Binary Sparse Coding for Interpretability

Lucia Quirke
EleutherAI
lucia@eleuther.ai

Stepan Shabalin
Georgia Institute of Technology

Nora Belrose
EleutherAI

Abstract

Sparse autoencoders (SAEs) are used to decompose neural network activations into sparsely activating features, but many SAE features are only interpretable at high activation strengths. To address this issue we propose to use binary sparse autoencoders (BAEs) and binary transcoders (BTCs), which constrain all activations to be zero or one. We find that binarisation significantly improves the interpretability and monosemanticity of the discovered features, while increasing reconstruction error. By eliminating the distinction between high and low activation strengths, we prevent uninterpretable information from being smuggled in through the continuous variation in feature activations. However, we also find that binarisation increases the number of uninterpretable ultra-high frequency features, and when interpretability scores are frequency-adjusted, the scores for continuous sparse coders are slightly better than those of binary ones. This suggests that polysemanticity may be an ineliminable property of neural activations.

1 Introduction

Recently, large language models have reached human-level reasoning across numerous tasks [15]. The field of interpretability seeks to bolster these models' safety and dependability by elucidating their internal structures and representations. Although early efforts focused on generating natural-language descriptions for individual neurons [22, 16, 17], it is now commonly understood that the majority of neurons are "polysemantic," responding to a variety of semantically unrelated inputs [2, 12].

Sparse autoencoders (SAEs) have recently shown promise in mitigating polysemanticity by decomposing activations into more coherent, interpretable components [6, 28, 14]. An SAE is a single-hidden-layer network trained to reconstruct its input activations while enforcing sparsity through penalties [6, 25], explicit constraints [14, 7], or an information-bottleneck objective [3]. The architecture comprises an encoder that maps activations into a sparse, overcomplete latent representation, and a decoder that reconstructs the original activations from this latent code.

In general, SAE features are qualitatively more interpretable than neurons. But many SAE features are uninterpretable at low activation strengths, while interpretability evaluations often focus on high activation strengths. There is a general concern that SAEs might "hide" uninterpretable information in the exact activation strength of each active feature. In this work, we address these issues by *binarising* SAE activations, constraining them to take values in the set $\{0, 1\}$. We develop a sigmoid-based straight through estimator (STE) for the gradient [4], allowing us to differentiate through the discontinuous binarisation operation.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

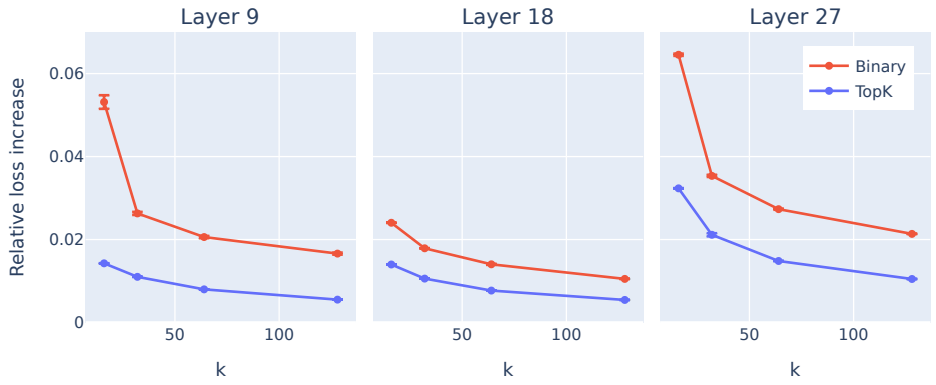

Figure 1: Binary vs. continuous next token prediction cross-entropy loss increase over various values of k. Each model are trained on 20 billion tokens.

2 Related work

Tamkin et al. [27] finetune neural network layers to add sparse binary bottlenecks with a similar structure to sparse autoencoders, and finds that the resulting features are highly interpretable and can be used to steer the model. Unlike sparse autoencoders, codebooks are components of the base model, use features' cosine similarities with the input rather than their dot products to select the top k, tie encoder and decoder weights, and do not use bias terms.

Ayonrinde et al. [3] quantise SAE activations into discrete bins after training for the purpose of encoding the data in the most efficient way possible. Instead of binarising they perform 8-bit quantisation.

Gallifant et al. [13] use an SAE feature binarisation step in their inference-time sequence classification pipeline; they set non-top-n feature activations to zero at each token, sum the feature vectors, then binarise each component of the result with a fixed threshold of 1.

3 Preliminaries

Sparse autoencoders consist of two parts: an encoder that transforms activation vectors into a sparse, higher-dimensional latent space, and a decoder that projects the latents back into the original space. Different activation functions can be used in the encoder, but in this work we will focus on the TopK function introduced by Gao et al. [14] due to its simplicity and efficacy. It zeros out all activations which are not in the top k highest activations on a given token, thereby directly enforcing a given level of sparsity. The functional form of a TopK SAE is thus

$$\hat{\boldsymbol{x}} = \mathbf{W}_{\text{dec}}\text{TopK}(\mathbf{W}_{\text{enc}}\boldsymbol{x} + \mathbf{b}_{\text{enc}}) + \mathbf{b}_{\text{dec}} \tag{1}$$

All parameters are trained jointly to minimise the reconstruction error $||\boldsymbol{x} - \hat{\boldsymbol{x}}||_2^2$. It is standard to initialise $\mathbf{W}_{\text{dec}}$ and $\mathbf{W}_{\text{enc}}$ to be transposes of each other, except that the rows of $\mathbf{W}_{\text{dec}}$ are constrained to be unit-norm [6]. We also initialise $\mathbf{b}_{\text{dec}}$ to the empirical mean or geometric median of the MLP outputs. This allows the fraction of variance explained to start out significantly higher than zero.

Later, Dunefsky et al. [11] proposed *transcoders* as an alternative method for extracting interpretable features from transformers. The architecture of a transcoder is identical to that of an SAE, but its target output is different—given the input of a feedforward component such as an MLP or attention block, a transcoder is trained to predict the component's output. For transcoders we do not constrain $\mathbf{W}_{\text{dec}}$ to have unit-norm rows, and because we are no longer trying to approximate an identity function $\mathbf{W}_{\text{dec}}$ is zero-initialised rather than being initialised with the transpose of $\mathbf{W}_{\text{enc}}$. We also initialize $\mathbf{b}_{\text{enc}}$ to the inverse of the empirical mean of the encoder $-\mathbf{W}_{\text{enc}}\mathbb{E}[\boldsymbol{x}]$ to ensure preactivations are centered at initialization.. The transcoder starts out training as a constant function with fraction of variance explained equal to zero.

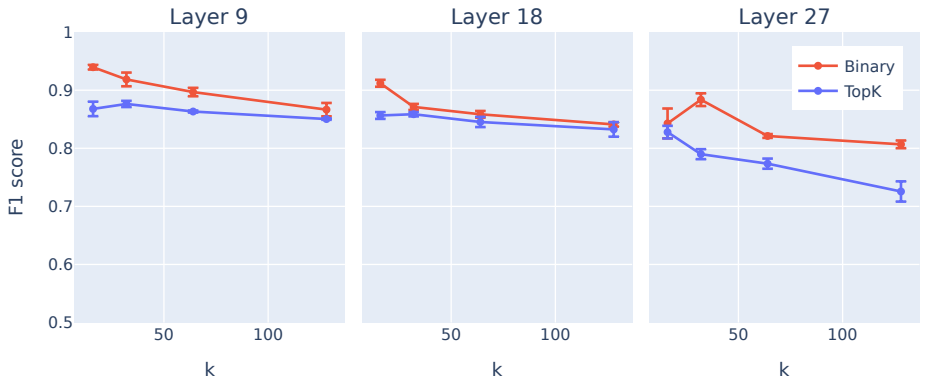

Figure 2: Unweighted fuzzing interpretability scores for binary vs. continuous skip-transcoders trained on SmolLM2-135M, with various values of k. By this metric, binary coders match or outperform continuous ones across all layers and values of k.

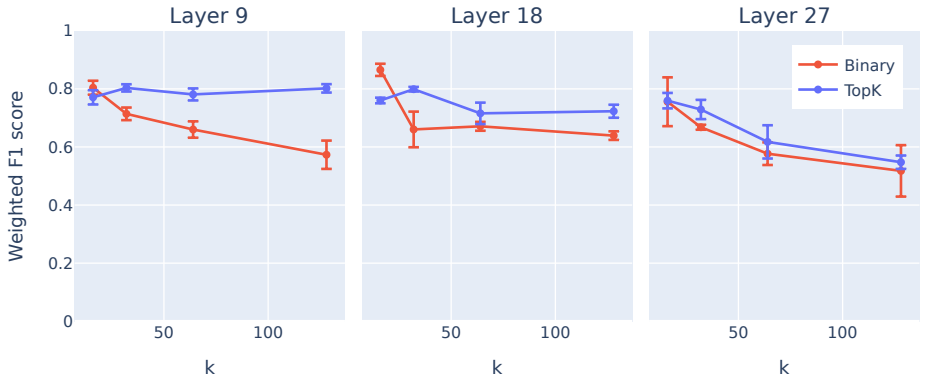

Figure 3: Frequency-weighted fuzzing interpretability scores for binary vs. continuous skip-transcoders trained on SmolLM2-135M, with various values of k. By this metric, binary coders are often less interpretable than continuous ones.

There exist transcoder variants that introduce additional parameters for the sake of increasing reconstruction accuracy. [1] adds decoder weights that connect residual streams at different layers, and Paulo et al. [24] shows that the simple modification of allowing a linear skip connection between the inputs and outputs of a transcoder improves reconstruction performance at no cost to interpretability scores. The **skip-transcoder** takes the functional form

$$\hat{\boldsymbol{y}} = \mathbf{W}_{\text{dec}}\text{TopK}(\mathbf{W}_{\text{enc}}\boldsymbol{x} + \mathbf{b}_{\text{enc}}) + \mathbf{W}_{\text{skip}}\boldsymbol{x} + \mathbf{b}_{\text{dec}} \tag{2}$$

where $\boldsymbol{x}$ is the input of the MLP and $\hat{y}$ is the reconstructed output of the MLP. The $\mathbf{W}_{\text{skip}}$ parameter is zero-initialised, so that skip-transcoders start out as constant functions just like normal transcoders.

Importantly, the work found that transcoders and skip-transcoders yield features with higher interpretability scores on average than SAEs. While transcoders are sometimes worse than SAEs in terms of reconstruction loss, the skip connection fixes this issue. For these reasons, we emphasise skip-transcoders in this work. Where our discussion refers to multiple model types, we will use the general term *sparse coder*.

4 Methods

4.1 Binary sparse coders

Binary sparse coders are a simple modification of standard sparse coders. We simply add a binarisation step after the TopK function is applied, forcing the activations to binary as well as sparse. The functional form for a binary SAE is thus

$$\hat{\boldsymbol{y}} = \mathbf{W}_{\text{dec}}\text{Binarise}[\text{TopK}(\mathbf{W}_{\text{enc}}\boldsymbol{x} + \mathbf{b}_{\text{enc}})] + \mathbf{b}_{\text{dec}} \tag{3}$$

where $\text{Binarise} : \mathbb{R}^N \to \mathbb{R}^N$ replaces each element of a vector with 1 if it is positive, otherwise replacing it with 0. The sigmoid function can be viewed as a continuous and differentiable relaxation of Binarise, and hence we use a sigmoid-based straight-through estimator [4] to backpropragate through it. We found that adding a temperature parameter of 2 to the sigmoid STE improved training stability.

4.2 Gumbel-Softmax and GroupMax

Reconstruction accuracy can be further improved by training with the Gumbel-Softmax gradient estimator [18], which enables differentiable approximate categorical sampling of pre-activations through a relaxation of the Gumbel-Max trick. We found that this method can create training instability when backpropagating through the top-k operator, so for the Gumbel variant we replaced the TopK with a novel activation function we call *GroupMax*, that groups latents into k groups and selects the largest element in each group. These *GroupMax* sparse coders reliably trained with Gumbel-Softmax without instability. The functional form for a binary GroupMax SAE is

$$\hat{\boldsymbol{y}} = \mathbf{W}_{\text{dec}}\text{Binarise}[\text{GroupMax}(\mathbf{W}_{\text{enc}}\boldsymbol{x} + \mathbf{b}_{\text{enc}})] + \mathbf{b}_{\text{dec}} \tag{4}$$

where $\text{GroupMax} : \mathbb{R}^N \to \mathbb{R}^N$ divides a vector into k contiguous and equally sized groups, then zeros out any activation that is not the largest of a group.

4.3 Training

Unless stated otherwise, all experiments train sparse coders on a total of 10 billion tokens with a batch size of 2^{20} tokens. The training data is drawn from the base model's original corpus. For SmolLM2-1.7B, which was trained with curriculum learning, we specifically use the final-stage dataset mixture from its pre-training curriculum.

We trained skip-transcoders, binary skip-transcoders, and the Gumbel GroupMax skip-transcoder variant at $k = 32$ for both SmolLM2-135M and SmolLM2-1.7B.

We also trained skip-transcoders, binary skip-transcoders, SAEs, and BAEs over a variety of k values for the SmolLM2-135M base model. We trained three seeds for each sparse coder and report mean values for our evaluation metrics; error bars denote the ± 1 standard error of the mean.

We additionally trained three seeds of the Gumbel GroupMax variant at $k = 32$, and finetuned it along with binary and continuous skip-transcoders of equal k using a KL divergence loss for use in sparse probing.

We find that fine-tuning transcoders with a KL divergence loss as proposed by Karvonen [19] can improve performance when the transcoder is patched into the base model.

We use the Adam optimiser [21] to train most sparse coders on SmolLM2-135M models, and the schedule-free Signum optimiser [5, 8] to train the SmolLM2-1.7B coders and to train binary SAEs, which exhibit near-complete index collapse when trained with Adam. For Adam we used a linear learning rate schedule with a peak value of 3×10^{-3} for transcoders, 5×10^{-3} for SAEs, 1000 warmup steps, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and no weight decay. For schedule-free Signum we used a constant learning rate of 3×10^{-3} and a momentum of 0.95. Note that in schedule-free training the momentum term is equivalent to the weight EMA employed by Gao et al. [14].

4.4 Evaluation

Our primary performance metric is the increase in next-token cross-entropy loss induced by patching the sparse coder into its base model. We also provide the trajectory of the fraction of variance unexplained (FVU) training objective (Appendix A.4.)

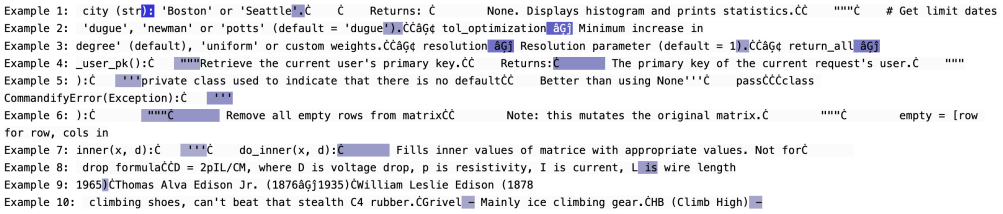

Figure 4: Activating examples from different deciles of a TopK skip-transcoder feature. The top activations appear in a coding context, while the lower activations appear in diverse contexts.

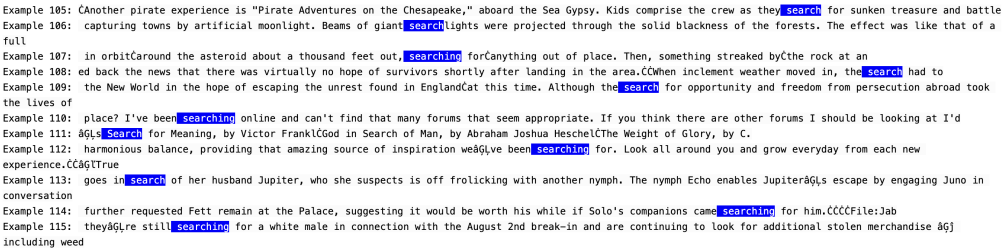

Figure 5: Activating examples for a binary skip-transcoder feature that achieved perfect auto-interpretability scores. The generated explanation is "The verb 'search' or its variants, often used in contexts of looking for something, someone, or information, and frequently associated with a sense of investigation, inquiry, or pursuit." Note that a counterexample to this explanation, a non-activating instance of "Search", appears mid-way through Example 111.

To evaluate the interpretability of the sparse coders, we employ the auto-interpretability techniques proposed in [23] to generate and evaluate feature explanations using LLMs. First, we collect sparse coder feature activations over 10 million tokens from the training dataset, filtering out activations associated with beginning of sentence (BOS) tokens. Next, we sample 100 features from each of four layers of the sparse coder to explain, for a total of 400 features. We generate an explanation for each feature by prompting an LLM with 40 decile-sampled labeled activating sequences of 256 tokens each, then attempt to classify balanced datasets containing 50 sequences with decile-sampled feature activations and 50 randomly sampled non-activating sequences by prompting an LLM to classify each sequence as activating or not activating in accordance with the generated feature explanation. We use Llama 3.1 for both explanation generation and sequence classification.

The results of the LLM binary classification tasks can be used to produce the mean classification accuracy, but this metric can be misleading because sparse coder features tend to exhibit extreme class imbalance where they are inactive far more than they are active. To account for this imbalance we report the mean F_1 score instead. The F_1 score for a feature is the harmonic mean of the precision and recall, which simplifies to

$$F_1 = \frac{2 \cdot \text{true positives}}{2 \cdot \text{true positives} + \text{false positives} + \text{false negatives}} \tag{5}$$

As proposed in [23], we calculate scores for both detection and fuzzing classifiers, which perform slightly different classification tasks. For the detection task the classification task is to determine whether or not a sequence contains a feature activation, and for fuzzing the classification task is to determine whether a highlighted token in a sequence is associated with a feature activation.

We also provide the frequency-adjusted F1 scores for each classifier, since features vary in their firing frequencies across many orders of magnitude.

In addition to the aforementioned metrics, we integrate the sparse probing evaluation from [20]. This evaluation tests representation of concepts in the SAE by training sparse classifiers of samples containing those concepts (Table 1.)

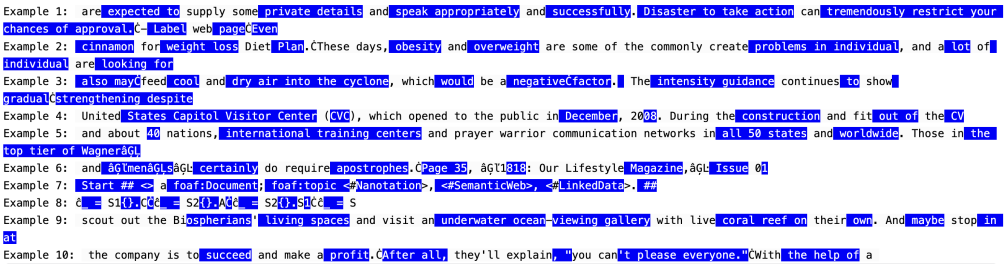

Figure 6: Activating examples from an ultra high frequency binary skip transcoder feature. The activations occurs on more than half of tokens and are not interpretable.

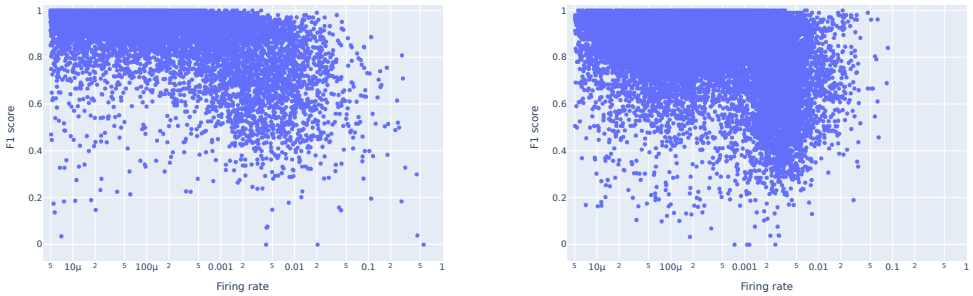

Figure 7: Feature firing rates for binary (left) and continuous (right) skip-transcoders on layer 9 of SmolLM2-135M, calculated over 10 million tokens. Firing rates are plotted on a log scale. The binary transcoder has ultra-high frequency features than the continuous transcoder, with the most frequent firing on over half of tokens. No continuous feature activates on more than 10% of the dataset.

5 Results

We find that binarisation improves unweighted average feature interpretability (Figure 2), but increases the cross-entropy loss when patched into the base model (Figure 1). Unfortunately, we find that binarisation does not increase interpretability when features are weighted by their firing frequencies. We find that binary transcoders indeed have more ultra-high frequency features than continuous ones do, and that these tend to have fairly low interpretability scores (Figure 7). We examine a concrete example of this phenomenon in Figure 6. We hypothesize that the sparse coder effectively learns to push the information that would usually be stored in low activation deciles into high-frequency, uninterpretable features. If this is true, it would suggest that polysemanticity is to a certain extent ineliminable: try to remove it, and it gets pushed somewhere else.

Binarised models also underperformed continuous ones in the sparse probing evaluation, however after finetuning both models the opposite is true; both binary skip-transcoders and their Gumbel variant outperformed finetuned continuous skip-transcoders (Table 1.)

We also explored the relationship between the interpretability score and the causal effect of a feature on the model's performance. To do this, we grouped features into bins based on their fuzzing score, taking care to keep the total fire count roughly constant within each bin, and then ablated each bin of features and measured the resulting increase in cross-entropy loss. We found that almost all bins had nearly the same effect on the loss, except for two very high interpretability bins (Figure 8). This is interesting and suggests that, at the extremes, there is a positive relationship between interpretability and causal influence. Across most bins, however, there is no relation between interpretability and causal importance.

5.1 Case study: a polysemantic code feature in SmolLM2-135M

This feature (Figure 4) seems to fire inside doc strings in Python code in the top seven deciles of its activation range, although it can fire on a few different kinds of tokens within that context. The autointerp explanation is '"Special characters and symbols used for various purposes such as denoting comments, indicating code blocks, separating parameters, and representing mathematical

Layer	Sparse probing (↑)				
	SST	BSST	GSST	FSST	BFSST
L0	69.4 ± 1.7	64.5 ± 1.7	$\mathbf{69.5 \pm 1.0}$	69.1 ± 2.1	64.0 ± 0.5
L9	60.3 ± 0.4	66.9 ± 3.5	67.6 ± 0.2	60.4 ± 0.7	$\mathbf{67.8 \pm 1.5}$
L18	64.7 ± 2.1	68.2 ± 1.0	68.7 ± 1.2	66.4 ± 0.5	$\mathbf{68.9 \pm 0.9}$
L27	74.7 ± 0.6	76.4 ± 0.8	76.8 ± 0.2	74.5 ± 0.5	$\mathbf{77.3 \pm 0.4}$

Table 1: Sparse probing evaluation from [20]. Results for SmolLM2-135M for sparse skip-transcoders, binary sparse skip-transcoders, Gumbel binary sparse skip-transcoders, and finetuned variants of binary sparse skip-transcoders and sparse skip-transcoders. Binary sparse skip-transcoders are surprisingly useful in this setting, with both finetuned binary sparse coder variants scoring more highly than finetuned continuous sparse coders. Each sparse coder is evaluated over 3 seeds.

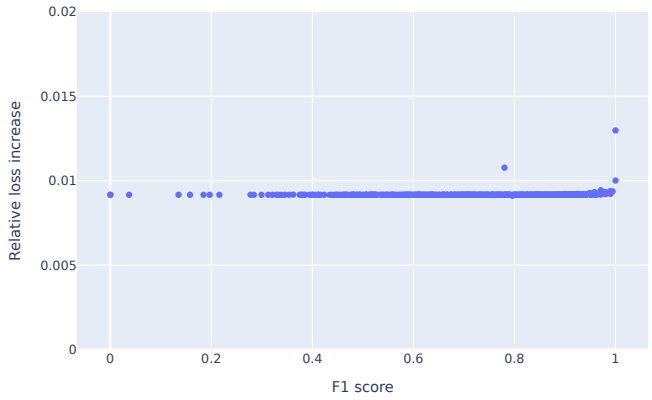

Figure 8: The effect on loss of ablating groups of features across the interpretability spectrum. Every feature in a binary transcoder for layer 9 of a SmolLM2-135M is ordered by its fuzzing F1 score and assigned a bin such that each bin has an approximately equal firing count. Mean losses from zero ablating each bin are then taken over 1024-token sequences ($N = 1024$.)

operations."‘. In the lower deciles, it fires in a wider range of non-code contexts. We would like to eliminate the polysemanticity we see in the lower deciles.

5.2 Case study: the $search$ feature in SmolLM2-135M

A non-cherrypicked feature was selected from the set of binary skip-transcoder features with perfect auto-interpretability fuzzing and detection scores (Fig. 5). The feature clearly activates on words related to searching ($search, searching, looking, quest, searched$) but close inspection of the sequences containing activating examples reveals several non-activating instances of these words, both preceding and following the activating instances. This shows that our interpretability scores can over-estimate the true interpretability of features. This is partially because we use random non-activating examples when generating and evaluating explanations, rather than non-activating examples adversarially selected to be semantically similar to the activating ones. If the scoring pipeline had used adversarial selection of negative examples, it might have detected instances of the "search" concept where this feature does not fire.

6 Discussion and future work

While binary sparse coders produce more interpretable features, this effect is cancelled out by their tendency to produce uninterpretable ultra-high frequency features. They also explain less of the variance in model activations than continuous sparse coders, and they incur a larger increase in loss when patched into the model. Overall, we cannot recommend binary sparse coders for most purposes. The primary result of this work is negative: we attempted to remove a source of polysemanticity in sparse coding techniques (low activation values), and it re-appeared in a new form somewhere else

(high-frequency features). This provides support for the idea that polysemanticity is an ineliminable feature of neural activations.

If polysemanticity turns out to be genuinely ineliminable, as we suspect but cannot prove, this would mean that the attempt to *fully* decompose neural activations into interpretable features is fundamentally misguided. It would mean the mind is irreducibly holistic, rather than being built from logical atoms, as philosophers like Hubert Dreyfus have argued for decades [9, 10].

More positively, future attempts to optimize the sparse coder architecture might include some mixture of binary and continuous features, since some concepts are best expressed as continuous values, and others discrete ones. While we could fix the proportion of binary features at the beginning of training as a hyperparameter, it may be best to dynamically determine the appropriate mixture in a data-dependent way. This might ameliorate the increase in fraction of variance unexplained we observe with binary sparse coders, and the problem of high-frequency features. Exploring mixed architectures is an interesting direction for future work. Notably, Smith et al. [26] investigated auxiliary loss terms that explicitly discourage the formation of high-frequency features, although they ultimately did not find the direction particularly promising.

Finally, the promising performance of binary sparse coders on sparse probing tasks could be investigated further.

7 Limitations

Our focus on a single dataset may limit the applicability of this work—all sparse coders were trained on a sample of the SmolLM2 training corpus. This is a high-quality dataset, but using a different dataset could change the results. Additionally, some of our sparse coders exhibited high numbers of dead neurons (Figure 19). If this issue were resolved it might also change our results.

Our results may also change at higher sparse coder widths. Sparse coder FVU follows a scaling law, where increasing the number of features increases the fraction of variance explained. But as the width tends to infinity, the sparse coder converges to a degenerate solution where each individual input-output pair in the dataset gets its own feature. The optimal width for a sparse coder should make a tradeoff between reconstruction accuracy and learning features that are sufficiently coarse-grained to be interesting. The reduction of expressivity caused by binarising sparse coder activations could result in an optimal width that is too large to train with a reasonable amount of compute.

References

[1] Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 2025. URL `https://transformer-circuits.pub/2025/attribution-graphs/methods.html`.

[2] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics*, 6:483–495, 2018.

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

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[5] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pages 560–569. PMLR, 2018.

[6] 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, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[7] Bart Bussmann, Patrick Leask, and Neel Nanda. Batchtopk sparse autoencoders. *arXiv preprint 2412.06410*, 2024. URL `https://arxiv.org/abs/2412.06410`.

[8] Aaron Defazio, Xingyu Yang, Ahmed Khaled, Konstantin Mishchenko, Harsh Mehta, and Ashok Cutkosky. The road less scheduled. *Advances in Neural Information Processing Systems*, 37:9974–10007, 2024.

[9] Hubert Dreyfus. What computers can't do. *British Journal for the Philosophy of Science*, 27(2), 1976.

[10] Hubert L Dreyfus. *What computers still can't do: A critique of artificial reason*. MIT press, 1992.

[11] Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable llm feature circuits. *arXiv preprint arXiv:2406.11944*, 2024.

[12] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

[13] Jack Gallifant, Shan Chen, Kuleen Sasse, Hugo Aerts, Thomas Hartvigsen, and Danielle S. Bitterman. Sparse autoencoder features for classifications and transferability, 2025. URL `https://arxiv.org/abs/2502.11367`.

[14] 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.

[15] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[16] Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *arXiv preprint arXiv:2305.01610*, 2023.

[17] Wes Gurnee, Theo Horsley, Zifan Carl Guo, Tara Rezaei Kheirkhah, Qinyi Sun, Will Hathaway, Neel Nanda, and Dimitris Bertsimas. Universal neurons in gpt2 language models. *arXiv preprint arXiv:2401.12181*, 2024.

[18] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=rkE3y85ee`.

[19] Adam Karvonen. Revisiting end-to-end sparse autoencoder training: A short finetune is all you need, 2025. URL `https://arxiv.org/abs/2503.17272`.

[20] Adam Karvonen, Can Rager, Johnny Lin, Curt Tigges, Joseph Bloom, David Chanin, Yeu-Tong Lau, Eoin Farrell, Arthur Conmy, Callum McDougall, Kola Ayonrinde, Matthew Wearden, Samuel Marks, and Neel Nanda. Saebench: A comprehensive benchmark for sparse autoencoders, 2024. URL `https://www.neuronpedia.org/sae-bench/info`. Accessed: 2025-01-17.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL `https://arxiv.org/abs/1412.6980`.

[22] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3), March 2020. ISSN 2476-0757. doi: 10.23915/distill.00024.001. URL http://dx.doi.org/10.23915/distill.00024.001.

[23] Gonçalo Paulo, Alex Mallen, Caden Juang, and Nora Belrose. Automatically interpreting millions of features in large language models, 2024. URL https://arxiv.org/abs/2410.13928.

[24] Gonçalo Paulo, Stepan Shabalin, and Nora Belrose. Transcoders beat sparse autoencoders for interpretability, 2025. URL https://arxiv.org/abs/2501.18823.

[25] 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.

[26] Lewis Smith, Sen Rajamanoharan, Arthur Conmy, Callum McDougall, Janos Kramar, Tom Lieberum, Rohin Shah, and Neel Nanda. Negative results for sparse autoencoders on downstream tasks and deprioritising sae research (mechanistic interpretability team progress update), March 2025. URL https://deepmindsafetyresearch.medium.com/negative-results-for-sparse-autoencoders-on-downstream-tasks-and-deprioritising-sae-research

[27] Alex Tamkin, Mohammad Taufeeque, and Noah D. Goodman. Codebook features: Sparse and discrete interpretability for neural networks, 2023. URL https://arxiv.org/abs/2310.17230.

[28] 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, 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. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

A Technical Appendices and Supplementary Material

A.1 Compute

Each experiment was run on nodes equipped with either 4 or 8 GPUs. The nodes were equipped with a mixture of NVIDIA A40, A100, and H100 GPUs from various cloud providers. Individual training runs took 3 to 15 hours, full-layer binned ablations took 4 hours, and collecting individual model interpretability statistics took around 1 hour per model. The total compute estimate is 500 A100 compute hours. Additionally, preliminary experiments used an estimated 100 A100 compute hours.

A.2 Results for SmolLM2-1.7B binary and continuous skip-transcoders

Metric	SmolLM2-1.7B TC	
	Binary	Continuous
Unweighted fuzzing F1	**0.880**	0.779
Unweighted detection F1	**0.8387**	0.670
Weighted fuzzing F1	**0.801**	0.658
Weighted detection F1	**0.702**	0.670

Table 2: Results for SmolLM2-1.7B binary and continuous skip-transcoders, expansion factor of 64, $k = 32$, averaged over four layers. In this single-seed run BTCs are consistently more interpretable.

A.3 Detection F1 scores

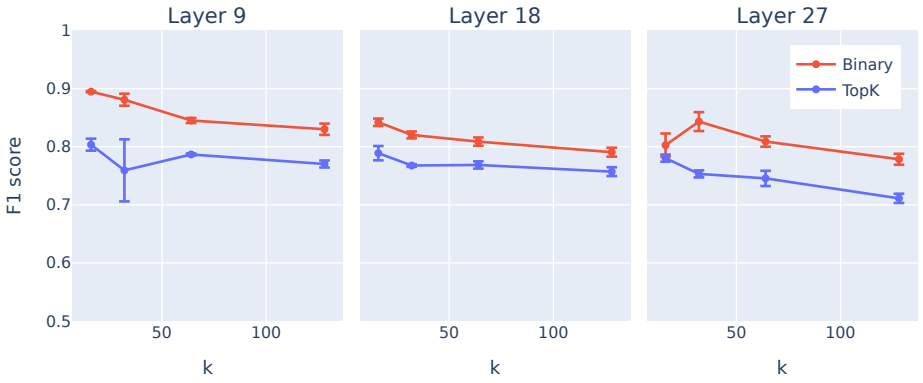

Figure 9: Unweighted detection F1 scores for SmolLM2-135M binary and continuous skip-transcoders.

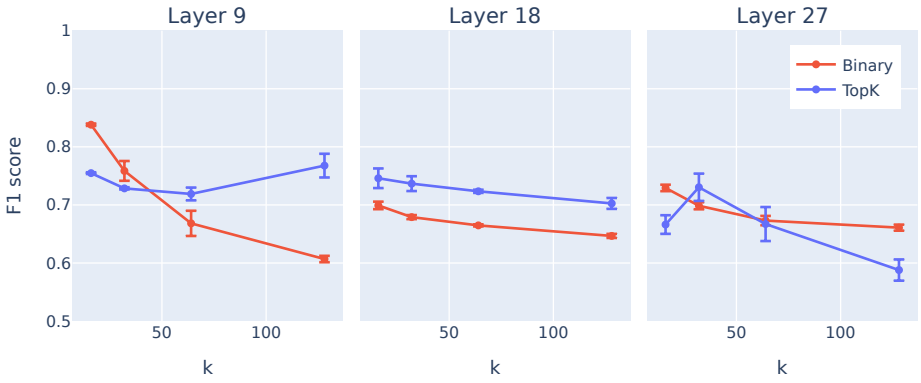

Figure 10: Unweighted detection F1 scores for SmolLM2-135M BAEs and SAEs.

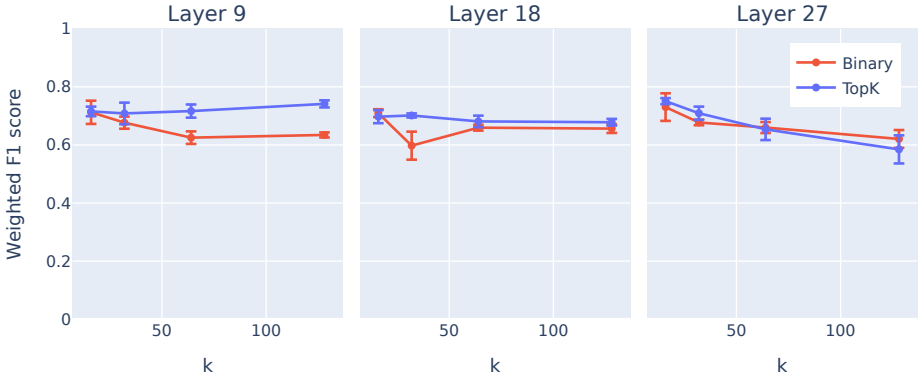

Figure 11: Frequency-weighted detection F1 scores for SmolLM2-135M binary and continuous skip-transcoders.

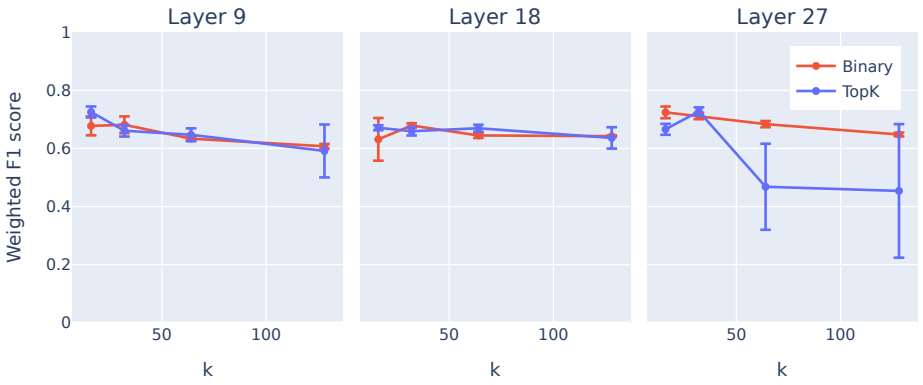

Figure 12: Frequency-weighted detection F1 scores for SmolLM2-135M BAEs and SAEs.

A.4 Fraction of variance unexplained

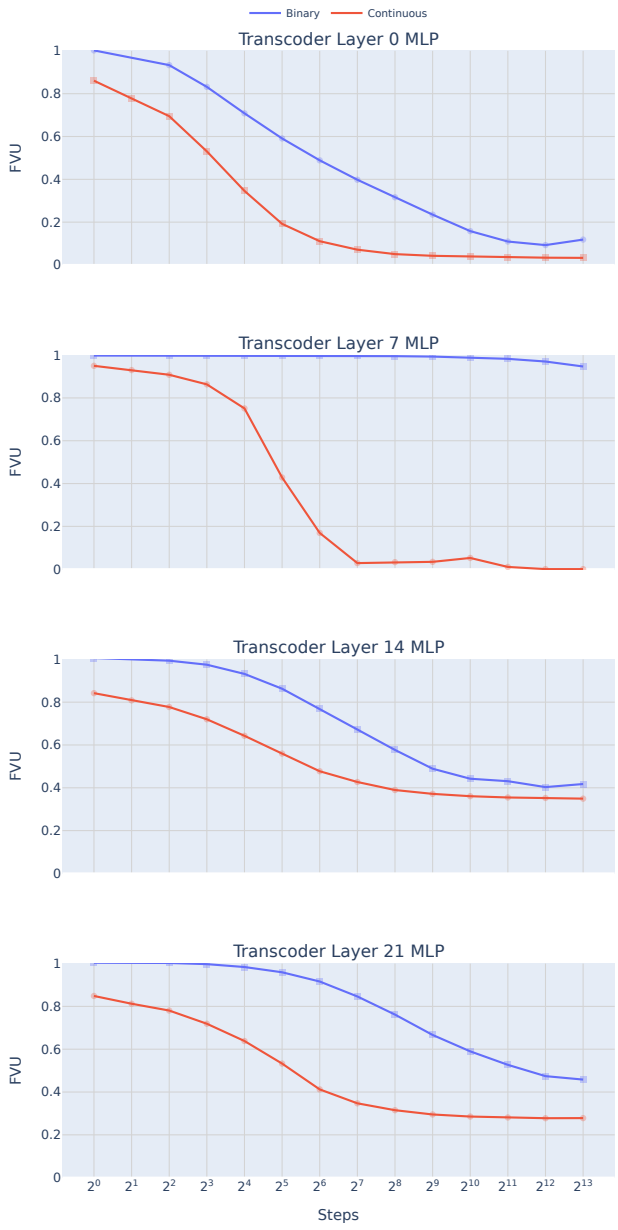

Figure 13: FVUs over training for SmolLM2-1.7B binary and continuous skip-transcoders.

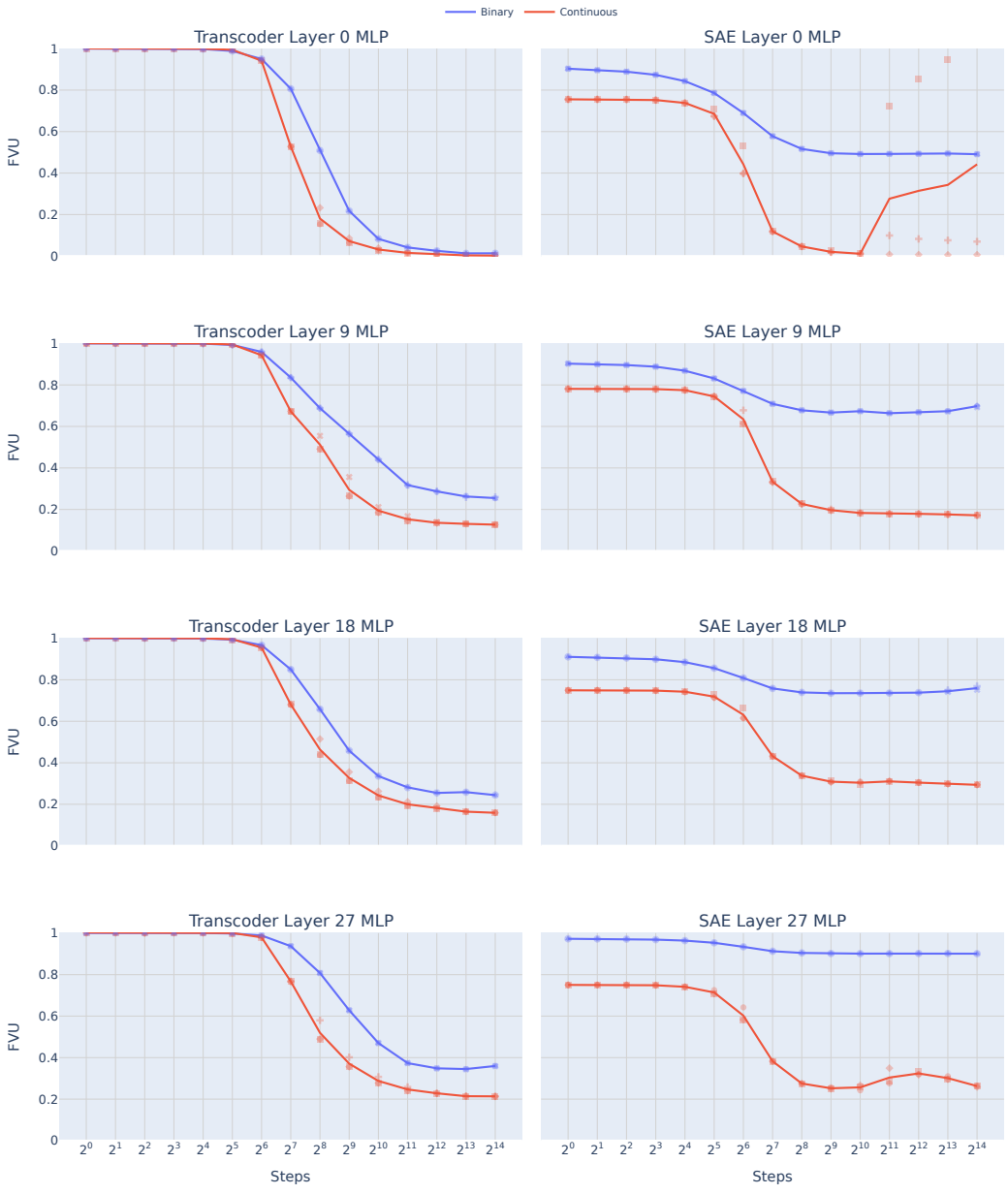

Figure 14: FVUs over training for all $k = 16$ pre-trained models.

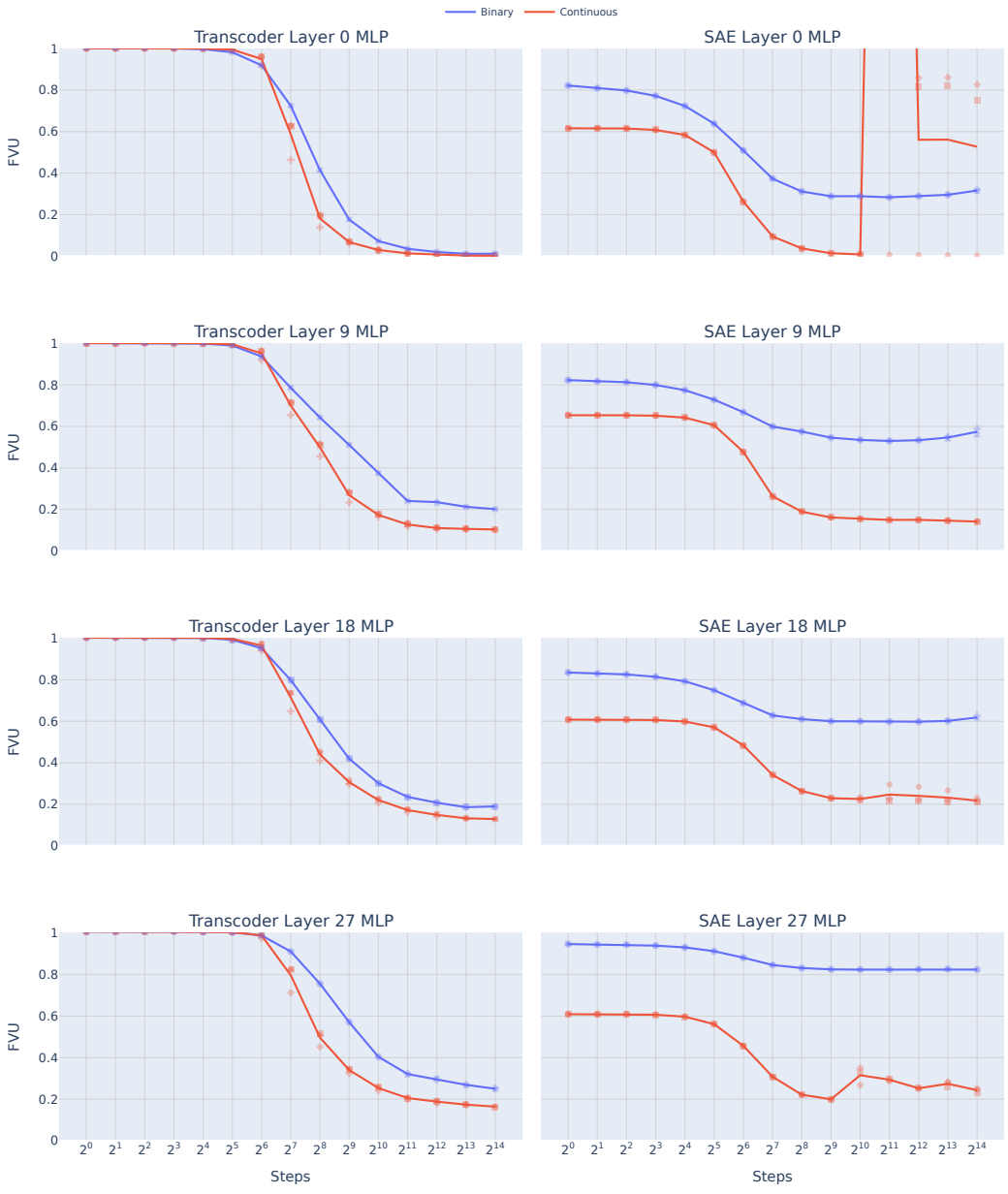

Figure 15: FVUs over training for all $k = 32$ pre-trained models.

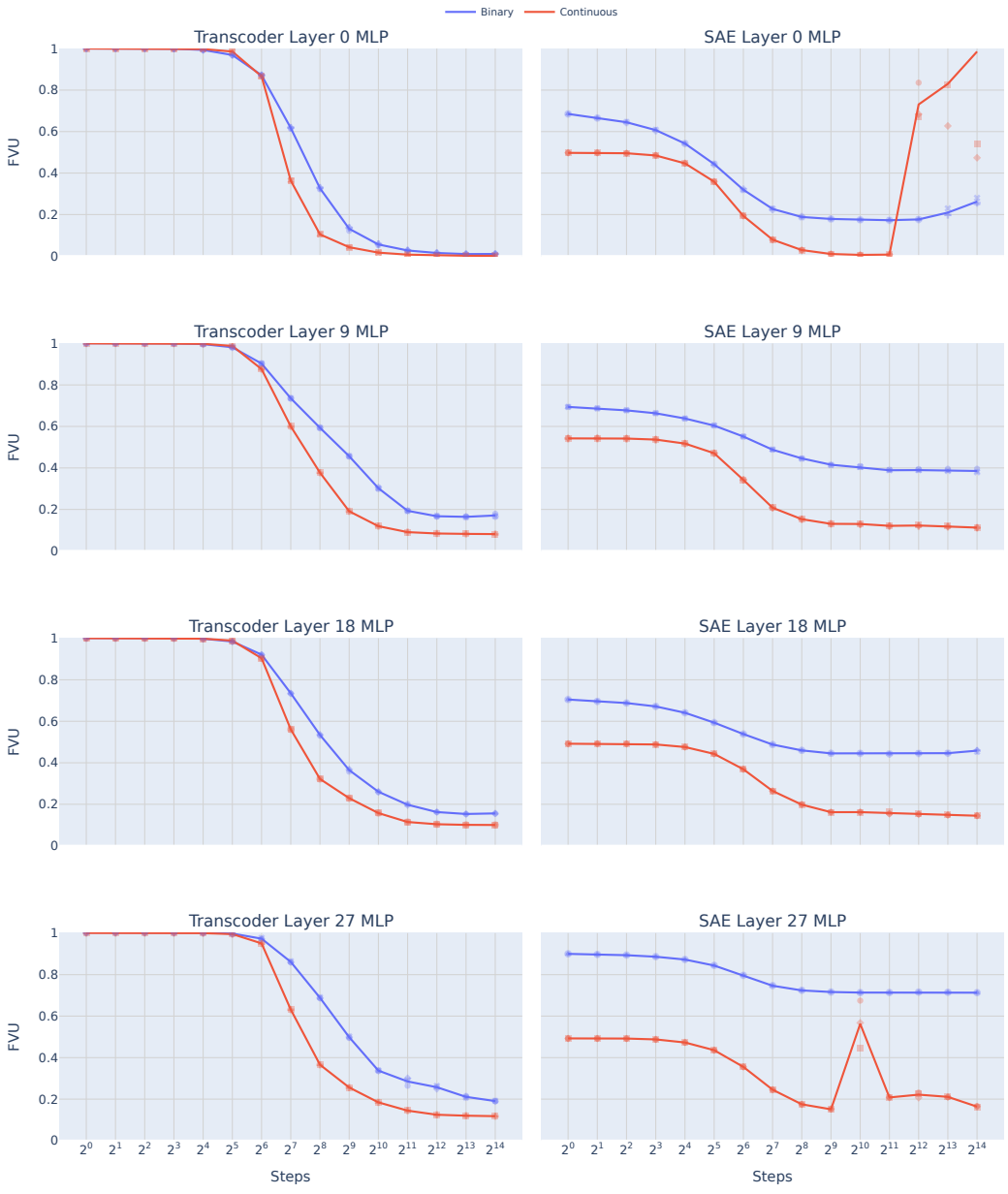

Figure 16: FVUs over training for all $k = 64$ pre-trained models.

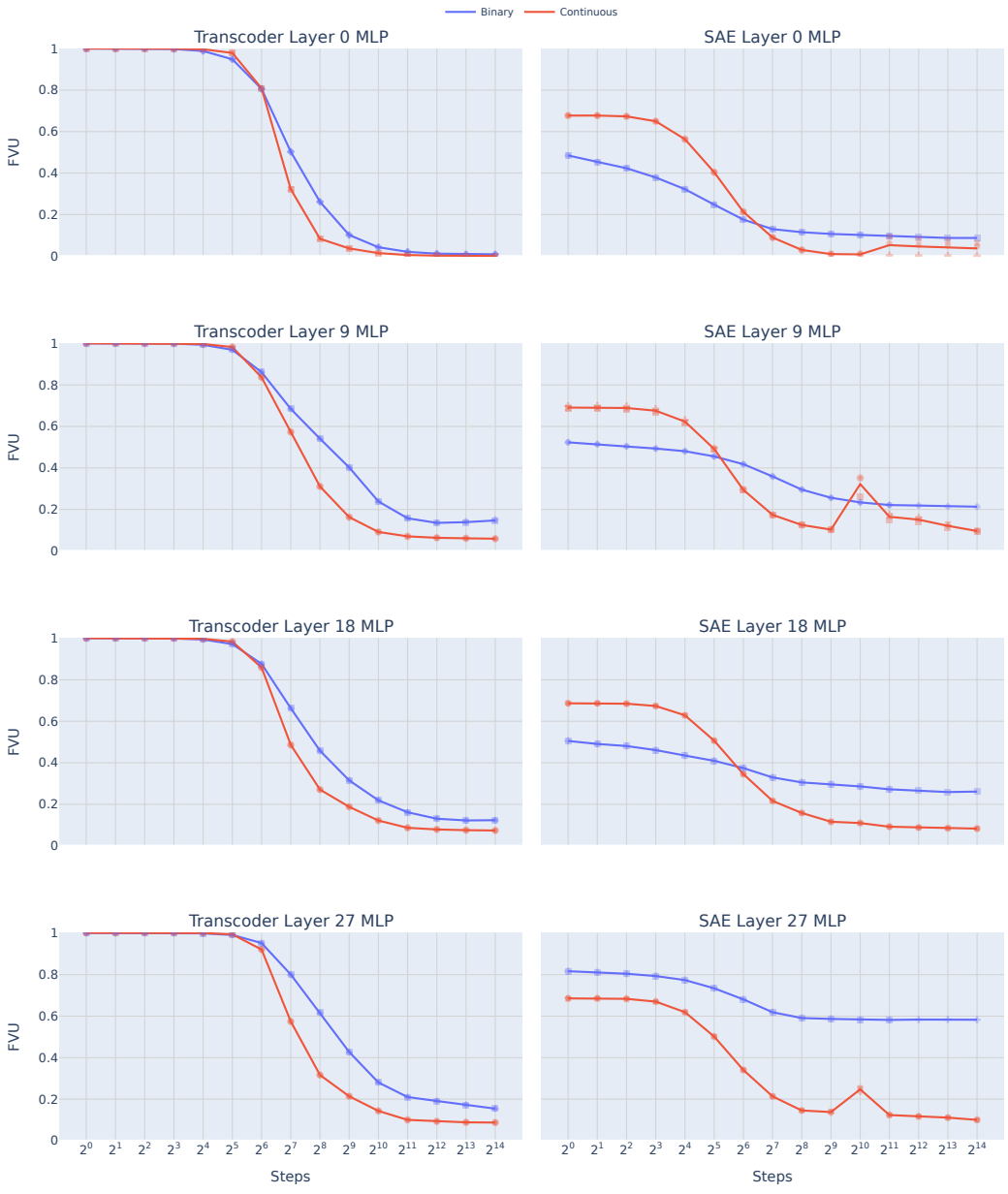

Figure 17: FVUs over training for all $k = 128$ pre-trained models.

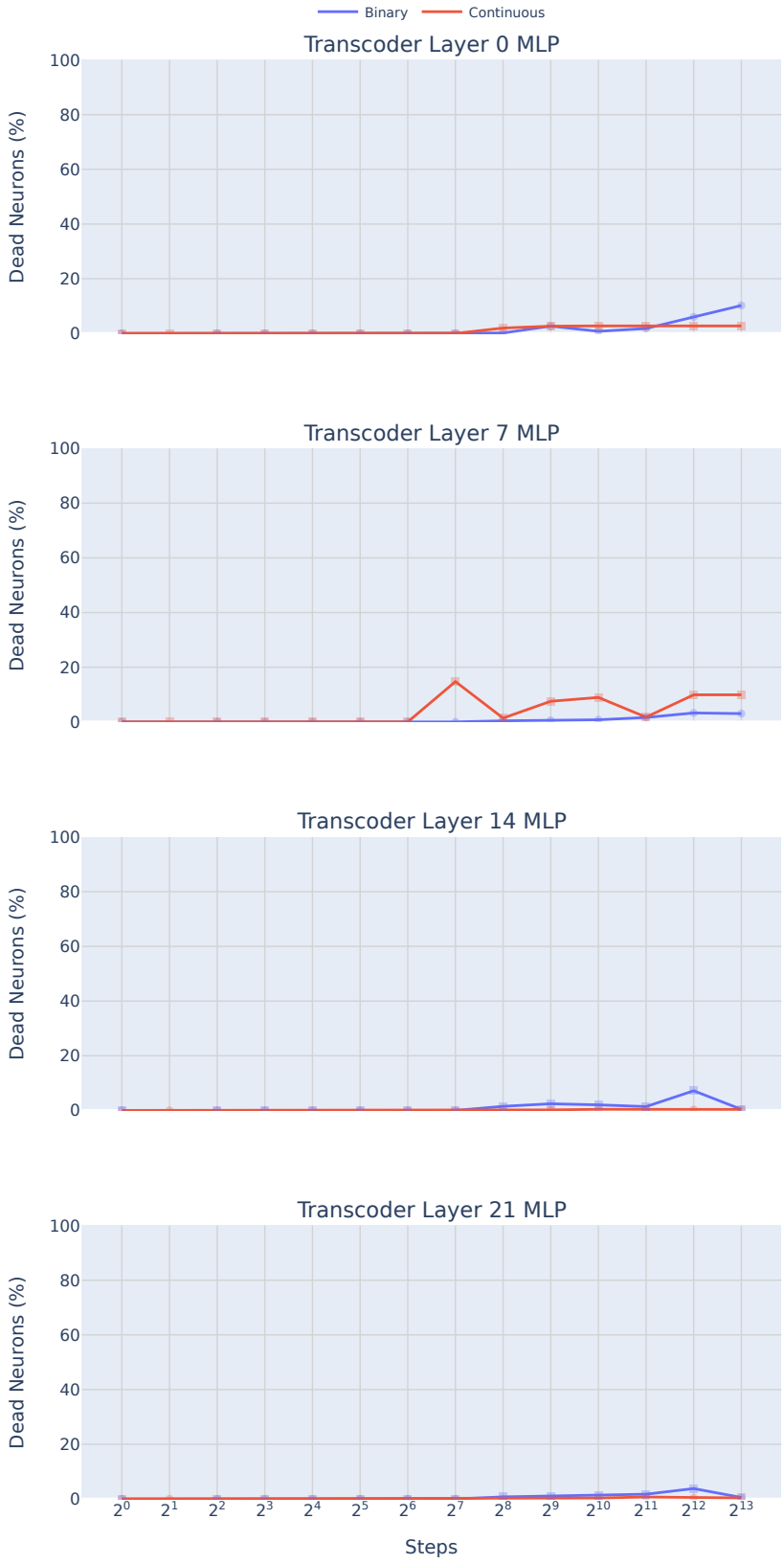

Figure 18: Dead neurons over training for SmolLM2-1.7B binary and continuous skip-transcoders.

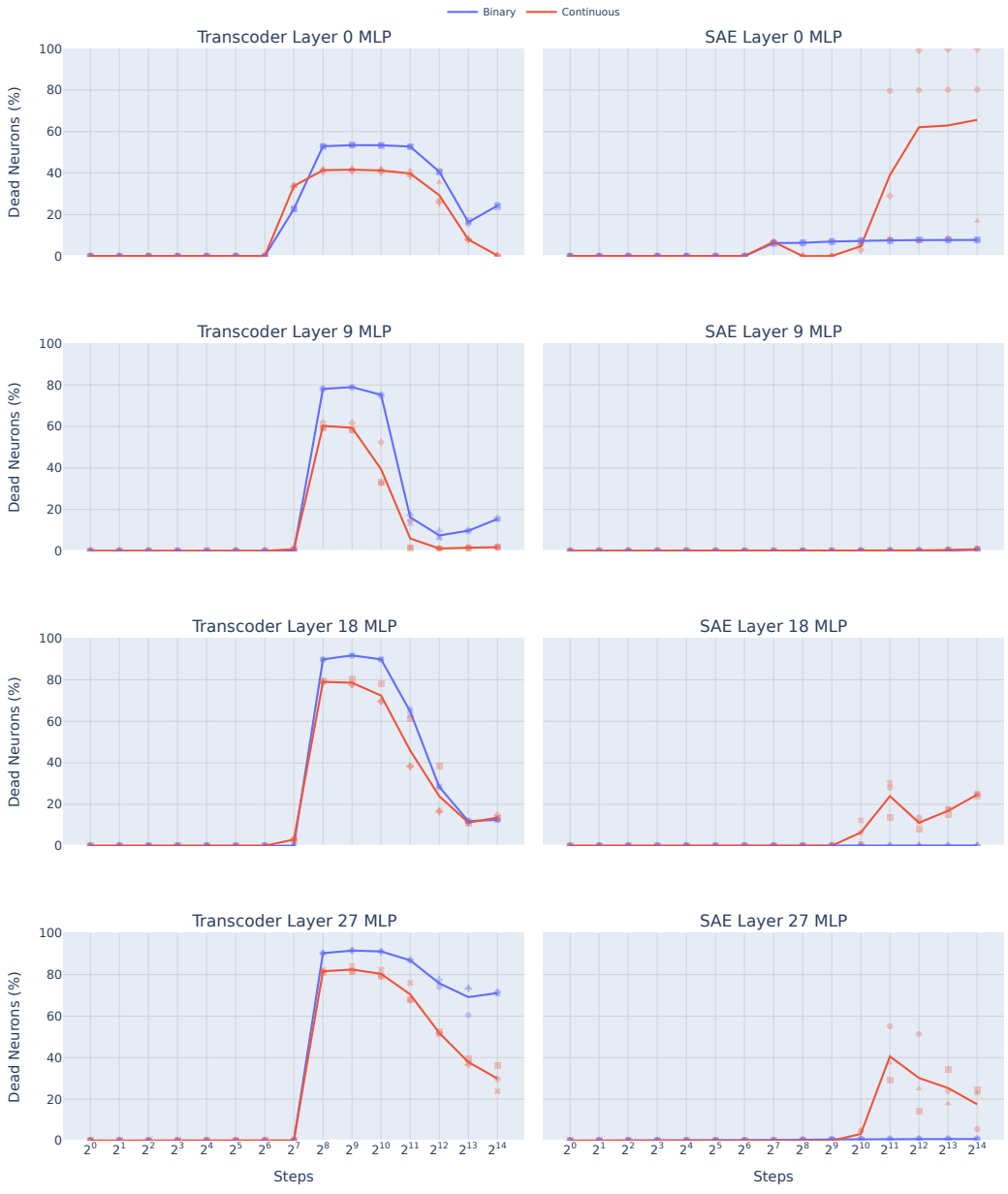

Figure 19: Dead neurons over training for all $k = 16$ pre-trained models.

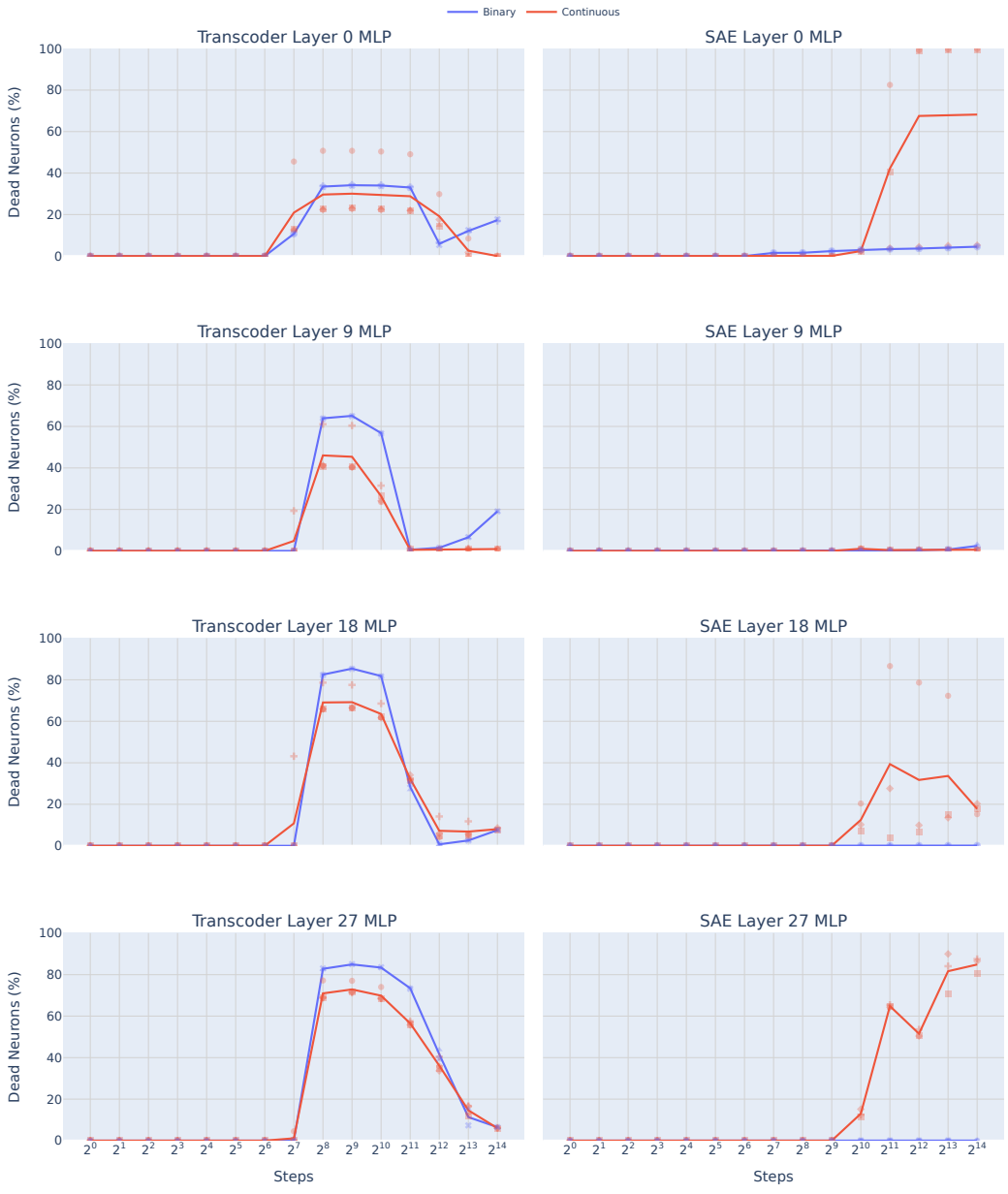

Figure 20: Dead neurons over training for all $k = 32$ pre-trained models.

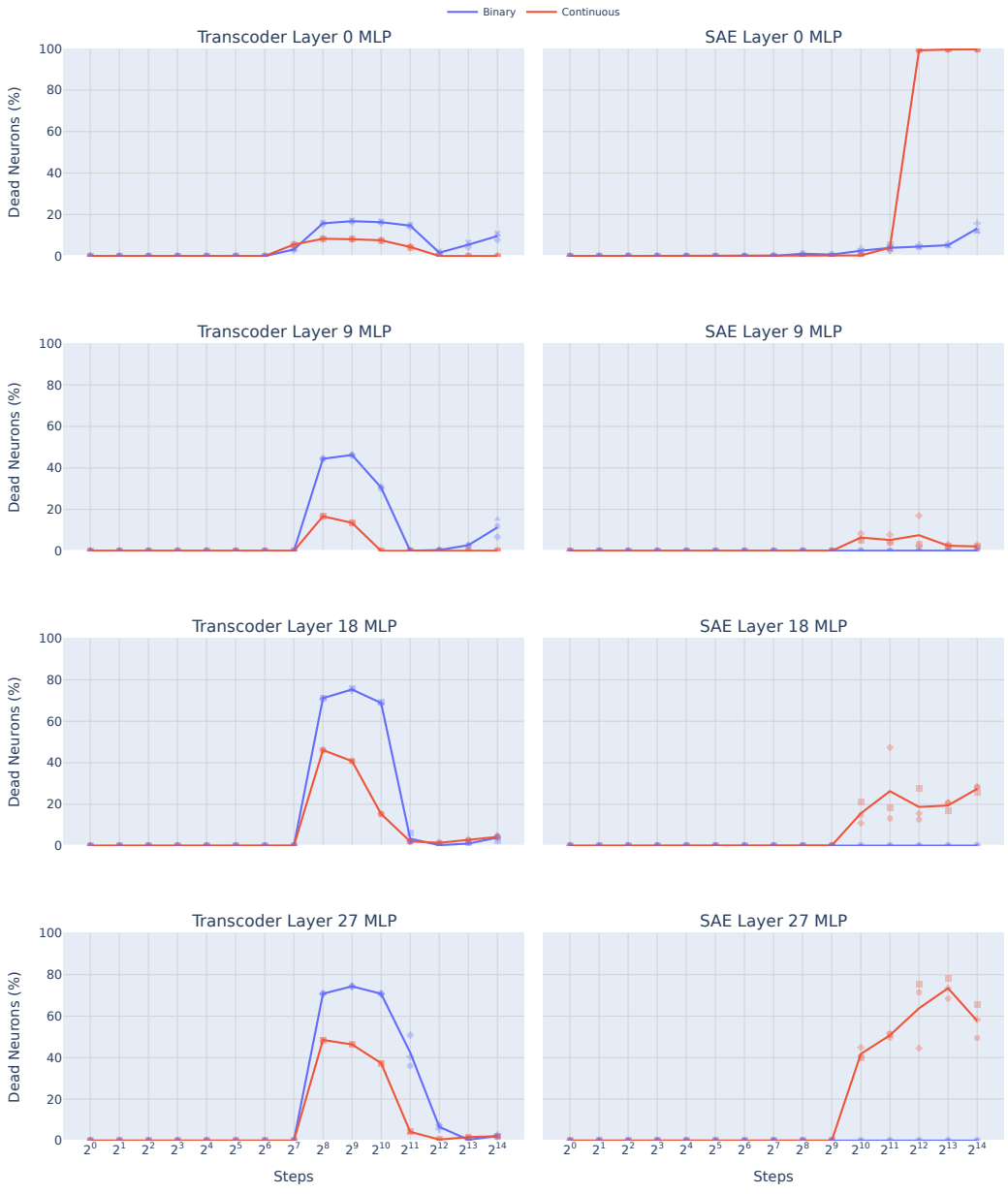

Figure 21: Dead neurons over training for all $k = 64$ pre-trained models.

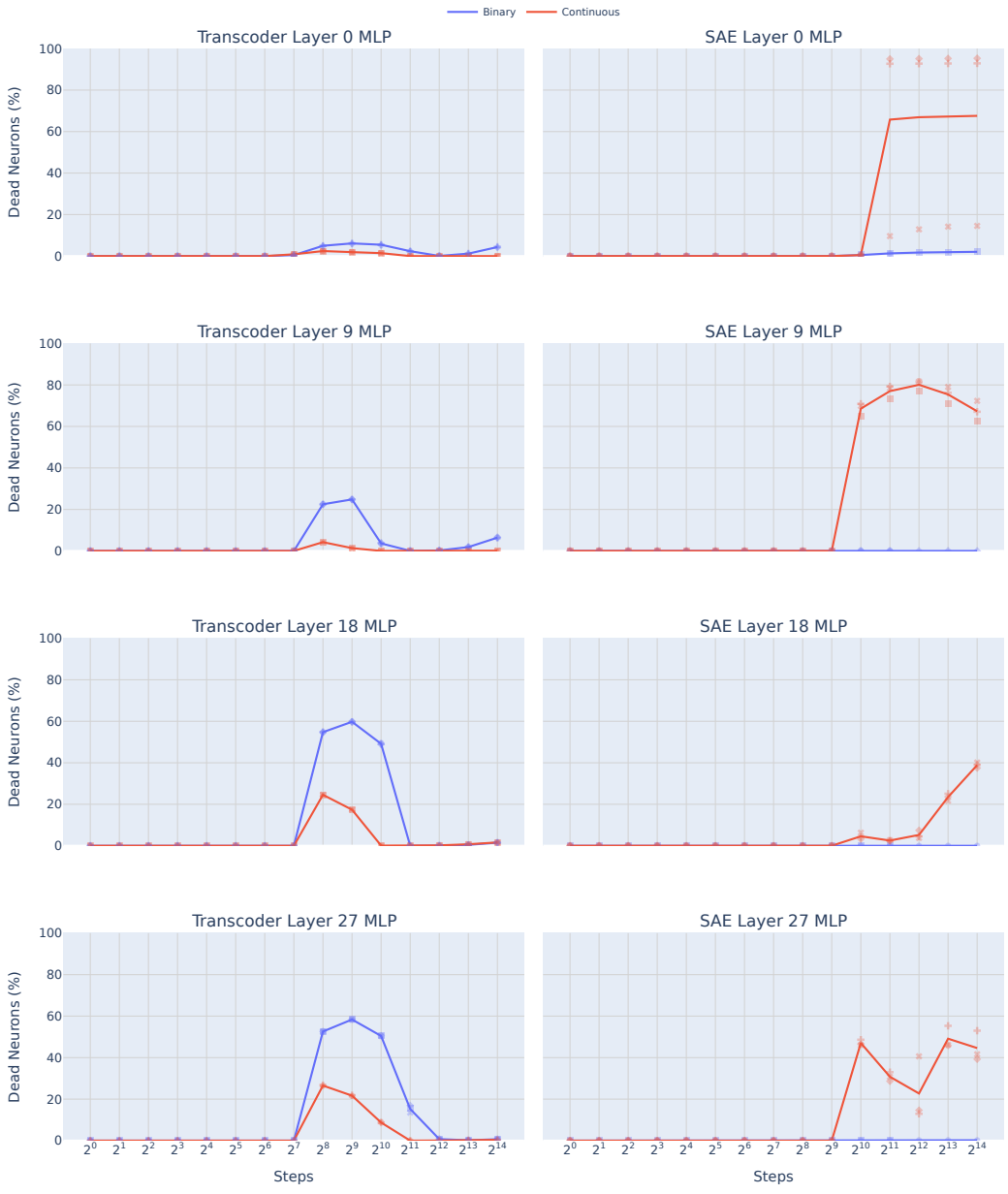

Figure 22: Dead neurons over training for all $k = 128$ pre-trained models.