# Sparse but Wrong: Incorrect L0 Leads to Incorrect Features in Sparse Autoencoders

**David Chanin**[1,3] **Adrià Garriga-Alonso**[2,3]

[1]University College London [2]FAR AI [3]MATS

## Abstract

Sparse Autoencoders (SAEs) extract features from LLM internal activations, meant to correspond to interpretable concepts. A core SAE training hyperparameter is L0: how many SAE features should fire per token on average. Existing work compares SAE algorithms using sparsity–reconstruction tradeoff plots, implying L0 is a free parameter with no single correct value aside from its effect on reconstruction. In this work we study the effect of L0 on SAEs, and show that if L0 is not set correctly, the SAE fails to disentangle the underlying features of the LLM. If L0 is too low, the SAE will mix correlated features to improve reconstruction. If L0 is too high, the SAE finds degenerate solutions that also mix features. Further, we present a proxy metric that can help guide the search for the correct L0 for an SAE on a given training distribution. We show that our method finds the correct L0 in toy models and coincides with peak sparse probing performance in LLM SAEs. We find that most commonly used SAEs have an L0 that is too low. Our work shows that L0 must be set correctly to train SAEs with correct features.

## 1 Introduction

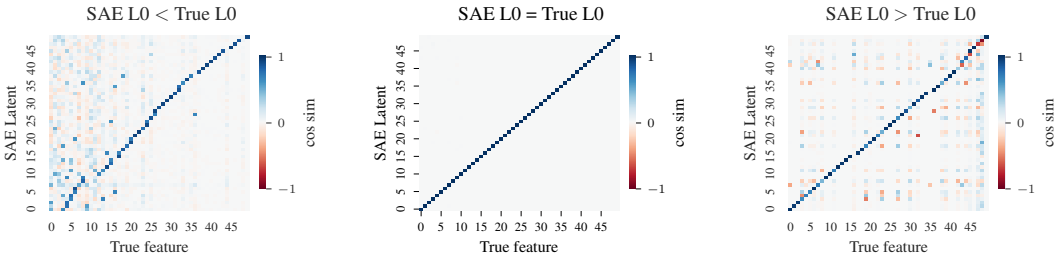
Sparse autoencoders (SAEs) decompose the dense, polysemantic activations of LLMs into interpretable latent features [8, 2] using sparse dictionary learning [17]. SAEs have the advantage of being unsupervised, and can be scaled to millions of neurons in its hidden layer (hereafter called "latents") [20, 13]. When training an SAE, practitioners must decide on the L0 of the SAE: that is, the sparsity; how many latents activate on average for a given input.[1] L0 is typically considered a neutral design choice: most of the literature evaluates SAEs at a range of L0 values, referring to this as a "sparsity–reconstruction tradeoff" [13, 18], implying any L0 is equally valid.

However, recent work shows the same trend: too low an L0 leads to worse SAE performance on downstream tasks [14, 4]. In this work, we explore the effect of L0 on SAEs. Starting by exploring a toy model with correlated features, we demonstrate that if the L0 is too low, the SAE can "cheat" by mixing together components of correlated features, achieving better reconstruction compared to an SAE with correct, disentangled features. We consider this to be a manifestation of feature hedging [6], where the SAE abuses feature correlations to compensate for insufficient resources to correctly model the underlying features. Furthermore, we show that if the L0 is too high, the SAE can learn degenerate mixtures of features. We show that it is possible to determine the correct L0 of an SAE by observing projection magnitude between the SAE decoder and training activations.

We validate these findings on Gemma-2-2b [19] and Llama-3.2-1b [9], demonstrating that the same decoder patterns we observe in our toy model experiments also manifest in LLM SAEs. We further validate that the optimal L0 we find with our method in Gemma-2-2b matches with peak performance

[1]TopK and BatchTopK SAEs [13, 3] set the L0 ($K$) directly, whereas L1 and JumpReLU [8, 2, 18] adjust it via a coefficient in the loss. In any case, all SAE trainers must decide on the target L0.

Mechanistic Interpretability Workshop at NeurIPS 2025.




(a) SAE L0 = 5 (b) SAE L0 = 11 (c) SAE L0 = 18

Figure 1: SAE decoder cosine similarity with true features. True features L0 = 11.

on sparse probing tasks [14]. Our findings show that L0 must be set correctly for SAEs to learn correct features, and implies that most SAEs used by researchers today have too low an L0.

Code is available at `https://github.com/chanind/sparse-but-wrong-paper`.

## 2 Background

**Sparse autoencoders (SAEs).** An SAE decomposes an input activation $\mathbf{x} \in \mathbb{R}^d$ into a hidden state $\mathbf{a}$ consisting of $h$ hidden neurons, called "latents". An SAE is composed of an encoder $\mathbf{W}_{\text{enc}} \in \mathbb{R}^{h \times d}$, a decoder $\mathbf{W}_{\text{dec}} \in \mathbb{R}^{d \times h}$, a decoder bias $\mathbf{b}_{\text{dec}} \in \mathbb{R}^d$, and encoder bias $\mathbf{b}_{\text{enc}} \in \mathbb{R}^h$, and a nonlinearity $\sigma$, typically ReLU or a variant like JumpReLU [18], TopK [13] or BatchTopK [3].

$$\mathbf{a} = \sigma(\mathbf{W}_{\text{enc}}(\mathbf{x} - \mathbf{b}_{\text{dec}}) + \mathbf{b}_{\text{enc}}) \tag{1}$$

$$\hat{\mathbf{x}} = \mathbf{W}_{\text{dec}}\mathbf{a} + \mathbf{b}_{\text{dec}} \tag{2}$$

## 3 Toy model experiments

We set up a toy model with 50 mutually orthogonal true features $F = \{f_0, \ldots, f_{49}\} \in \mathbb{R}^{100}$. Each feature $f_i$ has firing probability $P_i$. We set $P(f_0) = 0.345$, and we linearly decrease $P_i$ to $P_{49} = 0.05$, so feature firing probability decreases with feature number. We randomly generate a correlation matrix, so the firings of each feature are correlated with other features. Features fire with magnitude $\sim \mathcal{N}(1.0, 0.15)$. We sampling true feature firings, and summing all firing features to create a training input for the SAE. The goal of the SAE, then, is to learn these true features despite only being trained on their summed activations. Since we have ground-truth knowledge, we know the true number of features firing on average. We call this the *true L0*, which is 11 for this toy model.

Throughout this work, we use BatchTopK SAEs [3], as this architecture allows directly controlling L0 rather than controlling it indirectly via a L1 coefficient [8, 2] or L0 coefficient [18] in the loss. For these toy model experiments, we train SAEs on 15M synthetic samples using SAELens [1].

We train SAEs with L0 that is too low (L0=5), exactly correct (L0=11), and too high (L0=18). Results are shown in Figure 1. When the SAE L0 matches the true L0 (Figure 1b), the SAE exactly learns the true features. When the SAE L0 is too low (Figure 1a), the SAE mixes components of correlated features together, especially breaking latents tracking high-frequency features. When SAE L0 is too high (Figure 1c), the SAE learns degenerate solutions that mix features together. The further the L0 is from the true L0, the more broken the SAE becomes. Interestingly, when L0 is too high the SAE still learns many correct latents, but *when L0 is too low, every latent in the SAE is affected*.

### 3.1 MSE loss incentivizes low-L0 SAEs to learn incorrect features

Why do SAEs with too low L0 not learn the true features? We take the correct SAE from Figure 1b and set L0=5, to match the SAE from Figure 1a. We then generate 100k training samples and calculate the Mean Square Error (MSE) of both these SAEs. The SAE with incorrect latents from Figure 1a achieves a MSE of $2.73$, while the SAE with ground-truth correct latents achieves a much worse MSE of $4.88$. Thus, *MSE loss actively incentivizes low L0 SAEs to learn incorrect latents*.

### 3.2 The sparsity–reconstruction tradeoff

SAE architectures are commonly evaluated using a sparsity–reconstruction tradeoff plot [8, 13, 18], where the assumption is that having better reconstruction at a given sparsity is inherently better, and indicates that the SAE is correct. Afterall, we train SAEs to reconstruct inputs, so surely an SAE that has better reconstruction must therefore be a better SAE than one that has lower reconstruction?

Sadly, this is not the case. As we discussed in Section 3.1, when the L0 of the SAE is lower than optimal, the SAE can find ways to "cheat" by engaging in feature hedging [6], and get a better MSE score by mixing components of correlated features together. This results in an SAE where the latents are not monosemantic, and do not track ground-truth features.

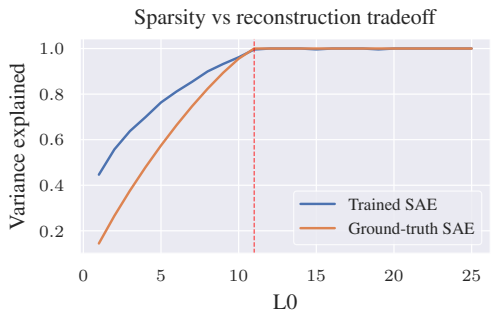



Figure 2: Sparsity ($L0$, lower is better) vs reconstruction (variance explained, higher is better) for learned SAEs and a ground-truth SAE. When L0 is less than the true L0 of the toy model (the dotted line), the trained SAE gets better reconstruction than the ground-truth SAE. Sparsity–reconstruction plots like this lead us to the incorrect conclusion that the ground-truth SAE is a worse SAE.

We next explore the sparsity–reconstruction tradeoff by training SAEs on our toy model at various L0s. Since we know the ground-truth features in our toy model, we construct a ground-truth SAE that perfectly represents these features. We vary the L0 of the ground truth SAE while leaving the encoder and decoder fixed at the correct features. We plot the variance explained vs L0 in Figure 2 for both SAEs. When the SAE L0 is lower than the true L0 of the toy model, the ground-truth SAE scores worse on reconstruction than the trained SAE! If we had an SAE training technique that gave us the ground truth correct SAE for a given LLM, sparsity–reconstruction plots would cause us to discard the correct SAE in favor of an incorrect SAE that mixes features together.

We show the cosine similarity of the SAE decoder latents with the ground truth features for the SAEs learned with L0=1 and L0=2 compared with the ground-truth SAE in Figure 3. Both these SAEs outperform the ground-truth SAE on variance explained by over 2x despite learning horribly polysemantic latents bearing little resemblance to the underlying true features of the model.

### 3.3 Detecting the true L0 using the SAE decoder

Figure 1 reveals that the SAE decoder latents contain mixes of underlying features, both when the L0 is too high and also when it is too low. As the SAE approaches the correct L0, each SAE latent has fewer components of multiple true features mixed in, becoming more monosemantic. Thus, we expect that the closer the SAE is to the correct L0, the more latents should be orthogonal relative to each other, as there are fewer components of shared correlated features mixed into latents. If we are far from the correct L0, then SAE latents contain components of many underlying features, and thus we expect latents to have higher cosine similarity with each other.

We call this metric *decoder pairwise cosine similarity*, $c_{\text{dec}}$, and define it as below:

$$c_{\text{dec}} = \frac{1}{\binom{h}{2}} \sum_{i=1}^{h-1} \sum_{j=i+1}^{h} |\cos(\mathbf{W}_{\text{dec},i}, \mathbf{W}_{\text{dec},j})| \tag{3}$$

where $\binom{h}{2} = \frac{h(h-1)}{2}$ is the total number of distinct pairs of latents in the SAE decoder.

Figure 3: SAE decoder cosine similarity with true features for the learned SAEs with L0=1 (left) and L0=2 (middle), compared with the ground-truth SAE (right). The learned SAEs score much better than the ground truth SAE on variance explained, despite their corrupted, polysemantic latents.

If SAE decoder latents are mixing lots of positive and negative components of correlated and anti-correlated features, then each SAE latent should become less orthogonal to each other SAE latent, as many latents will likely mix together similar features. This should mean that the absolute value of the cosine similarity between arbitrary latents should also increase the worse this mixing becomes.

We calculate pairwise calculate similarity $c_{\text{dec}}$ for each of the BatchTopK SAEs we trained on toy models from Section 3.3. Results are shown in Figure 4. We see that pairwise cosine similarity is minimized at the true L0.

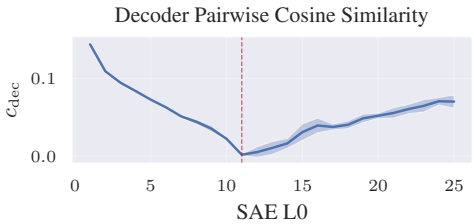



Figure 4: Decoder pairwise cosine similarity $c_{\text{dec}}$ evaluated on 5 seeds of toy model SAEs. The true L0 is indicated with a dotted line at 11. Shaded area is 1 stdev. $c_{\text{dec}}$ is minimized at the true L0.

We explore alternative metrics in Appendix A.9. Further toy model experiments are shown in Appendix A.7. Pytorch code implementing $c_{\text{dec}}$ is provided in Appendix A.16. We provide formal theoretical justification for the $c_{\text{dec}}$ metric in Appendix A.10.

## 4 LLM experiments

We train a series of BatchTopK SAEs [3] with $h = 32768$ on Gemma-2-2b [19] and Llama-3.2-1b [9] varying L0 and calculate $c_{\text{dec}}$. Each SAE is trained on 500M tokens from the Pile [12] using SAELens [1]. We also calculate k-sparse probing performance for these SAEs using the benchmark from Kantamneni et al. [14], consisting of over 100 sparse probing tasks. Results are shown in Figure 5.

The Llama SAE $c_{\text{dec}}$ plot looks very similar to the toy model, with a clear minimum point. The Gemma-2-2b layer 5 SAEs also show a sharp increase in $c_{\text{dec}}$ at low L0 as we saw in toy models, but has a long shallow region with the global minimum actually appearing in that shallow region. In both cases, the "elbow" in the $c_{\text{dec}}$ plots just before the jump due to low L0 is around L0 200, and this also corresponds to peak sparse probing performance. More plots and analysis of $c_{\text{dec}}$ curves are shown in Appendix A.14.

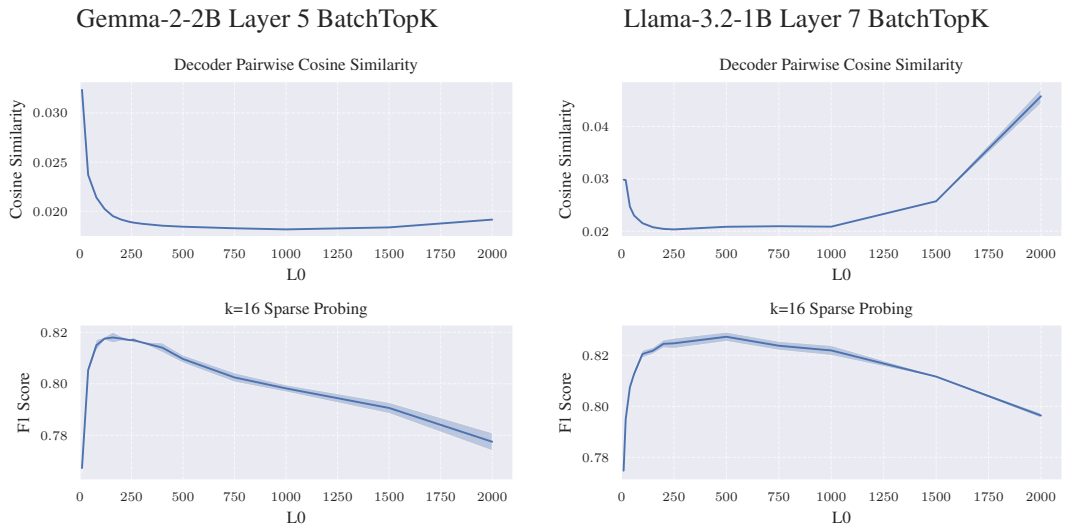



Figure 5: Decoder pairwise cosine similarity vs SAE L0 and K-sparse probing F1 vs L0 with 3 seeds per L0. (left) Gemma-2-2b layer 5 BatchTopK results. (right) Llama-3.2-1b layer 7 BatchTopK SAEs. In both cases, peak sparse probing performance occurs in the elbow just before $c_{\text{dec}}$ jumps due to low L0, although the shapes of the $c_{\text{dec}}$ plots vary at high L0.

## 5 Related work

Chanin et al. [6] explores feature hedging, showing SAEs mix correlated features into latents if the SAE is too narrow. We consider our work a version of feature hedging due to low L0. Till [21] shows SAEs may increase sparsity by inventing features. Chanin et al. [5] discuss the problem of feature absorption, where SAEs can improve their sparsity score by mixing hierarchical features together. Engels et al. [10] investigates SAE errors and finds that SAE error may be pathological and non-linear.

## 6 Discussion

While most practitioners of SAEs understand that having too high L0 is problematic, our work shows that having too low of L0 is perhaps even worse. Our work has several implications for the field. First, the L0 used by most SAEs is lower than it ideally should be, as a cursory search of open source SAEs on Neuronpedia [16] shows L0 less than 100 is very common even for SAEs trained on large models (see Appendix A.12). We further show that the sparsity–reconstruction tradeoff, as commonly discussed by most SAE papers [8, 13, 18], is misleading: when L0 is too low, an SAE with a correct dictionary achieves worse reconstruction than an incorrect SAE that mixes correlated features.

We presented a metric based on the correlation between the SAE decoder and input activations, $c_{\text{dec}}$, that can give us hints about the correct L0 for a given SAE. However, we do not view this as a perfect guide. As we saw in our results, while low L0 SAEs consistently have very high $c_{\text{dec}}$, the metric is not always high at high L0 (although it usually is). Still, we feel that this metric is a useful guide to avoid L0 that is clearly too low, and we hope this investigation into correlation-based SAE quality metrics can be built on further in future work.

While our metric currently requires training a sweep over L0 to optimize, we are hopeful that it may be possible to optimize this metric automatically during training (steps towards this are discussed in Appendix A.11). We further encourage anyone implementing ideas from this work to also experiment with our alternative metric, decoder pairwise cosine similarity, discussed in Appendix A.9.

## Acknowledgments and Disclosure of Funding

David Chanin was supported thanks to EPSRC EP/S021566/1 and the Machine Learning Alignment and Theory Scholars (MATS) program. We are grateful to Henning Bartsch and Lovkush Agarwal for feedback during the project.

## References

[1] Joseph Bloom, Curt Tigges, Anthony Duong, and David Chanin. Saelens. https://github.com/jbloomAus/SAELens, 2024.

[2] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, et al. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2, 2023.

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

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

[5] David Chanin, James Wilken-Smith, Tomáš Dulka, Hardik Bhatnagar, and Joseph Bloom. A is for absorption: Studying feature splitting and absorption in sparse autoencoders. *arXiv preprint arXiv:2409.14507*, 2024.

[6] David Chanin, Tomáš Dulka, and Adrià Garriga-Alonso. Feature hedging: Correlated features break narrow sparse autoencoders. *arXiv preprint arXiv:2505.11756*, 2025.

[7] Tom Conerly, Hoagy Cunningham, Adly Templeton, Jack Lindsey, Basil Hosmer, and Adam Jermyn. Dictionary learning optimization techniques. https://transformer-circuits.pub/2025/january-update, 2025.

[8] Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

[9] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar

Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo

Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

[10] Joshua Engels, Logan Riggs, and Max Tegmark. Decomposing the dark matter of sparse autoencoders. *arXiv preprint arXiv:2410.14670*, 2024.

[11] Joshua Engels, Eric J Michaud, Isaac Liao, Wes Gurnee, and Max Tegmark. Not all language model features are one-dimensionally linear. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=d63a4AM4hb`.

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

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

[14] Subhash Kantamneni, Joshua Engels, Senthooran Rajamanoharan, Max Tegmark, and Neel Nanda. Are sparse autoencoders useful? a case study in sparse probing. *arXiv preprint arXiv:2502.16681*, 2025.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Johnny Lin. Neuronpedia: Interactive reference and tooling for analyzing neural networks, 2023. URL `https://www.neuronpedia.org`. Software available from neuronpedia.org.

[17] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

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

[19] 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, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozińska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucińska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Görner, Mat

Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Cogan, Sarah Perrin, Sébastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomas Kocisky, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. Gemma 2: Improving open language models at a practical size, 2024. URL https://arxiv.org/abs/2408.00118.

[20] Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, Alex Tamkin, Esin Durmus, Tristan Hume, Francesco Mosconi, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. https://transformer-circuits.pub/2024/scaling-monosemanticity/, May 2024. Accessed on May 21, 2024.

[21] Demian Till. Do sparse autoencoders find true features? *LessWrong*, 2024. URL https://www.lesswrong.com/posts/QoR8noAB3Mp2KBA4B/do-sparse-autoencoders-find-true-features.

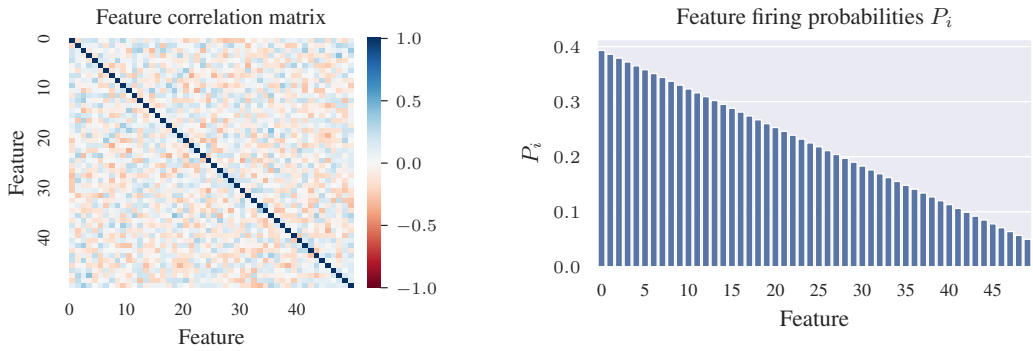



Figure 6: (left) random correlation matrix and (right) base feature firing probabilities for toy model.

# A Technical Appendices and Supplementary Material

## A.1 SAE training architecture definitions

In this work we focus on JumpReLU [7, 18] and BatchTopK [3] SAEs. For BatchTopK SAEs, there is no sparsity penalty as sparsity is enforced by the BatchTopK function. The auxiliary loss $\mathcal{L}_P$ for BatchTopK is as follows, where $e$ is the SAE training error residual, and $\hat{e}$ is a reconstruction using the top $k_{\text{aux}}$ dead latents (meaning the latents have not fired in more than $n_{\text{dead}}$ steps during training).

$$\mathcal{L}_p = \|e - \hat{e}\|_2^2$$

We follow the JumpReLU training setup from Conerly et al. [7], which involves both a sparsity loss $\mathcal{L}_s$ and a pre-activation loss for reviving dead latents, $\mathcal{L}_p$. $\mathcal{L}_s$ is defined as below, where $c$ is a scalar scaling factor:

$$\mathcal{L}_s = \sum_i \tanh(c * |a_i| \|\mathbf{W}_{\text{dec},i}\|_2)$$

The pre-activation loss $\mathcal{L}_p$ adds a small penalty for all dead features, where $a_{\text{pre}}$ refers to the pre-activation of the SAE passed into the JumpReLU:

$$\mathcal{L}_p = \sum_i \text{ReLU}(\tau_i - a_{\text{pre},i}) \|\mathbf{W}_{\text{dec},i}\|_2$$

The JumpReLU defines a pseudo-gradient relative to the threshold $\tau$ as follows, where $\epsilon$ is the bandwidth of the estimator:

$$\frac{\partial \text{JumpReLU}(x, \tau)}{\partial \tau} = \begin{cases} -\frac{\tau}{\epsilon} & \text{if } -\frac{1}{2} < \frac{x-\tau}{\epsilon} < \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

## A.2 Toy model SAE training details

We train on 15M samples with a batch size of 1024 for all toy model experiments, and a learning rate of 3e-4. We do not use any learning rate warm-up or decay. For all SAE latents vs true feature cosine similarity plots, we re-arrange the SAE latents so the latent indices match the feature indices in the plots, as this makes interpreting the plots easier without any loss of generality.

For the large toy model experiments in Section 3, we use a randomly generated correlation matrix and linearly decreasing feature firing probabilities, both shown in Figure 6.

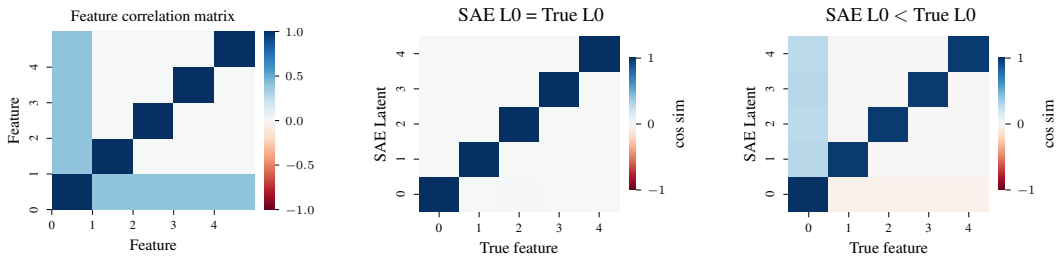



Figure 7: (left) Toy model feature correlation matrix showing positive correlations between features. (middle) SAE decoder cosine similarities with true feature when SAE L0 = 2, matching the true L0 of the toy model. (right) SAE decoder cosine similarities with true features when SAE L0 = 1.8. When L0 is too low, the SAE mixes components of features based on their firing correlations.

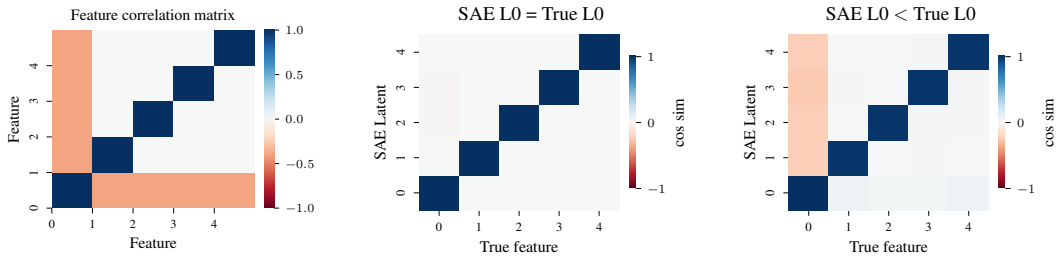



Figure 8: (left) Toy model feature correlation matrix showing negative correlations between features. (middle) SAE decoder cosine similarities with true feature when SAE L0 = 2, matching the true L0 of the toy model. (right) SAE decoder cosine similarities with true features when SAE L0 = 1.8,. When L0 is too low, the SAE mixes negative components of anti-correlated features.

### A.3 LLM SAE training details

For BatchTopK SAEs, we ensure that the decoder remains normalized with $||\mathbf{W}_{\text{dec}}||_2 = 1$ so $s_n^{\text{dec}}$ calculations use the same scale for every latent. We use a learning rate of $3e^{-4}$ with no warmup or decay.

For JumpReLU SAEs, we broadly follow the training procedure laid out by Conerly et al. [7]. However, we do not apply learning rate decay, and only warm $\lambda_s$ for 100M tokens to avoid the sparsity penalty changing throughout the majority of training. We use a learning rate of $2e^{-4}$, $c = 4$, $\lambda_p = 3e - 6$ and bandwidth $\epsilon = 2.0$ as recommended by [7].

### A.4 Small toy models: low L0 SAEs mix correlated and anti-correlated features

We begin with a small toy model with 5 true features ($g = 5$) in an input space of $d = 20$. We set each $p_i = 0.4$ such that on average 2 features are active per input, for a true L0 of 2. We begin with a simple correlation pattern between features, where $f_0$ is positively correlated with every feature $f_1$ through $f_4$, but otherwise there are no other correlations. We then train an SAE with $L0 = 2$, matching the true L0 of the model, and an SAE with slightly lower value of $L0 = 1.8$ (BatchTopK SAEs permit setting fractional L0). For the $L0 = 1.8$ SAE, we initialize it to the ground-truth solution, ensuring that the result of training is due to gradient pressure rather than just being a local minimum. We show the toy model feature correlation matrix as well as decoder cosine similarity plots with the true features for both SAEs in Figure 7.

When the SAE L0 matches the true L0, we see that the SAE perfectly learns the underlying true features. However, when SAE L0 is smaller than the true L0, the resulting SAE latents mix feature components together based on the correlation matrix. The latents tracking features $f_1$ through $f_4$ all mix in a *positive* component of $f_0$, but they have no components of each other.

Next, we invert the correlation, i.e. each feature $f_1$ through $f_4$ is negatively correlated with $f_0$ instead, while keeping everything else unchanged. We show the correlation matrix and SAE decoder cosine similarity with true features plots in Figure 8.

Now, we see the same pattern as with positive correlations except inverted. The latents tracking features $f_1$ through $f_4$ mix in a *negative* component of $f_0$, but have no component of each other.

This pattern is problematic because it means that if our L0 is too low, every SAE latent will contain positive components of every positively correlated feature, and negative components of every negatively correlated feature in the model. Negative correlations are particularly bad, as negative correlations are prevalent throughout language. For instance, we may expect a nonsensical negative component of "Harry Potter" to appear in the latent for "French poetry", since Harry Potter has nothing to do with French poetry. This will result in highly polysemantic and noisy SAE latents.

### A.5 Transitioning L0 during training

We explore the effect of transitioning the L0 of the SAE during training using the toy model from Section 3. This toy model has a true L0 of 11. We train BatchTopK SAEs with a final L0 of 11, but starting with L0 either too high or too low, and linearly transitioning to the correct L0 over the first 25k steps of training, leaving the SAE at the correct L0 for the final 5k steps of training. We use a starting L0 of 20 for the case where we start too high, and use a starting L0 of 2 for the case where we start too low. Results are shown in Figure 9.

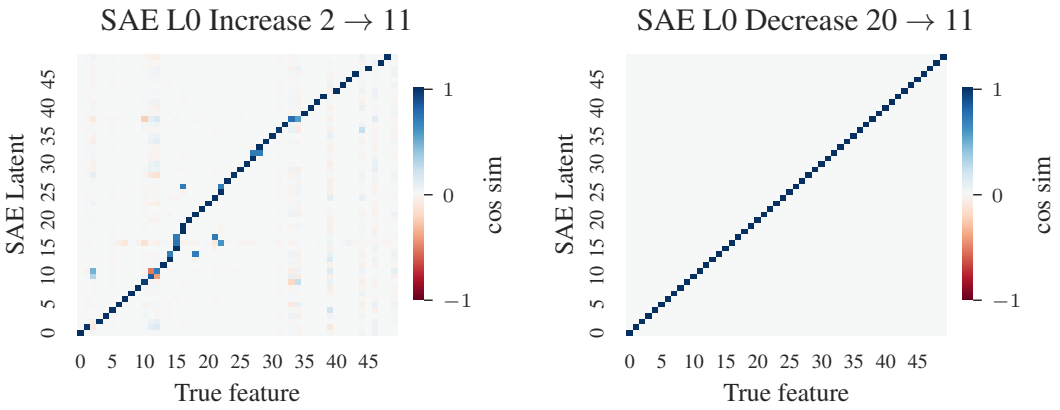



Figure 9: Transitioning L0 from too low (left) and too high (right) to the correct L0 during training. When the starting L0 is too high, the SAE still learns the correct features at the end of training. However, when L0 is too low, the SAE cannot recover fully and still learns many incorrect features at the end of training.

We see that decreasing the L0 of the SAE from a too high value to the correct value still results in the SAE learning correct features. However, when the SAE starts from a too low L0, the SAE cannot fully recover when the L0 is adjusted to the correct value later. It seems that the latents the SAE learns when L0 is too low is a local minimum that is difficult from the SAE to escape from even when the L0 is later corrected. This is likely because the latents learned when L0 is too low are optimized by gradient descent to achieve a higher MSE loss than is achievable by the correct latents under the same L0 constraint. However, when L0 is too high, there is no equivalent optimization pressure, and is thus less likely to be a local minimum.

### A.6 JumpReLU SAE toy model experiments

So far, we have only investigated BatchTopK SAES due to their ease of setting L0. We now validate that these same conclusions apply to JumpReLU SAEs. We train JumpReLU saes with a range of $\lambda_s$ to control the sparsity of the SAEs. We show plots of $\lambda_s$ vs L0 $n^{\text{th}}$ decoder projection vs L0 for these SAEs in Figure 10. We see that $n^{\text{th}}$ decoder projection vs L0 broadly follows the same pattern as we saw for BatchTopK SAEs, and is minimized at and slightly above the correct L0.

Interestingly, we see that the L0 does not change linearly with $\lambda_s$, but instead "sticks" near the correct L0. This is a testament to Anthropic's JumpReLU SAE training method [7], as a wide range of sparsity coefficients $\lambda_s$ cause the SAE to naturally find the correct L0.

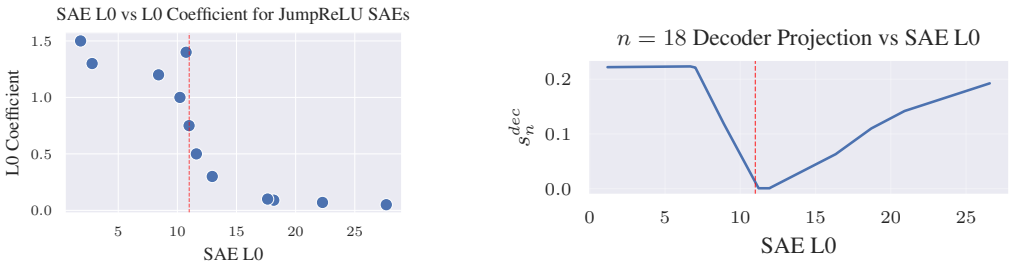



Figure 10: (left) L0 coefficient $\lambda_s$ vs L0 for JumpReLU SAEs. (right) $n^{\text{th}}$ decoder projection vs L0 for JumpReLU SAEs. The true L0, 11, is marked by a dotted line on the plot.

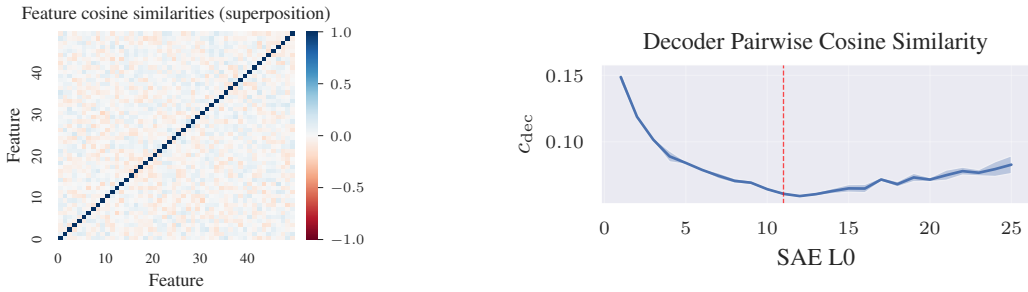



Figure 11: (Left) cosine similarity between the true features in the toy model. Due to superposition noise, each features overlaps slightly with many other features. (Right) decoder pairwise cosine similarity for SAEs trained at different L0s. The true L0 is marked with a dashed line.

### A.7 Extended large toy model experiments

In this section, we build on the results from the 50-latent toy model from Section 3.

#### A.7.1 Superposition noise

We now modify the large toy model to have superposition noise, as this is a more realistic setting for an LLM SAE to operate in. We reducing the dimensionality of the space to 40, lower than the number of features in the toy model (50). This forces each feature to slightly overlap other features in the space. The resulting feature cosine similarities are shown in Figure 11 (left).

We train 5 seeds of SAEs at a range of L0s on this superposition toy model, and calculate $c_{\text{dec}}$ in Figure 11 (right). We see that decoder pairwise cosine similarity is still roughly minimized at the true L0 of the toy model.

### A.8 Proof: Low L0 incentivizes feature mixing

We now provide a theoretical proof that when SAE L0 is less than the true L0 of the underlying features, MSE loss directly incentivizes the SAE to mix features together.

**Theorem 1.** *Consider a toy model with two orthonormal features $\mathbf{f}_1, \mathbf{f}_2 \in \mathbb{R}^d$ where $\mathbf{f}_1 \cdot \mathbf{f}_2 = 0$ and $\|\mathbf{f}_1\|_2 = \|\mathbf{f}_2\|_2 = 1$. Let $\mathbf{f}_1$ fire alone with probability $P_1$, $\mathbf{f}_2$ fire alone with probability $P_2$, and both fire together with probability $P_{12}$, where $P_1 + P_2 + P_{12} \leq 1$. Consider a tied SAE with 2 latents (i.e., $\mathbf{W}_{enc} = \mathbf{W}_{dec}^T$) and no biases that can fire at most 1 latent per input (L0 = 1). We assume this is less than the true L0 of the data (i.e., $\mathbb{E}[\text{active features}] = P_1 + P_2 + 2P_{12} > 1$), which occurs whenever features co-occur ($P_{12} > 0$). Then the SAE that minimizes expected MSE will have latents that mix $\mathbf{f}_1$ and $\mathbf{f}_2$, rather than learning them separately.*

*Proof.* We define our SAE with decoder $\mathbf{W}_{\text{dec}} = [\mathbf{l}_1, \mathbf{l}_2] \in \mathbb{R}^{d \times 2}$ where $\mathbf{l}_1, \mathbf{l}_2$ are the two latent directions. Since the SAE is tied and has no biases, the reconstruction of an input $\mathbf{x}$ using a single

active latent $\mathbf{l}_i$ (selected via Top-1 projection) is:

$$\hat{\mathbf{x}} = (\mathbf{l}_i \cdot \mathbf{x})\mathbf{l}_i \tag{4}$$

The reconstruction loss for a single sample is:

$$\mathcal{L}(\mathbf{x}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 = \|\mathbf{x} - (\mathbf{l}_i \cdot \mathbf{x})\mathbf{l}_i\|_2^2 \tag{5}$$

**Parameterization.** We parameterize the latents as:

$$\mathbf{l}_2 = \mathbf{f}_2 \tag{6}$$

$$\mathbf{l}_1 = \frac{\alpha\mathbf{f}_1 + (1-\alpha)\mathbf{f}_2}{\sqrt{\alpha^2 + (1-\alpha)^2}} \tag{7}$$

where $0 \leq \alpha \leq 1$ controls the mixture. When $\alpha = 1$, $\mathbf{l}_1 = \mathbf{f}_1$ (the correct, disentangled solution). When $0 \leq \alpha < 1$, $\mathbf{l}_1$ mixes both features.

**Case analysis.** We analyze the four possible cases:

**Case 1:** Only $\mathbf{f}_1$ fires (probability $P_1$). The input is $\mathbf{x} = m_1\mathbf{f}_1$ where $m_1 > 0$ is the magnitude. Latent $\mathbf{l}_1$ activates (since it has the largest projection). The reconstruction loss is:

$$\begin{aligned}
\mathcal{L}_1(\alpha) &= \|m_1\mathbf{f}_1 - (\mathbf{l}_1 \cdot m_1\mathbf{f}_1)\mathbf{l}_1\|_2^2 && (8)\\
&= \left\| m_1\mathbf{f}_1 - \frac{m_1\alpha}{\sqrt{\alpha^2 + (1-\alpha)^2}} \cdot \frac{\alpha\mathbf{f}_1 + (1-\alpha)\mathbf{f}_2}{\sqrt{\alpha^2 + (1-\alpha)^2}} \right\|_2^2 && (9)\\
&= \left\| m_1\mathbf{f}_1 - \frac{m_1\alpha^2}{\alpha^2 + (1-\alpha)^2}\mathbf{f}_1 - \frac{m_1\alpha(1-\alpha)}{\alpha^2 + (1-\alpha)^2}\mathbf{f}_2 \right\|_2^2 && (10)\\
&= m_1^2 \left[ \left(1 - \frac{\alpha^2}{\alpha^2 + (1-\alpha)^2}\right)^2 + \left(\frac{\alpha(1-\alpha)}{\alpha^2 + (1-\alpha)^2}\right)^2 \right] && (11)
\end{aligned}$$

Simplifying using $\alpha^2 + (1-\alpha)^2 = 1 - 2\alpha(1-\alpha)$:

$$\begin{aligned}
\mathcal{L}_1(\alpha) &= m_1^2 \left[ \left(\frac{(1-\alpha)^2}{\alpha^2 + (1-\alpha)^2}\right)^2 + \left(\frac{\alpha(1-\alpha)}{\alpha^2 + (1-\alpha)^2}\right)^2 \right] && (12)\\
&= m_1^2 \cdot \frac{(1-\alpha)^2[\alpha^2 + (1-\alpha)^2]}{[\alpha^2 + (1-\alpha)^2]^2} && (13)\\
&= m_1^2 \cdot \frac{(1-\alpha)^2}{\alpha^2 + (1-\alpha)^2} && (14)
\end{aligned}$$

**Case 2:** Only $\mathbf{f}_2$ fires (probability $P_2$). The input is $\mathbf{x} = m_2\mathbf{f}_2$. Latent $\mathbf{l}_2 = \mathbf{f}_2$ activates, giving perfect reconstruction:

$$\mathcal{L}_2 = 0 \tag{15}$$

**Case 3:** Both $\mathbf{f}_1$ and $\mathbf{f}_2$ fire (probability $P_{12}$). The input is $\mathbf{x} = m_1\mathbf{f}_1 + m_2\mathbf{f}_2$. Since L0 = 1, only one latent can activate. The SAE will choose $\mathbf{l}_1$ if $|\mathbf{l}_1 \cdot \mathbf{x}|^2 > |\mathbf{l}_2 \cdot \mathbf{x}|^2$. We have:

$$|\mathbf{l}_1 \cdot \mathbf{x}|^2 = \left(\frac{m_1\alpha + m_2(1-\alpha)}{\sqrt{\alpha^2 + (1-\alpha)^2}}\right)^2 = \frac{(m_1\alpha + m_2(1-\alpha))^2}{\alpha^2 + (1-\alpha)^2} \tag{16}$$

$$|\mathbf{l}_2 \cdot \mathbf{x}|^2 = m_2^2 \tag{17}$$

For simplicity, we assume $m_1 \geq m_2 > 0$, so $\mathbf{l}_1$ will activate when $\alpha$ is sufficiently large (e.g., for $\alpha = 1$, $|\mathbf{l}_1 \cdot \mathbf{x}|^2 = m_1^2 > m_2^2$). Assuming $\mathbf{l}_1$ activates, the reconstruction loss is:

$$\begin{aligned}
\mathcal{L}_3(\alpha) &= \|m_1\mathbf{f}_1 + m_2\mathbf{f}_2 - (\mathbf{l}_1 \cdot (m_1\mathbf{f}_1 + m_2\mathbf{f}_2))\mathbf{l}_1\|_2^2 && (18)\\
&= \left\| m_1\mathbf{f}_1 + m_2\mathbf{f}_2 - \frac{m_1\alpha + m_2(1-\alpha)}{\sqrt{\alpha^2 + (1-\alpha)^2}} \cdot \frac{\alpha\mathbf{f}_1 + (1-\alpha)\mathbf{f}_2}{\sqrt{\alpha^2 + (1-\alpha)^2}} \right\|_2^2 && (19)
\end{aligned}$$

Let $c = \frac{m_1\alpha + m_2(1-\alpha)}{\alpha^2+(1-\alpha)^2}$. Then:

$$\mathcal{L}_3(\alpha) = \|m_1\mathbf{f}_1 + m_2\mathbf{f}_2 - c\alpha\mathbf{f}_1 - c(1-\alpha)\mathbf{f}_2\|_2^2 \tag{20}$$

$$= (m_1 - c\alpha)^2 + (m_2 - c(1-\alpha))^2 \tag{21}$$

Expanding and simplifying (see detailed algebra below):

$$\mathcal{L}_3(\alpha) = \frac{[m_1(1-\alpha) - m_2\alpha]^2}{\alpha^2 + (1-\alpha)^2} \tag{22}$$

Note that when $m_1 = m_2 = m$, this simplifies to:

$$\mathcal{L}_3(\alpha) = \frac{m^2(1-2\alpha)^2}{\alpha^2 + (1-\alpha)^2} \tag{23}$$

which equals 0 when $\alpha = 0.5$ (perfect reconstruction when features are equally mixed) and equals $m^2$ when $\alpha = 1$ (complete failure to reconstruct $\mathbf{f}_2$).

**Case 4:** Neither feature fires (probability $P_0 = 1 - P_1 - P_2 - P_{12}$). Perfect reconstruction with $\mathcal{L}_4 = 0$.

**Expected loss.** The expected loss is:

$$\mathbb{E}[\mathcal{L}(\alpha)] = P_1\mathbb{E}_{m_1}[\mathcal{L}_1(\alpha)] + P_2\mathbb{E}_{m_2}[\mathcal{L}_2(\alpha)] + P_{12}\mathbb{E}_{m_1,m_2}[\mathcal{L}_3(\alpha)] \tag{24}$$

Assuming $\mathbf{l}_1$ activates in Case 1 and $\mathbf{l}_2$ in Case 2 (which holds for $\alpha > 0.5$):

$$\mathbb{E}[\mathcal{L}(\alpha)] = P_1\mathbb{E}_{m_1}\left[m_1^2\frac{(1-\alpha)^2}{\alpha^2+(1-\alpha)^2}\right] + P_{12}\mathbb{E}_{m_1,m_2}\left[\frac{[m_1(1-\alpha) - m_2\alpha]^2}{\alpha^2+(1-\alpha)^2}\right] \tag{25}$$

**Concrete example demonstrating feature mixing.** To make this concrete, suppose $m_1 = m_2 = 1$ (both features have equal magnitude when they fire). Assume equal probabilities $P_1 = P_{12} = 0.4$, which implies $P_2 = 0$ (or is negligible) and $P_0 = 0.2$.

For the disentangled solution ($\alpha = 1$, so $\mathbf{l}_1 = \mathbf{f}_1$):

$$\mathcal{L}_1(\alpha = 1) = 0 \quad \text{(perfect reconstruction when only } \mathbf{f}_1 \text{ fires)} \tag{26}$$

$$\mathcal{L}_3(\alpha = 1) = \frac{m^2(1-2)^2}{1^2+0^2} = m^2 = 1 \quad \text{(cannot reconstruct } \mathbf{f}_2 \text{ component)} \tag{27}$$

Expected loss: $\mathbb{E}[\mathcal{L}(\alpha = 1)] = (0.4 \times 0) + (0.4 \times 1) = 0.4$

For a mixed solution ($\alpha = 0.6$):

$$\mathcal{L}_1(\alpha = 0.6) = 1^2 \cdot \frac{(1-0.6)^2}{0.6^2+0.4^2} = \frac{0.16}{0.52} \approx 0.308 \tag{28}$$

$$\mathcal{L}_3(\alpha = 0.6) = 1^2 \cdot \frac{(1-2\times 0.6)^2}{0.6^2+0.4^2} = \frac{(-0.2)^2}{0.52} = \frac{0.04}{0.52} \approx 0.077 \tag{29}$$

Expected loss: $\mathbb{E}[\mathcal{L}(\alpha = 0.6)] = (0.4 \times 0.308) + (0.4 \times 0.077) \approx 0.1232 + 0.0308 = 0.154$

The mixed solution achieves $\mathbb{E}[\mathcal{L}(\alpha = 0.6)] \approx 0.154 < 0.4 = \mathbb{E}[\mathcal{L}(\alpha = 1)]$, demonstrating that MSE loss directly incentivizes feature mixing when L0 is constrained below the true L0.

**Optimal mixing coefficient.** More generally, for the case $m_1 = m_2 = m$, the expected loss is:

$$\mathbb{E}[\mathcal{L}(\alpha)] = \frac{m^2}{\alpha^2+(1-\alpha)^2}\left[P_1(1-\alpha)^2 + P_{12}(1-2\alpha)^2\right] \tag{30}$$

At the boundaries:

- At $\alpha = 1$ (disentangled): $\mathbb{E}[\mathcal{L}(1)] = P_{12}m^2$

- At $\alpha = 0.5$ (maximally mixed): $\mathbb{E}[\mathcal{L}(0.5)] = \frac{P_1 m^2 (0.5)^2}{0.5^2+0.5^2} = \frac{P_1 m^2 (0.25)}{0.5} = \frac{P_1 m^2}{2}$

When $P_{12} > P_1/2$, we have $\mathbb{E}[\mathcal{L}(0.5)] < \mathbb{E}[\mathcal{L}(1)]$, showing that mixing reduces loss when both features frequently co-occur. For instance, with $P_1 = P_{12} = 0.5$ and $m = 1$:

$$\mathbb{E}[\mathcal{L}(\alpha = 1)] = 0.5 \tag{31}$$
$$\mathbb{E}[\mathcal{L}(\alpha = 0.5)] = 0.25 \tag{32}$$

This demonstrates that when features frequently co-occur ($P_{12}$ is large), the MSE-optimal solution involves substantial feature mixing ($\alpha^* < 1$) rather than learning them disentangled ($\alpha = 1$), completing the proof. □

**Detailed algebra for Case 3.** Starting from:

$$\mathcal{L}_3(\alpha) = (m_1 - c\alpha)^2 + (m_2 - c(1-\alpha))^2 \tag{33}$$

where $c = \frac{m_1\alpha + m_2(1-\alpha)}{\alpha^2 + (1-\alpha)^2}$.

Expanding:

$$\mathcal{L}_3 = m_1^2 - 2m_1 c\alpha + c^2\alpha^2 + m_2^2 - 2m_2 c(1-\alpha) + c^2(1-\alpha)^2 \tag{34}$$
$$= m_1^2 + m_2^2 + c^2[\alpha^2 + (1-\alpha)^2] - 2c[m_1\alpha + m_2(1-\alpha)] \tag{35}$$

Note that $c[\alpha^2 + (1-\alpha)^2] = m_1\alpha + m_2(1-\alpha)$ by definition of $c$. Therefore:

$$\mathcal{L}_3 = m_1^2 + m_2^2 + c[m_1\alpha + m_2(1-\alpha)] - 2c[m_1\alpha + m_2(1-\alpha)] \tag{36}$$
$$= m_1^2 + m_2^2 - c[m_1\alpha + m_2(1-\alpha)] \tag{37}$$
$$= m_1^2 + m_2^2 - \frac{[m_1\alpha + m_2(1-\alpha)]^2}{\alpha^2 + (1-\alpha)^2} \tag{38}$$

Further simplification:

$$\mathcal{L}_3 = \frac{(m_1^2 + m_2^2)[\alpha^2 + (1-\alpha)^2] - [m_1\alpha + m_2(1-\alpha)]^2}{\alpha^2 + (1-\alpha)^2} \tag{39}$$

The numerator expands to:

$$(m_1^2 + m_2^2)[\alpha^2 + (1-\alpha)^2] - [m_1^2\alpha^2 + 2m_1m_2\alpha(1-\alpha) + m_2^2(1-\alpha)^2] \tag{40}$$
$$= m_1^2\alpha^2 + m_1^2(1-\alpha)^2 + m_2^2\alpha^2 + m_2^2(1-\alpha)^2 - m_1^2\alpha^2 - 2m_1m_2\alpha(1-\alpha) - m_2^2(1-\alpha)^2 \tag{41}$$
$$= m_1^2(1-\alpha)^2 + m_2^2\alpha^2 - 2m_1m_2\alpha(1-\alpha) \tag{42}$$
$$= [m_1(1-\alpha) - m_2\alpha]^2 \tag{43}$$

We can verify this factorization:

$$[m_1(1-\alpha) - m_2\alpha]^2 = m_1^2(1-\alpha)^2 - 2m_1m_2\alpha(1-\alpha) + m_2^2\alpha^2 \tag{44}$$

This matches. Therefore:

$$\mathcal{L}_3(\alpha) = \frac{[m_1(1-\alpha) - m_2\alpha]^2}{\alpha^2 + (1-\alpha)^2} \tag{45}$$

### A.9 Alternative metric: n$^{\text{th}}$ decoder projection

Figure 1 reveals that the SAE decoder latents contain mixes of underlying features, both when the L0 is too high and also when it is too low. As the SAE approaches the correct L0, each SAE latent has fewer components of multiple true features mixed in, becoming more monosemantic. Thus, we expect that when the SAE is at the correct L0, most latents should have near zero projection on arbitrary training inputs, because they usually do not contain the feature being tracked by that latent.

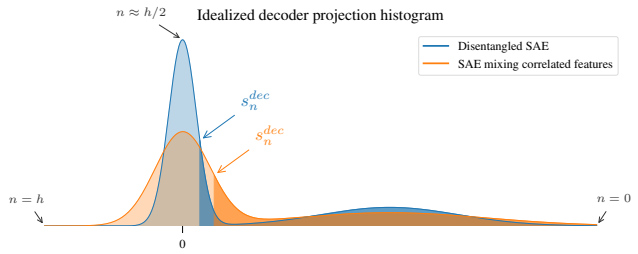



Figure 12: Idealized histogram of decoder projections on input activations demonstrating the intuition behind our $n^{\text{th}}$ decoder projection metric, $s_n^{\text{dec}}$. For an arbitrary input, most latents should be non-active and thus have low projection. When SAE latents are monosemantic, meaning they do not mix components of many features, we expect non-active latents to have a near-zero projection on arbitrary input activations. However, if SAE latents mix positive and negative components of many underlying features, then those latents will have larger projections on arbitrary inputs that contains those features. By picking an $N$ less than $h/2$ (corresponding roughly to the origin), a smaller $s_n^{\text{dec}}$ means latents have smaller projection on arbitrary inputs and thus are more monosemantic.

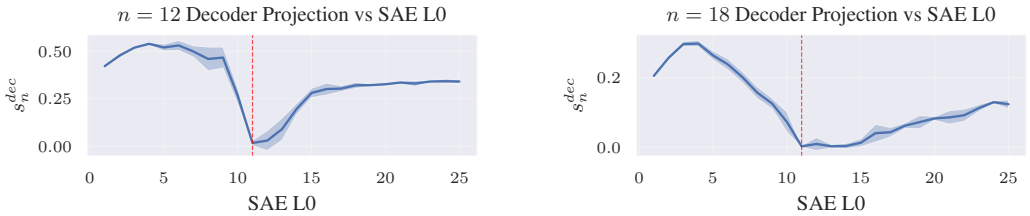



Figure 13: $n^{\text{th}}$ decoder projection vs SAE L0 for $n = 12$ (left) and $n = 18$ (right) on our toy model SAEs. The true L0, 11, is marked by a dotted line on the plots. Both settings of $n$ are minimized at the true L0, but the slopes of the metric change depending on N. The shaded area is 1 stdev.

If we are far from the correct L0, then SAE latents contain components of many underlying features, and we expect latents to project more strongly on arbitrary training inputs.

We now define a metric we call $n^{th}$ *decoder projection score*, or $s_n^{\text{dec}}$, that we can use to find the optimal L0 of the SAE. Given SAE inputs $\mathbf{x} \in \mathbb{R}^{b \times d}$ where $b$ is the batch size and $d$ is the input dimension, we first compute the decoder projections for all latents:

$$\mathbf{Z} = (\mathbf{x} - \mathbf{b}_{\text{dec}})\mathbf{W}_{\text{dec}}^{\top} \in \mathbb{R}^{b \times h} \tag{46}$$

where $\mathbf{b}_{\text{dec}} \in \mathbb{R}^{d}$ is the decoder bias and $\mathbf{W}_{\text{dec}} \in \mathbb{R}^{d \times h}$ is the decoder weight matrix with $h$ latent dimensions. To aggregate across the batch, we flatten $\mathbf{Z}$ to obtain $\mathbf{z} \in \mathbb{R}^{bh}$ and sort these values in descending order to get $\mathbf{z}_{\downarrow}$. The $n^{\text{th}}$ decoder projection is then defined as:

$$s_n^{\text{dec}} = \mathbf{z}_{\downarrow}[n \cdot b] \tag{47}$$

where $[n \cdot b]$ corresponds to selecting the element at index $n \cdot b$. The multiplication by $b$ accounts for the batch dimension, effectively selecting the $n^{\text{th}}$ highest projection value when considering all samples in the batch. For this to work $n$ should be sufficiently larger than a reasonable guess at the correct L0, as in a perfect SAE, the decoder for these latents should be uncorrelated with input activations. Picking any $n$ up to $h/2$ should work, as the majority of latents should have low projection on arbitrary input activations, so $h/2$ intuitively corresponds to 0 expected projection. The intuition behind $s_n^{dec}$ is shown visually in Figure 12, and is formalized in Appendix A.15.

We calculate $s_n^{\text{dec}}$ for $n = 12$ and $n = 18$, varying SAE L0 from 2 to 25 with 5 seeds per L0 in Figure 13. The metric is minimized at the true L0, 11, in both cases, although the shape changes depending on $n$. In both cases, the slope of $s_n^{\text{dec}}$ is flat when L0 is slightly higher than the true L0.

### A.9.1 LLM SAE results

Next, we $s_n^{\text{dec}}$ for each LLM SAE we evaluated in the paper along with k=16 sparse probing results. Gemma-2-2b layer 5 and Llama-3.2-1b layer 7 results are shown in Figure 14. The results roughly match what we saw with $c_{\text{dec}}$.

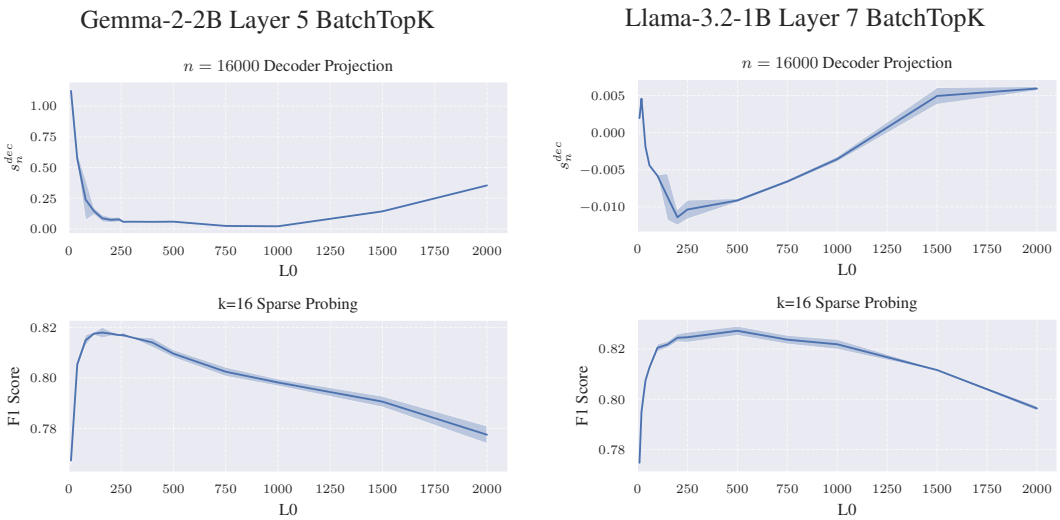



Figure 14: $n^{\text{th}}$ decoder projection and k=16 sparse probing results for BatchTopK SAEs trained on Gemma-2-2b layer 5 (left) and Llama-3.2-1b layer 7 (right). The metric is roughly minimized near peak sparse-probing performance. The shaded area is 1 stdev.

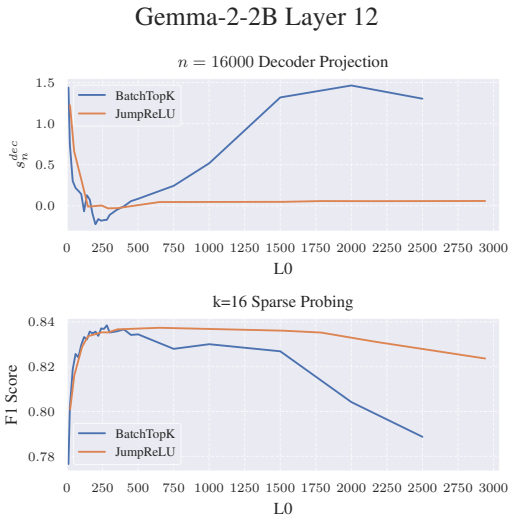



Figure 15: $n^{\text{th}}$ decoder projection and sparse probing results for BatchTopK and JumpReLU SAEs trained on Gemma-2-2b layer 12. The metric seems to align less well with k=16 sparse probing results.

BatchTopK and JumpReLU results for Gemma-2-2b layer 12 are shown in Figure 15. The results look similar to what we saw for $c_{\text{dec}}$.

### A.9.2 Which metric is better?

We choose to focus on $c_{\text{dec}}$ as it is a simpler metric both to understand and implement as it has no hyperparameters. However, we expect that when an SAE is near the correct L0, there are likely many indicators that all should point to similar results. Any metric which can detect correlated features being mixed into SAE latents should give roughly similar results.

### A.10 Theoretical Justification for $c_{\text{dec}}$ Metric

We provide a theoretical justification for why the decoder pairwise cosine similarity metric, $c_{\text{dec}}$, serves as a proxy for detecting feature mixing in SAEs.

**Theorem 2.** *Consider two SAEs with identical dictionary size $h$, where SAE 1 learns disentangled features and SAE 2 mixes a correlated feature into its latents. Let the underlying true features $\{\mathbf{f}_1, \ldots, \mathbf{f}_h, \mathbf{g}\}$ be an orthonormal set in $\mathbb{R}^d$, where $\mathbf{f}_i$ are unique features and $\mathbf{g}$ is a dense or frequent feature correlated with multiple $\mathbf{f}_i$. We model the decoder weights $\mathbf{W}_{dec,i}$ (normalized to unit length) for the two SAEs as:*

$$\textit{SAE 1 (Disentangled):} \quad \mathbf{W}_i^{(1)} = \mathbf{f}_i \tag{48}$$

$$\textit{SAE 2 (Mixed):} \quad \mathbf{W}_i^{(2)} = \sqrt{1-\gamma_i^2}\mathbf{f}_i + \gamma_i \mathbf{g} \tag{49}$$

*where $\gamma_i \in [-1, 1]$ represents the mixing coefficient for latent $i$. Assume there exists a subset of latents $S \subseteq \{1, \ldots, h\}$ with $|S| \geq 2$ such that for all $i \in S$, $\gamma_i \neq 0$. Then, the expected pairwise cosine similarity is strictly greater for SAE 2 than SAE 1:*

$$c_{dec}(\textit{SAE 2}) > c_{dec}(\textit{SAE 1}) \tag{50}$$

*Proof.* Recall the definition of decoder pairwise cosine similarity:

$$c_{\text{dec}} = \frac{1}{\binom{h}{2}} \sum_{i=1}^{h-1} \sum_{j=i+1}^{h} |\cos(\mathbf{W}_{\text{dec},i}, \mathbf{W}_{\text{dec},j})| \tag{51}$$

Since the decoder weights are normalized, $\cos(\mathbf{W}_{\text{dec},i}, \mathbf{W}_{\text{dec},j}) = \mathbf{W}_{\text{dec},i}^\top \mathbf{W}_{\text{dec},j}$.

**Case 1: SAE 1 (Disentangled).** For any distinct pair $i \neq j$, the weights are $\mathbf{W}_i^{(1)} = \mathbf{f}_i$ and $\mathbf{W}_j^{(1)} = \mathbf{f}_j$. Since the underlying features are orthonormal:

$$\mathbf{W}_i^{(1)\top} \mathbf{W}_j^{(1)} = \mathbf{f}_i^\top \mathbf{f}_j = 0 \tag{52}$$

Thus, for SAE 1:

$$c_{\text{dec}}(\text{SAE 1}) = 0 \tag{53}$$

**Case 2: SAE 2 (Mixed).** Consider the dot product for a distinct pair $i, j$:

$$\mathbf{W}_i^{(2)\top} \mathbf{W}_j^{(2)} = (\sqrt{1-\gamma_i^2}\mathbf{f}_i + \gamma_i \mathbf{g})^\top (\sqrt{1-\gamma_j^2}\mathbf{f}_j + \gamma_j \mathbf{g}) \tag{54}$$

$$= \sqrt{(1-\gamma_i^2)(1-\gamma_j^2)}(\mathbf{f}_i^\top \mathbf{f}_j) + \gamma_j \sqrt{1-\gamma_i^2}(\mathbf{f}_i^\top \mathbf{g}) + \gamma_i \sqrt{1-\gamma_j^2}(\mathbf{g}^\top \mathbf{f}_j) + \gamma_i \gamma_j (\mathbf{g}^\top \mathbf{g}) \tag{55}$$

Using the orthonormality of the set $\{\mathbf{f}_1, \ldots, \mathbf{f}_h, \mathbf{g}\}$:

- $\mathbf{f}_i^\top \mathbf{f}_j = 0$
- $\mathbf{f}_i^\top \mathbf{g} = 0$ and $\mathbf{g}^\top \mathbf{f}_j = 0$
- $\mathbf{g}^\top \mathbf{g} = 1$

The expression simplifies to:

$$\cos(\mathbf{W}_i^{(2)}, \mathbf{W}_j^{(2)}) = \gamma_i \gamma_j \tag{56}$$

The metric $c_{\text{dec}}$ is the average of absolute cosine similarities:

$$c_{\text{dec}}(\text{SAE 2}) = \frac{1}{\binom{h}{2}} \sum_{i<j} |\gamma_i \gamma_j| \tag{57}$$

Since we assumed there exists a subset $S$ where $\gamma_i \neq 0$, there exists at least one pair $(i, j)$ where $|\gamma_i \gamma_j| > 0$. All other terms are non-negative. Therefore:

$$c_{\text{dec}}(\text{SAE 2}) > 0 = c_{\text{dec}}(\text{SAE 1}) \tag{58}$$

This confirms that mixing a shared feature component into multiple latents strictly increases the $c_{\text{dec}}$ metric. □

**Remark 1.** *In real-world scenarios with superposition noise, the baseline orthogonality $\mathbf{f}_i^\top \mathbf{f}_j$ is not exactly zero but follows a distribution with mean zero and variance $\approx 1/d$. However, systematic feature mixing introduces a structured non-zero component ($\gamma_i \gamma_j$) that typically dominates the random superposition noise, causing a measurable rise in $c_{dec}$ as observed in Figure 4 and Figure 5.*

### A.11 Automatically finding the correct L0 during training

A natural next step of our finding that the correct L0 occurs when Nth decoder projection, $s_n^{\text{dec}}$, metric is minimized is to use this to find the correct L0 automatically during training. This is a meta-learning task, as the L0 is a hyperparameter of the training process. We find there are several challenges to directly using $s_n^{\text{dec}}$ as an optimization target:

- **Small gradients directly above correct L0** In our plots of $s_n^{\text{dec}}$ from both toy models and Gemma-2-2b, we find that the metric is relatively flat in a region start at the correct L0 and extending to higher L0 values. We thus need a way to traverse this flat region and stop once the metric starts to increase again.
- **The impact of changing L0 is delayed** We find that it takes many steps after changing L0 for $s_n^{\text{dec}}$ to also change, meaning it is easy to overshoot the target L0 or oscillate back and forth.
- **Dropping L0 too low can harm the SAE** As we saw in Appendix A.5, if the L0 is too low the SAE can permanently end up in poor local minima. We thus want to avoid dropping below the correct L0, even temporarily, to avoid permanently breaking the SAE. We therefore need to start with L0 too high and slowly decrease it until we find the correct L0.
- **Noise during training** We find that while $s_n^{\text{dec}}$ shows clear trends after training for many steps, it can be noisy on each training sample. So our optimization needs to be robust to this noise.

Taking these requirements into account, we present an optimization procedure to find the L0 that minimizes $s_n^{\text{dec}}$ automatically during training. We first estimate the gradient of $s_n^{\text{dec}}$, hereafter referred to as to as the metric, $m$, with respect to L0, ${}^{dm}/_{dL0}$. We first define an evaluation step $t$ as a set number of training steps (we evaluate every 100 training steps). At $t$ we change L0 by $\delta_{L_0}$. At the next evaluation step, $t+1$, we evaluate $m$. We use a sliding average of $s_n^{\text{dec}}$ over the past 10 training steps to calculate $m$ to help account for noise. We the estimate ${}^{dm}/_{dL0}$ as:

$$\frac{dm}{dL0} = \frac{m_{t+1} - m_t}{\delta_{L0}}$$

Next, we add a small negative bias to this gradient estimate to encourage our estimate to push L0 lower even if the loss landscape is relatively flat. We use a bias magnitude $0 < b < 1$ that is multiplied by the magnitude of our gradient estimate, so that our biased estimate can never change the sign of the gradient estimate, but can gently nudge it to be more negative in flat, noisy regions of the loss landscape. We find $b = 0.1$ works well. Thus, our biased gradient estimate ${}^{dm_b}/_{dL0}$ is calculated as below:

$$\frac{dm_b}{dL0} = \frac{dm}{dL0} - b\left|\frac{dm}{dL0}\right|$$

We then provide this gradient to the Adam optimizer [15] with default settings, and allow it to change the L0 parameter.

We add the following optional modifications to this algorithm. First, we clip the gradient estimates ${}^{dm}/_{dL0}$ to be between -1 and 1. We also set a minimum and maximum $\delta_{L_0}$. The minimum is added

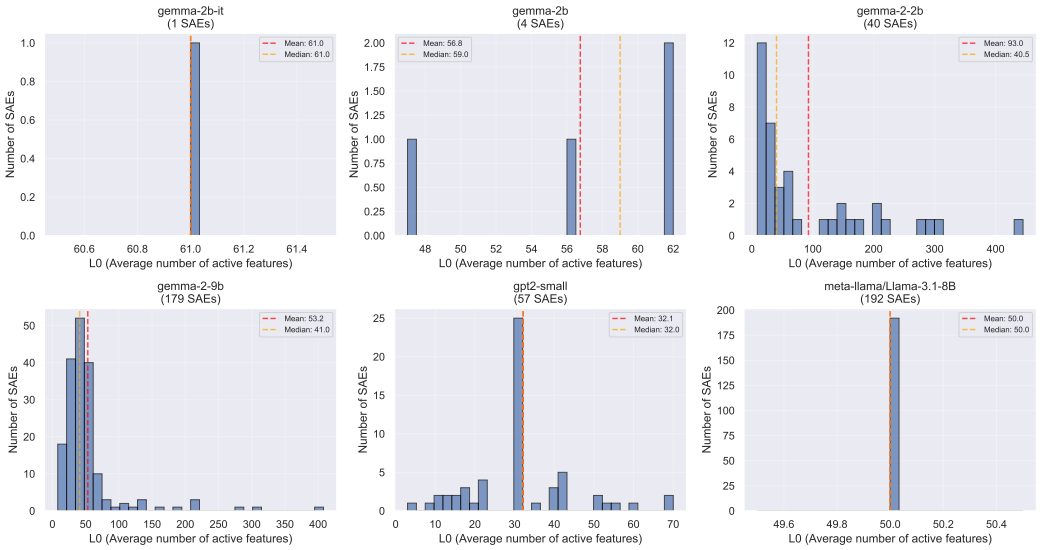



Figure 16: L0 of SAEs on Neuronpedia with known L0 listed in SAELens.

to avoid the denominator of our gradient estimate being near 0, and the maximum is chosen to keep the L0 from changing too quickly. In practice, we find a minimum $\delta_{L_0}$ between 0 and 1 seems to work well, and a maximum $\delta_{L_0}$ between 1 and 5 seems to work well.

We find that this optimization strategy works very well in toy models, but requires a lot of hyperparameter tuning to work in real LLMs, limiting its utility. The starting L0, $n$ for $s_n^{\text{dec}}$, $b$, learning rate for the Adam optimizer, and min and max $\delta_{L_0}$ values all have a big impact on how fast and how aggressively the optimization works. The slope of $m$ around the correct L0 is shallow, so it is easy to overshoot. We also find that different values of $n$ take more or less time to converge during training. We expect it is possible to further simplify and improve this process in future work.

### A.12 L0 of open-source SAEs on Neuronpedia

We analyze common open-source SAEs as provided by Neuronpedia [16] and SAELens [1]. We include all SAEs cross-listed in both SAELens and Neuronpedia with an L0 reported in SAELens. We show the results as a histogram in Figure 16. Our analysis shows that for layer 12 of Gemma-2-2b, the correct L0 should be around 200-250. However, we find that most open-source SAEs have L0 below 100, much lower than our analysis expects to be ideal.

### A.13 Limitations

We limited the scope of our investigation to features satisfying the linear representation hypothesis, and do not investigate how SAEs react if the underlying features are actually non-linear [11]. However, we do not feel that non-linear features are necessary for SAEs to fail to work properly, as we demonstrate in this paper. We also do not consider the nuances of how unbalanced correlations impact the SAE, as simple correlations are already enough to cause problems. However, we do expect that different sorts of correlations may affect SAEs differently, and would encourage future work to look into this. Finally, we only investigated a few layers of popular LLMs, as running sweeps of SAE training at every layer of the LLM was too prohibitively expensive for this work. Nevertheless, we have no reason to expect any meaningfully different behavior in decoder projection at other LLM layers.

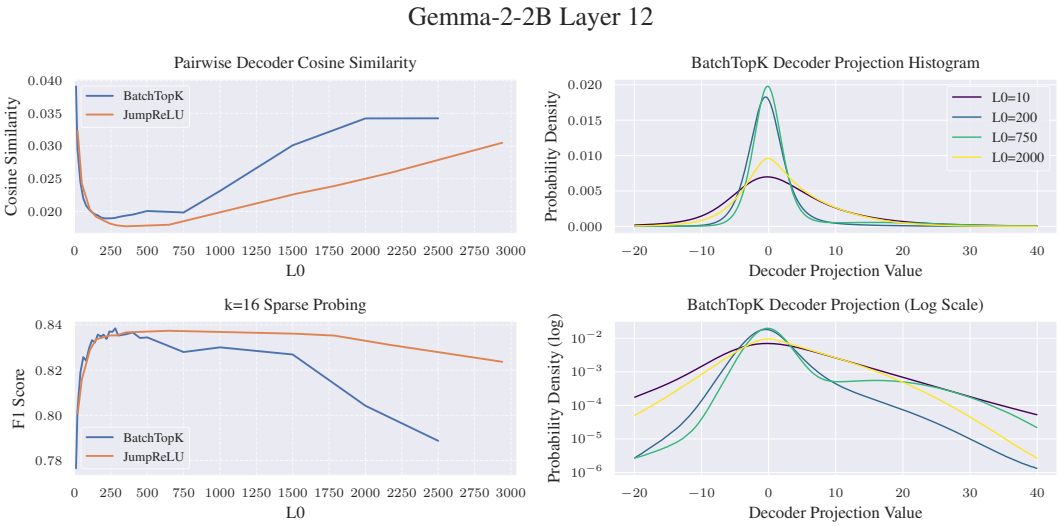



Figure 17: Gemma-2-2b layer 12, with (left) decoder pairwise cosine similarity and K-sparse probing F1 for BatchTopK and JumpReLU SAEs, and (right) normalized decoder projection histograms for BatchTopK SAEs. Histograms are truncated to -20 and 40 to highlight projections near the origin.

## A.14 Extended LLM SAE results

### A.14.1 JumpReLU vs BatchTopK SAEs

We next explore how JumpReLU and BatchTopK SAEs compare with decoder pairwise cosine similarity plots. We train a suite of SAEs on 1B tokens on Gemma-2-2b layer 12. We plot $c_{\text{dec}}$ for a range N values as well as k-sparse probing results for JumpReLU and BatchTopK SAEs in Figure 17 (left).

JumpReLU and BatchTopK SAEs behave similarly at low L0, with the high $c_{\text{dec}}$ at low L0 corresponding to poor sparse-probing performance. However, we see notable differences at high L0. The BatchTopK SAEs have a global $c_{\text{dec}}$ minimum around 200, but JumpReLU SAEs $c_{\text{dec}}$ minimum appears closer to 250-300. As we saw in Figure 5 as well, using the "elbow" of the plots just before $c_{\text{dec}}$ jumps due to low L0 seems to roughly correspond to peak k-sparse probing performance.

For JumpReLU SAEs, we see that $c_{\text{dec}}$ rises much less than BatchTopK SAEs at high L0, and indeed, JumpReLU SAEs also perform much better than BatchTopK SAEs at sparse probing when L0 is high. We suspect this is due to JumpReLU SAEs being able to "stick" near the correct threshold per latent like we saw in our toy models section. We investigate the differences in learned SAEs between JumpReLU and BatchTopK further in Appendix A.14.2.

### A.14.2 JumpReLU vs BatchTopK dynamics in LLMs

We begin by comparing the encoder bias between JumpReLU and BatchTopK in Figure 18. We see that BatchTopK SAEs rely much more heavily on the encoder bias than JumpReLU SAEs seem to, with a much wider variance in values and a sharper decrease compared to JumpReLU. We expect this is because BatchTopK cannot coordinate the cutoff threshold with the encoder directly as JumpReLU can, since there is no gradient available to directly change the threshold of BatchTopK SAEs.

Next, we inspect the threshold values between JumpReLU and BatchTopK in Figure 19. Here as well, we see dramatic differences between BatchTopK and JumpReLU SAEs. The threshold for BatchTopK is much higher than it is for JumpReLU, and the threshold decreases as L0 increses. This makes sense, since using a lower cutoff means more latents can fire. However, JumpReLU seems to unintuitively have the opposite trend, with the threshold actually *increasing* with L0. We saw in Figure 18 that the encoder bias for JumpReLU (and BatchTopK) SAEs increases as well as L0 increases, so perhaps this increase in threshold for JumpReLU SAEs with increasing L0 is just to offset that trend somewhat. We also notice that the variance in JumpReLU SAE thresholds also increases as L0 increases, supporting our hypothesis that one of the reasons JumpReLU SAEs seem

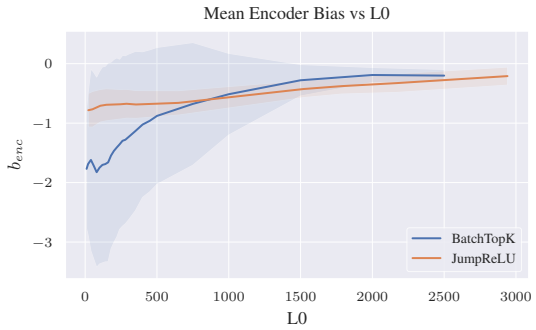



Figure 18: Mean encoder bias vs L0. Shaded area in plots corresponds to 1 stdev.

to handle high L0 better than BatchTopK is because the thresholds are able to dynamically adjust to near the correct cutoff point per latent, aleviating the situation we saw in BatchTopK SAEs where we can be at both too high and too low L0 at the same time (Section A.14.3).

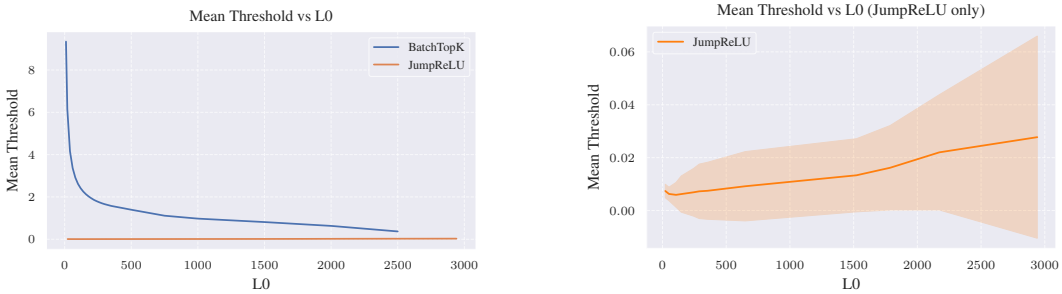



Figure 19: Threshold vs L0 for JumpReLU and BatchTopK SAEs. Shaded area in plots corresponds to 1 stdev. Interestingly, JumpReLU threshold is much lower than the BatchTopK threshold, and actually increases as L0 increases. We plot just JumpReLU on its own (right) since it is otherwise difficult to see these trends, as the threshold is so much smaller than BatchTopK.

### A.14.3 Can L0 be both too low and too high simultaneously?

In Figure 17 (right), we plot decoder histogram projection plots for BatchTopK SAEs on Gemma-2-2b layer 12 with L0 10, 200, 750, and 2000. As we expected from Figure 12, when L0 is very low (10) or very high (2000), we see a wide gaussian around 0, indicating that decoder latents are mixing correlated features together. At L0=200, we see a much more narrow distribution around 0, as we expect when near the correct L0. However, at L0=750, we see an interesting phenomenon, where there is an even narrower distribution than at L0=200, but also a large hump starting at projection above 10 (more visible in the log plot).

We suspect this indicates at L0=750, some latents become more monosemantic while other latents mix underlying features becoming less monosemantic. This likely means that the L0 is too high for some latents while simultaneously being too low for other latents. There is no reason why every latent has the same optimal L0 threshold, so there is likely a range of L0s where some latents are firing more than they ideally should while other latents are firing less than they ideally should. We also suspect this is part of why JumpReLU SAEs seem to perform much better at high L0, since JumpReLU SAEs can adjust firing threshold per-latent while BatchTopK SAEs cannot.

### A.14.4 Extended Nth decoder projection plots

In this section we document $n^{\text{th}}$ decoder projection plots for multiple values of N for each sweep of L0 we performed. We show Llama-3.2-1b layer 7 plots in Figure 22, Gemma-2-2b layer 5 in Figure 20, and Gemma-2-2b layer 12 in Figure 21. We note that in all cases, low L0 behavior is similar: no matter the value of N, $s_n^{\text{dec}}$ increases dramatically at low L0. However, the high L0 behavior is less consistent. We always see a similar "elbow" in the plots at roughly the same place

regardless of N, but sometimes this elbow corresponds to a clear global minimum, and sometimes the high L0 behavior is very shallow. We find that using a N near $h/2$ ( 16k in our cases) seems to give the best results.

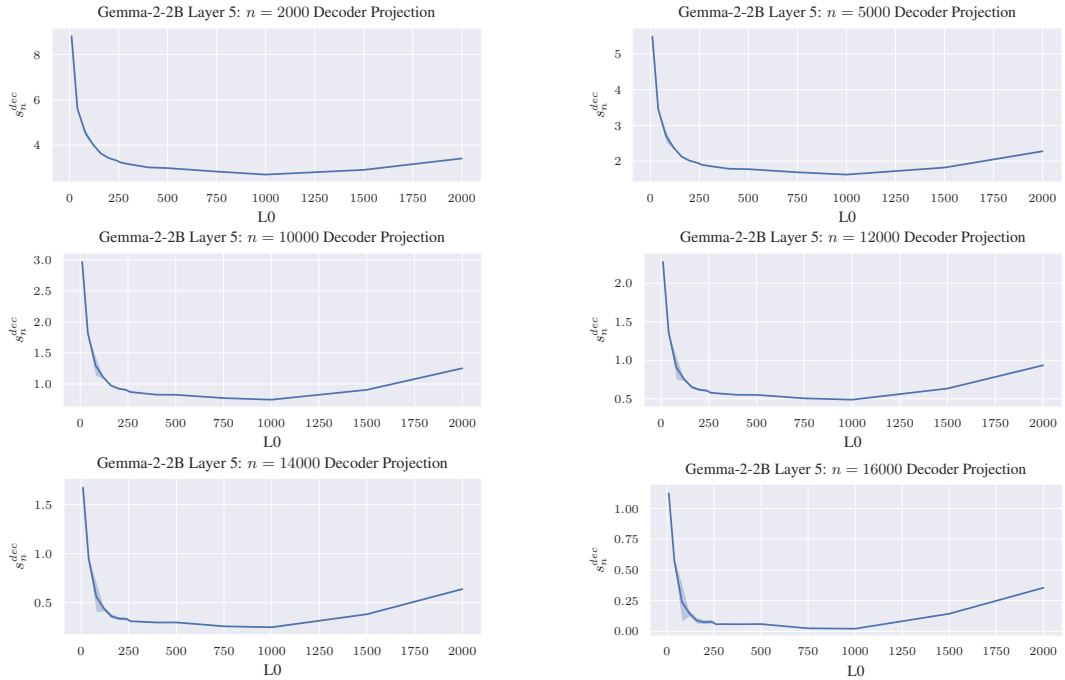



Figure 20: Extended Nth decoder projection plots. Gemma-2-2b, layer 5, 32k latents. These plots never have a clear global minimum at the "elbow" point, but the "elbow" is always at the same point regardless of choice of N.

### A.15 Theoretical Justification for $s_n^{\text{dec}}$ Metric

We provide a theoretical justification for why the $n^{\text{th}}$ decoder projection metric, $s_n^{\text{dec}}$, successfully identifies when SAE latents are mixing correlated features.

**Theorem 3.** *Consider two SAEs with identical dictionary sizes $h$ and input dimension $d$, where SAE 1 has greater feature mixing than SAE 2. Specifically, let the decoder projections onto a feature $f$ for non-active latents follow:*

$$\textit{SAE 1:} \quad z_i^{(1)} \sim \mathcal{N}(0, \sigma_0^2 + \sigma_1^2) \tag{59}$$

$$\textit{SAE 2:} \quad z_i^{(2)} \sim \mathcal{N}(0, \sigma_0^2 + \sigma_2^2) \tag{60}$$

*where $\sigma_0^2$ represents the base variance from superposition noise, and $\sigma_1^2 > \sigma_2^2$ represents the variance from feature mixing. Then for $n < h/2$, we have $\mathbb{E}[s_n^{dec}]$ is larger for SAE 1 than SAE 2.*

*Proof.* Let $f \in \mathbb{R}^d$ be an underlying true feature with $\|f\|_2 = 1$. Consider an SAE with decoder $\mathbf{W}_{\text{dec}} \in \mathbb{R}^{d \times h}$ and decoder bias $\mathbf{b}_{\text{dec}} \in \mathbb{R}^d$. For an input activation $\mathbf{x} \in \mathbb{R}^d$, the projection of latent $i$ onto the input is:

$$z_i = (\mathbf{x} - \mathbf{b}_{\text{dec}})^\top \mathbf{W}_{\text{dec},i} \tag{61}$$

**Decomposition of decoder latents.** We decompose each decoder latent $\mathbf{W}_{\text{dec},i}$ into three components:

$$\mathbf{W}_{\text{dec},i} = \alpha_i f + \beta_i g_i + \epsilon_i \tag{62}$$

where:

- $\alpha_i f$ is the component aligned with feature $f$ (the intended feature for latent $i$ if $i$ is the correct latent, or mixing if $i$ is incorrect)

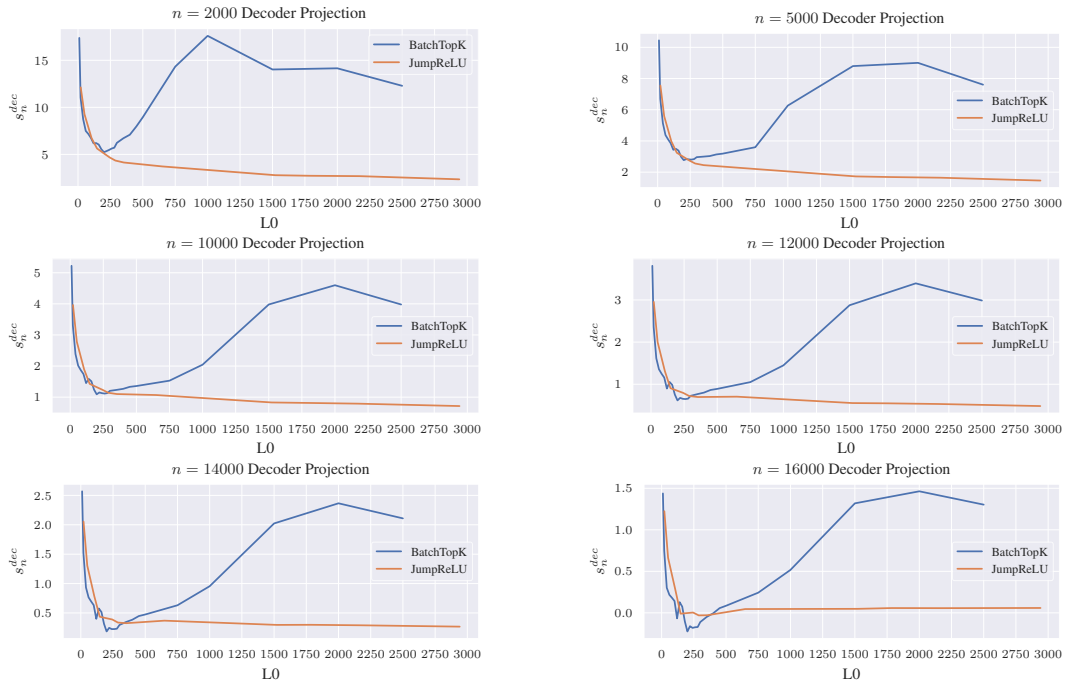



Figure 21: Extended Nth decoder projection plots. Gemma-2-2b, layer 12, 32k latents for both JumpReLU and BatchTopK. For BatchTopK, regardless of the choice of N, all plots are minimized around the same L0 range, 200-250. For JumpReLU, there is a clear "elbow" at roughly the same L0, but this elbow is only a clear minimum at N=16k.

- $\beta_i g_i$ represents components of other correlated/anti-correlated features mixed into latent $i$, where $g_i$ is orthogonal to $f$
- $\epsilon_i$ represents superposition noise, also orthogonal to $f$

**Distribution of projections for non-active latents.** Consider latents that should not activate for feature $f$ (i.e., latents $i$ where $\alpha_i$ should ideally be near zero). For an input $\mathbf{x}$ containing feature $f$ with magnitude $m_f$, we can write:

$$\mathbf{x} - \mathbf{b}_{\text{dec}} = m_f f + \mathbf{r} \tag{63}$$

where $\mathbf{r}$ contains all other feature contributions orthogonal to $f$.

The projection of latent $i$ becomes:

$$z_i = (m_f f + \mathbf{r})^\top (\alpha_i f + \beta_i g_i + \epsilon_i) = m_f \alpha_i + \mathbf{r}^\top (\beta_i g_i + \epsilon_i) \tag{64}$$

For non-active latents in a well-trained SAE, we expect:

- $\alpha_i \approx 0$ for the intended feature component
- $\beta_i g_i$ represents unintended mixing of correlated features
- $\epsilon_i$ represents superposition noise

**Modeling as Gaussian mixtures.** Under the assumptions that:

1. Feature magnitudes $m_f$ and residual components $\mathbf{r}$ vary across the input distribution
2. The number of latents $h$ is large
3. Feature mixing coefficients $\beta_i$ arise from optimization pressure to compensate for insufficient L0

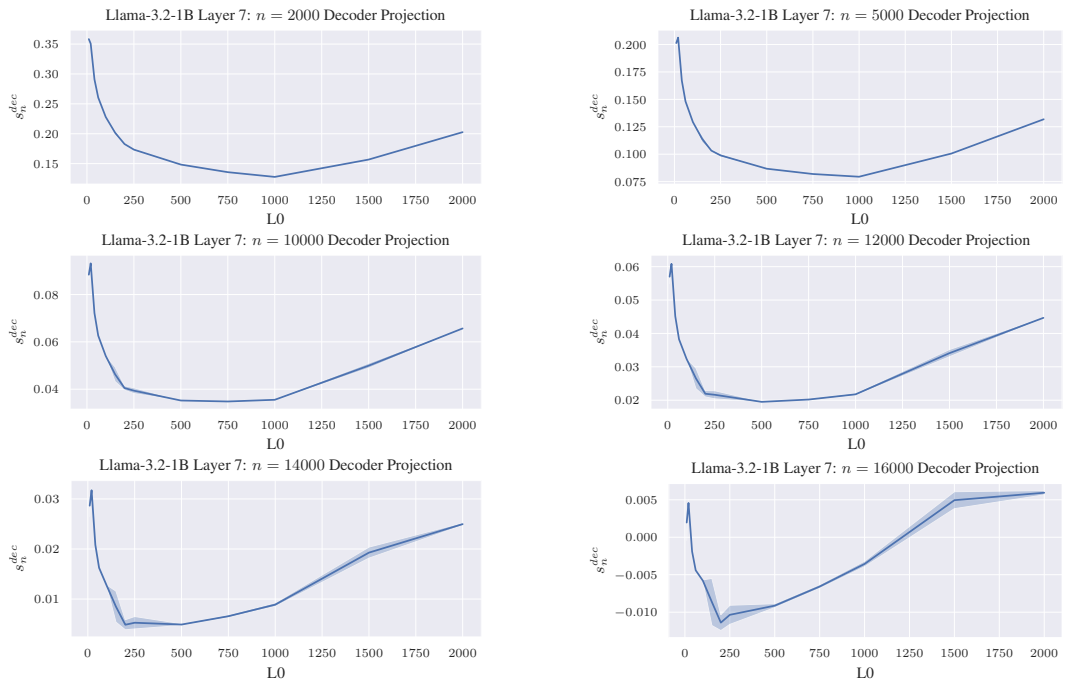



Figure 22: Extended Nth decoder projection plots. Llama-3.2-1b, layer 7, 32k latents. The plots begin to have a sharp minimum around N=14k, but the "elbow" of the plots before the decoder projection increases at low L0 is always around the same location.

By the Central Limit Theorem, the distribution of projections $z_i$ for non-active latents approximately follows:

$$z_i \sim \mathcal{N}(0, \sigma^2_{\text{base}} + \sigma^2_{\text{mix}}) \tag{65}$$

where:

- $\sigma^2_{\text{base}}$ captures variance from superposition noise ($\epsilon_i$)
- $\sigma^2_{\text{mix}}$ captures variance from feature mixing ($\beta_i g_i$)

**Comparing two SAEs.** Consider two SAEs:

- **SAE 1** (high feature mixing): $z_i^{(1)} \sim \mathcal{N}(0, \sigma^2_{\text{base}} + \sigma_1^2)$ where $\sigma_1^2$ is large
- **SAE 2** (low feature mixing): $z_i^{(2)} \sim \mathcal{N}(0, \sigma^2_{\text{base}} + \sigma_2^2)$ where $\sigma_2^2$ is small

with $\sigma_1^2 > \sigma_2^2$.

**Computing $s_n^{\text{dec}}$.** For a batch of size $b$, we have $bh$ projection values. After sorting in descending order, the $n^{\text{th}}$ decoder projection is:

$$s_n^{\text{dec}} = z_{\downarrow}[n \cdot b] \tag{66}$$

This corresponds to the $(n \cdot b)/(bh) = n/h$ quantile of the distribution. For $n < h/2$, this is the $(n/h)^{\text{th}}$ quantile on the positive side of the distribution.

**Quantile comparison.** For a standard normal distribution $Z \sim \mathcal{N}(0, 1)$ and $\sigma_1 > \sigma_2 > 0$, the $p^{\text{th}}$ quantile satisfies:

$$Q_p(\mathcal{N}(0, \sigma_1^2)) = \sigma_1 \cdot Q_p(\mathcal{N}(0, 1)) > \sigma_2 \cdot Q_p(\mathcal{N}(0, 1)) = Q_p(\mathcal{N}(0, \sigma_2^2)) \tag{67}$$

for $p > 0.5$ (corresponding to positive quantiles).

Since $n < h/2$ implies $p = n/h < 0.5$, we are actually looking at the $(1-p)^{\text{th}}$ quantile on the right tail due to descending sort. This gives us:

$$\mathbb{E}[s_n^{\text{dec}}]_{\text{SAE 1}} > \mathbb{E}[s_n^{\text{dec}}]_{\text{SAE 2}} \tag{68}$$

Therefore, SAEs with greater feature mixing (larger $\sigma^2_{\text{mix}}$) will have larger values of $s_n^{\text{dec}}$, justifying its use as a metric for detecting feature mixing. □

**Remark 2.** *The choice of $n \approx h/2$ in practice corresponds to sampling from a region where the distribution is sensitive to changes in variance (roughly near the median), while being sufficiently far from the extreme tails to maintain statistical stability. Values of $n$ too close to 0 would sample from the extreme right tail where variance is high, while $n$ too close to $h$ would sample from regions dominated by active latents rather than the non-active latents we wish to characterize.*

**Remark 3.** *This theoretical analysis assumes that decoder projections follow approximately Gaussian distributions. While this is a simplification, our empirical results in both toy models (where we have full control) and LLM SAEs support this assumption, as evidenced by the decoder projection histograms in Figure 17.*

### A.16 Pytorch pseudocode for metrics

```
def pairwise_decoder_cosine_similarity(sae):
    norm_dec = torch.nn.functional.normalize(sae.W_dec, dim=1)
    dec_sims = torch.mm(norm_dec, norm_dec.T)
    triu_mask = torch.triu(
        torch.ones_like(dec_sims),
        diagonal=1,
    ).bool()
    return dec_sims[triu_mask].abs().mean()
```

Figure 23: Pytorch pseudocode for decoder pairwise cosine similarity

```
def nth_decoder_projection(input_acts, sae, n):
    dec_proj = (input_acts - sae.b_dec) @ sae.W_dec.T
    sorted_dec_proj = dec_proj.flatten().sort(descending=True)
    index = n * dec_proj.shape[0]
    return sorted_dec_proj.values[index]
```

Figure 24: Pytorch pseudocode for n$^{\text{th}}$ decoder projection

We present Pytorch pseudocode for n$^{\text{th}}$ decoder projection in Figure 24 and decoder pairwise cosine similarity in Figure 23.