD quantifies, on average, the worst-case distance within each group. Tables 5, 6, and 7 in Appendix C report the resulting groups and their corresponding AMAD values for the tested models.

### 3.3 Computational Savings

The computational cost, in FLOPs, of training a SAE can be divided into two main components:

- *Activation caching ($A$)*: The computation required to generate the model's activations, which are used for training the SAE.
- *SAE training ($T$)*: The computation involved in optimizing a SAE using the cached activations.

Thus, the total cost of training SAEs across all residual stream layers of a model is given by $A + LT$. Since both baseline and Group-SAEs share the same architecture and undergo the same training process for a single SAE, the total cost of training all Group-SAEs [2] is $A + GT$.

The resulting compute savings, $\Delta(G)$, quantifying the relative change in total FLOPs when applying Group-SAEs instead of per-layer SAEs, is defined as:

$$\Delta(G) = \frac{A + GT}{A + LT} - 1. \tag{5}$$

By definition, if $G = L$, then $\Delta(G) = 0$, meaning no savings. Conversely, as $G$ decreases, savings increase, reaching a maximum of $(T - LT)/(A + LT)$ when $G = 1$.

Since our method does not alter either $A$ or $T$, the efficiency gains of Group-SAEs are primarily determined by the $G/L$ ratio.

[2] We do not account for the cost of computing angular distance when selecting groups, as we rely on activations already sampled for training, making the additional computational overhead negligible.

## 4 Experiments

Our work is primarily focused on addressing the following research questions:

**Q1** Do SAEs trained on groups of layers activations maintain reconstruction quality and downstream performance?

**Q2** Does selecting the number of groups $G$ based on the Average Maximum Angular Distance (AMAD) ensure an optimal balance between computational efficiency and model performance?

**Q3** How do Group-SAEs affect the interpretability of the SAE latent representations?

To address these questions, we compare the performance of standard SAEs and Group-SAEs across a range of metrics and alternative grouping strategies.

### 4.1 Experimental setting

We denote $\text{SAE}_l$ as the baseline SAE trained to reconstruct the activations of layer $l$. For every $g = 1, \ldots, G$, with $G \in \{1, \ldots, L-1\}$ and $L$ being the number of layers of a model, let $[g_G]$ represent the set of layers belonging to the $g$-th group within the partition of G groups. We then define $\text{SAE}_g^G$ as the SAE trained to reconstruct the activations for all layers in $[g_G]$.

**Models, Dataset and Hyperparameters** Following Lawson et al. (2024), we train both SAEs and Group-SAEs with the Fraction of Variance Unexplained (FVU) as reconstruction loss. Defined as

$$\text{FVU}(\mathbf{x}) = \frac{\|\mathbf{x} - \widehat{\mathbf{x}}\|_2^2}{\text{Var}(\mathbf{x})}, \tag{6}$$

we prefer it to standard MSE loss as it accounts for the different magnitudes of activations coming from different layers of the model. We employ Top-$K$ activation[3] with $K = 128$ and expansion factor of $c = 16$ on the residual stream after the MLP contribution of three models of varying sizes from the Pythia family (Biderman et al., 2023): Pythia 160M, Pythia 410M, and Pythia 1B. To train all the SAEs, we sample 1 billion tokens from the Pile dataset (Gao et al., 2020) and process them with a context size of 1024.

[3] The Top-$K$ activation function is directly applied on the features obtained with Equation1, where $\sigma = \text{Top-}K \circ \text{ReLU}$.

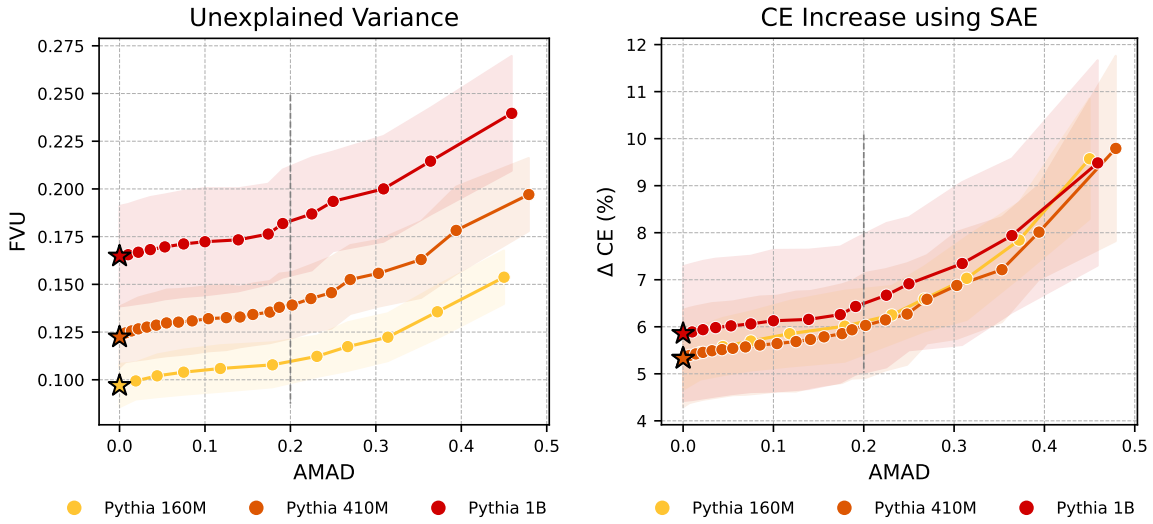



Figure 2: (Left) FVU and (Right) $\Delta\text{CE}(\%)$ over $\text{AMAD}(G)$ for every $G \in \{1, \ldots, L-1\}$. The highlighted star markers represent the baseline SAEs (i.e., with no grouping), while the other points correspond to Group-SAEs, ordered from left to right by increasing AMAD, which reflects a decrease in the number of groups. The shaded area indicates one std

For each model, we compute all partitions $G \in \{1, \ldots, L-1\}$ and train a Group-SAE for all groups of layers in them. We exclude the last layer from all partitions because it resides in the unembedding space and, based on our empirical findings, consistently exhibits a distinct reconstruction error pattern. As a result, it requires a separate SAE. Additionally, we compare our grouping strategy with two **baseline techniques** aimed to reduce the computational cost of training SAEs: (1) training Group SAEs on evenly spaced groups, and (2) training smaller SAEs on all layers. Hyperparameters for all the experiments and training details can be found in Appendix A and B respectively.

**Evaluation.** We evaluate SAE performance across three key areas: reconstruction, downstream, and interpretability.

For both reconstruction and downstream evaluations, we use a subset of the Pile dataset (distinct from the training set) comprising 1 million tokens.

For reconstruction, we compare each $\text{SAE}_g^G$ with its corresponding baseline $\text{SAE}_l$ for every layer $l \in [g_G]$. We report the average Fraction of Variance Unexplained (FVU, Equation 6) as our reconstruction metric.

To evaluate downstream performance, we measure the effect of replacing a layer's activation with its SAE reconstruction on the next-token prediction. Specifically, we compute the average relative change in next-token Cross-Entropy:

$$\Delta\text{CE} = \frac{\text{CE}(\text{M}(P \mid \mathbf{x}^l \leftarrow \widehat{\mathbf{x}}^l)) - \text{CE}(\text{M}(P))}{\text{CE}(\text{M}(P))}, \tag{7}$$

where M denotes the model, $P$ is the input prompt, and $\text{M}(P \mid \mathbf{x}^l \leftarrow \widehat{\mathbf{x}}^l)$ indicates the model output when the true activation $\mathbf{x}^l$ at layer $l$ is replaced with the SAE reconstruction $\widehat{\mathbf{x}}^l$.

For interpretability, we adopt the automated pipeline proposed by Paulo et al. (2024a). First, an *explainer* language model (LM) generates natural language explanations of the SAE latent representations. Then, a separate *scorer* LM evaluates these explanations. In our experiments, both the explainer and scorer are implemented using `gemini-2.0-flash-001`[4]. Specifically, for each SAE, we randomly sample 64 features and cache their latent activations over a 10M token sample from the Pile. For each latent, the explainer is shown 20 distinct examples, 10 activating the latent and 10 sampled randomly, each consisting of 32 tokens. Two binary scoring strategies are employed:

- *Detection*: A language model determines whether a given sequence activates an SAE latent according to the provided explanation.
- *Fuzzing*: Activating tokens are marked within each example, and a language model is

[4] https://deepmind.google/technologies/gemini/flash/

Table 1: FVU and ΔCE for different approaches across model sizes. Our proposed grouping strategy based on the AMAD achieves lower FVU and ΔCE compared to the baselines: Group SAEs with evenly spaced groups and smaller SAEs trained on all layers. Note that both the *Evenly Spaced* and *Smaller SAEs* strategies have the same number of training FLOPs as our AMAD-based grouping strategy.

| Approach | Pythia-160M | | Pythia-410M | | Pythia-1B | |
|---|---|---|---|---|---|---|
| | FVU | $\Delta\text{CE}_{\%}$ | FVU | $\Delta\text{CE}_{\%}$ | FVU | $\Delta\text{CE}_{\%}$ |
| Group SAEs (AMAD with $\widehat{G}$ groups) | **0.108** | 6.01 | **0.138** | **5.94** | **0.182** | **6.43** |
| Group SAEs (Evenly spaced with $\widehat{G}$ groups) | 0.114 | **5.40** | 0.145 | 6.01 | 0.189 | 6.63 |
| Smaller SAEs (All layers) | 0.115 | 7.37 | 0.146 | 7.10 | 0.188 | 8.10 |

prompted to assess whether the marked sentences are correctly identified.

Figure 8 shows a sentence example for each strategy. For every metric (FVU, ΔCE and Detection/Fuzzing) and for each $G \in \{1, \dots, L-1\}$, we first compute all metrics at the layer level, then aggregate the results for each partition $g$ within $G$ by computing the mean and the standard deviation weighted by the number of layers in that partition.

### 4.2 Results

In the following paragraphs, we aim to empirically answer the research questions outlined in Section 4.

**Q1: What is the impact of grouping layers (Group-SAEs) on reconstruction quality and downstream task performance?** In Figure 2, we plot the average FVU and the cross-entropy difference (ΔCE) as functions of the AMAD for different group configurations. The highlighted star markers represent the baseline models (i.e., with no grouping), while the other points correspond to grouped models. The points are ordered from left to right by increasing AMAD, which reflects a decrease in the number of groups, with $G$ ranging from $L-1$ down to 1. From Figure 2, a *notable turning point* emerges around $\text{AMAD}(G) \approx 0.2$: increasing AMAD beyond this threshold leads to a more rapid loss in performance. In particular, training a single SAE on all the model layers ($G = 1$), although achieving the best computational saving, also incurs the worst reconstruction and downstream performance.

To further validate our method, in Appendix E, we further inspect the quality of features learned by Group-SAEs by measuring their similarity to the features learned by Baseline-SAEs. As expected, for each baseline $\text{SAE}_l$, we found average similarity to peak with the Group-SAE trained on a group containing $l$. Finally, in Appendix F we show how features of a Group-SAE distribute across the activations of layers of their respective group, thus supporting the rationale behind our proposed method.

**Q2: Does selecting the number of groups $G$ based on the AMAD ensure an optimal balance between computational efficiency and model performance?** Motivated by the insights from the previous paragraph, the optimal $G$ is chosen as $\widehat{G} = \inf\{G \mid \text{AMAD}(G) < 0.2\}$. In Figure 3 we show both FVU and ΔCE plotted against the fraction of PFLOPs relative to the baseline. Again, star markers denote baseline SAEs, whereas circles represent Group-SAEs. Here, moving from right to left indicates reducing PFLOPs (i.e., training fewer SAEs overall). The points are ordered from right to left by decreasing PFLOPs, which reflects a decrease in the number of groups, from $L-1$ down to 1. The highlighted square markers correspond to Group-SAEs with $\widehat{G}$ groups; they substantially reduce training costs up to more than 50% with only a moderate performance penalty: $\text{FVU}(\bigstar - \blacksquare) \approx -0.01$ and $\Delta\text{CE}_{\%}(\bigstar - \blacksquare) \approx -0.6$ for all three evaluated models.

To ensure that our grouping strategy and the selection of $\widehat{G}$ based on AMAD offer an optimal balance between computational efficiency and performance, we compare them against two baselines: 1) *Evenly Spaced Group SAEs*: Group SAEs trained such that each partition contains nearly equal numbers of layers; 2) *Smaller SAEs*: A separate, smaller SAE is trained for each layer. All methods are adjusted to incur equal computational costs[5]. Results in Table 1 shows that the proposed method outper-

[5]For Evenly Spaced Group SAEs, we use the same number of groups $\widehat{G}$; for Smaller SAEs, we set the expansion factor as $c' = c \cdot \widehat{G}/T$, matching the FLOPs of a Group-SAE with $\widehat{G}$ groups.

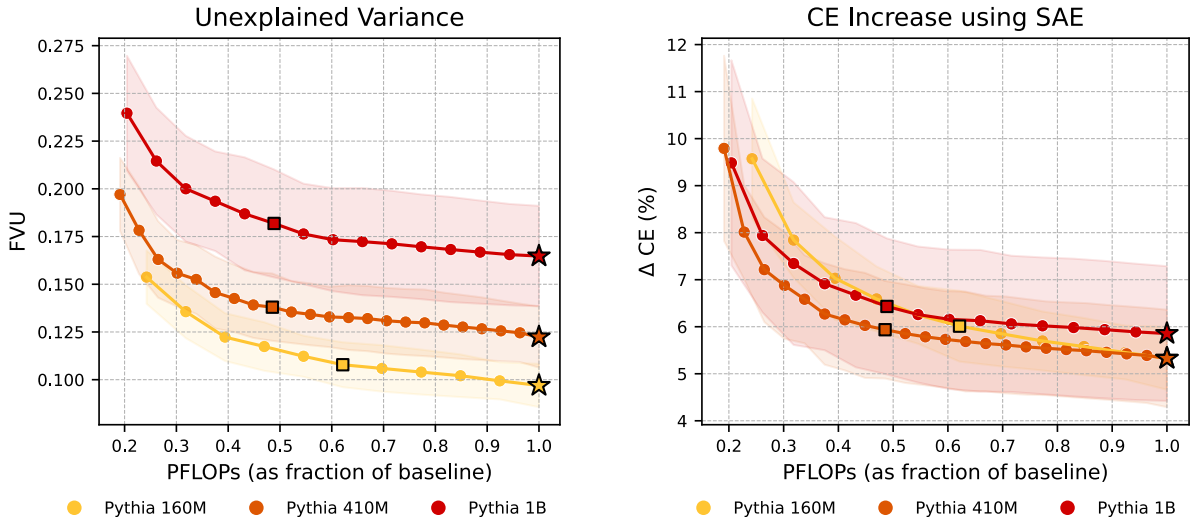



Figure 3: (Left) FVU and (Right) $\Delta$CE(%) over the fraction of training PFLOPs with respect to the baseline. The highlighted star markers represent the baseline SAEs (i.e., with no grouping), while the other points correspond to Group-SAEs, ordered from right to left by decreasing PFLOPs, which reflects a decrease in the number of groups. The highlighted square markers represent the Group-SAEs with a number of groups $\widehat{G} = \inf\{G \mid \text{AMAD}(G) < 0.2\}$

.

forms the two additional baselines across nearly all models and evaluation metrics, with only a single exception observed in the case of Pythia-160M. Importantly, this exception does not arise from the idea of grouping layers but from the chosen grouping strategy. Indeed, our method consistently outperforms the standard per-layer approach with smaller Standard SAEs. We observe that these advantages are particularly noticeable for the $\Delta$CE metric, related to the downstream performance. Additionally, Table 2 presents the computational costs and savings, as defined in Section 3.3, of Group-SAEs compared to the baselines when the optimal number of groups $G$ is selected as $\widehat{G}$.

Table 2: Comparison of FLOPs ($10^{18}$) required for caching activations and training Baseline and Group SAEs on 1B tokens, covering all layers with an expansion factor of 16 and $\widehat{G} = \inf\{G \mid \text{AMAD}(G) < 0.2\}$.

| Model | $\widehat{\mathbf{G}}$ | A+LT | A+$\widehat{\mathbf{G}}$T | $\boldsymbol{\Delta}_{\%}(\widehat{\mathbf{G}})$ |
|---|---|---|---|---|
| Pythia 160M | 6 | 1.34 | **0.77** | $-42.5\%$ |
| Pythia 410M | 9 | 4.73 | **2.21** | $-53.3\%$ |
| Pythia 1B | 6 | 12.48 | **5.77** | $-53.7\%$ |

**Q3: How do Group-SAEs affect the interpretability of the SAE latent representations?** To assess the interpretability of the learned SAE latents we employ the auto-interpretability pipeline proposed by (Paulo et al., 2024b). For each SAE latent, first, an *explainer* Language Model is asked to propose a natural language explanation of it given both activating and non-activating examples. Then, given the explanation, a *scorer* Language Model is tasked with predicting the set of sentences that should activate the target latent (*detection*) and the sentences containing highlighted tokens that activate the target latent (*fuzzing*). In Figure 4 we plot both the detection and fuzzing scores for all the evaluated models. In the figures, square markers denote Group-SAEs with $\widehat{G}$ groups, while star markers indicate the baseline SAEs. We observe a similar trend as in reconstruction and downstream evaluations: detection and fuzzing scores improve more rapidly as AMAD($G$) decreases—provided it remains above the turning point—after which the scores plateau at an approximately constant level. This result further validates our selection of $\widehat{G}$ based on AMAD, suggesting that the interpretability of features in the baseline and Group-SAEs differs only marginally.

## 5 Conclusion

This work introduces a novel approach to efficiently train SAEs for LLMs by clustering layers based on their angular distance and training a single SAE for each group. Through this method, we achieved up to a 50% reduction in training costs without com-

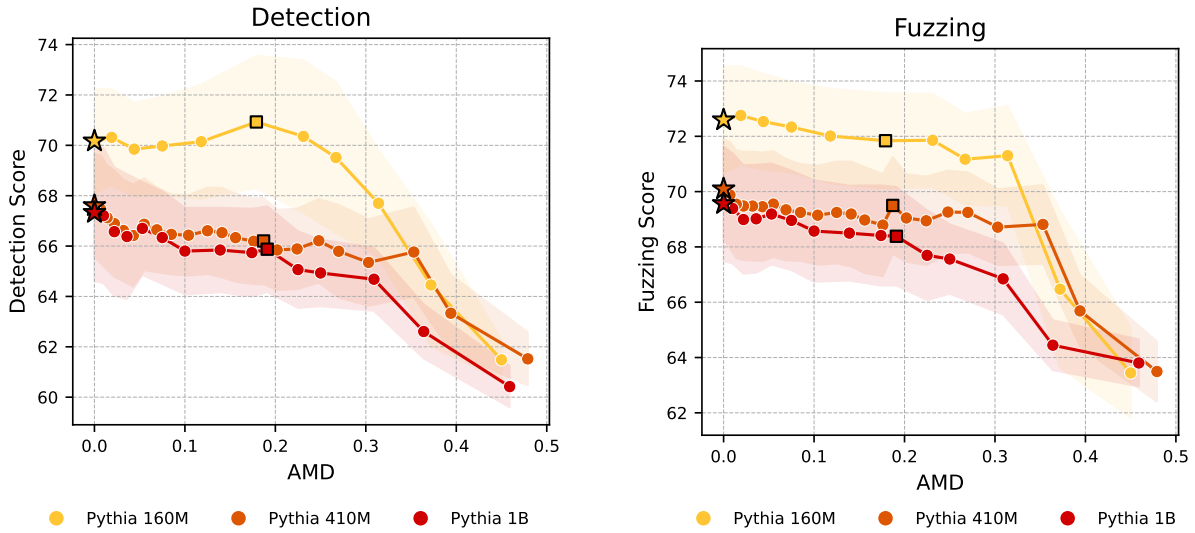



Figure 4: Auto-Interpretability scores following the automated pipeline defined by (Paulo et al., 2024b) over AMAD$(G)$ for every $G \in \{1, \ldots, L-1\}$. The highlighted star markers represent the baseline SAEs (i.e., with no grouping), while the other points correspond to Group-SAEs, ordered from left to right by increasing AMAD, which reflects a decrease in the number of groups. The highlighted square markers represent the Group-SAEs with a number of groups $\widehat{G} = \inf\{G \mid \text{AMAD}(G) < 0.2\}$. (Left) Detection and (Right) Fuzzing scores, as defined in the Evaluation paragraph of Section 4.

promising reconstruction quality or performance on downstream tasks. The results demonstrate that activations from adjacent layers in LLMs share common features, enabling effective reconstruction with fewer SAEs.

Our findings also show that the SAEs trained on grouped layers perform comparably to layer-specific SAEs in terms of reconstruction and downstream metrics. Furthermore, the automated interpretability evaluations confirmed the interpretability of the features learned by our SAEs, underscoring their utility in disentangling neural activations.

The methodology proposed in this paper opens avenues for more scalable interpretability tools, facilitating deeper analysis of LLMs as they grow in size. Future work will focus on further optimizing the number of layer groups and scaling the approach to even larger models.

## Limitations

Although we evaluated our approach across various groups and model sizes, our primary focus here is on experiments using a fixed expansion factor of $c = 16$ and TopK as activation function. Although we don't expect the choices of these hyperparameters to influence the results of this work, we left investigations of this phenomenon for future work. We also limit the scope of our study to models from the Pythia family. Although we recognize that architectural and training differences across model families may influence the behavior of Group-SAEs, we defer a comprehensive cross-model analysis to future research. Exploring the generality of our findings across diverse architectures, such as LLaMA, Qwen, or Mistral, is an important next step. Finally, our interpretability evaluation remains limited, primarily due to the high economic cost of annotating large numbers of features. While we observe promising patterns, a more comprehensive and systematic interpretability analysis is left for future work.

## Reproducibility statement

To support the replication of our empirical findings on training SAEs via layer groups and to enable further research on understanding their inner works, we plan to release all the code and SAEs used in this study upon acceptance.

## References

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International*

*Conference on Machine Learning*, pages 2397–2430. PMLR.

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*.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. 2022. Toy models of superposition. *Transformer Circuits Thread*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2024. Scaling and evaluating sparse autoencoders. *Preprint*, arXiv:2406.04093.

Davide Ghilardi, Federico Belotti, Marco Molinari, and Jaehyuk Lim. 2024. Accelerating sparse autoencoder training via layer-wise transfer learning in large language models. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 530–550, Miami, Florida, US. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, and Artem Korene. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. 2024. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, and 7 others. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362.

Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. 2024. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*.

J. D. Hunter. 2007. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization. *Preprint*, arXiv:1412.6980.

Connor Kissane, Ryan Krzyzanowski, Andrew Conmy, and Neel Nanda. 2024. SAEs (usually) transfer between base and chat models. AI Alignment Forum.

Tim Lawson, Lucy Farnik, Conor Houghton, and Laurence Aitchison. 2024. Residual stream analysis with multi-layer saes. *Preprint*, arXiv:2409.04185.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, Janos Kramar, Anca Dragan, Rohin Shah, and Neel Nanda. 2024. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 278–300, Miami, Florida, US. Association for Computational Linguistics.

Jack Lindsey, Adly Templeton, Jonathan Marcus, Thomas Conerly, Joshua Batson, and Christopher Olah. 2024. Sparse crosscoders for cross-layer features and model diffing. *Transformer Circuits*. * Equal contribution.

Alireza Makhzani and Brendan Frey. 2014. k-sparse autoencoders. In *International Conference on Learning Representations (ICLR)*, Banff, AB, Canada.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

Neel Nanda, Andrew Lee, and Martin Wattenberg. 2023. Emergent linear representations in world models of self-supervised sequence models. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 16–30, Singapore. Association for Computational Linguistics.

Frank Nielsen and Frank Nielsen. 2016. Hierarchical clustering. *Introduction to HPC with MPI for Data Science*, pages 195–211.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. 2020. Zoom in: An introduction to circuits. *Distill*.

Kiho Park, Yo Joong Choe, and Victor Veitch. 2023. The linear representation hypothesis and the geometry of large language models. In *Causal Representation Learning Workshop at NeurIPS 2023*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

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

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

Jason Phang, Haokun Liu, and Samuel R. Bowman. 2021. Fine-tuned transformers show clusters of similar representations across layers. *Preprint*, arXiv:2109.08406.

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.

Lee Sharkey, Bilal Chughtai, Joshua Batson, Jack Lindsey, Jeff Wu, Lucius Bushnaq, Nicholas Goldowsky-Dill, Stefan Heimersheim, Alejandro Ortega, Joseph Bloom, Stella Biderman, Adria Garriga-Alonso, Arthur Conmy, Neel Nanda, Jessica Rumbelow, Martin Wattenberg, Nandi Schoots, Joseph Miller, Eric J. Michaud, and 10 others. 2025. Open problems in mechanistic interpretability. *Preprint*, arXiv:2501.16496.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. *Preprint*, arXiv:1312.6199.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, and 89 others. 2024a. Gemma: Open models based on gemini research and technology. *Preprint*, arXiv:2403.08295.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, and 178 others. 2024b. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Michael L. Waskom. 2021. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Zeyu Yun, Yubei Chen, Bruno A Olshausen, and Yann LeCun. 2023. Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors. *Preprint*, arXiv:2103.15949.

Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer.

## A Hyperparameters

We train both SAEs and Group-SAEs using Top-$K$ activation[6] with $K = 128$ and expansion factor of $c = 16$ on the residual stream after the MLP contribution of three models of varying sizes from the Pythia family (Biderman et al., 2023): Pythia 160M, Pythia 410M, and Pythia 1B. To train all the SAEs, we sample 1 billion tokens from the Pile dataset (Gao et al., 2020) and process them with a context size of 1024. We use Adam optimizer (Kingma and Ba, 2017) with default $\beta$ parameters and set the learning rate equal to 2e-4/$\sqrt{(m/2^{14})}$ as specified in Gao et al. (2024). We use a batch size of $131072, 65536$ and $32768$ for the three models, respectively, to maximize computational usage. Following (Bricken et al., 2023) we constrain the decoder columns (i.e. the feature directions) to have unit norm. Additionally, we normalize the activations to have mean squared $\ell_2$ norm of 1 during SAE training, as specified in (Rajamanoharan et al., 2024), by first estimating the norm scaling factor over 5 million tokens of our train set.

Table 3: Pythia model details.

| Pythia model | Non-Embedding Params | Layers | Model Dim | Heads |
|---|---|---|---|---|
| 160M | 85,056,000 | 12 | 768 | 12 |
| 410M | 302,311,424 | 24 | 1024 | 16 |
| 1.0B | 805,736,448 | 16 | 2048 | 8 |

Table 4: Training and fine-tuning hyperparameters

| **Hyperparameter** | **Value** |
|---|---|
| c | 16 |
| Top-$K$ K | 128 |
| $\alpha_{\text{aux}}$ | $1/32$ |
| Hook name | resid-post |
| Batch size | 131'072 (Pythia-160M)<br>65'536 (Pythia-410M)<br>32'768 (Pythia-1B) |
| Adam $(\beta_1, \beta_2)$ | $(0.9, 0.999)$ |
| Context size | 1024 |
| lr | 2e-4/$\sqrt{(m/2^{14})}$ |
| lr scheduler | constant |
| Dead latents threshold | 10M |
| # tokens (Train) | 1B |
| Checkpoint freq | 100K |
| Decoder column normalization | Yes |
| Activation normalization | Mean squared $\ell_2$ norm equal to 1 during SAE training |
| FP precision | 32 |
| Prepend BOS token | No |

The experiments were carried out on a cluster of 8 AMD MI250X. The longest experimental run took approximately 24 hours. Our experiments were carried out using `PyTorch` (Paszke et al., 2019) and the `sparsify` library.[7] We performed our data analysis using `NumPy` (Harris et al., 2020) and `Pandas` (Wes McKinney, 2010). Our figures were made using `Matplotlib` (Hunter, 2007) and `Seaborn` (Waskom, 2021).

[6] The Top-$K$ activation function is directly applied on the features obtained with Equation1, where $\sigma = \text{Top-}K \circ \text{ReLU}$.

[7] https://github.com/EleutherAI/sparsify

## B SAEs Training Details

Following Lawson et al. (2024), given $\mathbf{X}, \widehat{\mathbf{X}} \in \mathbb{R}^{B \times n}$ being the input activation batch and its SAE reconstruction, respectively, we train our SAEs with the following loss:

$$\mathcal{L}(\mathbf{X}) = \text{FVU}(\mathbf{X}, \widehat{\mathbf{X}}) + \alpha_{\text{aux}} \cdot \text{AuxK}(\mathbf{X}, \widehat{\mathbf{X}}) \tag{8}$$

The first term of the loss is the Fraction of Variance Unexplained, or:

$$\text{FVU}(\mathbf{X}, \widehat{\mathbf{X}}) = \frac{\|\mathbf{X} - \widehat{\mathbf{X}}\|_F}{\|\mathbf{X} - \overline{\mathbf{X}}\|_F} \tag{9}$$

where $\|\cdot\|_F$ is the Frobenius norm and $\overline{\mathbf{X}} = \frac{1}{B}\mathbf{1}_B\mathbf{1}_B^\top\mathbf{X}$ is a matrix where each row corresponds to the mean of $\mathbf{X}$ along the batch dimension. The second term of the loss is an auxiliary loss to prevent the formation of dead latents during training and is defined as:

$$\text{AuxK}(\mathbf{X}, \widehat{\mathbf{X}}) = \frac{\|\mathbf{E} - \widehat{\mathbf{E}}\|_F}{\|\mathbf{X} - \overline{\mathbf{X}}\|_F} \tag{10}$$

Here, $\mathbf{E} = \mathbf{X} - \widehat{\mathbf{X}}$ is the reconstruction error of the main model, and $\widehat{\mathbf{E}}$ is its reconstruction using the top-$\text{K}_{\text{aux}}$ dead latents. A dead latent $\mathbf{f}_i(\mathbf{x})$ is a latent that didn't fire, i.e. $\mathbf{f}_i(\mathbf{x}) = 0$, for a predefined number of tokens (10M in our experiments). Following Gao et al. (2024), we choose $K_{\text{aux}}$ as the minimum between the number of dead latents and $m/2$, and $\alpha = 1/32$.

To ensure a fair comparison with baselines, we allocate 1 billion training tokens for each $\text{SAE}_l$ and $\text{SAE}_g^G$. For baseline SAEs, activations are always taken from a single fixed layer. In contrast, for Group-SAEs, activations are drawn from a randomly selected layer within the set $[g_G]$. In this way, we ensure that each Group- and baseline SAEs process exactly 1 billion tokens and activations.

## C Additional Angular Distances and Layers Groups

We use the same angular distance formulation of Gromov et al. (2024):

$$d_\theta\left(\mathbf{x}^i, \mathbf{x}^j\right) = \frac{1}{\pi} \arccos\left(\frac{\mathbf{x}^i \cdot \mathbf{x}^j}{\|\mathbf{x}^i\|_2 \|\mathbf{x}^j\|_2}\right) \tag{11}$$

for every $i, j \in \{1, ..., L\}$, where $\mathbf{x}^l$ are the $l$-th residual stream activations after the MLP's contribution.

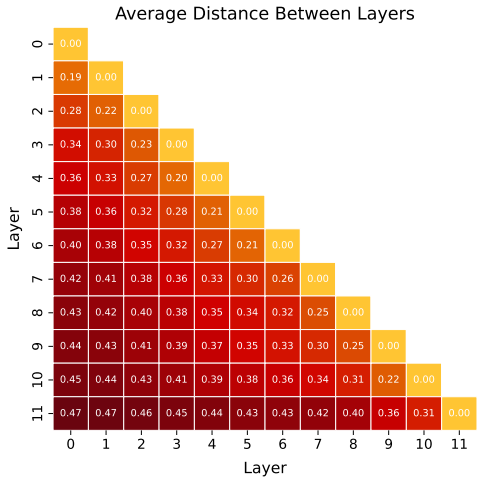



Figure 5: Average angular distance between all layers of the Pythia-160M model, as defined in Equation 11. The angular distances are computed over 10M tokens from the training dataset. The angular distances are bounded in $[0, 1]$, where an angular distance equal to $0$ means equal activations, $0.5$ means activations are perpendicular and an angular distance of $1$ means that the activations point in opposite directions.

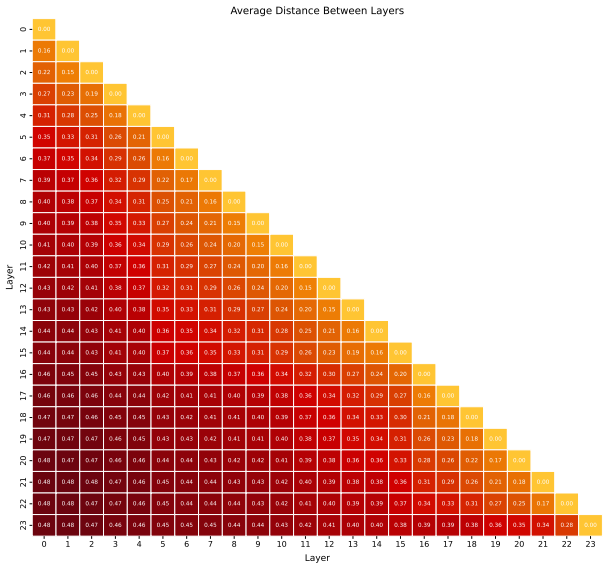



Figure 6: Average angular distance between all layers of the Pythia-410M model, as defined in Equation 11. The angular distances are computed over 10M tokens from the training dataset. The angular distances are bounded in $[0, 1]$, where an angular distance equal to $0$ means equal activations, $0.5$ means activations are perpendicular and an angular distance of $1$ means that the activations point in opposite directions.

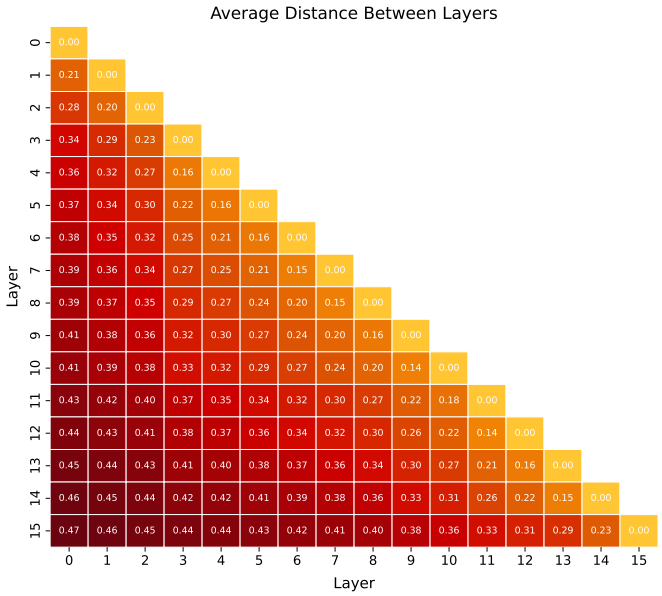



Figure 7: Average angular distance between all layers of the Pythia-1B model, as defined in Equation 11. The angular distances are computed over 10M tokens from the training dataset and are bounded in $[0, 1]$. An angular distance equal to $0$ means equal activations, $0.5$ means activations are perpendicular and an angular distance of $1$ means that the activations point in opposite directions.

| G | Groups | AMAD |
|---|---|---|
| 1 | 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 | 0.450 |
| 2 | 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1 | 0.372 |
| 3 | 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 0 | 0.314 |
| 4 | 2, 2, 2, 0, 0, 0, 0, 1, 1, 3, 3 | 0.267 |
| 5 | 0, 0, 0, 4, 4, 2, 2, 1, 1, 3, 3 | 0.231 |
| 6 | 3, 3, 5, 4, 4, 2, 2, 0, 0, 1, 1 | 0.179 |
| 7 | 3, 3, 5, 1, 1, 2, 2, 6, 4, 0, 0 | 0.118 |
| 8 | 3, 3, 5, 1, 1, 0, 0, 6, 4, 7, 2 | 0.075 |
| 9 | 1, 1, 5, 0, 0, 8, 7, 6, 4, 3, 2 | 0.044 |
| 10 | 0, 0, 5, 9, 7, 8, 3, 6, 4, 1, 2 | 0.019 |

Table 5: Layer groups for every $G$ up to $L - 1$ for Pythia-160M

| G | Groups | AMAD |
|---|---|---|
| 1 | 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 | 0.479 |
| 2 | 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1 | 0.394 |
| 3 | 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1 | 0.353 |
| 4 | 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 | 0.303 |
| 5 | 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 1, 1, 1, 1, 1, 4, 4, 4, 2, 2, 2, 2 | 0.270 |
| 6 | 5, 5, 5, 1, 1, 1, 1, 3, 3, 3, 3, 0, 0, 0, 0, 0, 4, 4, 4, 2, 2, 2, 2 | 0.248 |
| 7 | 5, 5, 5, 0, 0, 0, 0, 1, 1, 1, 1, 6, 6, 3, 3, 3, 4, 4, 4, 2, 2, 2, 2 | 0.224 |
| 8 | 2, 2, 2, 5, 5, 7, 7, 1, 1, 1, 1, 6, 6, 3, 3, 3, 4, 4, 4, 0, 0, 0, 0 | 0.202 |
| 9 | 2, 2, 2, 5, 5, 7, 7, 0, 0, 0, 0, 6, 6, 3, 3, 3, 1, 1, 1, 8, 8, 4, 4 | 0.187 |
| 10 | 2, 2, 2, 5, 5, 7, 7, 8, 8, 9, 9, 6, 6, 1, 1, 1, 0, 0, 0, 3, 3, 4, 4 | 0.176 |
| 11 | 2, 2, 2, 5, 5, 7, 7, 8, 8, 9, 9, 6, 6, 0, 0, 0, 10, 4, 4, 3, 3, 1, 1 | 0.156 |
| 12 | 0, 0, 0, 2, 2, 7, 7, 8, 8, 9, 9, 6, 6, 11, 5, 5, 10, 4, 4, 3, 3, 1, 1 | 0.141 |
| 13 | 12, 9, 9, 2, 2, 7, 7, 8, 8, 4, 4, 6, 6, 11, 5, 5, 10, 1, 1, 3, 3, 0, 0 | 0.125 |
| 14 | 12, 9, 9, 0, 0, 7, 7, 8, 8, 4, 4, 6, 6, 11, 2, 2, 10, 1, 1, 3, 3, 13, 5 | 0.104 |
| 15 | 12, 9, 9, 14, 8, 7, 7, 3, 3, 4, 4, 6, 6, 11, 2, 2, 10, 0, 0, 1, 1, 13, 5 | 0.085 |
| 16 | 12, 9, 9, 14, 8, 3, 3, 1, 1, 4, 4, 6, 6, 11, 2, 2, 10, 15, 7, 0, 0, 13, 5 | 0.069 |
| 17 | 12, 9, 9, 14, 8, 1, 1, 0, 0, 4, 4, 6, 6, 11, 2, 2, 10, 15, 16, 7, 3, 13, 5 | 0.055 |
| 18 | 12, 4, 4, 14, 17, 0, 0, 8, 9, 1, 1, 6, 6, 11, 2, 2, 10, 15, 16, 7, 3, 13, 5 | 0.043 |
| 19 | 12, 4, 4, 14, 17, 18, 13, 8, 9, 1, 1, 2, 2, 11, 0, 0, 10, 15, 16, 7, 3, 6, 5 | 0.032 |
| 20 | 12, 1, 1, 14, 17, 18, 13, 8, 19, 0, 0, 2, 2, 11, 9, 10, 4, 15, 16, 7, 3, 6, 5 | 0.022 |
| 21 | 12, 1, 1, 14, 17, 18, 13, 8, 19, 20, 11, 0, 0, 5, 9, 10, 4, 15, 16, 7, 3, 6, 2 | 0.014 |
| 22 | 12, 0, 0, 14, 17, 18, 13, 8, 19, 20, 11, 21, 16, 5, 9, 10, 4, 15, 7, 3, 1, 6, 2 | 0.007 |

Table 6: Layer groups for every $G$ up to $L - 1$ for Pythia-410M

| G | Groups | AMAD |
|---|---|---|
| 1 | 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 | 0.459 |
| 2 | 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1 | 0.364 |
| 3 | 1, 1, 1, 1, 1, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0 | 0.309 |
| 4 | 0, 0, 0, 0, 0, 2, 2, 2, 2, 1, 1, 1, 1, 3, 3 | 0.250 |
| 5 | 4, 4, 1, 1, 1, 2, 2, 2, 2, 0, 0, 0, 0, 3, 3 | 0.225 |
| 6 | 4, 4, 1, 1, 1, 0, 0, 0, 0, 2, 2, 5, 5, 3, 3 | 0.191 |
| 7 | 1, 1, 0, 0, 0, 4, 4, 2, 2, 6, 6, 5, 5, 3, 3 | 0.174 |
| 8 | 0, 0, 7, 3, 3, 4, 4, 2, 2, 6, 6, 5, 5, 1, 1 | 0.139 |
| 9 | 8, 4, 7, 3, 3, 1, 1, 2, 2, 6, 6, 5, 5, 0, 0 | 0.100 |
| 10 | 8, 9, 7, 1, 1, 0, 0, 2, 2, 6, 6, 5, 5, 4, 3 | 0.075 |
| 11 | 8, 9, 7, 0, 0, 10, 3, 2, 2, 6, 6, 5, 5, 4, 1 | 0.053 |
| 12 | 8, 9, 7, 11, 6, 10, 3, 0, 0, 2, 2, 5, 5, 4, 1 | 0.036 |
| 13 | 8, 9, 7, 11, 6, 10, 3, 12, 5, 0, 0, 2, 2, 4, 1 | 0.022 |
| 14 | 8, 9, 7, 11, 13, 10, 3, 12, 5, 6, 2, 0, 0, 4, 1 | 0.010 |

Table 7: Layer groups for every $G$ up to $L - 1$ for Pythia-1b

## D Auto Interpretability

To evaluate the interpretability of features of baseline and Group SAEs, we adopt automated pipeline from (Paulo et al., 2024b), focusing on *detection* and *fuzzing* scores. First, an *explainer* language model (LM) generates natural language explanations of the SAE latent representations. Then, a separate *scorer* LM evaluates these explanations.

Then, detection scoring assesses whether a language model can identify entire sequences that activate a specific latent, given its interpretation. This method evaluates the model's ability to distinguish between activating and non-activating contexts, offering insights into the precision and recall of the interpretation. Fuzzing scoring, on the other hand, operates at the token level, prompting the model to pinpoint specific tokens within sequences that trigger latent activations. This approach closely mirrors simulation scoring and is particularly effective in evaluating the model's token-level understanding of latent activations.

In our experiments, we use `gemini-2.0-flash-001` as the base model for both the explainer and the scorer. For each SAE, we randomly select 64 features and cache their latent activations across 10M tokens from the Pile (Gao et al., 2020). To generate annotations, we present the explainer with 20 distinct examples per feature—10 that activate the latent and 10 randomly sampled—each comprising 32 tokens.

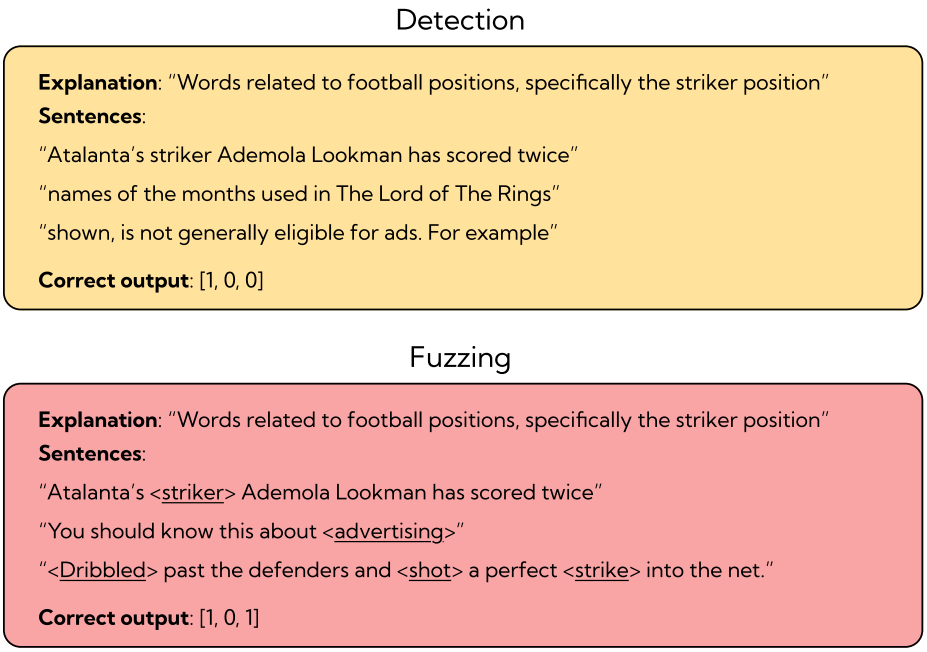



Figure 8: Examples of each of the auto-interpretability techniques: Detection and Fuzzing. In detection, the objective is to find the sentences in which the feature is active. In fuzzing, the objective is to spot the highlighted tokens referring to the target feature.

## E Feature Similarity Analysis

In this section, we perform a novel study to understand how features of Group SAEs are distributed with respect to baseline ones. The question we want to answer is the following: *"Given a feature $i$ of a baseline SAE trained at layer $l$, does a feature $j$ similar to it also appear in a Group SAE trained on the group that contains that layer?"*.

We investigate feature similarity using two complementary metrics: Mean Maximum Concordance (MMCO) and Mean Maximum Cosine Similarity (MMCS). MMCO measures similarity based on feature activations, while MMCS captures alignment in feature directions.

### E.1 Mean Maximum Concordance (MMCO)

Let $[k] = \{1, ..., k\}$. Let $N$ denote the number of tokens and $m$ the total number of SAE features. For a particular pair of layers $l_1, l_2 \in \{1, ..., L-1\}$, for each token $t \in \{1, \dots, N\}$, let $F_t^{\text{Baseline},l_1}, F_t^{\text{Group},l_2} \subset \{1, \dots, m\}$ denote the set of features activated[8] in the Baseline SAE at layer $l_1$ and in the Group SAE at layer $l_2$ respectively. For each feature $i, j \in \{1, \dots, m\}$ we define the per-feature occurrence sets as

$$B_i^{l_1} = \{\, t \mid i \in F_t^{\text{Baseline},l_1} \,\}, \tag{12}$$

$$G_j^{l_2} = \{\, t \mid j \in F_t^{\text{Group},l_2} \,\}. \tag{13}$$

From $B_i^{l_1}$ and $G_j^{l_2}$, we compute the co-occurrence matrix $\text{AND}^{l_1,l_2} \in \mathbb{R}^{m \times m}$, where the $(i, j)$-th entry captures the precise co-occurrence count of feature $i$ (in the Baseline SAE) with feature $j$ (in the Group SAE). That is,

$$\text{AND}_{ij}^{l_1,l_2} = |B_i^{l_1} \cap G_j^{l_2}|. \tag{14}$$

Moreover, we compute the $\text{OR}^{l_1,l_2} \in \mathbb{R}^{m \times m}$ matrix, where the $(i, j)$-th entry represents the number of tokens in which at least one of the two features is active. By the inclusion–exclusion principle, we have

$$\text{OR}_{ij}^{l_1,l_2} = |B_i^{l_1}| + |G_j^{l_2}| - \text{AND}_{ij}^{l_1,l_2}. \tag{15}$$

A common similarity measure, the Jaccard index, is then defined as

$$\text{Jaccard}_{ij}^{l_1,l_2} = \frac{\text{AND}_{ij}^{l_1,l_2}}{\text{OR}_{ij}^{l_1,l_2} + \varepsilon}, \tag{16}$$

where $\varepsilon$ is a small constant introduced to avoid division by zero. Finally, the Mean Maximum Concordance (MMCO) is given by:

$$\text{MMCO}^{l_1,l_2} = \frac{1}{m} \sum_{i=1}^{m} \max_{j \in [m]} \text{Jaccard}_{ij}^{l_1,l_2}. \tag{17}$$

Computing the MMCO across layers between Baseline and Group SAEs allows us to visualize their concordance with a heatmap. We plot the MMCO across layers for Pythia-160M, Pythia-410M and Pythia-1B in Figures 9, 10 and 11 respectively.

[8] A feature $i$ is activated on the representation $\mathbf{x}_t$ of the token $t$ if $\mathbf{f}_i(\mathbf{x}_t) > 0$.

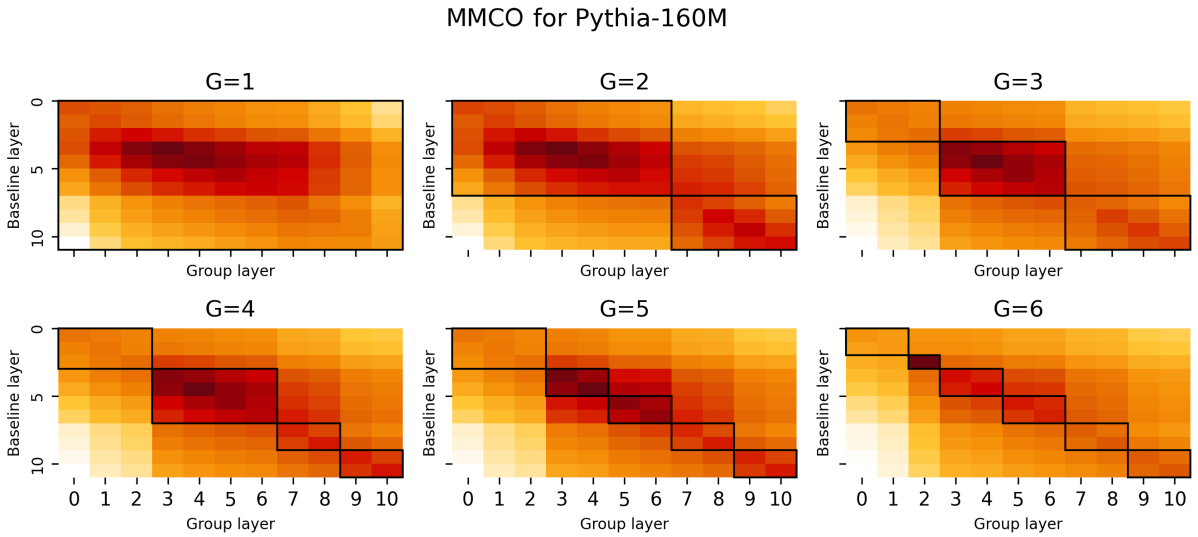



Figure 9: Heatmaps of Pythia-160M Mean Maximum Concordance (MMCO) for each group $G \in \{1, \ldots, \widehat{G}\}$ over 1 million test tokens. Darker regions indicate higher MMCO; the black squares highlight single groups in each $G$-group partition.

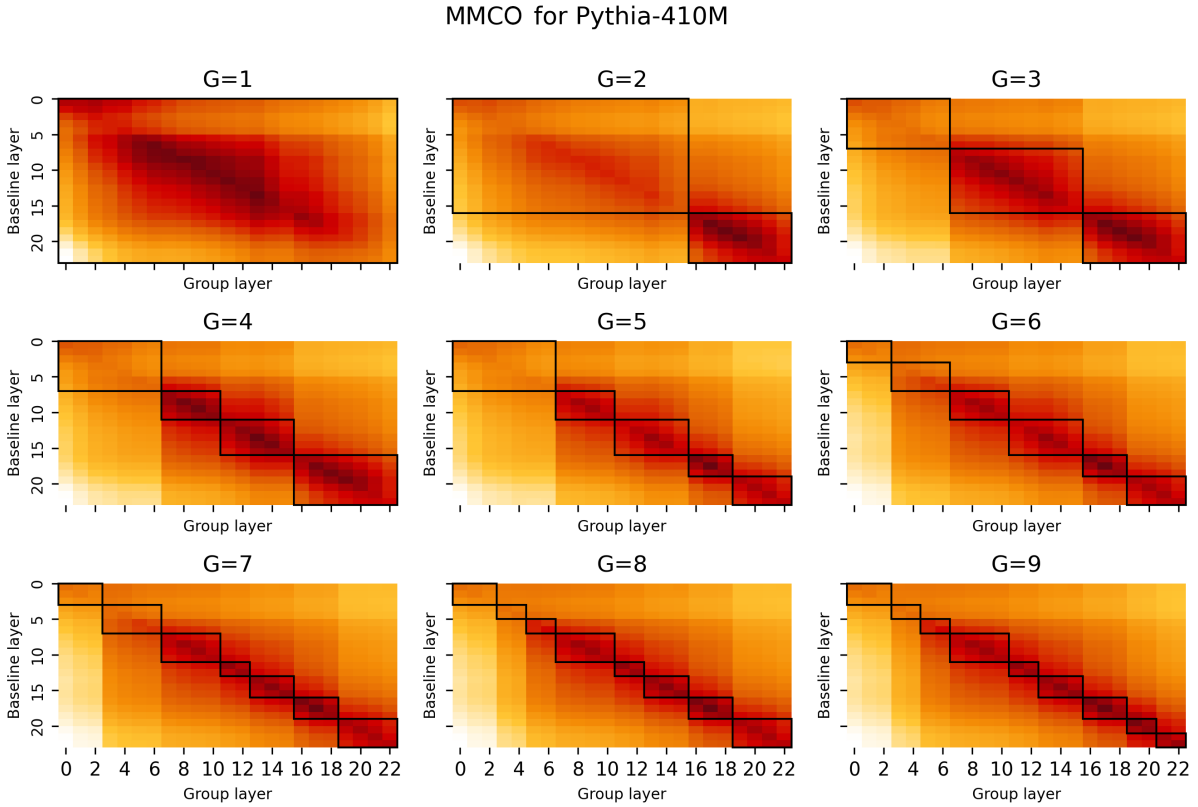



Figure 10: Heatmaps of Pythia-410M Mean Maximum Concordance (MMCO) for each group $G \in \{1, \ldots, \widehat{G}\}$ over 1 million test tokens. Darker regions indicate higher MMCO; the black squares highlight single groups in each $G$-group partition.

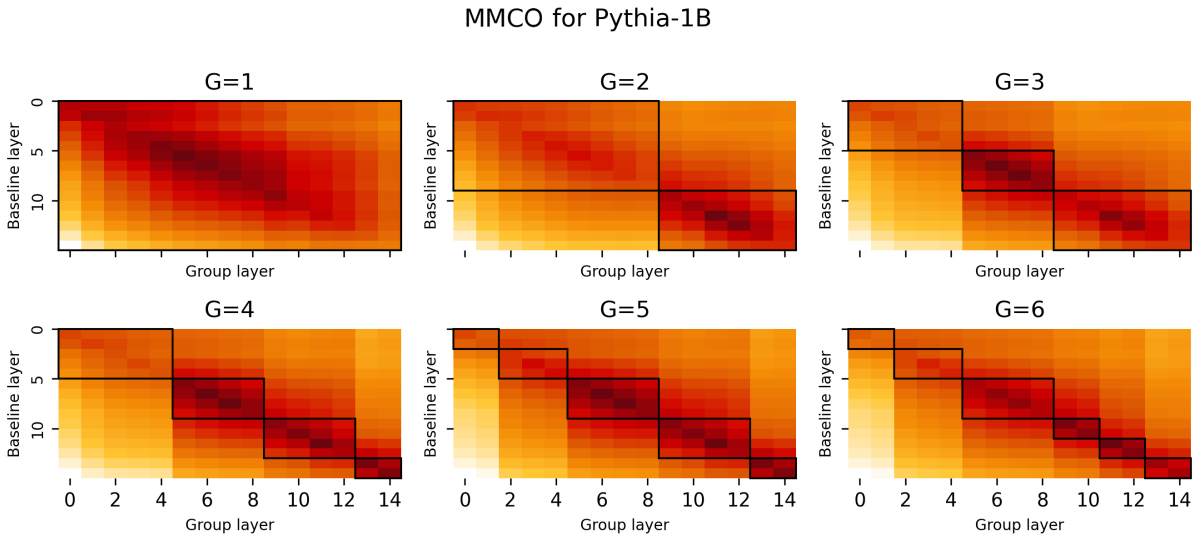



Figure 11: Heatmaps of Pythia-1B Mean Maximum Concordance (MMCO) for each group $G \in \{1, \ldots, \widehat{G}\}$ over 1 million test tokens. Darker regions indicate higher MMCO; the black squares highlight single groups in each $G$-group partition.

### E.2 Mean Maximum Cosine Similarity

Following (Ghilardi et al., 2024), we adopt Mean Maximum Cosine Similarity (MMCS) to assess the extent to which Baseline and Group SAEs learn similar feature directions. For any two SAEs, $\text{SAE}_i$ and $\text{SAE}_j$, we compute the MMCS between their decoder matrices $\mathbf{W}_d^i, \mathbf{W}_d^j \in \mathbb{R}^{n \times m}$ as these matrices encode the directions of the learned features:

$$\text{MMCS}(\mathbf{W}_d^i, \mathbf{W}_d^j) = \frac{1}{m} \sum_{k=1}^{m} \max_{l \in \{1,\ldots,m\}} \left( \cos\left( \widetilde{\mathbf{w}}_k^i, \widetilde{\mathbf{w}}_l^j \right) \right) \tag{18}$$

where $\widetilde{\mathbf{w}}_k^i$ and $\widetilde{\mathbf{w}}_l^j$ are the $k$-th and $l$-th columns of the normalized decoder matrices $\widetilde{\mathbf{W}}_d^i$ and $\widetilde{\mathbf{W}}_d^j$, respectively. The directionality of the maximum operation is important for interpretation: we first find, for each feature in $\text{SAE}_i$, the most similar feature in $\text{SAE}_j$ (by cosine similarity), and then average these maximum similarities across all features of $\text{SAE}_i$. In our analysis, we specifically compute $\text{MMCS}(\mathbf{W}_d^{\text{Baseline}}, \mathbf{W}_d^{\text{Group}})$, meaning that the resulting value represents the average highest similarity that each Baseline SAE feature has with any feature in the Group SAE.

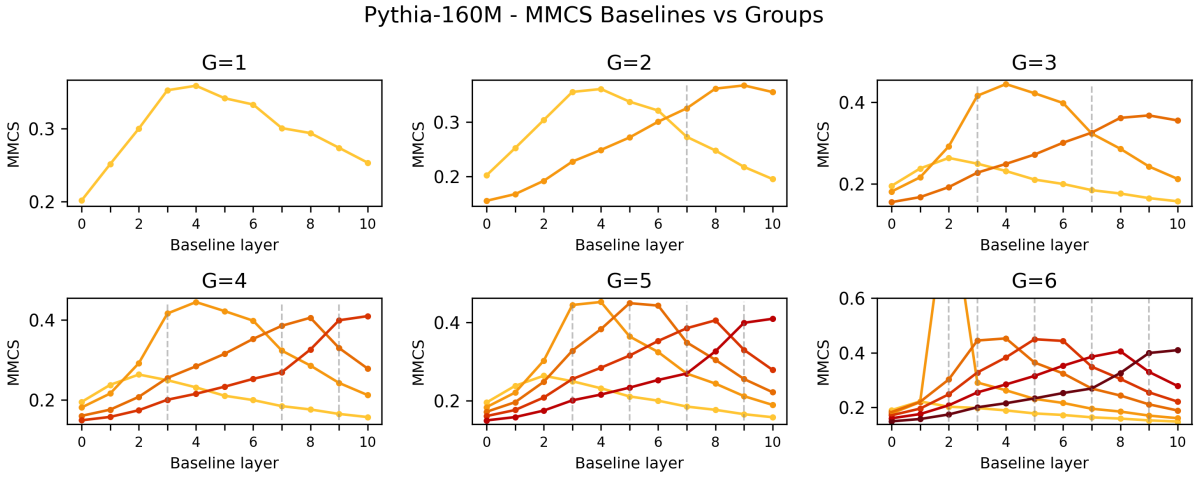



Figure 12: Mean Maximum Cosine Similarity (MMCS) between all the learned features of baseline and group SAEs for each group $G \in \{1, \ldots, \widehat{G}\}$ of Pythia-160M. Colors represent the different Group SAEs of a given partition.

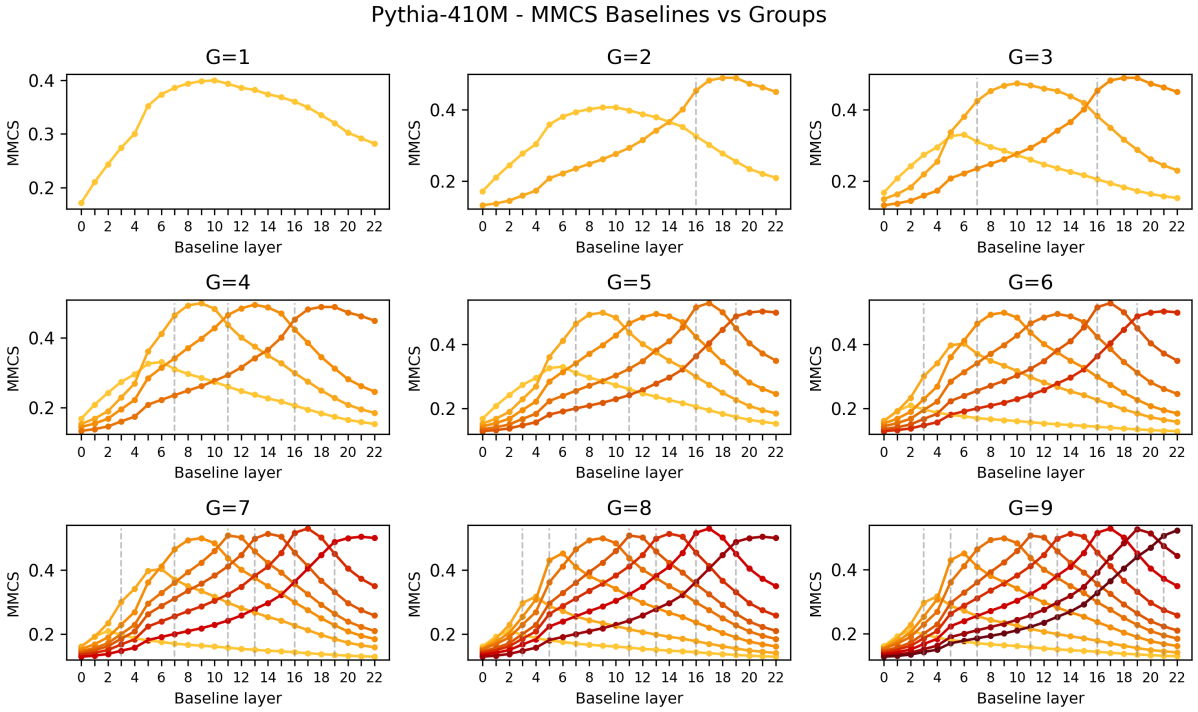



Figure 13: Mean Maximum Cosine Similarity (MMCS) between all the learned features of baseline and Group SAEs for each group $G \in \{1, \ldots, \widehat{G}\}$ of Pythia-410M. Colors represent the different Group SAEs of a given partition.

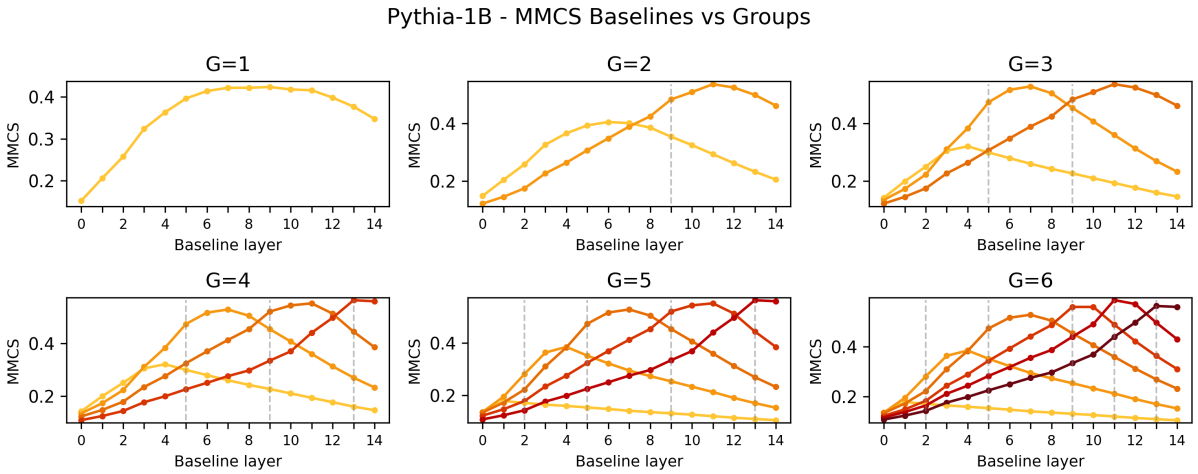



Figure 14: Mean Maximum Cosine Similarity (MMCS) between all the learned features of baseline and Group SAEs for each group $G \in \{1, \ldots, \widehat{G}\}$ of Pythia-1B. Colors represent the different Group SAEs of a given partition.

## F Feature Distribution Analysis

Following (Lawson et al., 2024), we perform a study to understand how features distribute across layers of a given group. Previous work from (Lindsey et al., 2024) showed that activations of a given feature usually peak at a specific layer. To measure this phenomenon, for each Group SAE of a given partition in $G$ groups, we sample 1 million tokens from the test set and compute feature distributions across the layers of its group.

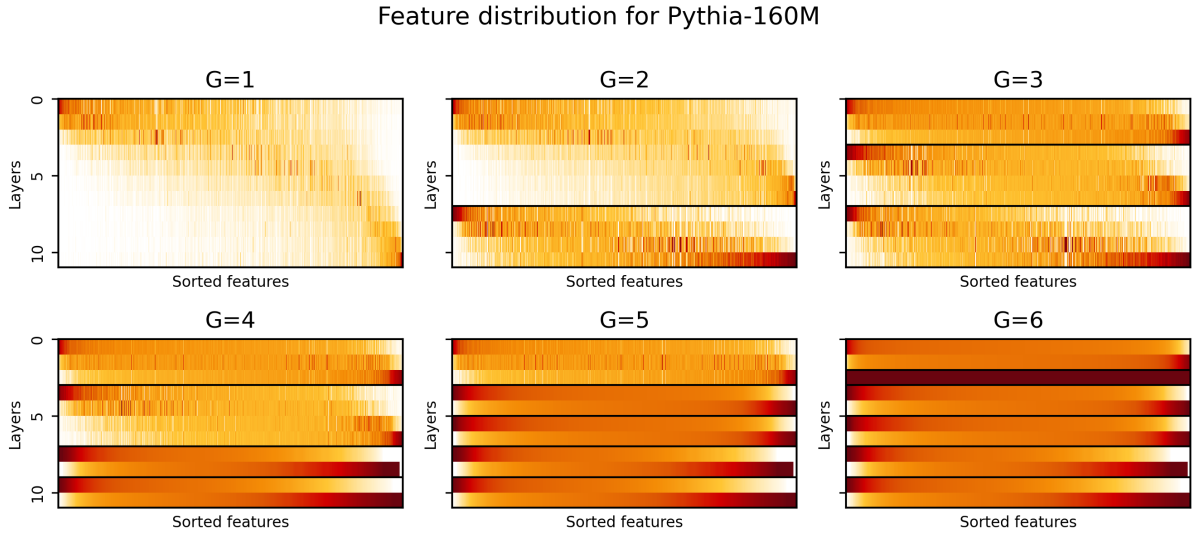



Figure 15: Pythia-160M feature activations distribution for every group $G \in \{1, ..., \widehat{G}\}$ over 1 million tokens from the test set. Darker regions indicate higher feature activation density.

Heatmaps in Figure 15, 16, and 17 show distributions of features activations for all the models and Group SAEs of partitions from 1 to $\widehat{G}$. In the images, we sort the features by the average layer they activate the most. Darker regions indicate higher feature activation density. Looking at the charts several considerations can be drawn:

- Features activating for the first and last layers of a given group tend to be more specific for that layers (i.e. their activation frequencies peak at those layers).
- Features at early layers of a model are more spread across their respective group.
- Bigger models tend to have features more spread across the layers of a given group with respect to smaller models.

In summary, while feature distributions tend to peak at a specific layer (with this being more evident in smaller models and later layers), they also spread across close ones. This result agrees with findings from (Lindsey et al., 2024) while still leaving the potential for Group SAEs to make SAE training more efficient.

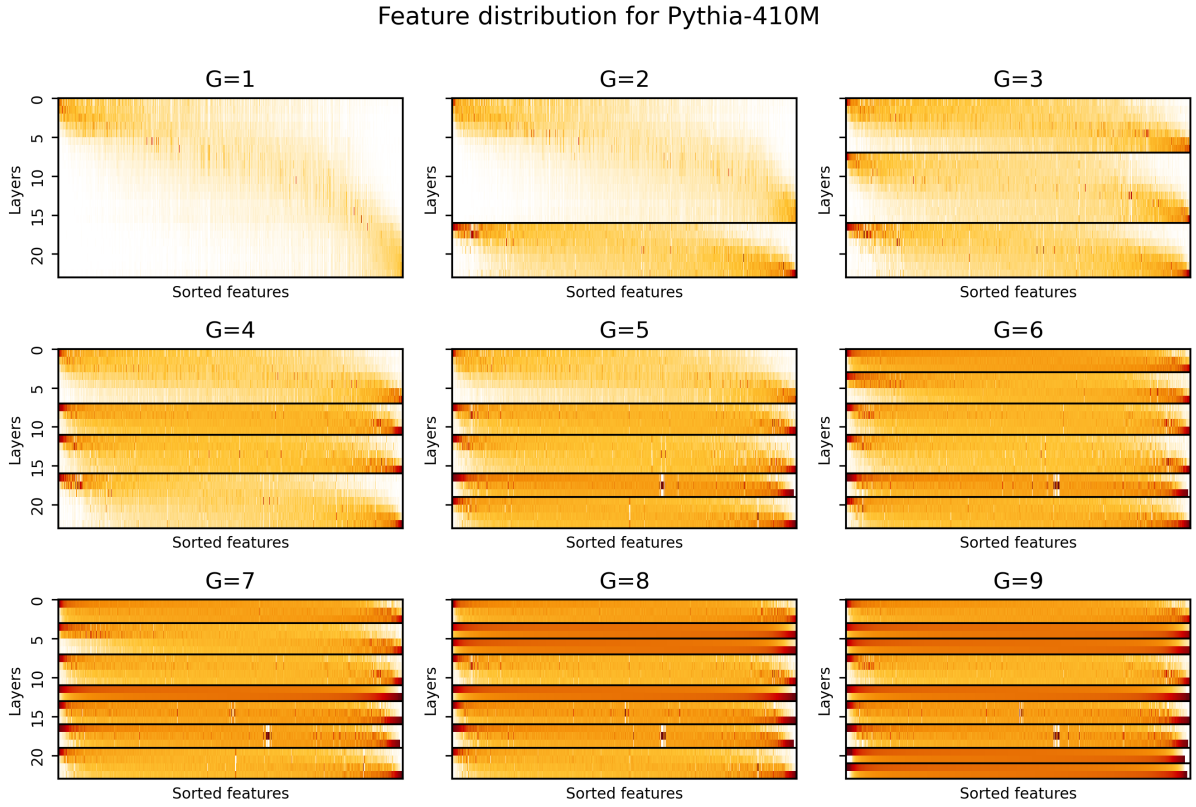



Figure 16: Pythia-410M feature activations distribution for every group $G \in \{1, ..., \widehat{G}\}$ over 1 million tokens from the test set. Darker regions indicate higher feature activation density.

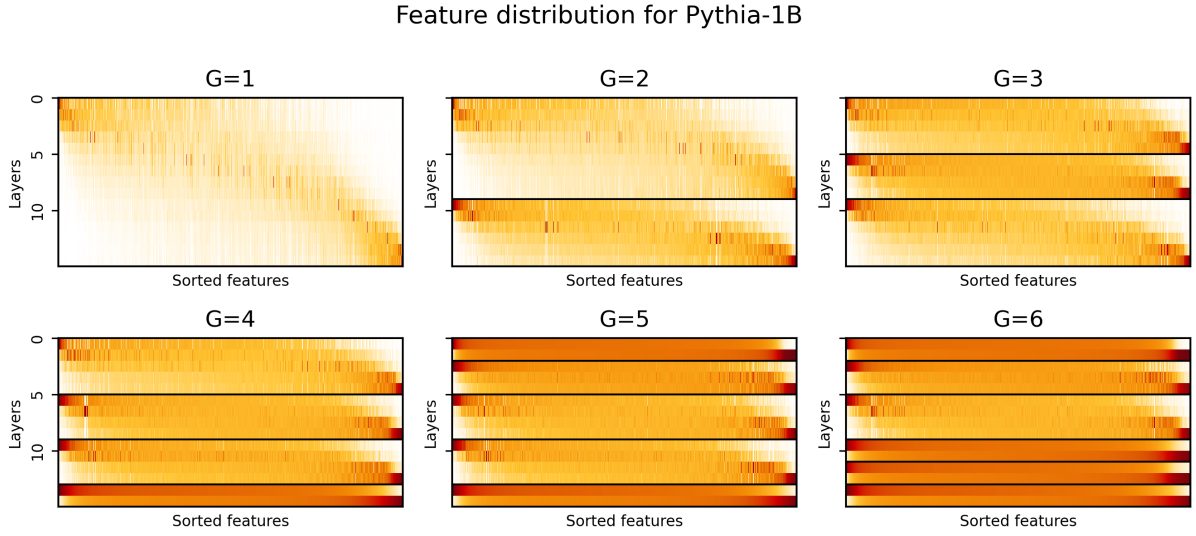



Figure 17: Pythia-1b feature activations distribution for every group $G \in \{1, ..., \widehat{G}\}$ over 1 million tokens from the test set. Darker regions indicate higher feature activation density.