SPARSE COMPOSITIONALITY IDENTIFIES FEATURES OF COMPUTATION

Anonymous authors
Paper under double-blind review

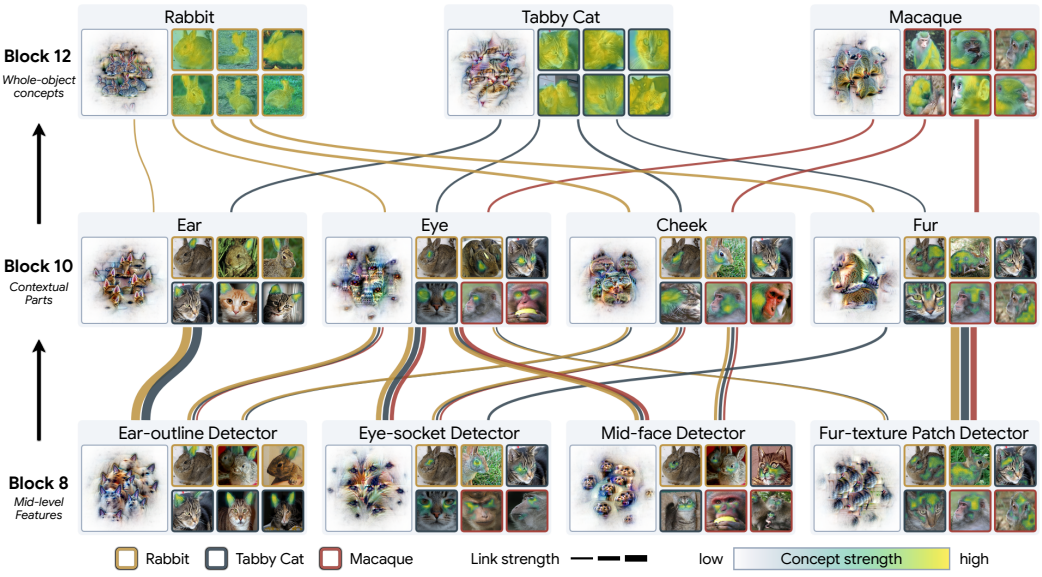

Figure 1: **Recovering compositional structure from DINOv3.** How does a model compute the presence of e.g., a rabbit in an image? Generic mid-level features for fur texture and facial regions at block 8 compose into contextually enriched part-level features at block 10, which in turn compose into whole-object level features.

ABSTRACT

Does a reconstruction objective recover the features that participate in a network's computation, or a basis that merely reconstructs its activations? The distinction matters if, as a growing body of work suggests, learners generalize through compositional sparsity. The features that explain such a network are then the intermediate functions it composes, and nothing guarantees that a featurizer trained to reconstruct will single them out. In this work, we make the distinction measurable by planting a ground-truth feature DAG, whose intermediate features and compositional relations are known, inside a transformer through per-layer supervision. Reconstruction-based featurizers, including sparse autoencoders, do not reliably identify these intermediate features as individual latents, although the same architectures recover them with direct supervision. This suggests that reconstruction alone cannot identify the planted basis even when the featurizer has the capacity to represent it, isolating a critical identifiability failure. We address this by constraining the featurizer dictionary to represent features relevant to downstream computation by pressuring the features to align with the Jacobian. On the synthetic model, this featurizer identifies the ground-truth basis to reveal the planted compositional hierarchy where baselines could not. We then apply this approach to DINOv3, where no intermediate ground-truth is available. Without any supervision, the resulting attribution graphs yield part-to-whole object circuits: features for a rabbit's eye, nose, snout, and chest funnel into a downstream rabbit feature. Intervening on targeted upstream features changes the responses of connected downstream features, supporting the functional relevance of these links. For instance, patching a stripe concept on the horse body shifts a frozen ImageNet readout toward the zebra. Together, our results show that grounding features in downstream gradients addresses shortcomings of optimizing reconstruction alone, providing a path from featurizing representations to featurizing computation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

"But clever neurons are not enough," noted Barlow in 1972, reflecting on the limits of explaining perception solely through the responses of individual cells and not on *how* these cells interact to produce perception. Interpretability faces an analogous problem. Featurizers, auxiliary models used to explain the internal representations of models, are typically trained to reconstruct activations. They therefore describe what a layer represents without determining how the recovered features participate in downstream computation. This motivates the two questions at the center of this work. First, do the features found by a featurizer correspond to meaningful components of the computation, or do they merely provide a convenient basis for reconstruction? And second, can more faithful computational features be recovered by constraining the way they compose?

The second question builds on one account of the success of deep networks: that they learn structured algorithms by composing simpler computations. Under compositional sparsity, a complex function is organized as a graph of simpler functions, each using only a few inputs and passing its outputs to subsequent stages (Poggio & Fraser, 2024; Danhofer et al., 2025). Related work on compositional generalization examines whether models can recombine learned constituents in configurations that were not seen during training (Okawa et al., 2023), and due to its connection to generalization ability, understanding the circumstances in which models can recover compositional structure has been widely studied (Dziri et al., 2023; Hupkes et al., 2020; Griffiths et al., 2025; Marcus, 2001; Quilty-Dunn et al., 2023; Lake et al., 2016; Smolensky, 1990; Fodor & Pylyshyn, 1988; Lake & Baroni, 2018; Lake, 2019; McCoy et al., 2026; Yang et al., 2025; McCoy & Griffiths, 2024; Lippl et al., 2025; Andreas et al., 2016; Koh et al., 2020; Webb et al., 2024; Ellis et al., 2023; Webb et al., 2025; Assouel et al., 2025; Li et al., 2026; Zerroug et al., 2022; Fu et al., 2025; Lepori et al., 2024; 2023; Song et al., 2026). However, knowing that a model can generalize in this way does not directly tell us how these computations are implemented internally within the network, nor on which tasks models learn such an implementation. For a compositional interpretation, we would want to isolate both the features involved and recover the structural relations between features that the model computes to produce its behavior (Gauderis et al., 2026).

Neural networks can, in principle, represent these parts in many different ways. One approach to recovering these parts follows the linear representation hypothesis (LRH) which treats concepts as directions in activation space (Park et al., 2023; Elhage et al., 2022). This motivates dictionary learning methods like sparse autoencoders (SAEs) which learn large dictionaries of sparsely activating, interpretable concept directions by reconstructing activations with sparse codes (Olshausen & Field, 1996; Cunningham et al., 2023; Bricken et al., 2023; Templeton et al., 2024; Gao et al., 2024). However, being able to reconstruct activations or decode a concept does not tell us whether or how the model uses that information for downstream computation (Belinkov, 2022; Elazar et al., 2021; Ravichander et al., 2021; Hewitt & Liang, 2019; Wu et al., 2025; Geiger et al., 2022; 2025).

To this end, the circuits perspective provides an explicit way to describe these interactions by treating features as nodes in a graph and the connections between them as functional relations (Olah et al., 2020; Ameisen et al., 2025; Marks et al., 2025; Kowal et al., 2024). Function-aware featurizers move the same idea into feature learning: end-to-end SAEs preserve model behavior, transcoders predict downstream activations, gradient SAEs use downstream gradients to select active dictionary features and Jacobian SAEs encourage a sparse feature-to-feature Jacobian across an MLP (Braun et al., 2024; Dunefsky et al., 2024; Olmo et al., 2025; Farnik et al., 2025). Our focus is whether such computational constraints help identify the intermediate variables of a compositional computation. We first study this in a controlled synthetic setting and then apply the approach to DINOv3, where no intermediate ground truth is available. Our contributions are:

- **A testbed for identifying intermediate variables of a sparse compositional computation.** We construct a sparse compositional DAG in the form of a layered Boolean circuit, where every intermediate variable and parent–child relation is known. We plant the intermediate features of this DAG into a transformer by supervising them through linear probes attached to the residual stream at successive transformer blocks. This allows us to directly test whether reconstruction-based featurizers can identify the planted features. We find that they do not, reaffirming the problem of feature identifiability in sparse representation learning (Pacela et al., 2026). However, nudging the featurizer latents toward the known DAG nodes with direct supervision allows the same architectures to recover the planted basis while maintaining sparse reconstruction. The target basis is

therefore compatible with sparse reconstruction, but reconstruction-based training alone does not identify it.

- **Jacobian pressure.** To address the identifiability problem, we introduce a Jacobian coverage loss that uses gradients to encourage the dictionary to capture directions that the downstream features are locally sensitive towards. This additional constraint improves recovery of the planted basis. We then use gradients computed through the learned dictionaries to iteratively trace features backward across layers, recovering the planted compositional hierarchy more faithfully.
- **Tracing part-to-whole composition at scale.** We then apply this method to DINOv3, a vision foundation model (Siméoni et al., 2025), to examine the organization of learned visual features (Bau et al., 2017; Zhang et al., 2018; Kowal et al., 2024; Olah et al., 2020; Rajaram et al., 2024; Fel et al., 2025a;b; Doshi et al., 2025; 2026). We find attribution graphs that show part-to-whole routes, as shown in Figure 1, where features for a rabbit's eye, nose, snout, and chest funnel into a downstream rabbit feature (see Figure 21). We further examine the functional relevance of these relations through targeted interventions: manipulating selected upstream concepts changes the responses of connected downstream concepts.

2 A Toy Model of Compositional Computation

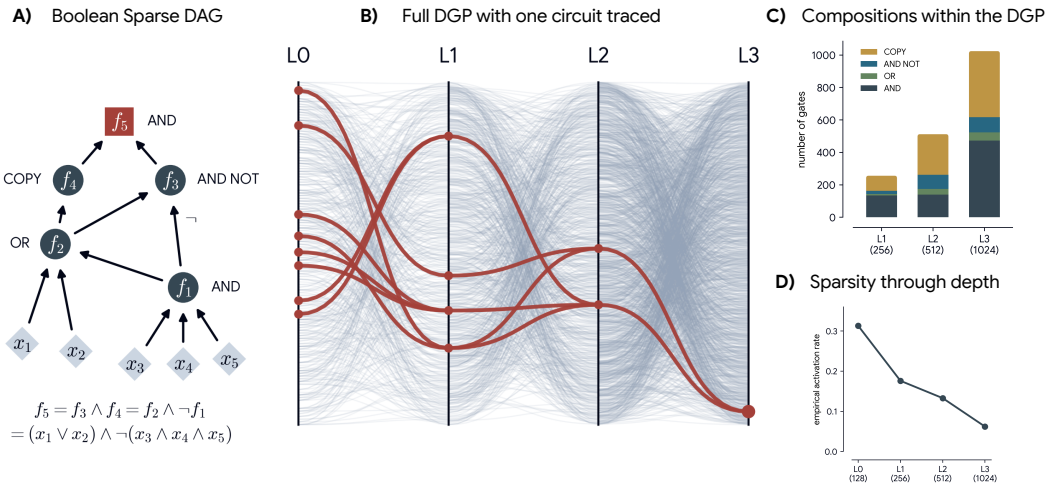

Figure 2: **A sparse Boolean feature DAG defines a known data-generating process. (A)** A schematic circuit illustrates how Boolean gates compose five Boolean leaf activations into a higher-level feature. **(B)** The full four-level DAG, with one constituent circuit highlighted in red. **(C)** The number and type of gates at each composed level. **(D)** The mean fraction of active features at each DAG layer over 10,000 held-out inputs.

To test whether a featurizer can recover the internal variables of a compositional computation, we need a setting in which those variables are known. We construct a sparse Boolean feature DAG that, together with a known input distribution, defines a data-generating process (DGP) with exact feature targets at every level (Figure 2). We then plant successive DAG levels into a transformer using depth-matched linear probes, making these targets accessible at the assigned transformer depths (Figure 3).

A sparse Boolean feature DAG. Each input $\mathbf{x} \in \mathbb{R}^{128}$ is drawn from $\mathcal{N}(0, I_{128})$ and grouped into eight 16-dimensional tokens. The DAG has four levels $\ell \in \{0, 1, 2, 3\}$, with widths $(n_0, n_1, n_2, n_3) = (128, 256, 512, 1024)$. We write $\mathbf{f}^{(\ell)}(\mathbf{x}) \in \{0, 1\}^{n_\ell}$ for the feature activations at level ℓ, and $f_i^{(\ell)}(\mathbf{x})$ for its ith feature (the ground-truth). Level 0 consists of features that are thresholded random projections of the input: half of these features read from two to four adjacent tokens, while the remaining half read from all eight. For each feature j at level $\ell \in \{1, 2, 3\}$, let $p_1, \ldots, p_k$ index its parents in level $\ell - 1$. The feature applies a Boolean gate to their activations:

$$f_j^{(\ell)}(\mathbf{x}) = g_j^{(\ell)}\left(f_{p_1}^{(\ell-1)}(\mathbf{x}), \ldots, f_{p_k}^{(\ell-1)}(\mathbf{x})\right), \quad g_j^{(\ell)} \in \{\text{AND}, \text{OR}, \text{AND-NOT}, \text{COPY}\}. \tag{1}$$

Here $k = 1$ for COPY and $k \in \{2, 3, 4\}$ for AND, OR, and AND-NOT. AND-NOT returns one when its first $k - 1$ inputs are one and its final input is zero. The resulting DAG contains 1,920

features and 3,062 edges (Figure 2B). Its mean activation rate falls from 31.3% at level 0 to 6.2% at level 3, giving increasingly sparse feature activations through depth (Figure 2D). The full DAG construction details are in Appendix A.

Planting feature levels across transformer depth. To make each DAG level accessible at a designated depth, we train a six-block encoder-only transformer (Figure 3, left). For each block $b \in \{0, \ldots, 5\}$, we RMS-normalize its eight token states and average across tokens to obtain $\mathbf{h}^{(b)}(\mathbf{x}) \in \mathbb{R}^d$, where $d = 128$ is the transformer hidden dimension. We assign feature level ℓ to transformer block $\ell + 2$, so levels 0 through 3 map to transformer blocks 2 through 5. The transformer and probes are optimized jointly by averaging binary cross-entropy over the four feature levels.

$$\mathbf{s}^{(\ell)} = W^{(\ell)}\mathbf{h}^{(\ell+2)} + \mathbf{c}^{(\ell)}, \qquad \frac{1}{4}\sum_{\ell=0}^{3} \mathrm{BCE}\left(\mathbf{s}^{(\ell)}, \mathbf{f}^{(\ell)}\right). \tag{2}$$

The binary cross entropy loss is averaged equally across the four levels and is the transformer's only training objective. "Planting" supervises linear accessibility at selected blocks, but neither hard-codes the Boolean gates nor confines a feature to one block. On held-out inputs, the jointly trained probes achieve high F_1 across all four target levels (Figure 3, right). We compute F_1 for each target feature and then average, giving every feature equal weight. These results verify that the planted features are linearly accessible at their designated depths. Appendix A gives the complete architecture, pooling operation, optimization, and evaluation protocol.

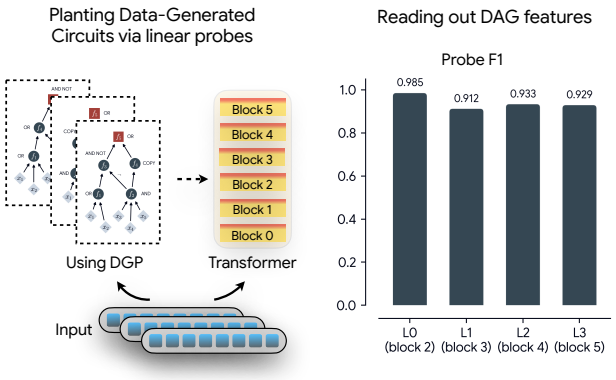

Figure 3: **Planting data-generated circuits across transformer depth. Left:** Each input is processed as eight tokens. The DGP provides the feature targets and linear probes plant these inside the transformer. **Right:** Held-out F_1 scores.

3 Sparse Reconstruction Does Not Identify the Planted Basis

Reconstruction-based featurizers. For DAG level ℓ at block $b = \ell + 2$, most of the methods learn a sparse code and reconstruct the pooled transformer state:

$$\mathbf{z}^{(b)} = G_b\left(E_b\left(\mathbf{h}^{(b)}\right)\right), \qquad \widehat{\mathbf{h}}^{(b)} = D_b\mathbf{z}^{(b)} + \mathbf{c}_b, \qquad \mathcal{L}_{\mathrm{rec}}^{(b)} = \frac{1}{d}\mathbb{E}_{\mathbf{x}}\left[\left\|\mathbf{h}^{(b)} - \widehat{\mathbf{h}}^{(b)}\right\|_2^2\right]. \tag{3}$$

Here E_b is an encoder, G_b imposes sparsity, and D_b is the learned dictionary. Here we compare six featurizers: per-example TopK (Gao et al., 2024), batch-level BatchTopK (Bussmann et al., 2024), learned-threshold JumpReLU (Rajamanoharan et al., 2024), relaxed archetypal (RA) SAE variant (Fel et al., 2025a), which constrains the dictionary geometry to the convex hull of the data, matching pursuit (MP) SAE (Costa et al., 2025) which replaces the learned encoder with greedy residual fitting, and a TopK transcoder (Dunefsky et al., 2024) which predicts block b from block $b-1$. At level ℓ, each method uses $m_\ell = 2n_\ell$ latents, giving widths $(256, 512, 1024, 2048)$ from L0 to L3. Since every pooled transformer state is 128-dimensional, these dictionaries have expansion factors of $2\times$, $4\times$, $8\times$, and $16\times$, respectively. They are therefore overcomplete with respect to the representation dimension as well as the planted feature set. For methods with a fixed activity budget, we choose k based on the average ground-truth L_0 at each level, while JumpReLU learns its thresholds under an L_0 penalty. Appendix B gives the complete objectives and training protocol.

Measuring one-to-one feature recovery. Reconstruction asks whether the learned code preserves the transformer state but it does not ask whether an individual latent corresponds to an individual ground-truth feature. We measure this using the injective Hungarian assignment which maps every ground-truth feature to a distinct learned latent over held-out inputs, thus preventing one latent from being credited for multiple target ground-truths. Let $\mathbf{y}_i^{(\ell)}$ be the ground-truth feature i across these inputs, and let $\mathbf{a}_j^{(\ell)}$ be the binary activity $\mathbf{1}[z_j^{(\ell+2)} > 0]$ of learned latent j. We define the Hungarian F1 as:

$$S_{ij}^{(\ell)} = F_1\left(\mathbf{y}_i^{(\ell)}, \mathbf{a}_j^{(\ell)}\right), \qquad F_{1,\mathrm{Hung}}^{(\ell)} = \frac{1}{n_\ell} \max_{\substack{\pi:\{1,\dots,n_\ell\}\to\{1,\dots,m_\ell\} \\ \pi \text{ injective}}} \sum_{i=1}^{n_\ell} S_{i,\pi(i)}^{(\ell)}. \tag{4}$$

Reconstruction fidelity does not imply feature identifiability. Table 1 shows that good reconstruction does not necessarily imply identifiability of the planted features. At block 5, for example, TopK reconstructs the representation with $R^2 = 0.93$, but its Hungarian F_1 is only 0.16. We find a similar gap across the other featurizers: changing the sparsity mechanism, constraining the dictionary toward archetypal geometry, replacing amortized inference with matching pursuit, or predicting from the preceding block does not resolve this gap. However, these low one-to-one scores do not mean that information about the planted features has been lost. Dense linear probes can still decode these features from the code as a whole, achieving F_1 scores above 0.8 for most featurizers (see Appendix B.3). That is, the ground-truth features remain decodable from combinations of latents, even though the individual latents do not align with them one-to-one.

Method	Block 2 (L0)		Block 3 (L1)		Block 4 (L2)		Block 5 (L3)	
	R^2	F_1	R^2	F_1	R^2	F_1	R^2	F_1
TopK	0.940	0.573	0.794	0.382	0.940	0.294	0.927	0.162
BatchTopK	0.943	0.577	0.976	0.332	0.995	0.189	0.990	0.078
JumpReLU	0.942	0.560	0.930	0.291	0.950	0.160	0.914	0.050
RA	0.926	0.480	0.947	0.327	0.980	0.317	0.982	0.206
MP	0.918	0.471	0.976	0.295	0.998	0.200	0.999	0.139
Transcoder	0.902	0.565	0.883	0.375	0.976	0.429	0.929	0.310
Transformer probe	–	0.985	–	0.912	–	0.933	–	0.929

Table 1: Reconstruction and feature identifiability of the six baseline featurizers across transformer blocks. R^2 is held-out reconstruction R^2 (for the transcoder, it measures prediction of target block b from block $b-1$); F_1 is the strict one-to-one Hungarian F_1 between featurizer latents and the ground-truth features read out at that block, on 10,000 held-out samples. RA = relaxed archetypal SAE, MP = matching pursuit.

To test whether this result depends on featurizer scale or model capacity, we vary the dictionary width and sparsity, and also train the featurizers on transformers with widths ranging from 128 to 1024. For each method and block, we report the highest ground-truth Hungarian F_1, making this an intentionally optimistic comparison. While the F_1 scores improved slightly at the deeper layers, neither increasing the model width nor tuning the featurizer brings one-to-one recovery up to the level achieved by the linear probes. This gap remains even when we choose the best activation threshold for each feature–latent pair (see Appendix B.4 for the full results).

The planted basis is expressible by the same featurizers. One explanation for the low Hungarian F_1 scores is that the featurizer itself is incapable of doing sparse reconstruction using the ground-truth basis. We test this with an oracle control. We initialize the dictionary encoder directions corresponding to the first n_ℓ latents using the planted probe directions (equation 2) and directly supervise the post-gate activations of these latents:

$$\mathcal{L}_{\mathrm{sup}}^{(b)} = \mathcal{L}_{\mathrm{rec}}^{(b)} + \beta\, \mathbb{E}_{\mathbf{x}}\left[\mathrm{BCE}\left(\mathbf{z}_{1:n_\ell}^{(b)}, \mathbf{f}^{(\ell)}\right)\right]. \tag{5}$$

Direct supervision substantially improves Hungarian F_1 while maintaining high reconstruction R^2 (Figure 4). The ground-truth basis is therefore compatible with the sparse featurizer codes. Their unsupervised failure is not due to a lack of expressivity. Reconstruction simply gives the featurizers

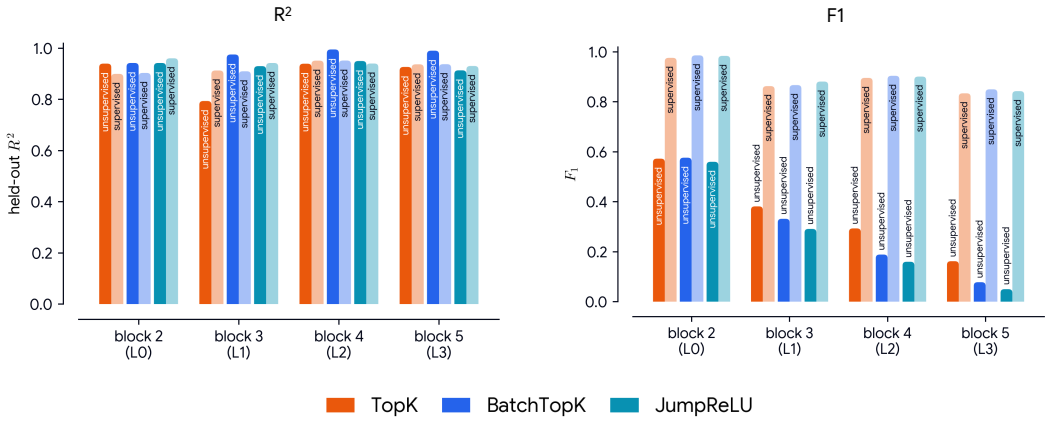

Figure 4: **Feature identifiability is not an expressivity bottleneck.** Directly supervised TopK, BatchTopK, and JumpReLU featurizers recover the planted feature basis while retaining high reconstruction R^2.

no reason to prefer the planted basis over other sparse bases that can reproduce the same activations. To distinguish among these bases, we next introduce a signal based on how these activations are used in downstream computation.

4 Grounding Featurizers in Downstream Computation

A computation-aligned featurizer should capture variations in the activation that affect downstream features and we use this downstream computation to distinguish among the many possible sparse feature bases that can be used to reconstruct the same activations. Gradients identify these variations locally. Let $s_c^{\text{down}}(\mathbf{x})$ denote the scalar score of a downstream target c present in the input. For the block-4 featurizer, this score is the block-5 probe logit of a present ground-truth feature in the input. We then carry this signal to earlier blocks by training the featurizers in reverse order (Figure 5A). Once the block-4 featurizer is trained, we freeze it and use the encoder preactivations of its active latents as downstream scores for block 3. We repeat this step from block 3 to block 2. We call this reverse training procedure *backchaining*. Since the next block acts on the token activations, we differentiate its score with respect to a shared change across all eight tokens:

$$\mathbf{g}_c^{(b)}(\mathbf{x}) = \sum_{t=1}^{8} \frac{\partial s_c^{\text{down}}}{\partial \mathbf{h}_t^{(b)}}(\mathbf{x}). \tag{6}$$

The resulting gradient is a d-dimensional ($\mathbb{R}^{128}$) direction in the block-b activation space. It describes how the activation can be changed to most strongly affect downstream feature c, locally around input $\mathbf{x}$. We amortize these input-specific gradients into the featurizer dictionary by using them as an additional constraint during featurizer training. We use the ReLU TopK SAE as the featurizer here. Let $\mathbf{q}_j^{(b)}$ denote the unit-normalized featurizer direction associated with latent j and $\mathcal{A}^{(b)}(\mathbf{x})$ denote the set of latents selected by TopK for input $\mathbf{x}$ in the forward pass. We will apply the Jacobian pressure on these $\mathcal{A}^{(b)}(\mathbf{x})$ latents via a coverage loss. For each present downstream target c, the coverage loss asks whether its gradient aligns with at least one direction in $\mathcal{A}^{(b)}(\mathbf{x})$.

$$\mathcal{L}_{\text{cov}}^{(b)} = \mathbb{E}_{\mathbf{x},\, c \text{ present}} \left[1 - \max_{j \in \mathcal{A}^{(b)}(\mathbf{x})} \left| \left\langle \frac{\mathbf{g}_c^{(b)}(\mathbf{x})}{\|\mathbf{g}_c^{(b)}(\mathbf{x})\|_2}, \mathbf{q}_j^{(b)} \right\rangle \right| \right]. \tag{7}$$

This loss is small when at least one selected upstream latent aligns with the gradient of each active downstream target. This term is trained alongside the reconstruction objective. The toy compositional setup also uses auxiliary population-level coverage terms; Appendix C gives the full objective and training settings. The coverage loss in equation 7 is the core term that we later scale to a vision foundation model. Further details are given in Appendix C.

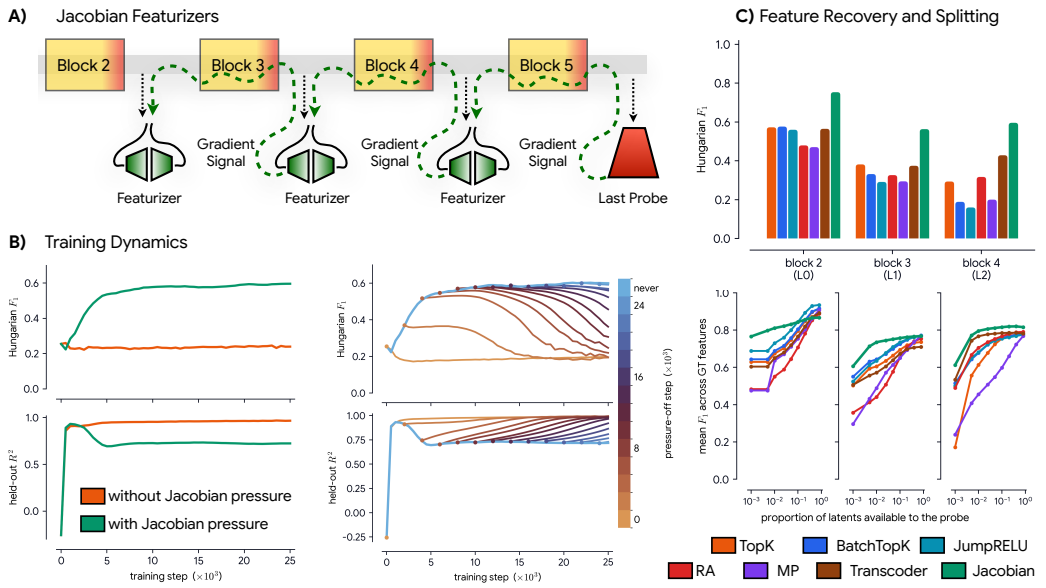

Figure 5: **Downstream-gradient pressure improves recovery of the planted features.** **(A)** Dictionaries are trained sequentially from block 4 to block 2, beginning with the block-5 planting head; dashed green arrows show the gradient signal used at each stage. **(B)** Left: block-4 Hungarian F_1 and held-out reconstruction R^2 during training with and without Jacobian pressure. Right: the same metrics when all Jacobian-specific terms are removed at different training steps; dots mark when the pressure is removed. **(C)** Top: Hungarian F_1 compared with the reconstruction-based featurizers from Section 3. Bottom: target-specific sparse-probe F_1 as more dictionary coordinates are made available to the probe.

Downstream gradients improve feature identifiability. At block 4, Jacobian pressure from the last probe raises Hungarian F_1 relative to the matched reconstruction-only control, although reconstruction R^2 is lower (Figure 5B, left). To test whether this improved feature identifiability depends specifically on the Jacobian pressure, we remove it at different training steps while allowing the reconstruction to continue. Once the pressure is removed, Hungarian F_1 falls as reconstruction improves; removing it earlier returns the dictionary towards the high-reconstruction and low-recovery regime (Figure 5B, right). Continued downstream pressure is therefore needed to maintain the observed alignment in this setting.

After backchaining to blocks 3 and 2, the Jacobian dictionaries reach the highest Hungarian F_1 among the featurizers compared at all three blocks (Figure 5C, top). We also use sparse probes to test how many learned latents are needed to decode each ground-truth feature (Appendix C.5). When each probe uses only one latent, the Jacobian dictionaries give the highest mean F_1 at all three blocks (Figure 5C, bottom). This suggests less splitting of the planted features across dictionary latents. Since different targets may reuse the same latent, sparse probes complement rather than replace the one-to-one Hungarian evaluation. These results show that downstream gradients can help align the learned basis with the planted ground-truth features, although this comes at some cost to reconstruction fidelity. This is consistent with prior work suggesting that not all variation in activations is equally important for downstream behavior (Braun et al., 2024; Marks et al., 2025; Gurnee et al., 2026).

5 Intermediate Computations in DINOv3

Featurizing DINOv3. We next apply the Jacobian pressure to a frozen DINOv3 ViT-B/16 (Siméoni et al., 2025) containing 12 transformer blocks, where intermediate ground-truth features are not directly available. We use block-sparse featurizers (BSFs) (Fel et al., 2026), with 16,384 concept groups of three latents. At each image patch, a TopK gate selects the 32 groups (our L0) with the largest norms. A frozen reconstruction-trained BSF from the last block provides the downstream

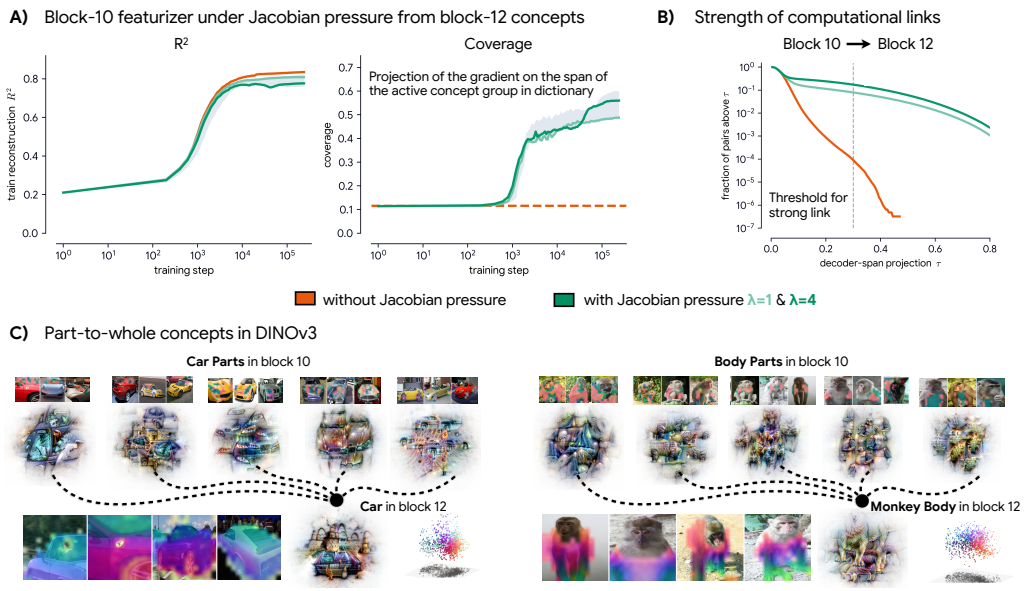

Figure 6: **Jacobian pressure reveals part-to-whole connections in DINOv3.** **(A)** Block-10 reconstruction R^2 and decoder coverage during training. Green curves show featurizers trained with Jacobian coefficients 1 and 4; orange shows the regular reconstruction-only featurizer. The dashed orange line shows coverage measured after training. **(B)** Fraction of gradient projections over the decoder spans of TopK-selected groups that exceeds each threshold. **(C)** Block-10 features for car parts and monkey body parts connect to car and monkey body concepts at block 12.

targets for a featurizer trained on block 10. Hence, gradients propagate through two transformer blocks (hop 2). To compute these gradients, we start from cached activations at the upstream block (block 10 for example) and run only the intervening transformer blocks (blocks 11 and then 12). We compute these gradients on the fly at each training step and hold them fixed while updating the upstream featurizer. We retain patch, CLS, and register tokens so that attention operates on the full input. To compute the Jacobian gradients at each step, we use B images and sample up to S active downstream concept groups per image. For each sampled group, we make a separate copy of the image's activations and compute the target score by summing that group's norms over its active patches. Since these copies are processed independently, a single batched Vector–Jacobian product (VJP) gives a separate gradient for each image and downstream group. For each image and sampled group, we retain the M upstream patches with the largest gradient norms for the coverage loss. In the BSF variant, this extends equation 7 by encouraging the span of at least one TopK-selected upstream group to contain the gradient direction at each supervised patch. Local coverage is the only Jacobian loss added to reconstruction.

Jacobian pressure improves gradient coverage. Figure 6A shows featurizers trained on block-10 activations with gradient pressure from active block-12 concepts, identified by a frozen reconstruction-only BSF. At each training step, we compute gradients for 32 active concepts from each of 16 images. For each image and concept, we select the 96 block-10 patches with the largest gradient norms to apply the coverage loss on. We show two variants in Figure 6A (green lines) corresponding to different coefficients for this coverage loss (1 and 4). The shaded area shows the range across 32 training configurations, where we vary the coverage weight and the numbers of images, downstream concepts, and upstream patches that we use to apply the Jacobian pressure. Remaining training configurations are shown in Appendix D. The featurizers with the Jacobian pressure increase coverage of the downstream gradients during training but with lower reconstruction R^2. The regular BSF achieves higher reconstruction R^2 with much lower gradient coverage (Figure 6A) measured only after training is completed. This pattern is consistent with our synthetic results.

Feature graphs and causal interventions. We project each normalized gradient onto the decoder span of every TopK-selected upstream group at the corresponding patch. The length of this projection measures how well the group can express a direction that locally changes the downstream

concept. Figure 6B shows the fraction of these gradient–group pairs (6.3 million pairs over 2,048 images, 32 active downstream concepts and 96 patches) whose projection exceeds each threshold. Taking a projection above 0.3 to indicate a strong link (because a random dictionary never exceeds 0.20), we find that 7.9% and 17.3% of the pairs exceed this threshold for coverage coefficients 1 and 4, respectively, compared with 0.009% for the regular BSF.

To qualitatively visualize these connections, we aggregate strong link scores across patches and object-specific images (of cars and monkeys for example) and inspect the linked features using their activation overlays and feature visualizations. The examples in Figure 6C illustrate part-to-whole relations: block-10 features responding to parts of a car connect to a block-12 car feature, while features responding to parts of monkey bodies connect to a downstream monkey body feature. Additional concept visualizations and their interactions are shown in Appendices D.1 and D.2. We then test whether intervening on these linked features causally affects downstream computation by scaling their decoded contributions to block-10 activations and measuring changes in the connected block-12 concepts. For example, amplifying a rabbit ear feature increases the downstream rabbit body response, while subtracting its contribution reduces it (Appendix D.3). We also add a stripe-feature direction obtained from zebra training images to block-10 activations of 50 validation horse images (see Appendix D.4). We apply this direction at patches where a horse-body feature is TopK-selected and run the remaining frozen transformer blocks. As we increase the scale of the added direction, the zebra–horse logit gap increases and a frozen ImageNet linear probe eventually classifies all 50 images as zebra (Figure 7). A neutral feature direction (neither horse- or zebra-specific) added at the same patches with the same norm produces no zebra predictions. In the Appendix D.4, we compare this stripe direction with the difference of means and a direction learned by a separate supervised horse–zebra probe. With all directions applied at the same patches, the stripe direction reaches 50% zebra predictions at a lower intervention strength than the class mean-difference direction, but a higher strength than the supervised probe direction.

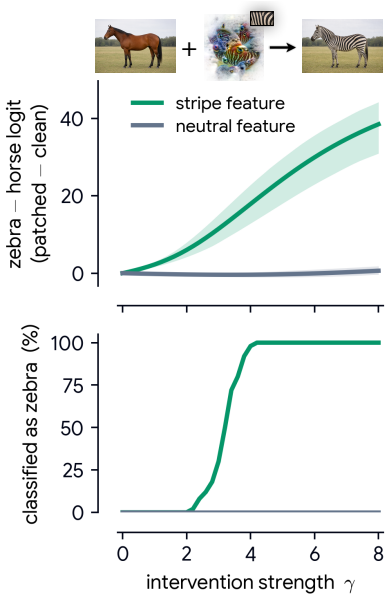

Figure 7: **Steering horse predictions toward zebra.** Top: Logit gap; Bottom: percentage of 50 validation horse images classified as zebra by the recognition probe.

6 Discussion

In this work, we find that optimizing featurizers directly for finding sparse features that have a role in downstream computation helps to identify causally important features. In our synthetic setup, the Jacobian pressure improves identifiability of planted compositional features that reconstruction alone does not reliably recover as individual latents. We apply this pressure using downstream gradients, which describe how small changes in activations affect later computation and hence give the featurizer information about how the model uses its representations. When this gradient pressure is removed, recovery declines even as reconstruction improves. In DINOv3, featurizers trained with this Jacobian pressure reveal part-to-whole connections and compositional circuits, wherein generic, reusable components combine together to form whole object categories (e.g., "cat" computed from "fur" and "upright ears" concepts; Figure 1). The prevalence and causal relevance of the compositional computations we find support arguments made by Poggio & Fraser (2024); Danhofer et al. (2025), that deep learning should be explained via such compositionally sparse programs.

Evidence for compositional circuits is exciting scientifically, as well as practically. Future work could involve shaping and recombining computational parts to design and guide how models make predictions during finetuning. It would be interesting to pursue a similar methodology, and apply these ideas on language models, where interpretable and reusable components have been observed naturally (Olsson et al., 2022; Wang et al., 2022; Merullo et al., 2024; Gurnee et al., 2026). Leveraging rich structure in neural networks, one that we can interpret, may help us avoid off-target effects during finetuning (Qi et al., 2024; Betley et al., 2025) and provide assurances on the model performance. Guiding model finetuning using these methods is an exciting direction for further work.

AI USE STATEMENT

We used generative AI tools to help write and organize experimental code for our synthetic DAG and data-generation procedure, and to scale up and parallelize featurizer training on the vision foundation model. We also used these tools to refine hypotheses, formulate mathematical definitions, obtain feedback on experiments, identify relevant literature and research gaps, and edit the manuscript for readability. In the vision experiments, we used automated interpretation (autointerp) to generate descriptions for the block-8, block-10, and block-12 feature groups visualized in the appendix, including concepts shared across classes. We also used these descriptions alongside activation overlays to help identify the stripe concept. We have reviewed and tested the code, checked the mathematical definitions and equations, verified references against the original papers, and checked the results. We take responsibility for the final content of this work, including its text, methods, and results.

REPRODUCIBILITY STATEMENT

We describe the synthetic data-generating process, transformer architecture, and planting procedure in Appendix A. Appendix B describes the baseline featurizers along with their training and evaluation protocols, and the width–sparsity sweep results. Appendix C describes gradient construction and the Jacobian losses, with loss coefficients and the dictionary settings in Tables 3 and 4 respectively. Section 5 and Appendix D describe the DINOv3 featurizer configurations. The causal intervention experiments are described in Appendices D.3 and D.4.

REFERENCES

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.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 39–48, 2016.

Rim Assouel, Declan Campbell, Yoshua Bengio, and Taylor Webb. Visual symbolic mechanisms: Emergent symbol processing in vision language models. *arXiv preprint arXiv:2506.15871*, 2025.

Horace B Barlow. Single units and sensation: a neuron doctrine for perceptual psychology? *Perception*, 1(4):371–394, 1972.

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 2022.

Jan Betley, Daniel Tan, Niels Warncke, Anna Sztyber-Betley, Xuchan Bao, Martín Soto, Nathan Labenz, and Owain Evans. Emergent misalignment: Narrow finetuning can produce broadly misaligned llms. *arXiv preprint arXiv:2502.17424*, 2025.

Dan Braun, Jordan Taylor, Nicholas Goldowsky-Dill, and Lee Sharkey. Identifying functionally important features with end-to-end sparse dictionary learning. *ArXiv e-print*, 2024.

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.

Bart Bussmann, Patrick Leask, and Neel Nanda. Batchtopk sparse autoencoders. *ArXiv e-print*, 2024.

Valérie Costa, Thomas Fel, Ekdeep Singh Lubana, Bahareh Tolooshams, and Demba Ba. From flat to hierarchical: Extracting sparse representations with matching pursuit. *arXiv preprint arXiv:2506.03093*, 2025.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

David A Danhofer, Davide D'Ascenzo, Rafael Dubach, and Tomaso Poggio. Position: A theory of deep learning must include compositional sparsity. *arXiv preprint arXiv:2507.02550*, 2025.

Fenil R Doshi, Thomas Fel, Talia Konkle, and George Alvarez. Visual anagrams reveal hidden differences in holistic shape processing across vision models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Fenil R Doshi, Thomas Fel, Talia Konkle, and George Alvarez. Bi-orthogonal factor decomposition for vision transformers. *arXiv preprint arXiv:2601.05328*, 2026.

Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable llm feature circuits. *Advances in Neural Information Processing Systems*, 37:24375–24410, 2024.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of transformers on compositionality. *Advances in neural information processing systems*, 36:70293–70332, 2023.

Yanai Elazar, Shauli Ravfogel, Alon Jacovi, and Yoav Goldberg. Amnesic probing: Behavioral explanation with amnesic counterfactuals. *Transactions of the Association for Computational Linguistics*, 9:160–175, 2021. doi: 10.1162/tacl_a_00359. URL https://aclanthology.org/2021.tacl-1.10/.

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

Kevin Ellis, Lionel Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. *Philosophical Transactions of the Royal Society A*, 381(2251):20220050, 2023.

Lucy Farnik, Tim Lawson, Conor Houghton, and Laurence Aitchison. Jacobian sparse autoencoders: Sparsify computations, not just activations. *arXiv preprint arXiv:2502.18147*, 2025.

Thomas Fel, Ekdeep Singh Lubana, Jacob S Prince, Matthew Kowal, Victor Boutin, Isabel Papadimitriou, Binxu Wang, Martin Wattenberg, Demba Ba, and Talia Konkle. Archetypal sae: Adaptive and stable dictionary learning for concept extraction in large vision models. *Proceedings of the International Conference on Machine Learning (ICML)*, 2025a.

Thomas Fel, Binxu Wang, Michael A Lepori, Matthew Kowal, Andrew Lee, Randall Balestriero, Sonia Joseph, Ekdeep S Lubana, Talia Konkle, Demba Ba, et al. Into the rabbit hull: From task-relevant concepts in dino to minkowski geometry. *arXiv preprint arXiv:2510.08638*, 2025b.

Thomas Fel, Matthew Kowal, Mozes Jacobs, Dron Hazra, Usha Bhalla, Lee Sharkey, Lucius Bushnaq, Satchel Grant, Tal Haklay, Thomas Icard, et al. Structuring sparsity: Block-sparse featurizers capture visual concept manifolds. *arXiv preprint arXiv:2606.25234*, 2026.

Jerry A Fodor and Zenon W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

Shuhao Fu, Andrew Jun Lee, Anna Wang, Ida Momennejad, Trevor Bihl, Hongjing Lu, and Taylor W Webb. Evaluating compositional scene understanding in multimodal generative models. *arXiv preprint arXiv:2503.23125*, 2025.

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.

Ward Gauderis, Thomas Dooms, Steven T Homer, Kola Ayonrinde, and Geraint A Wiggins. From mechanistic to compositional interpretability. *arXiv preprint arXiv:2605.08934*, 2026.

Atticus Geiger, Zhengxuan Wu, Hanson Lu, Josh Rozner, Elisa Kreiss, Thomas Icard, Noah Goodman, and Christopher Potts. Inducing causal structure for interpretable neural networks. In *International Conference on Machine Learning*, pp. 7324–7338. PMLR, 2022.

Atticus Geiger, Duligur Ibeling, Amir Zur, Maheep Chaudhary, Sonakshi Chauhan, Jing Huang, Aryaman Arora, Zhengxuan Wu, Noah Goodman, Christopher Potts, et al. Causal abstraction: A theoretical foundation for mechanistic interpretability. *Journal of Machine Learning Research*, 26(83):1–64, 2025.

Thomas L Griffiths, Brenden M Lake, R Thomas McCoy, Ellie Pavlick, and Taylor W Webb. Whither symbols in the era of advanced neural networks? *Trends in Cognitive Sciences*, 2025.

Wes Gurnee, Nicholas Sofroniew, Adam Pearce, Mateusz Piotrowski, Isaac Kauvar, Runjin Chen, Anna Soligo, Paul Bogdan, Euan Ong, Rowan Wang, Ben Thompson, David Abrahams, Subhash Kantamneni, Emmanuel Ameisen, Joshua Batson, and Jack Lindsey. Verbalizable representations form a global workspace in language models. *Transformer Circuits Thread*, 2026. URL https://transformer-circuits.pub/2026/workspace/index.html.

John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2733–2743, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1275. URL https://aclanthology.org/D19-1275/.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International conference on machine learning*, pp. 5338–5348. Pmlr, 2020.

Matthew Kowal, Richard P Wildes, and Konstantinos G Derpanis. Visual concept connectome (vcc): Open world concept discovery and their interlayer connections in deep models. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pp. 2873–2882. PMLR, 2018.

Brenden M Lake. Compositional generalization through meta sequence-to-sequence learning. *Advances in neural information processing systems*, 32, 2019.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *arXiv preprint arXiv:1604.00289*, 2016.

Michael Lepori, Thomas Serre, and Ellie Pavlick. Break it down: Evidence for structural compositionality in neural networks. *Advances in Neural Information Processing Systems*, 36:42623–42660, 2023.

Michael A Lepori, Alexa R Tartaglini, Wai K Vong, Thomas Serre, Brenden M Lake, and Ellie Pavlick. Beyond the doors of perception: Vision transformers represent relations between objects. *Advances in neural information processing systems*, 37:131503–131544, 2024.

Yihao Li, Saeed Salehi, Lyle Ungar, and Konrad Kording. Does object binding naturally emerge in large pretrained vision transformers? *Advances in Neural Information Processing Systems*, 38: 3394–3423, 2026.

Samuel Lippl, Thomas McGee, Kimberly Lopez, Ziwen Pan, Pierce Zhang, Salma Ziadi, Oliver Eberle, and Ida Momennejad. Algorithmic primitives and compositional geometry of reasoning in language models. *arXiv preprint arXiv:2510.15987*, 2025.

Gary F. Marcus. *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. The MIT Press, 2001.

Samuel Marks, Can Rager, Eric Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. In *International Conference on Learning Representations*, volume 2025, pp. 23888–23923, 2025.

R Thomas McCoy and Thomas L Griffiths. Meta-learning as a bridge between neural networks and symbolic bayesian models. *Behavioral & Brain Sciences*, 2024.

R Thomas McCoy, Paul Soulos, Tal Linzen, and Paul Smolensky. The emergent symbolic structure of artificial neural networks. *arXiv preprint arXiv:2608.29530*, 2026.

Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. Circuit component reuse across tasks in transformer language models. In *International Conference on Learning Representations*, volume 2024, pp. 18349–18377, 2024.

Maya Okawa, Ekdeep S Lubana, Robert Dick, and Hidenori Tanaka. Compositional abilities emerge multiplicatively: Exploring diffusion models on a synthetic task. *Advances in Neural Information Processing Systems*, 36:50173–50195, 2023.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Jeffrey Olmo, Jared Wilson, Max Forsey, Bryce Hepner, Thomas Vincent Howe, and David Wingate. Features that make a difference: Leveraging gradients for improved dictionary learning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7624–7634, 2025.

Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Vitória Barin Pacela, Shruti Joshi, Isabela Camacho, Simon Lacoste-Julien, and David Klindt. Stop probing, start coding: Why linear probes and sparse autoencoders fail at compositional generalisation. *arXiv preprint arXiv:2603.28744*, 2026.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. *arXiv preprint arXiv:2311.03658*, 2023.

Tomaso Poggio and Maia Fraser. Compositional sparsity of learnable functions. *Bulletin of the American Mathematical Society*, 61(3), 2024.

Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! In *International Conference on Learning Representations*, volume 2024, pp. 30988–31043, 2024.

Jake Quilty-Dunn, Nicolas Porot, and Eric Mandelbaum. The best game in town: The reemergence of the language-of-thought hypothesis across the cognitive sciences. *Behavioral and Brain Sciences*, 46:e261, 2023. doi: 10.1017/S0140525X22002849.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, Janos Kramar, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders. *ArXiv e-print*, 2024.

Achyuta Rajaram, Neil Chowdhury, Antonio Torralba, Jacob Andreas, and Sarah Schwettmann. Automatic discovery of visual circuits. *arXiv preprint arXiv:2404.14349*, 2024.

Abhilasha Ravichander, Yonatan Belinkov, and Eduard Hovy. Probing the probing paradigm: Does probing accuracy entail task relevance? In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 3363–3377, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.295. URL https://aclanthology.org/2021.eacl-main.295/.

Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.

Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.

Brian Song, Michael A Lepori, and Ellie Pavlick. Linguistic context recodes visual representations in vision-language models. *arXiv preprint arXiv:2608.00035*, 2026.

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.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

Taylor Webb, Shanka Subhra Mondal, and Ida Momennejad. A brain-inspired agentic architecture to improve planning with llms. *Nature Communications*, 16(1):8633, 2025.

Taylor W Webb, Steven M Frankland, Awni Altabaa, Simon Segert, Kamesh Krishnamurthy, Declan Campbell, Jacob Russin, Tyler Giallanza, Randall O'Reilly, John Lafferty, et al. The relational bottleneck as an inductive bias for efficient abstraction. *Trends in Cognitive Sciences*, 28(9): 829–843, 2024.

Zhengxuan Wu, Aryaman Arora, Atticus Geiger, Zheng Wang, Jing Huang, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. Axbench: Steering llms? even simple baselines outperform sparse autoencoders, 2025. URL https://arxiv.org/abs/2501.17148.

Yukang Yang, Declan Campbell, Kaixuan Huang, Mengdi Wang, Jonathan Cohen, and Taylor Webb. Emergent symbolic mechanisms support abstract reasoning in large language models. *arXiv preprint arXiv:2502.20332*, 2025.

Aimen Zerroug, Mohit Vaishnav, Julien Colin, Sebastian Musslick, and Thomas Serre. A benchmark for compositional visual reasoning. *Advances in neural information processing systems*, 35: 29776–29788, 2022.

Quanshi Zhang, Ruiming Cao, Feng Shi, Ying Nian Wu, and Song-Chun Zhu. Interpreting cnn knowledge via an explanatory graph. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

A TOY-MODEL CONSTRUCTION AND TRAINING DETAILS

A.1 FEATURE-DAG CONSTRUCTION

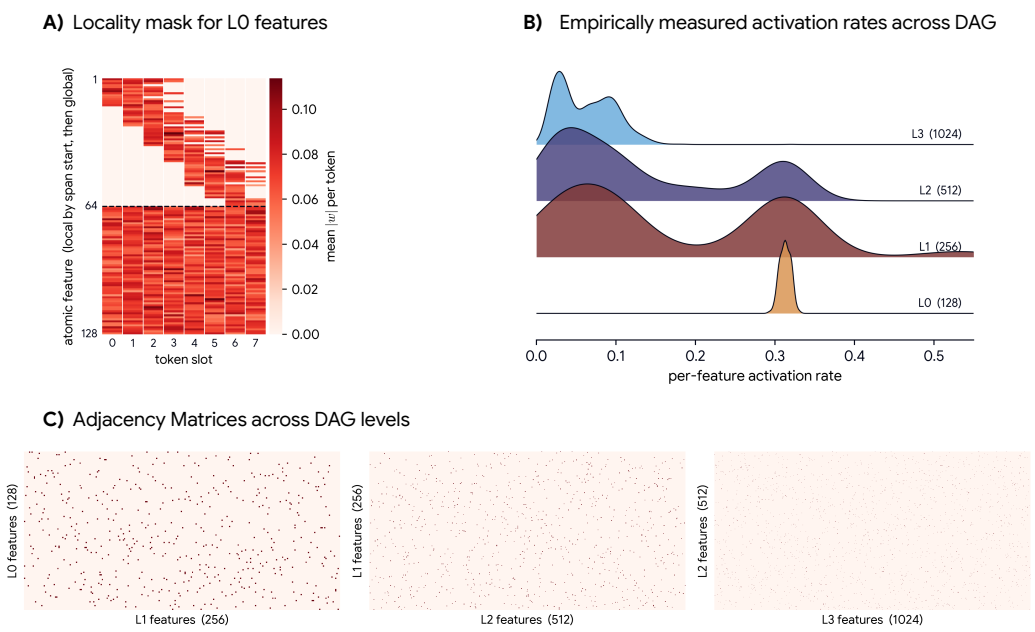

Figure 8: **Construction and statistics of the sparse Boolean feature DAG. (A)** Mean absolute atomic-feature weight within each token slot. There are a total of 128 features in L0. 64 of these features read locally while the remaining 64 features read from all eight tokens. **(B)** Per-feature activation-rate distributions at levels L0–L3, measured on 10,000 held-out inputs. **(C)** Unsigned adjacency matrices for the transitions L0→L1, L1→L2, and L2→L3. Dark entries denote parent–child edges.

Each input $\mathbf{x} \in \mathbb{R}^{128}$ is sampled from $\mathcal{N}(0, I_{128})$ and divided into eight contiguous 16-dimensional token slots. The four feature levels have widths $(n_0, n_1, n_2, n_3) = (128, 256, 512, 1024)$. Level 0 contains thresholded linear features

$$f_i^{(0)}(\mathbf{x}) = \mathbf{1}\left[(\mathbf{m}_i \odot \mathbf{w}_i)^\top \mathbf{x} > \tau_i\right], \tag{8}$$

where $\mathbf{w}_i \sim \mathcal{N}(0, I_{128}/128)$ and $\mathbf{m}_i \in \{0,1\}^{128}$ is a locality mask. Half of the masks select between two and four adjacent token slots; the remaining masks select all eight. Figure 8A visualizes the resulting locality pattern through the mean absolute weight within each token slot. We calibrate each threshold τ_i on 10,000 inputs by taking the $(1 - 40/128)$ quantile of the corresponding masked projection. Each atomic feature therefore activates with probability approximately $40/128 = 31.25\%$ on the calibration distribution.

At each higher level, we use AND, OR, AND-NOT, and COPY gates with probabilities 0.6, 0.1, 0.1, and 0.2. COPY selects one parent while the other gates select two, three, or four parents uniformly from the preceding level. Let $q_\ell = 40/n_\ell$ denote the nominal activation rate at level ℓ. We estimate the activation rate of each gate from the empirical parent rates using a product-of-marginals approximation. If this estimate falls outside $[0.15q_\ell, 4q_\ell]$, we resample for up to 50 attempts. After constructing each level, we check how often its features activate on the calibration inputs. Features that are active on fewer than 1.5% of these inputs are replaced by two-parent AND gates, with parents sampled from the most active third of the preceding level. We repeat this repair for at most five rounds. After filtering and repair, the counts for $(\text{AND}, \text{OR}, \text{AND-NOT}, \text{COPY})$ are $(135, 9, 19, 93)$ at level 1, $(140, 35, 87, 250)$ at level 2, and $(473, 51, 93, 407)$ at level 3. The resulting DAG has 1,920 features and 3,062 edges. On 10,000 held-out inputs, the mean activation rate at levels 0 through 3 are 31.3%, 17.6%, 13.3%, and 6.2%. Figure 8B shows the corresponding distribution across individual features at each level.

Finally we also extract an unsigned adjacency matrix $A^{(\ell)} \in \{0,1\}^{n_{\ell-1} \times n_\ell}$ for every pair of adjacent levels, as shown in Figure 8C:

$$A_{ij}^{(\ell)} = \begin{cases} 1, & \text{if feature } i \text{ at level } \ell - 1 \text{ is a parent of feature } j \text{ at level } \ell, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

A.2 Transformer architecture and Planting the DAG

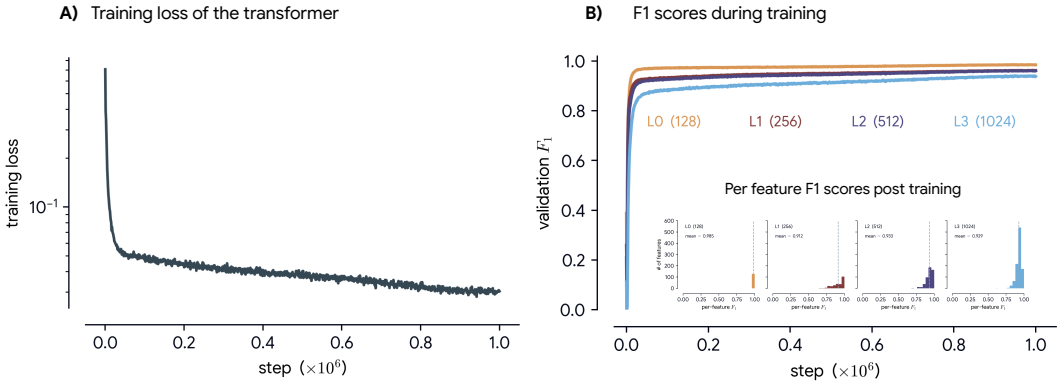

Figure 9: **Training dynamics. (A)** Training loss of the transformer. **(B)** F1 scores for the features increase during training. Inset fig shows F1 score for each feature.

The model is a six-block encoder-only transformer with residual width 128, four attention heads, and a 512-dimensional SwiGLU MLP. Each 16-dimensional input token is mapped into the residual stream by a bias-free linear embedding. Every block contains pre-RMSNorm self-attention with rotary position embeddings, followed by a pre-RMSNorm SwiGLU MLP; both sublayers use residual connections. Let $\mathbf{r}_t^{(b)}(\mathbf{x}) \in \mathbb{R}^{128}$ denote the state of token t after zero-indexed block b. The representation exposed to the probes and subsequent featurizers is

$$\mathbf{h}^{(b)}(\mathbf{x}) = \frac{1}{8} \sum_{t=1}^{8} \mathrm{RMSNorm}\left(\mathbf{r}_t^{(b)}(\mathbf{x})\right) \in \mathbb{R}^{128}. \tag{10}$$

Feature levels 0 through 3 are assigned to transformer blocks 2 through 5. For level ℓ, the corresponding probe parameters satisfy $W^{(\ell)} \in \mathbb{R}^{n_\ell \times 128}$ and $\mathbf{c}^{(\ell)} \in \mathbb{R}^{n_\ell}$, producing logits $\mathbf{s}^{(\ell)}(\mathbf{x}) \in \mathbb{R}^{n_\ell}$ as defined in Equation 2. The binary cross-entropy is then computed from the logits and averaged over examples and features within each level; the four level losses are then weighted equally. The transformer and probes are trained jointly for one million steps using fresh batches of 256 inputs (training dynamics shown in Figure 9A). We use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1, and gradient clipping at norm 1.0. The learning rate warms linearly to 3×10^{-4} over 5,000 steps and then follows a cosine schedule to 3×10^{-5}.

Evaluation of feature accessibility. For the bars in Figure 3, we generate 10,000 fresh inputs and use a fixed 70/30 split. The frozen, jointly trained probes are evaluated on the held-out 3,000 inputs. Predictions are obtained by thresholding the sigmoid probability at 0.5. We compute F_1 independently for every target feature and then average across features, so each feature contributes equally. The resulting scores for levels 0 through 3 are 0.985, 0.912, 0.933, and 0.929. The distribution of per-feature F1 scores is shown in Figure 9B. To test whether planting localizes each feature level to a particular depth, we freeze the transformer and train fresh linear probes from every block to every DAG level. We fit each

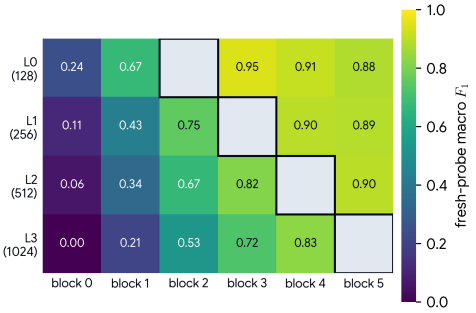

Figure 10: **Linear Probe F1 scores**. Fresh probes trained to readout the DAG features from activations of different blocks.

probe on 70% of a separate set of 10,000 examples and evaluate it on the remaining 30% (Figure 10). We observe a staircase pattern: higher-level features become progressively more accessible as we approach their assigned blocks, while features from earlier levels remain decodable in later blocks. Thus, planting makes the target features accessible at the intended depths, but does not restrict them to just those depths.

B Feature Recovery

Throughout, block $b = \ell + 2$ is paired with DAG level ℓ, and $\mathbf{h}^{(b)} \in \mathbb{R}^{128}$ denotes the pooled transformer state.

B.1 Featurizer families

Sparse autoencoders. We train a separate featurizer at each of blocks 2–5. The blockwise models have independent parameters and no cross-block objective. For TopK, BatchTopK, and JumpReLU, the encoder and decoder are

$$\mathbf{u}^{(b)} = E_b\left(\mathbf{h}^{(b)} - \mathbf{c}_b\right) + \mathbf{a}_b, \qquad \mathbf{p}^{(b)} = \mathrm{ReLU}\left(\mathbf{u}^{(b)}\right),$$
$$\mathbf{z}^{(b)} = G_b\left(\mathbf{p}^{(b)}\right), \qquad \widehat{\mathbf{h}}^{(b)} = D_b\mathbf{z}^{(b)} + \mathbf{c}_b, \tag{11}$$

where $E_b \in \mathbb{R}^{m_\ell \times 128}$, $D_b \in \mathbb{R}^{128 \times m_\ell}$, and G_b is the sparsity gate. TopK retains the k_ℓ largest entries for each input (Gao et al., 2024). BatchTopK retains the largest Bk_ℓ entries across a training batch of size B and uses an exponential-moving-average threshold at evaluation time (Bussmann et al., 2024). JumpReLU instead learns a positive threshold θ_j for each latent, using a kernel pseudo-gradient for the thresholds (Rajamanoharan et al., 2024). We use the Silverman kernel with bandwidth 0.15 and an L_0 coefficient of 2×10^{-2}. The standard SAE objective is reconstruction error plus a small reanimation term that pushes upward the pre-activations of latents that are inactive across the current batch. JumpReLU additionally includes its L_0 penalty. The reanimation coefficient is 3×10^{-4}, while ghost-gradient and AuxK losses are disabled in the reported runs. For the unsupervised TopK, BatchTopK, and JumpReLU models, decoder atoms are normalized after each optimizer step. Decoder normalization is disabled for the directly supervised models so that it does not immediately rescale their probe-based initialization. The transcoder also does not normalize its decoder.

Relaxed archetypal (RA) SAE. The relaxed archetypal SAE (RA) uses TopK codes but anchors its dictionary atoms to relaxed convex combinations of data-derived prototypes (Fel et al., 2025a). Inputs are standardized using fixed per-block statistics. We obtain 8,000 candidate prototypes by applying k-means to 50,000 cached transformer states and use relaxation radius $\delta = 1$ with a learned multiplier. The objective is reconstruction error plus the same reanimation term used for the standard SAEs.

Matching pursuit. Matching pursuit (MP) has no learned encoder. It constructs each code by repeatedly selecting the dictionary atom most aligned with the current residual and then updating that residual (Costa et al., 2025). The number of pursuit steps at each level is the same as the activity budget used for the fixed-sparsity featurizers. We use dropout 0.1 during training, and a residual-based atom-revival term with coefficient 10^{-2}. Its coefficients may be signed, and an atom can be selected more than once. Inputs are standardized at training and evaluation time; reconstructions and decoder atoms are transformed back to the original activation coordinates for evaluation.

Predictive transcoder. The transcoder uses a TopK bottleneck to predict the pooled state at block b from the preceding block (Dunefsky et al., 2024):

$$\mathbf{z}^{(b)} = \mathrm{TopK}\left(\mathrm{ReLU}\left(E_b\mathbf{h}_{\mathrm{in}}^{(b-1)} + \mathbf{a}_b\right)\right), \qquad \widehat{\mathbf{h}}^{(b)} = D_b\mathbf{z}^{(b)} + \mathbf{c}_b. \tag{12}$$

B.2 Width and sparsity

Every featurizer has twice as many latents as ground-truth features. Since the transformer state is 128-dimensional at every block, the dictionaries become increasingly overcomplete relative to the

Table 2: **Dictionary widths and sparsity settings.** The fixed setting is 1.5 times the mean ground-truth activity at each level. TopK, BatchTopK, RA, and the transcoder use it as an activity budget; BatchTopK satisfies it on average across a batch. MP uses it as its number of pursuit steps and can select an atom more than once. JumpReLU instead learns its thresholds under an L_0 penalty.

Level	Block	n_ℓ	m_ℓ	$m_\ell/128$	Groundtruth L0	k_ℓ
L0	2	128	256	2×	40	60
L1	3	256	512	4×	45	68
L2	4	512	1024	8×	68	102
L3	5	1024	2048	16×	64	96

representation space. Table 2 gives the exact widths and sparsity settings. The operating points were selected after width–sparsity sweeps. TopK, BatchTopK, RA, and the transcoder were evaluated at widths $\{1, 2, 4\} \times n_\ell$ and budgets between 0.5 and 1.5 times the planted activity. MP used widths $\{1, 2\} \times n_\ell$ and $\{1, 1.5\}$ times the planted activity as pursuit-step settings. JumpReLU was swept over L_0 coefficients from 10^{-2} to 7×10^{-2}. The selected JumpReLU models activate approximately 70–82 latents per input across blocks. TopK, BatchTopK, JumpReLU, and the transcoders are trained for 50,000 Adam steps from caches containing one million examples, with batch size 2,048. The learning rate warms linearly to 3×10^{-4} over 1,000 steps and then decays cosinusoidally to 3×10^{-5}; gradients are clipped at norm 1.0. RA and MP use caches containing 500,000 examples and a 500-step warmup, with the remaining optimizer settings unchanged. Seed 42 controls cache generation and RA k-means sampling, but parameter initialization and minibatch-index sampling are not explicitly globally seeded. All headline metrics are evaluated on 10,000 independently generated inputs using seed 12345. The figure reports one selected trained run per family rather than an average across initialization seeds.

B.3 Dense Probes

To test whether feature information is retained but distributed, we fit linear logistic probes from the complete learned code to all planted features. For the final comparison checkpoints, the dense probe is trained and evaluated on 10,000 examples, split 70/30 into 7,000 training and 3,000 held-out examples. We standardize each latent using the probe-training set, then optimize the probe for 1000 Adam steps with batch size 1,024, learning rate 3×10^{-2}, with class-rebalanced binary cross-entropy (each feature's positive class weighted by its inverse prevalence, clamped to at least 1). Sigmoid probabilities are thresholded at 0.5, and F_1 is computed for each planted feature before averaging. This protocol gives dense F_1 scores that are around 0.8 even at the deeper blocks (Figure 11).

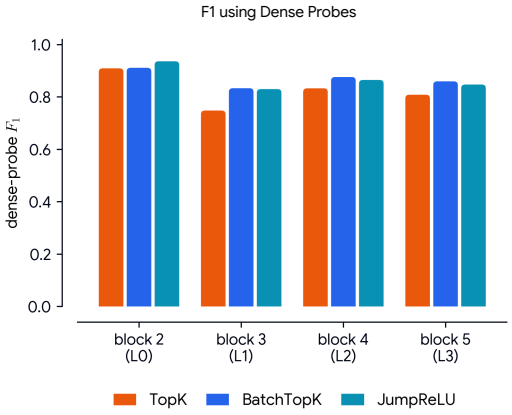

Figure 11: **Held-out F_1 of dense linear probes using the complete SAE code at each block.**

B.4 Robustness of feature-recovery failure

Dictionary width and sparsity. We test whether the recovery failure depends on the choice of the dictionary size by sweeping its width and sparsity for each method and transformer block. For TopK, BatchTopK, RA, and the transcoder, we use widths $\{1, 2, 4\} \times n_\ell$ and sparsity budgets between 0.5 and 1.5 times the mean ground-truth L_0. JumpReLU uses the same widths while sweeping its L_0 coefficient from 10^{-2} to 7×10^{-2}. We then select the configuration with the highest ground-truth Hungarian F_1 making this an intentionally optimistic comparison. Figures 12 and 13 show the complete sweeps and Figure 14 shows the scores at the best configurations. Even the best configuration in each sweep shows a big gap in one-to-one feature recovery while the reconstruction remains high for many of these configurations. Relaxing the sparsity constraint generally improves

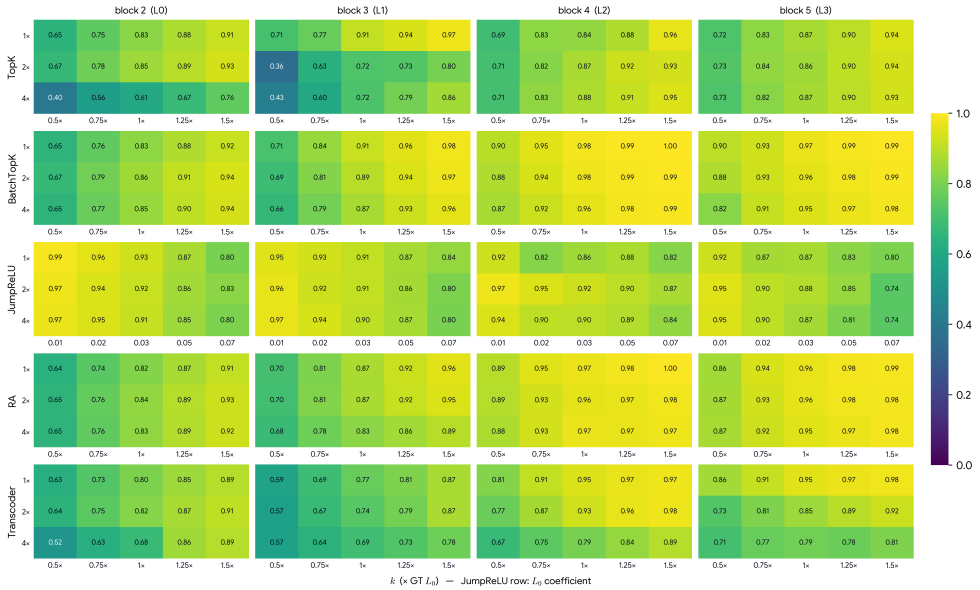

Figure 12: **Reconstruction across dictionary widths and sparsity settings.** Each cell reports held-out R^2. Rows give dictionary width relative to the number of ground-truth features. Columns give the sparsity budget relative to the mean ground-truth L_0.

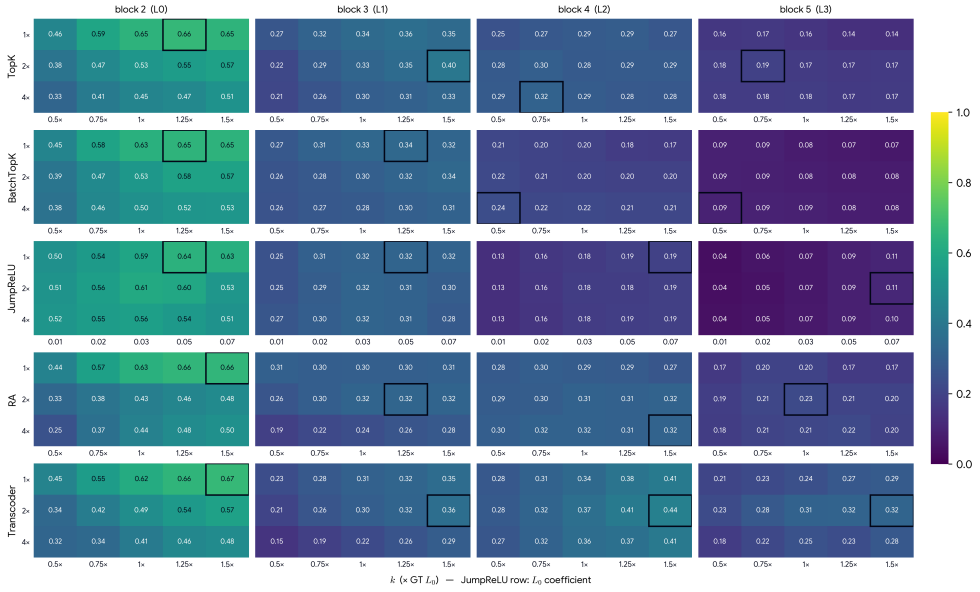

Figure 13: **Feature identifiability across dictionary widths and sparsity settings.** Each cell reports ground-truth Hungarian F_1. Rows give dictionary width relative to the number of ground-truth features. Columns give the sparsity budget relative to the mean ground-truth L_0. Outlined cells mark the configuration with the highest Hungarian F_1 for each method and block.

reconstruction, but no width–sparsity setting closes the recovery gap. In several cases, the highest Hungarian F_1 is obtained with the narrowest dictionary, also indicating that the failure cannot be explained simply by insufficient dictionary width.

Transformer width. The recovery failure could also arise because the 128-dimensional transformer compresses the DAG features too strongly. We therefore repeat the experiment with transformer

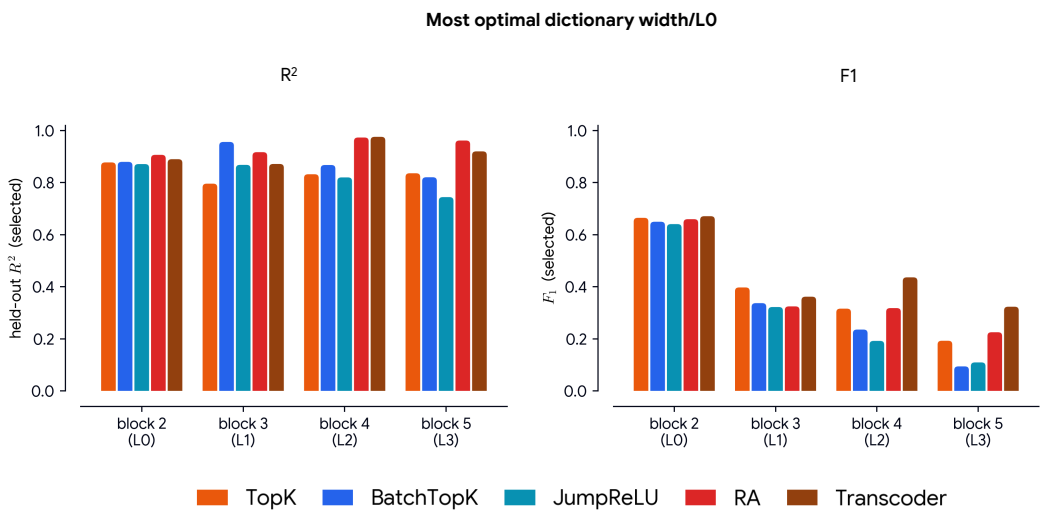

Figure 14: **Recovery at most optimal dictionary width and sparsity.** For each method and transformer block, we select the configuration with the highest ground-truth Hungarian F_1. The right panel reports this selected F_1, while the left reports the held-out R^2 of the same configuration.

widths $d \in \{128, 256, 512, 1024\}$ while keeping the feature DAG and transformer depth fixed. The probes retain high F_1 across all widths, with small improvements at the deeper blocks as the model becomes wider (Figure 15, left). For TopK and BatchTopK, we again sweep dictionary width and activity and select the configuration with the highest ground-truth Hungarian F_1. Increasing transformer width improves recovery at some deeper blocks, but even the best sweep-selected featurizers remain far below the probes. The recovery gap therefore cannot be explained simply by limited transformer capacity.

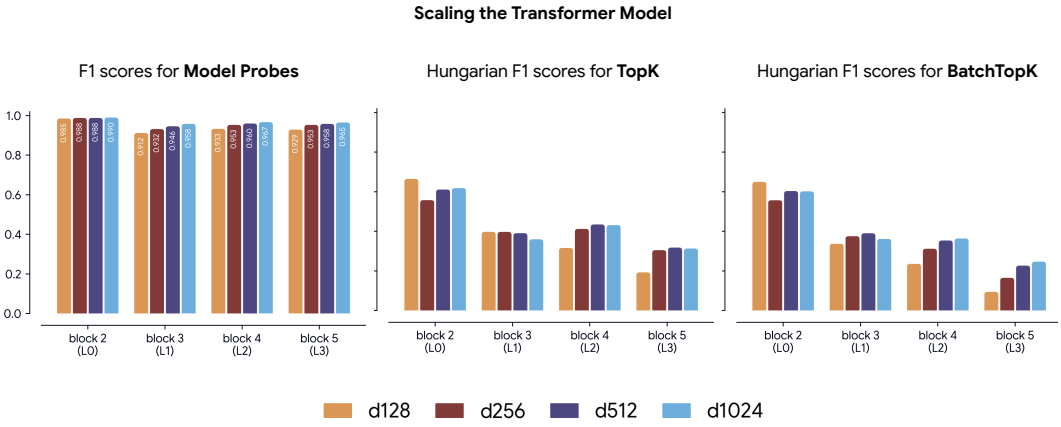

Figure 15: **Feature identifiability across different transformer widths.** Left: F_1 of the transformer probes. Middle and Right: Highest Hungarian F_1 achieved by TopK and BatchTopK across sweeps of dictionary width and sparsity, for each transformer width and block.

Sensitivity to the recovery criterion. We test whether our choice of the threshold to binarize a latent underestimates recovery by selecting the best threshold separately for every feature–latent pair. Reapplying Hungarian matching to these threshold-calibrated scores does not give any major improvement (Figure 16). If we remove the one-to-one constraint, each ground-truth feature can choose its best-matching latent. This gives higher scores, but the same latent may be counted as recovering multiple features, and therefore no longer measures recovery into unique latents.

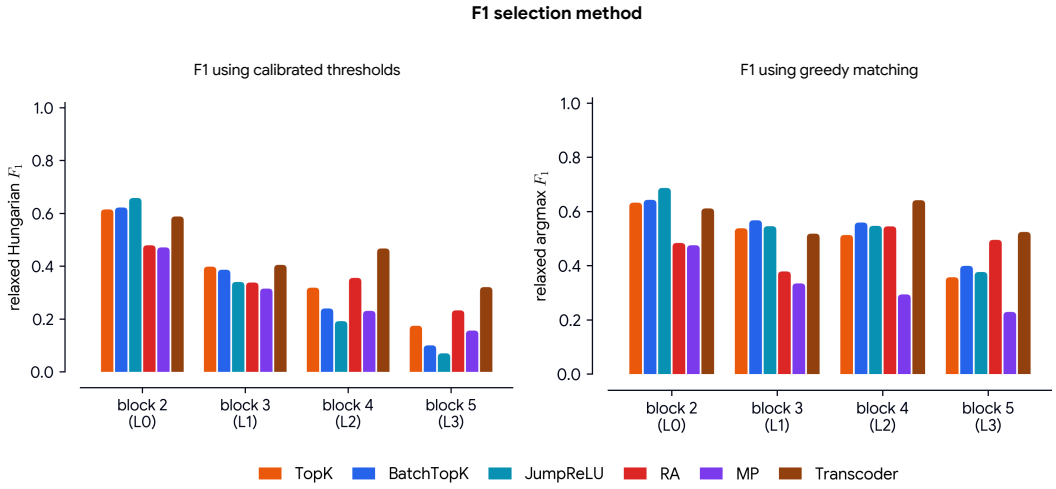

Figure 16: **Feature identifiability across calibrated thresholds and relaxed matching.** Left: Hungarian F_1 after selecting the best activation threshold for each ground-truth feature–latent pair. Right: Mean F_1 when each ground-truth feature selects its best-matching latent using greedy matching, with latent reuse allowed.

B.5 Directly supervised oracle control

To test whether the SAE architectures can express a sparse reconstruction aligned using the ground-truth features that we planted as the basis, we train directly supervised versions of TopK, BatchTopK, and JumpReLU. Let $W^{(\ell)} \in \mathbb{R}^{n_\ell \times 128}$ be the planting-probe weights at level ℓ, with each row normalized to unit norm. We initialize the first n_ℓ encoder and decoder features from these directions:

$$E_b[1\!:\!n_\ell, :] \leftarrow W^{(\ell)}, \qquad D_b[:, 1\!:\!n_\ell] \leftarrow W^{(\ell)\top}. \tag{13}$$

The remaining latents retain their random initialization, and all parameters remain trainable. We then add the direct feature loss in Equation 5, implemented as binary cross-entropy with logits on the post-gate activations of the first n_ℓ latents, with $\beta = 60$. The supervised models retain the same width and sparsity mechanisms as their unsupervised counterparts. Across the three gates, they reach Hungarian F_1 between approximately 0.83 and 0.99 while retaining reconstruction R^2 between approximately 0.90 and 0.96. The planted ground-truth feature basis can therefore coexist with high-quality sparse reconstruction in these architectures. In addition to the direct feature loss, it initializes the supervised encoder and decoder directions from the planting probes, disables decoder normalization, and reduces the reanimation coefficient from 3×10^{-4} to 10^{-4}. The block-2 JumpReLU model also lowers its initial log-threshold from 1.2862 to 0.16 so that the probe-initialized latents can fire. The purpose of this experiment is to establish that the featurizer expressivity is not the main bottleneck.

C JACOBIAN PRESSURE ON THE FEATURIZERS

In this section, we describe how we obtain downstream gradients and use them to train a featurizer through local coverage. We also give details on the additional losses and training settings used for the synthetic setup in Section 4. In the vision experiments, local coverage is the only Jacobian term added to the featurizer objective.

C.1 GRADIENT CONSTRUCTION AND BACKCHAINING

We train the featurizers in reverse order, starting at block 4. At this block, the downstream scores are the 1,024 logits of the block-5 probe head. For each input, we use the gradients of those logits whose ground-truth features are present. We then freeze the block-4 featurizer and use it to supply downstream scores for block 3. Here, a downstream feature is present when its sparse activation is positive, $z_c^{(4)} > 0$, and the score we differentiate is its encoder preactivation $u_c^{(4)}$. We repeat this procedure after freezing the block-3 featurizer to train block 2. The gradient is computed through the actual transformer block between the upstream activation and the downstream score. As in Equation 6, we add the same perturbation to all eight upstream token states and differentiate with respect to that shared perturbation. The resulting gradient is the sum of the eight token gradients and has dimension 128. The perturbation acts before RMS normalization and pooling, whereas the SAE reconstructs the normalized, pooled state defined in Equation 10. The transformer, probe head, and any downstream featurizer remain frozen while the upstream featurizer is getting trained. Ground-truth features at blocks 2–4 are used for selecting the featurizer width and sparsity, but do not enter the featurizer losses.

C.2 NOTATION

We list the notations used in this appendix here. In the training losses, b is the block being featurized, c indexes a downstream feature, and j indexes an atom in the upstream dictionary. We omit the block superscript when the block is clear from context, writing, for example, $\mathbf{g}_c$ for $\mathbf{g}_c^{(b)}$.

Notations being used from the previous sections.

Symbol	Meaning
$\mathbf{x}$, $\mathbf{x}_i$	An input; i indexes an example in a minibatch.
d	Transformer hidden dimension, with $d = 128$ in the synthetic setup.
$\mathbf{h}^{(b)}$, $\mathbf{h}_t^{(b)}$	The pooled activation and the raw state of token t at block b, respectively.
E_b, D_b	Encoder and decoder matrices of the block-b SAE.
$u_j^{(b)}$, $z_j^{(b)}$	Encoder preactivation and sparse activation of atom j, respectively. The sparse activation is obtained after ReLU and TopK.
k, $\mathcal{A}^{(b)}(\mathbf{x})$	The TopK budget and the set of k selected atom indices for input $\mathbf{x}$.
s_c^{down}, $\mathbf{g}_c^{(b)}$	The downstream score and its gradient with respect to a shared perturbation of the upstream token states (Equation 6).
$\mathbf{q}_j^{(b)}$	A unit-normalized dictionary direction: either encoder row $E_b[j,:]$ or decoder column $D_b[:,j]$, divided by its norm.
$\mathcal{L}_{\text{rec}}$, $\mathcal{L}_{\text{cov}}$	Mean-squared reconstruction error and local Jacobian coverage loss (Equation 3 and Equation 7).

Notation introduced in this appendix. The first two rows describe the dictionary and its alignment with a downstream gradient. The remaining rows collect the notation for the additional losses and evaluation. Their definitions and use are explained in the corresponding paragraphs below.

Symbol	Meaning
M	Total dictionary width.
α_{xcj}	Absolute cosine alignment of downstream gradient c with atom j on input $\mathbf{x}$: $\alpha_{xcj} = \lvert\langle \mathbf{g}_c(\mathbf{x})/\lVert\mathbf{g}_c(\mathbf{x})\rVert_2, \mathbf{q}_j\rangle\rvert$.
$\widetilde{\mathbf{g}}_c$, $\mathbf{\Sigma}_c$	Rescaled gradient and its uncentered second moment across inputs (Equation 16).
$\mathbf{v}_{cr}$, ν_{cr}, w_{cr}	The rth retained eigenvector of $\mathbf{\Sigma}_c$, its eigenvalue, and its loss weight. We retain six directions, so $r = 1, \ldots, 6$.
$\mathcal{B}_c$	Sampled active inputs for downstream feature c. This collection can include repeated examples.
A_{cj}^{local}, A_{cj}^{global}	Total alignment assigned to atom j for downstream feature c, accumulated over sampled active inputs or population gradient directions, respectively.
p_{cj}	The alignment distribution over dictionary atoms: $p_{cj} = A_{cj}/\sum_{j'=1}^{M} A_{cj'}$, using either version of A_{cj}.
$\mathcal{L}_{\text{leak}}$	Local leakage loss.
$\mathcal{L}_{\text{cov,global}}$	Population-level coverage loss.
$\mathcal{L}_{\text{conc,local}}$	Concentration loss using alignment accumulated over sampled active inputs.
$\mathcal{L}_{\text{conc,global}}$	Concentration loss using alignment accumulated over population gradient directions.
$\mathcal{D}$, $\mathcal{R}$	Atoms inactive throughout the current reconstruction minibatch and their reanimation loss.
ϵ	A small positive constant used to prevent division by zero.
W_{jc}^{dense}, η	Dense probe weight from latent j to planted target c, and the requested fraction of latents retained by a sparse probe.

We use $\mathbb{E}$ for averaging, $\mathbf{1}[\cdot]$ for an indicator, and $|\mathcal{D}|$ for the number of atoms in a set. The alignment distribution p_{cj} is different from the SAE's post-ReLU activations $\mathbf{p}^{(b)}$ in Equation 11.

C.3 Local Jacobian coverage

We use the ReLU TopK SAE from Equation 11. Table 4 gives the dictionary sizes at each block. For a present downstream feature c on input $\mathbf{x}$, we ask whether at least one selected atom aligns with its gradient. Using the absolute cosine alignment α_{xcj} defined above, the local coverage loss is

$$\mathcal{L}_{\text{cov}} = \mathbb{E}_{\mathbf{x},\, c \text{ present}} \left[1 - \max_{j \in \mathcal{A}(\mathbf{x})} \alpha_{xcj} \right]. \tag{14}$$

This is Equation 7. We use the TopK index set directly when evaluating the loss.[1] When both encoder and decoder directions are used, we compute the loss separately for each and average the two losses. The choice at each block is given in Table 4. Decoder columns are normalized after every optimizer step. Before training each featurizer, we cache the upstream activations and, for each input, the gradients of its active downstream features. These stored activations and gradients form our gradient cache. The activations are used for reconstruction and to determine the TopK-selected atoms for the local losses. We do this for all blocks. At each training step, we first select 128 different downstream features uniformly from those with active examples in the cache. For each selected feature, we sample eight cached inputs on which it is active, allowing repeated inputs. This gives common and rare features equal sampling weight.

C.4 Additional training choices for the synthetic setup

The toy setup also uses losses that encourage the selected atoms to be aligned to the gradient directions across the population and concentrate this alignment on fewer atoms. The full training loss is the weighted sum of reconstruction error, local coverage, and these additional Jacobian terms, and reanimation. Table 3 gives all coefficients at each block.

Local leakage. Local coverage asks for one aligned atom in the selected set, TopK $\mathcal{A}(\mathbf{x})$. We also penalize alignment carried by atoms outside that set. Using the same α_{xcj} as above, this loss is

$$\mathcal{L}_{\text{leak}} = \mathbb{E}_{\mathbf{x},c} \left[1 - \frac{\sum_{j \in \mathcal{A}(\mathbf{x})} \alpha_{xcj}^2}{\max\left(\sum_{j=1}^{M} \alpha_{xcj}^2, \epsilon \right)} \right], \tag{15}$$

Population-level coverage. To capture gradient directions that are strongest across inputs, we compute each downstream feature's gradients on 200,000 inputs, including inputs where that feature is inactive. For each feature, we rescale its gradients to have mean norm $\sqrt{d}$ and compute their uncentered second moment:

$$\widetilde{\mathbf{g}}_c(\mathbf{x}) = \frac{\sqrt{d}\, \mathbf{g}_c(\mathbf{x})}{\mathbb{E}_{\mathbf{x}} \|\mathbf{g}_c(\mathbf{x})\|_2}, \qquad \mathbf{\Sigma}_c = \mathbb{E}_{\mathbf{x}} \left[\widetilde{\mathbf{g}}_c(\mathbf{x}) \widetilde{\mathbf{g}}_c(\mathbf{x})^\top \right]. \tag{16}$$

We retain the six leading eigenvectors $\mathbf{v}_{cr}$ and their eigenvalues ν_{cr}. Each of these directions should align with an atom somewhere in the dictionary, without restricting the choice to an input's TopK set. We therefore minimize

$$\mathcal{L}_{\text{cov,global}} = -\frac{\sum_{c,r} w_{cr} \max_{1 \le j \le M} |\langle \mathbf{q}_j, \mathbf{v}_{cr} \rangle|}{\sum_{c,r} w_{cr}}. \tag{17}$$

Larger eigenvalues receive more weight: we use $w_{cr} = \nu_{cr}$ at block 2 and $w_{cr} = \sqrt{\nu_{cr}}$ at blocks 3 and 4. Like we do for local coverage, encoder and decoder losses are averaged when both are used.

Sparse Concentration. We also want each downstream feature's alignment to be concentrated on a small set of atoms across inputs. For each feature c, we first add up its alignment with each atom j. The local version sums over the sampled active inputs $\mathcal{B}_c$, counting an atom only when it is selected

[1] TopK is applied after ReLU. If fewer than k preactivations are positive, the selected set can include zero-valued entries; we do not remove these entries for the local losses. The selected indices are treated as fixed when differentiating the losses.

by TopK. The population version sums over the six gradient directions:

$$A_{cj}^{\text{local}} = \sum_{\mathbf{x}\in\mathcal{B}_c} \mathbf{1}[j \in \mathcal{A}(\mathbf{x})]\alpha_{xcj},$$
$$A_{cj}^{\text{global}} = \sum_{r=1}^{6} w_{cr}|\langle \mathbf{q}_j, \mathbf{v}_{cr}\rangle|, \qquad j = 1, \ldots, M. \tag{18}$$

For either version, we divide each atom's total by the sum over atoms, giving $p_{cj} = A_{cj}/\sum_{j'=1}^{M} A_{cj'}$. This is the fraction of feature c's total alignment carried by atom j. To encourage most of the alignment to be carried by a few atoms, we minimize the entropy of this distribution:

$$\mathcal{L}_{\text{conc}} = \mathbb{E}_c\left[-\frac{1}{\log M}\sum_{j=1}^{M} p_{cj}\log p_{cj}\right]. \tag{19}$$

The normalized entropy is zero when one atom carries all the alignment and one when it is spread equally over all M atoms. Using the local alignment totals gives $\mathcal{L}_{\text{conc,local}}$, while using the population totals gives $\mathcal{L}_{\text{conc,global}}$. Both use the same encoder or decoder choices as their corresponding coverage losses. Table 3 gives their coefficients; only block 4 uses global concentration.

Reconstruction and reanimation. Reconstruction uses all dictionary atoms and minimizes the mean-squared error between the pooled activation and its reconstruction. To encourage inactive atoms to be used, we also push up the encoder preactivations of atoms that are zero throughout the reconstruction minibatch. Let $\mathcal{D}$ be this set of inactive atoms. The reanimation term is

$$\mathcal{R} = -\frac{1}{|\mathcal{D}|}\sum_{j\in\mathcal{D}}\sum_{i} u_j(\mathbf{x}_i), \tag{20}$$

and is zero when $\mathcal{D}$ is empty. We sum over minibatch examples and average over inactive atoms. Using preactivations allows this term to act even when ReLU outputs are zero.

Table 3: **Loss coefficients for the synthetic setup.** The training objective is the weighted sum of the listed terms. A zero means that the term is not used at that block.

Loss	Block 2	Block 3	Block 4
Reconstruction, $\mathcal{L}_{\text{rec}}$	3×10^{-5}	3×10^{-5}	10^{-4}
Local coverage, $\mathcal{L}_{\text{cov}}$	20	10	15
Local leakage, $\mathcal{L}_{\text{leak}}$	0.5	0	0.5
Local concentration, $\mathcal{L}_{\text{conc,local}}$	0	0.3	0.3
Population coverage, $\mathcal{L}_{\text{cov,global}}$	10	10	15
Population concentration, $\mathcal{L}_{\text{conc,global}}$	0	0	1
Reanimation of atoms receiving Jacobian pressure	3×10^{-5}	3×10^{-5}	3×10^{-5}
Reanimation of free atoms	10^{-8}	10^{-8}	10^{-8}

C.5 Training, controls, and evaluation

Training settings.

We randomly initialize each featurizer and train it for 25,000 Adam steps at learning rate 3×10^{-4}. Reconstruction minibatches contain 512 examples, and gradients are clipped at norm 1. The reconstruction cache contains 500,000 examples; the population-gradient cache contains 200,000 examples. Evaluation uses 10,000 examples. The controls match the corresponding Jacobian model's dictionary size, TopK budget, activation rule, initialization seed, optimizer, and number of training steps. They set all five Jacobian loss coefficients to zero, use reconstruction coefficient 1, and use reanimation coefficient 3×10^{-4} for all atoms. This compares the two full training objectives, including their reconstruction and reanimation weights. The controls are not selected to match the Jacobian models' reconstruction R^2.

In the toy setup, only a smaller subset of the dictionary atoms receive the Jacobian pressure, details in Table 4. The remaining atoms remain free from this Jacobian pressure but all atoms, including these

free atoms, share the same TopK gate and contribute to reconstruction. We compute reanimation separately for these two groups. We leave this split out of the equations above to keep the notation simple.

Removing Jacobian pressure during training. To test whether continued Jacobian pressure is needed to maintain feature identifiability, we repeat block-4 training with the pressure removed at steps $0, 2000, 4000, \ldots, 24000$, together with a run where it is never removed. From the chosen step onward, we set all five Jacobian loss coefficients to zero. Reconstruction continues with coefficient 10^{-4}, using the existing parameters and optimizer state. These weights differ from those of the reconstruction-only control above, including when pressure is removed from the start. Each switch condition is trained separately using the same initialization seed. Figure 5B reports these individual trajectories.

Target-specific sparse probes. To measure how many latents are needed to decode each planted feature, we fit probes to the complete code. For each model and block, we encode 20,000 training examples and 10,000 evaluation examples, then standardize each latent using its training-set statistics. We first fit a dense logistic probe to all the assigned ground-truth features at that block for 500 AdamW steps, with batch size 1,024, learning rate 3×10^{-3}, unweighted binary cross-entropy, and an explicit 10^{-4} mean-squared weight penalty. For each target c, we rank latents by $|W_{jc}^{\text{dense}}| \, \text{std}(z_j)$, with the standard deviation measured after standardization. If the dictionary has M atoms, a probe at proportion η retains the $\max(1, \text{round}(\eta M))$ highest-ranked latents for that target. We use

$$\eta \in \{0.001, 0.005, 0.01, 0.025, 0.05, 0.1, 0.2, 0.4, 0.8\}.$$

We initialize the masked probe from the dense probe and refit it for 500 full-batch Adam steps at learning rate 10^{-2} with prevalence-weighted binary cross-entropy. Predictions use a sigmoid threshold of 0.5, and F_1 is averaged over ground-truth features. At $\eta = 0.001$, every model in Figure 5C uses one latent per target. Different targets can select the same latent, so this measures compact decodability while allowing reuse of dictionary coordinates; Hungarian F_1 requires one-to-one matching.

Table 4: **Dictionary sizes and direction choices for the synthetic experiments.** Both parts of the dictionary share one TopK gate. "Both" means that encoder and decoder losses are averaged. Concentration terms use the direction choices of the corresponding coverage term.

Setting	Block 2	Block 3	Block 4
Atoms receiving Jacobian pressure	256	512	1024
Free atoms	100	100	200
Total atoms	356	612	1224
TopK budget	68	68	110
Directions for local terms	encoder	encoder	both
Directions for population terms	encoder	both	both

D DINOV3 FEATURIZERS WITH JACOBIAN PRESSURE

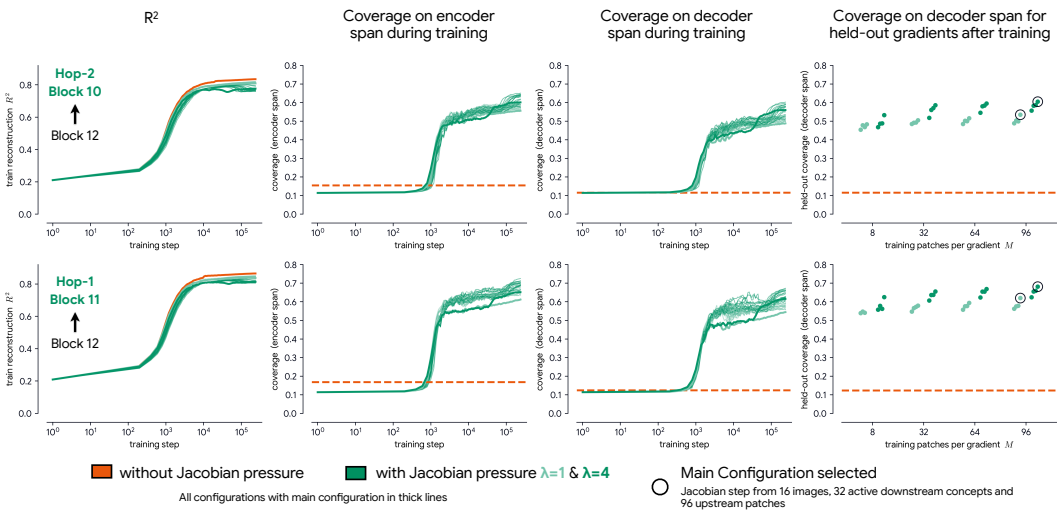

Figure 17: **Reconstruction and coverage across training configurations.** Rows show block 10 (hop 2, top) and block 11 (hop 1, bottom), both supervised by the frozen block-12 BSF. The first three columns show reconstruction R^2 and encoder and decoder coverage during training; the last shows held-out decoder coverage. Green denotes Jacobian pressure with coverage coefficients 1 and 4; orange denotes the regular BSF. Thick lines and circled points mark the setting with 16 images, 32 downstream concepts per image, and 96 upstream patches per image and concept. Dashed orange lines show coverage measured after training.

We evaluate 32 training configurations at each block, varying the coverage coefficient (1 or 4), images per Jacobian step (8 or 16), downstream concepts per image (16 or 32), and upstream patches per image and concept (8, 32, 64, or 96). In the rightmost column, all featurizers at a given block are evaluated on the same held-out gradients using 96 patches per image and concept; the horizontal axis gives the number of patches used during training. Across both blocks, Jacobian pressure improves coverage with lower reconstruction R^2, and using more patches during training generally improves held-out coverage.

D.1 Featurizing Object Concepts

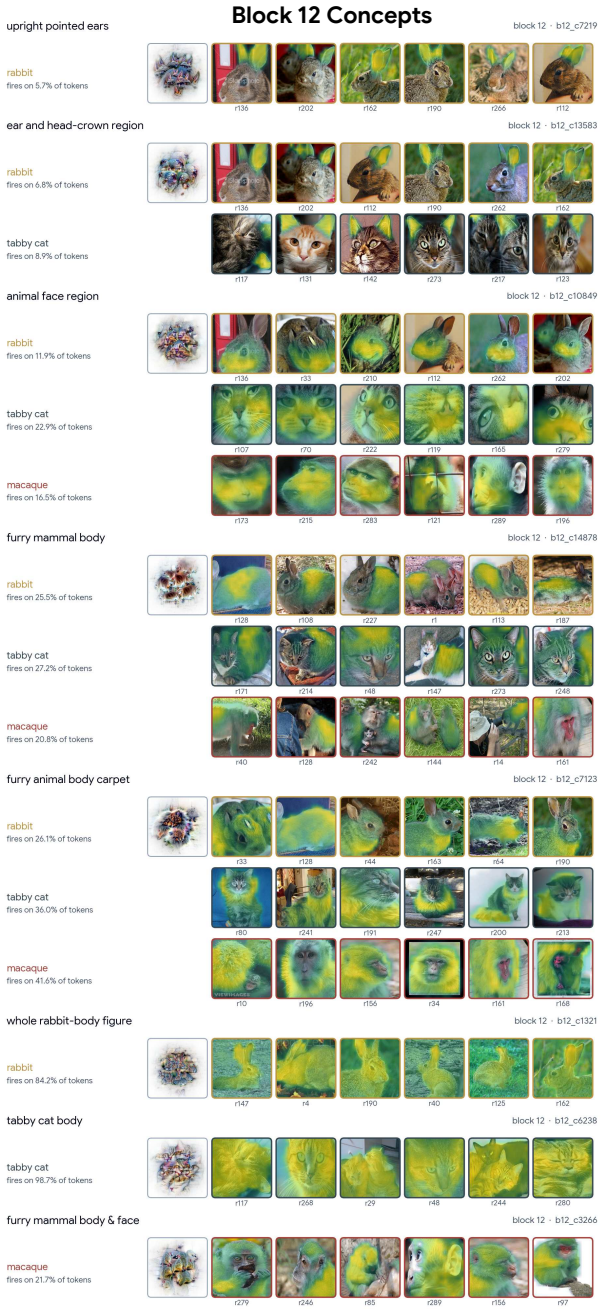

Figure 18: **Concepts at block 12 of DINOv3.** Feature visualizations and activation overlays for selected concept groups from the reconstruction-trained Block Sparse Featurizer. Examples include generic parts shared across rabbits, tabby cats, and macaques, as well as whole-object concepts. Percentages give each group's token firing rate within the indicated class.

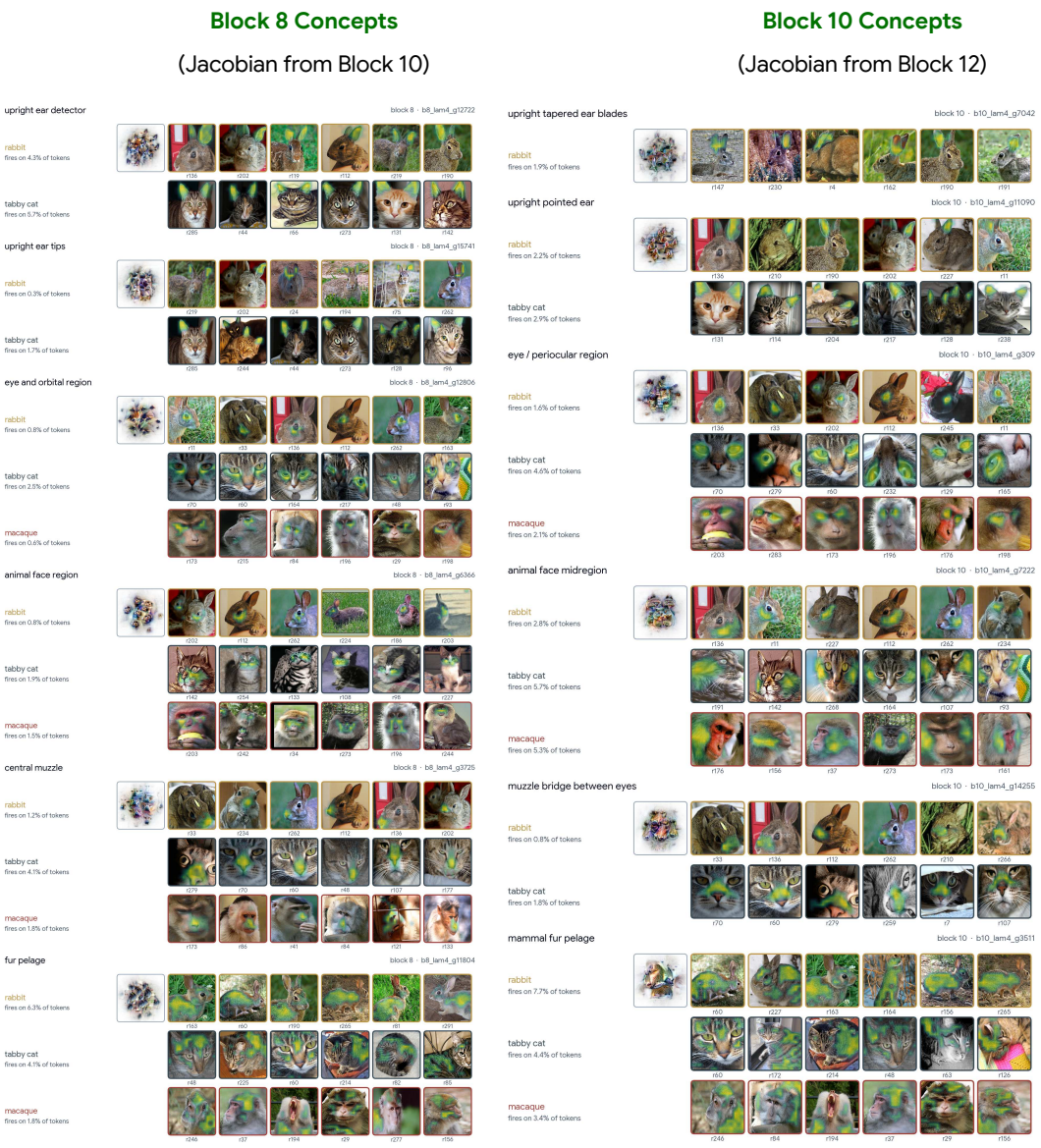

Figure 19: **Part concepts recovered with Jacobian pressure.** Left: Concept groups from the block-8 featurizer, trained with gradients from the block-10 featurizer that was itself trained with Jacobian pressure. Right: Concept groups from the block-10 featurizer, trained with gradients from the block-12 featurizer. Feature visualizations and activation overlays show responses to ears, eyes, muzzle regions, and fur, with several groups responding to corresponding parts across animal classes. Percentages give each group's token firing rate within the indicated class.



Figure 20: **Computational links shared across animal classes.** Rows show selected concept pairs from blocks 8 → 10 and 10 → 12. Numbers give link strengths computed from gradient alignment across patches and images within each class and the circle areas are proportional to these scores.

D.2 Part-to-Whole Connections for a Rabbit Concept

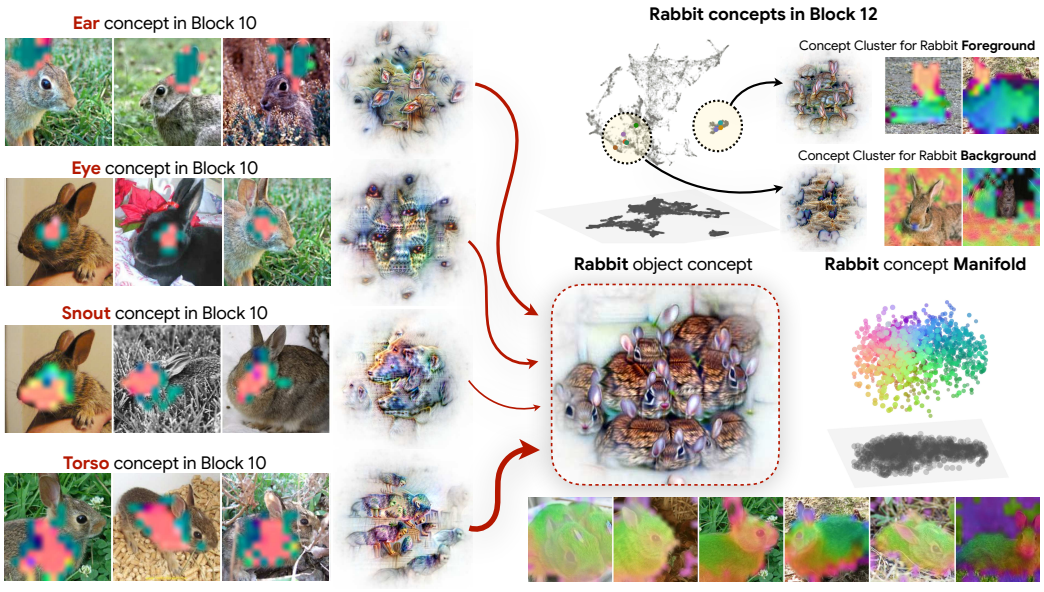

Figure 21: **Featurizing computation through compositionality.** Features for a rabbit's ear, eye, snout, and torso at block 10 of DINOv3 combine with different strengths into a rabbit-object feature at block 12. The upper-right plot shows clusters of downstream concepts associated with rabbit foreground and background; the lower-right plot shows the rabbit-object feature's activation manifold.

D.3 CONCEPT TO CONCEPT STEERING

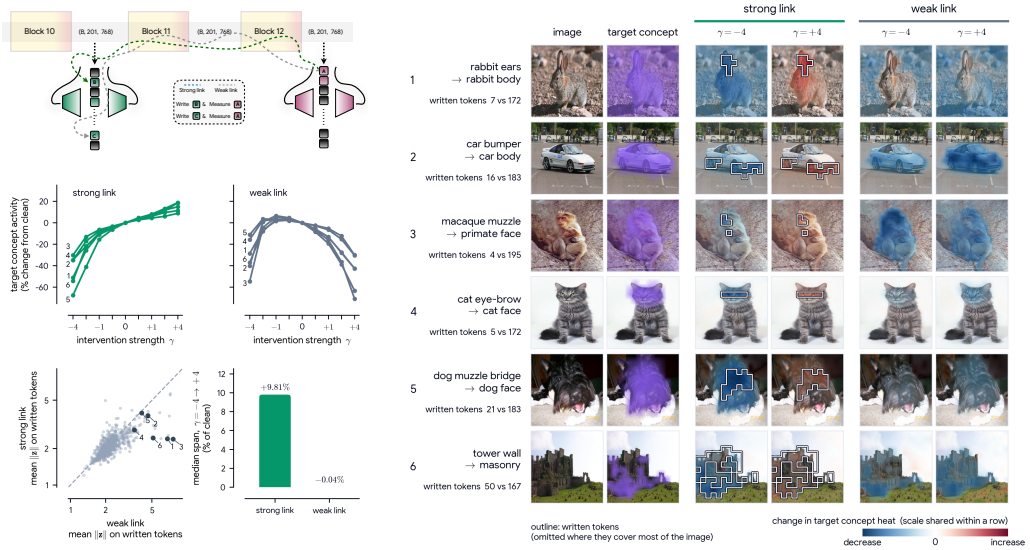

Figure 22: **Concept-to-concept steering in DINOv3.** Top left: Intervention schematic. Curves show percentage changes in the block-12 target response for six selected examples and their controls. Right: Purple overlays show the target's original activation. Red and blue overlays show increases and decreases in the target concept's activation at each patch after intervention at $\gamma = \pm 4$. Bottom left: Mean code norms on intervened patches for linked and control groups, shown for 668 pairs (light points) and the six selected examples (numbered points). The adjacent bars show the median response span across the 668 pairs, defined as the difference between the image-averaged percentage changes at $\gamma = +4$ and $\gamma = -4$.

We test whether intervening on a block-10 concept changes the response of a connected block-12 concept. At each patch, a BSF group's decoded contribution is the sum of its three decoder directions, weighted by the corresponding latent activations. This is the part of the reconstruction term $D_b \mathbf{z}^{(b)}$ supplied by that group. We add γ times this contribution, converted to the original activation scale, to the block-10 activations at patches where the group is TopK-selected. These decoded contributions and intervention patches are computed from the original image and held fixed across intervention strengths. We then run the remaining frozen transformer blocks and measure the block-12 target using its group norms before TopK, summed over the patches where it was originally TopK-selected. We report percentage changes from the original target response. Controls are other block-10 groups that are also TopK-active on the same images, tested against the same target using their own decoded contributions and active patches. In the 668-pair comparison, controls are selected from groups with no recorded strong links to the target and matched as closely as possible in mean code norm. In the six illustrated examples, the control groups have larger mean code norms than the linked groups. For these selected examples, adding a linked group's contribution increases the downstream response and subtracting it reduces the response, while control interventions end up reducing the response in both directions. Across the 668 pairs, the median response span between $\gamma = -4$ and $\gamma = +4$ is 9.81% for linked groups and -0.04% for controls.

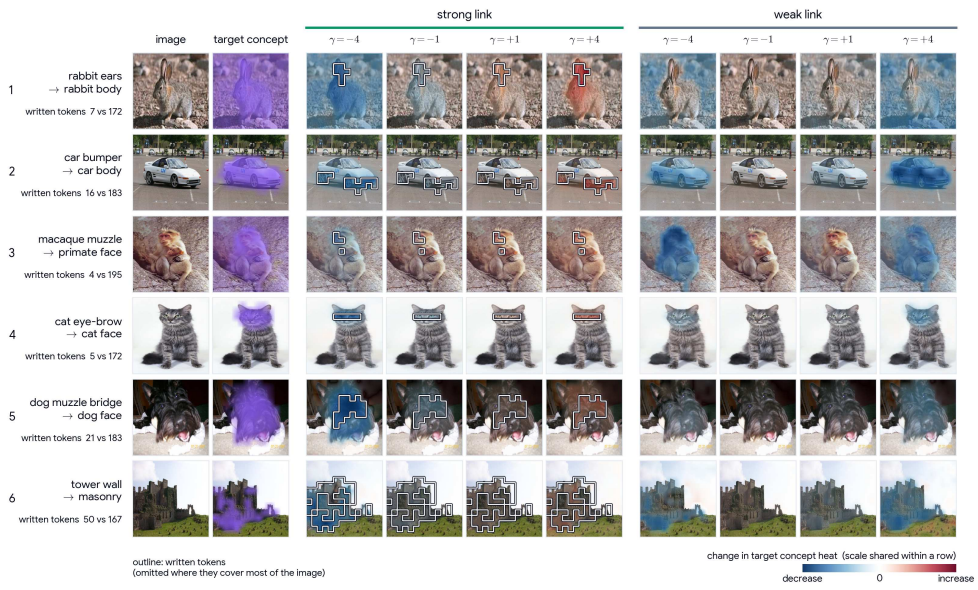

Figure 23: **Concept-to-concept steering across intervention strengths.** Rows show the six selected examples from Figure 22 at $\gamma = -4, -1, +1, +4$. Purple overlays show the target's original activation. Red and blue overlays show increases and decreases in the target concept's activation at each patch after intervention. Color scales are shared within each row. White outlines mark intervened patches.

D.4 STEERING HORSE PREDICTIONS TOWARD ZEBRA

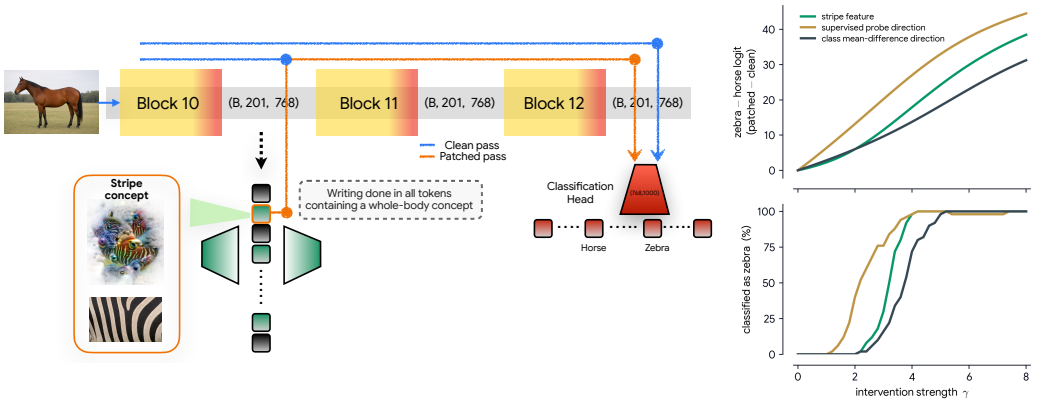

Figure 24: **Horse-to-zebra steering and baseline comparisons.** Left: we add a stripe-feature direction to block-10 activations at patches where a horse-body feature is TopK-selected, then run blocks 11 and 12. Right: mean change in the zebra–horse logit gap relative to the unmodified activations (top) and percentage of 50 validation horse images classified as zebra (bottom). All directions use the same patches and matched norms, and predictions come from the same frozen ImageNet linear probe.

We use the block-10 featurizer trained with coverage coefficient 4. Using zebra and horse images, we first shortlist block-10 concept groups that are frequently TopK-selected on zebras but rarely on horses. We then identify a stripe group from these candidates using its activation overlays and agreement between two independent automated feature descriptions. We collect the stripe group's three-dimensional codes from zebra patches where it is TopK-selected. Each code is three-dimensional because it contains one activation value for each of the group's three latents.

Next, we use SVD to find the dominant activation pattern across these three latents, retaining one direction in their three-dimensional code space. We flip this direction if necessary so that it points toward the average stripe code, then scale it to the median code norm. This gives one representative code whose three values describe how the group's latents activate at a typical overall strength. We then use these three values to weight the group's three decoder directions and sum them, producing one representative vector in the 768-dimensional activation space. We rescale this decoded representative vector back to the original block-10 activation scale and then add γ times it at patches where the horse-body group is TopK-selected. The same vector is used across all validation horse images.

We compare the stripe vector with two baselines: the difference between mean zebra and horse patch activations at block 10, and the weight direction of a binary linear probe trained to distinguish these classes from image-averaged block-10 patch activations. This binary probe provides an intervention direction while the predictions are measured by a separate frozen ImageNet linear probe at block 12. With matched norms and patches, the stripe direction first reaches at least 50% zebra predictions at intervention strength $\gamma = 3.2$, compared with 3.8 for the class mean difference and 2.2 for the supervised probe direction. The stripe intervention classifies all 50 horse images as zebra from $\gamma = 4.2$ onward over the tested range.