EXECUTABLE ANNOTATIONS OF SAE FEATURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse autoencoder (SAE) features are used in practice through their descriptions. While features fire at a prevalence of 10^{-4}, LLM descriptions are evaluated at 1:1. Rescored at natural prevalence, descriptions collapse to 0.01–0.05 F_1 and describe concepts that are 40–200$\times$ more prevalent than the feature.

In this work, we show that annotating a feature is equivalent to learning a Boolean function over the one-hot encoding of a token and its context. We annotate every feature of three GemmaScope dictionaries by learning this function using a monotone linear threshold function (LTF) over (offset, token) literals, and evaluate it directly as a per-token detector on held-out text. These annotations outscore LLM descriptions on the field's own benchmarks, and score $\sim$5–24$\times$ higher at natural prevalence. Scored at this scale, the dictionary resolves into strata that balanced evaluation cannot distinguish, and the LTF weights expose multilingual concepts, boilerplate templates, and superposition directly. Applied without modification to a protein language model, the same method recovers established structural motifs such as zinc-finger spacing, Kelch repeats, and P-loop motifs from sequence-only representations. 147k features across the three Gemma dictionaries are annotated and evaluated within 7 hours on a single quad-GH200 node, with no LLM in the pipeline. All annotations and code will be released publicly.

1 INTRODUCTION

How often does text mention the 'Golden Gate Bridge'? Qualitatively, we know that the phrase is rarer than the word 'bridge', since not every bridge is the Golden Gate. Quantitatively, a grep of the phrase answers the question. But what about the prevalence of the *concept* of the Golden Gate Bridge? Perhaps it includes the Presidio, and everything within five miles of the span; perhaps the fog, and the paint. "We drove across the bridge to Napa" might carry the concept, but only for someone who lives west of the Bay. To answer the question, every token within its context needs to be labeled yes or no. The way this boundary is drawn is intrinsically subjective.

Sparse autoencoders (SAEs) decompose a large language model's (LLM) activations into features, each interpreted as a concept (Bricken et al., 2023; Templeton et al., 2024; Gao et al., 2025). If features do capture concepts, those concepts, unlike the ones judged by humans, have precise boundaries: LLMs and SAEs are deterministic, and whether a feature fires is determined entirely by the tokens in context. In most applications, a feature is used through an LLM-written description, scored by an LLM judge on held-out text (Bills et al., 2023; Huang et al., 2023; Paulo et al., 2025; Karvonen et al., 2025). For the description to be accurate, the concept it names should be as rare as the feature, on the order of 10^{-4}.

Here we directly consider determinism and the natural prevalence of a feature as guiding principles, and annotate it as a monotone linear threshold function (LTF) over a token and its context. We evaluate descriptions and LTFs on held-out text at both standard and natural prevalence, and find that descriptions capture concepts much broader than the features they describe. Annotating every feature of three GemmaScope dictionaries, we find that most features are conjunctive. The weights of the LTF are legible, and encode multilingual concepts, phrasal grammar, boilerplate templates, register, and superposition. Applied to a protein language model with residue letters in place of tokens (Lin et al., 2023; Simon & Zou, 2025), the same fit yields 28 features that each carry the same annotation shifted by one residue and, when assembled, recover the known signature of a protein motif.

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.

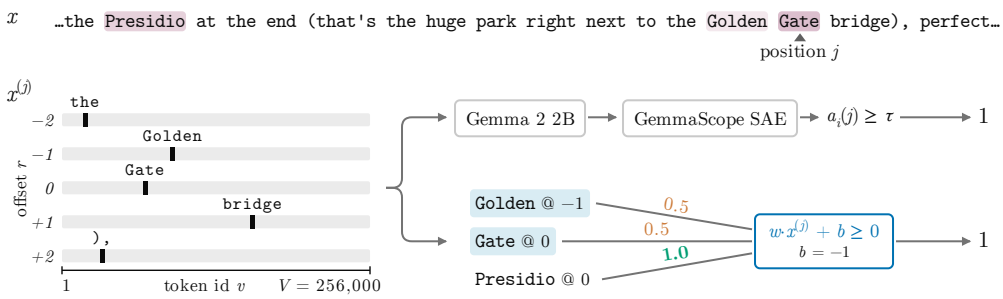


Figure 1: A visualization of the annotation of an abstract concept. The LTF takes the contextualized one-hot representation $x^{(j)}$ as input and classifies whether an SAE feature fires.

2 What features compute

2.1 Features are Boolean functions

A language model is a deterministic function of its input tokens, and so is the SAE attached to it. Whether a feature fires on a given token is entirely determined by the tokens in context. The input is a sequence of L tokens from a vocabulary of size V, represented as a one-hot matrix $X \in \{0,1\}^{L \times V}$. A SAE with dictionary size D trained on the model's residual stream decomposes each token's representation into features. Feature i has activation $a_i(j) \geq 0$ at position j. The contextualized one-hot representation at position j is the window of X centered at j with half-width R, written $x^{(j)} \in \{0,1\}^{(2R+1) \times V}$, where $x^{(j)}_{r,v} = 1$ if the token at position $j + r$ is v.

Since the model and the SAE are deterministic, the activation $a_i(j)$ is fully determined by $x^{(j)}$. [1] We threshold at a level τ to obtain a binary label $y_i(j) = \mathbf{1}[a_i(j) \geq \tau]$. This is a Boolean function $f_i : \{0,1\}^{(2R+1) \times V} \to \{0,1\}$, and annotating a feature means describing f_i.

Consider a feature that detects mentions of the Golden Gate Bridge (Templeton et al., 2024). It fires when 'Golden' appears at offset -1 and 'Gate' at offset 0, or when 'Presidio' appears at offset 0. Each entry $x^{(j)}_{r,v}$ is a literal, and the description is an OR of ANDs over those literals. This structure is called a disjunctive normal form (DNF), and every Boolean function has one (O'Donnell, 2014).

Finding the shortest DNF from examples is NP-hard (Pitt & Valiant, 1988; Feldman, 2009), and in our setting the number of literals $d = (2R+1)V$ is on the order of 10^8. However, the one-hot structure of $x^{(j)}$ constrains f_i. At each offset exactly one token is present, so negations are redundant and f_i is always a positive DNF. We replace the search with a fit, approximating f_i by a monotone linear threshold function (Muroga, 1971),

$$\hat{y}_j = \mathbf{1}\left[\sum_{r,v} w_{r,v}\, x^{(j)}_{r,v} + b \geq 0\right], \qquad w \geq 0. \tag{1}$$

The non-negativity constraint matches the monotone structure of f_i. Under this model, $w(\texttt{Golden}, -1) = 0.5$, $w(\texttt{Gate}, 0) = 0.5$, $w(\texttt{Presidio}, 0) = 1.0$, and $b = -1.0$ recovers the Golden Gate feature. The pair 'Golden' then 'Gate' scores 0 and fires; 'Presidio' alone also scores 0 and fires; 'Golden' alone scores -0.5 and does not.

Linear threshold functions over sparse literals are the standard model in ad-click prediction (McMahan et al., 2013), text classification (Wang & Manning, 2012), and clinical risk scoring (Ustun & Rudin, 2019). Algorithms for learning DNF pass through the same representation (Klivans & Servedio, 2004). Monotonicity is inherited from the positive DNF structure.

[1] R spans the full context in our experiments, so $x^{(j)}$ contains every token the models condition on. For the autoregressive model, $a_i(j)$ depends only on offsets $r \leq 0$; literals at $r > 0$ that receive weight are correlates of the firing.

Table 1: Judge scores for the LLM description and the classifier, 1000 features per SAE. Detection is scored on balanced accuracy and SAEBench on accuracy, over the features both methods scored. Standard errors are below 0.005. †As published by Karvonen et al. (2025), on their own random sample of 1000 features with their judge.

SAE	Detection			SAEBench		
	Haiku 4.5 $66.72	Opus 5 $459.48	Classifier —	Published† —	Haiku 4.5 $2.69	Classifier —
L19, 65k (L_0=63)	0.831	0.851	**0.902**	0.870	0.886	**0.922**
L19, 16k (L_0=73)	0.849	0.875	**0.893**	0.846	0.854	**0.892**
L12, 65k (L_0=72)	0.820	0.841	**0.880**	0.842	0.862	**0.900**

2.2 An annotation is a classifier

In this work, we fit (w, b) by minimizing class-weighted logistic loss with an ℓ_1 penalty on the weights,

$$\mathcal{L}(w, b) = -\sum_j [c_i\, y_i(j) \log \sigma(s_j) + (1 - y_i(j)) \log(1 - \sigma(s_j))] + \lambda \sum_{r,v} w_{r,v}, \tag{2}$$

where $s_j = \sum_{r,v} w_{r,v}\, x_{r,v}^{(j)} + b$, $\mathcal{P}_i$ and $\mathcal{N}_i$ are the sets of positive and negative tokens, and $c_i = |\mathcal{N}_i|/|\mathcal{P}_i|$ balances the classes. Under $w \geq 0$ the ℓ_1 penalty reduces to a linear term, making the full objective smooth and convex (Appendix A.1).

Each solution along the regularization path is scored on a held-out tune split at natural prevalence. Class weighting enters a well-specified logistic model only through the intercept (Appendix A.2), so the fitted w is independent of the feature's rarity. The decision threshold is chosen separately: $F_1(\theta)$ is piecewise constant, and a single sort of the tune scores finds the exact optimum (Appendix A.2).

The output is the triple (w, b, θ). Under the ℓ_1 penalty w is sparse, and the annotation reads as a ranked list of (offset, token, weight) entries. This setup is similar in spirit to Neuron to Graph (Foote et al., 2023), which builds a token-pattern graph for each neuron and scores it by predicting its activations, and to compositional explanations of neurons (Mu & Andreas, 2020).

2.3 Fitting at scale

The objective is convex, but it has $\sim 10^8$ variables. The optimality conditions at $w = 0$ yield a closed-form enrichment test per literal (Appendix A.3). A literal survives only if it is enriched among positives, with λ setting the bar. We screen further with a Bonferroni-corrected Poisson test (Appendix A.3). For a typical feature, screening cuts the number of literals from $\sim 10^8$ to $\sim 10^3$.

We solve on the surviving literals with L-BFGS-B over a geometric grid of 75 λ values, warm-starting each fit from the previous solution (Appendix A.4). Each solve runs on a strong-rule candidate set (Tibshirani et al., 2012), and we verify the optimality conditions across all survivors, adding any violator and re-solving until none remain.

3 Evaluating annotations

3.1 At evaluation prevalence

We use two published protocols: detection (Paulo et al., 2025) and the SAEBench autointerp score (Karvonen et al., 2025). In both, an explainer LLM reads excerpts where the feature fires and writes a description; an LLM judge then labels fresh excerpts against it. Detection is scored on balanced accuracy over excerpts drawn 1:1 from activating and non-activating documents; SAEBench on accuracy over excerpts drawn 4:10. Excerpt counts and the full construction are pinned in Appendix C.1.

We take three SAEs from GemmaScope (Lieberum et al., 2024), trained on the residual stream of Gemma 2 2B: dictionary sizes 16k and 65k at layer 19, and 65k at layer 12. From each, we draw 1000

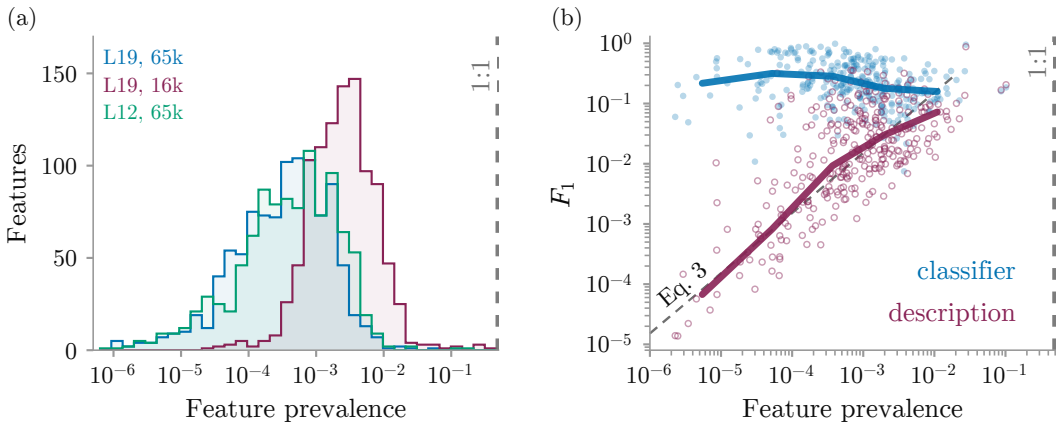


Figure 2: Prevalence and scores at natural prevalence. (a) The distribution of feature prevalence π_i for the 1000 features per SAE on held-out text. The dashed line marks the 1:1 evaluation ratio. (b) Per-feature F_1 vs prevalence, for the 300 features in Section 3.2. The solid lines are binned medians, and the dashed guide is Equation 3 at the descriptions' median TPR and FPR.

features randomly from those that fire at least 200 times in held-out text, matching the sample size of SAEBench. We label a token positive if the feature fires at all, and fit as in Section 2 (Appendix A.0). Unless stated otherwise, descriptions and judgments come from Claude Haiku 4.5.

While the classifier is fit on tokens at natural prevalence, the evaluations score excerpts drawn 1:1 in detection and 4:10 in SAEBench. The window and the ratio skew the class imbalance: an excerpt activates if any of its tokens does, and the drawn ratios are far above natural prevalence. To account for this imbalance, we tune the decision threshold on the tune split at the evaluation's ratio, leaving (w, b) fixed (Appendix A.2). The tuned classifier is then scored on the same held-out excerpts the judge labels.

The average accuracy scores are reported in Table 1. The classifier scores higher than the description on both detection and SAEBench. On detection, it is higher for 69–83% of the features across the three SAEs. On SAEBench, it is higher or equal for 79–80%. We also report the published SAEBench scores on their own set of 1000 features. Compared to their evaluation, our judge scores $\sim$0.01–0.02 higher. With Claude Opus 5 writing and judging the descriptions, the scores rise by 0.02–0.03 and remain below the classifier on all three SAEs.

3.2 At natural prevalence

Both protocols evaluate at a fixed ratio, 1:1 in detection and 4:10 in SAEBench, to judge the description. As shown in Figure 2, the actual firing rate is two to three orders of magnitude rarer than the evaluation ratios. The description corresponds to a concept, and for the description to be accurate, the concept's prevalence should match the feature's.

Consider again the Golden Gate feature of Section 2 and two descriptions of it: "mentions of the Golden Gate Bridge" and "mentions a bridge". At 1:1, random excerpts contain almost no bridges, so the judge never has to tell the two apart, and the two score nearly the same. At natural prevalence, most mentions of a bridge are not the Golden Gate.

To quantify this, we define $\pi_i = |\mathcal{P}_i| \,/\, (|\mathcal{P}_i| + |\mathcal{N}_i|)$ as the prevalence of feature i. At a prevalence of a few in ten thousand, accuracy no longer separates descriptions: labeling every token negative scores $1 - \pi_i$.

Under class imbalance, it is standard to consider F_1 (Davis & Goadrich, 2006) instead. In the limit that the positive class is rare, F_1 takes the form (Appendix B):

$$F_1 \;\approx\; \frac{2\ \mathrm{TPR}}{\mathrm{FPR}}\,\pi_i\ , \tag{3}$$

Table 2: Median token-level scores at natural prevalence, 100 features per SAE. The description's rates are measured on the judged sample, and its F_1 is computed from those rates and the exact counts of firing and non-firing tokens and the classifier is scored on the held-out test split (Appendix C.3).

		LLM Description ($91.08)			Classifier		
SAE	median π_i	TPR	FPR	F_1	TPR	FPR	F_1
L19, 65k (L_0=63)	4.2×10^{-4}	0.80	7.4×10^{-2}	0.0098	0.26	3.2×10^{-4}	**0.24**
L19, 16k (L_0=73)	2.5×10^{-3}	0.53	9.5×10^{-2}	0.046	0.25	1.6×10^{-3}	**0.21**
L12, 65k (L_0=72)	4.5×10^{-4}	0.66	9.3×10^{-2}	0.012	0.21	4.1×10^{-4}	**0.19**

where TPR is the true-positive rate (fraction of firing tokens the description covers) and FPR is the false-positive rate (fraction of non-firing tokens it covers). In Equation 3, TPR and FPR are conditioned on the class, so neither depends on π_i. The false-positive rate of "mentions a bridge" is set by how often text mentions bridges; it does not shrink with the feature's prevalence.[2] As π_i falls, F_1 falls with it.

To measure the F_1 at natural prevalence, we take the descriptions of the features from Section 3.1, and measure the true-positive and false-positive rates. Similar to fuzzing (Paulo et al., 2025), the LLM judge reads an excerpt with one token at the center marked and answers whether the feature fires on the marked token (Appendix C.3). Out of the 1000 features per SAE in the previous section, 100 are drawn at random. For each feature 50 firing tokens and 2000 non-firing tokens are selected from held-out text, with additional samples being drawn for twelve of the features to get an accurate estimate (Appendix C.3). The classifier trained at natural prevalence is also evaluated on the same held-out text.

The median TPR and FPR, along with the F_1 score over the 100 features, are reported in Table 2. The LLM descriptions score a median F_1 of 0.01 for the two 65k SAEs and 0.05 for the 16k SAE. The classifier scores $\sim$16-24 times higher for the two 65k SAEs and $\sim$5 times for the 16k SAE. The FPR illustrates the mismatch between the feature firing rate and the prevalence of a concept described by an LLM. The concepts the descriptions pick out cover $\sim$40–200 times more text than the features fire on. Figure 2 shows the per-feature F_1 against the firing fraction. The description's F_1 rises with the prevalence, tracking Equation 3 at the median rates.

The same comparison at the excerpt level is reported in Appendix D.

4 A census of GemmaScope

We now fit a classifier for every feature of the three SAEs. The label threshold τ is set at 0.15 of each feature's scaled activation, and firing tokens below it are dropped from the population, so weak firings count neither as positives nor negatives. The fits are scored on samples of the 250-million-token test split, and the full results are reported in Appendix E.

Fitting takes $\sim$7 hours on a single quad-GH200 node and annotates 146,553 of the 147,456 features. Consistent with our results on the 300 features from Table 2, the median F_1 is 0.23–0.29 across the SAEs, $\sim$460–1,600 times higher than the baseline $2\pi_i$, the score of labeling every token positive. More than 39,000 features score above an F_1 of 0.5, and more than 114,000 above 0.1.

4.1 Most features are conjunctive

Given the form of the classifier, we can read off whether a feature fires on a single token or requires a phrase. Weights are non-negative, so each token can only add evidence toward the threshold. If one token clears it alone, nothing in the surrounding context can prevent the classifier from firing. We define k^* as the smallest number of literals, taking at most one per offset, that together clear the threshold and compute it through a simple greedy pass over the weights. The distribution of k^* for

[2]This assumes no covariate shift, i.e. the distribution of non-firing tokens is the same at evaluation and natural prevalence. Both are drawn from the same held-out corpus in our experiments.

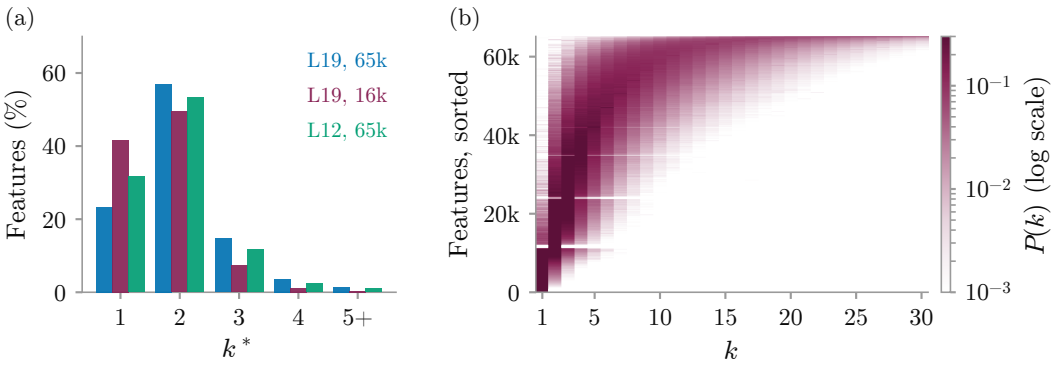


Figure 3: (a) The distribution of k^*, the minimum number of literals needed to cross the threshold, for each SAE. This is a property of the classifier's weights alone. (b) The smallest number of literals needed to cross the threshold at each true positive of the layer-19 65k SAE. Each row is one feature, rows are averaged in blocks of 128 and sorted by mean k, with color showing $P(k)$ on a log scale. White bands are features that always clear at a single k.

all features across the three SAEs is shown in Figure 3(a). Across the three dictionaries, 58–77% of features are conjunctive, requiring more than one literal to fire, with the mode and median of k^* both at 2. We also find that for layer 19, widening the dictionary from 16k to 65k drops the single-literal share from 42% to 23% (Table 4).

The median classifier retains roughly 10^{-6} of the possible literals (Table 4). This is suggestive of phrasal structure, i.e. the classifier selects a small vocabulary of tokens across offsets and requires at least two to co-occur. To verify this, we consider the true positives of the layer-19 65k SAE on held-out text. At each one, both the feature and the classifier fire, so literals in the window must sum above the threshold. We compute the smallest number needed to get there. Note that these sets of literals, or phrases, are sufficient in that the classifier fires whenever they occur, regardless of the surrounding context. Figure 3(b) shows each feature's distribution of k over its true positives. We find that most of these instances are short phrases of a few literals, with 75% requiring $k \leq 3$. The examples range from single tokens such as 'sauce' at $k = 1$, to phrases like 'ice cream' at $k = 2$ and 'Thanks in advance' at $k = 3$, all the way to a run of fives at $k = 11$ (Appendix F).

This structure sets the false-positive rate. For the classifier to mislabel a token, the surrounding text must contain the phrase without the feature firing. A single common token appears everywhere while two specific tokens at a fixed spacing are exceedingly rare ('Golden' is common but 'Golden' followed by 'Gate' is not). The low false-positive rates in Table 2 are consistent with this observation: the phrases the classifier detects are genuinely rare. By Equation 3, an F_1 that does not vanish with π_i requires a false-positive rate of order π_i.

4.2 Case studies

All examples here are from the layer-19 65k dictionary of GemmaScope. We start by considering the simplest example, a feature that detects a token:

f60094 $F_1 = 1.000$: BackColor@0 **2.00**

Here the feature and the F_1 are shown first, followed by the literals and their weights, with ␣ marking a token boundary. Any literal with weight at least one clears the threshold alone, and is colored green.

We find a range of disjunctions:

f23858 $F_1 = 0.998$: ␣that@0 **9.93**, ␣That@0 **5.40**, That@0 **4.73**, that@0 **2.00**

f4400 $F_1 = 0.989$: 9@0 **1.94**, ␣nine@0 **1.01**, 9@−1 0.95, ␣@−2 0.04

f7134 $F_1 = 0.968$: 5@0 **1.93**, ␣five@0 **1.00**, 5@−1 **1.00**

where the feature detects the same word across casings, or the same number as a digit or a word.

Next we find conjunctions, simple phrases such as:

f27607 $F_1 = 0.983$: ␣as@0 0.98, ␣such@−1 0.52, such@−1 0.11, ␣such@−2 0.04 · · · · · ␣such␣as

f54579 $F_1 = 0.981$: ␣of@0 0.98, ␣because@−1 0.55, ␣Because@−1 0.24, +1 · · · · · · ␣because␣of

f807 $F_1 = 0.952$: ␣of@0 0.98, ␣some@−1 0.38, ␣Some@−1 0.04 · · · · · · · · · · · · · · ␣some␣of

Here no single literal clears the threshold, so we include the caught phrase in blue. The phrase need not be contiguous:

f62929 $F_1 = 1.000$: Text@−1 0.67, ␣System@−3 0.67 · · · · · · · · · · · · · · · · · · · ␣System · · · Text

Here · · · marks the token between `System` and `Text`. The feature fires on `System.Text`.

Beyond phrases, we find examples of denotations:

f36902 $F_1 = 0.928$: delta@0 **1.20**, ␣delta@0 **1.17**, δ@0 **1.12**, ␣δ@0 **1.09**, +58

f45978 $F_1 = 0.886$: Delta@0 **1.23**, Δ@0 **1.11**, ␣Δ@0 **1.08**, ␣Delta@0 0.99, ␣delta@0 0.98, +37

f669 $F_1 = 0.936$: sigma@0 **13.67**, ␣Gaussian@0 **7.41**, ␣Sigma@0 **5.40**, ␣sigma@0 **4.94**, σ@0 **4.81**, +1

f4501 $F_1 = 0.921$: rho@0 **1.76**, ␣Rho@0 **1.52**, ␣rho@0 **1.48**, ␣Rhodes@0 **1.43**, ρ@0 **1.36**, ␣resistivity@0 **1.24**, +46

where the σ feature carries `Gaussian` and the ρ feature carries `resistivity`. ρ also fires on `Rhodes`, which appears to be a collision.

Some features are multilingual:

f2902 $F_1 = 0.971$: ␣Zhang@0 **1.36**, Zhang@0 **1.28**, zhang@0 **1.21**, ␣Jang@0 **1.05**, 张@0 **1.02**, +14

f14407 $F_1 = 0.894$: 以@0 **23.93**, ␣به@0 **19.73**, 向@0 **12.65**, به@0 **12.65**, ␣naar@0 **2.00**

张 is the word Zhang in Chinese, and `Jang` is its Korean form. The feature f14407 is the concept 'towards' (expressed in non-English).

A lot of features capture boilerplate templates:

f1722 $F_1 = 0.952$: ␣granted@−5 0.47, ␣hereby@−6 0.47, +1 · · · ␣hereby␣granted · · · ␣charge

f34794 $F_1 = 0.918$: ␣Copyright@−2 0.26, c@0 0.25, ␣*@−3 0.23, +4 <bos> · · · ␣*␣Copyright␣(c)

f10462 $F_1 = 0.951$: +7 · · · · · · · · · · ␣applicable␣law · · · ␣agreed · · · ␣writing · · · ␣software

Similar to rho and Rhodes, some features carry two unrelated concepts, often understood as superposition (Elhage et al., 2022):

f13544 $F_1 = 0.798$: ␣Minnesota@0 **1.39**, ␣Norwegian@0 **1.32**, hydroxy@0 **1.30**, ␣Norway@0 **1.30**, ␣Mn@0 **1.28**, ␣hydroxy@0 **1.26**, ␣throm@0 **1.25**, ␣Alaska@0 **1.23**, ␣Minneapolis@0 **1.23**, ␣MN@0 **1.22**, ␣Vikings@0 **1.21**, Mn@0 **1.21**, ␣manganese@0 **1.20**, mn@0 **1.17**, +202

Here the Minnesota associations (Vikings, Scandinavian heritage) collide with manganese and its biology, presumably through the shared abbreviation MN.

There exist a couple of instances of learned stereotypes (Templeton et al., 2024), for example:

f34073 $F_1 = 0.680$: ␣spell@0 **1.29**, ␣wizard@0 **1.26**, ␣Wizard@0 **1.24**, ␣witch@0 **1.23**, ␣conj@0 **1.19**, ␣Mage@0 **1.19**, ␣Prophet@0 **1.18**, ␣Witch@0 **1.18**, …, ␣gypsy@0 **1.02**, +236

This feature contains ␣`gypsy` with weight 1.02.

We end with our favorite feature:

f6692 $F_1 = 0.931$: ␣television@0 **1.66**, ␣laboratory@0 **1.59**, ␣telephone@0 **1.56**, ␣statistics@0 **1.52**, ␣automobile@0 **1.48**, ␣photographs@0 **1.48**, ␣Laboratory@0 **1.44**, ␣Television@0 **1.42**, ␣bicycle@0 **1.41**, ␣mathematics@0 **1.41**, ␣Mathematics@0 **1.40**, ␣adolescent@0 **1.40**, ␣Metropolitan@0 **1.39**, ␣laboratories@0 **1.39**, ␣photograph@0 **1.38**, ␣petroleum@0 **1.38**, Television@0 **1.35**, ␣metropolitan@0 **1.35**, ␣examinations@0 **1.34**, ␣tuberculosis@0 **1.33**, +102

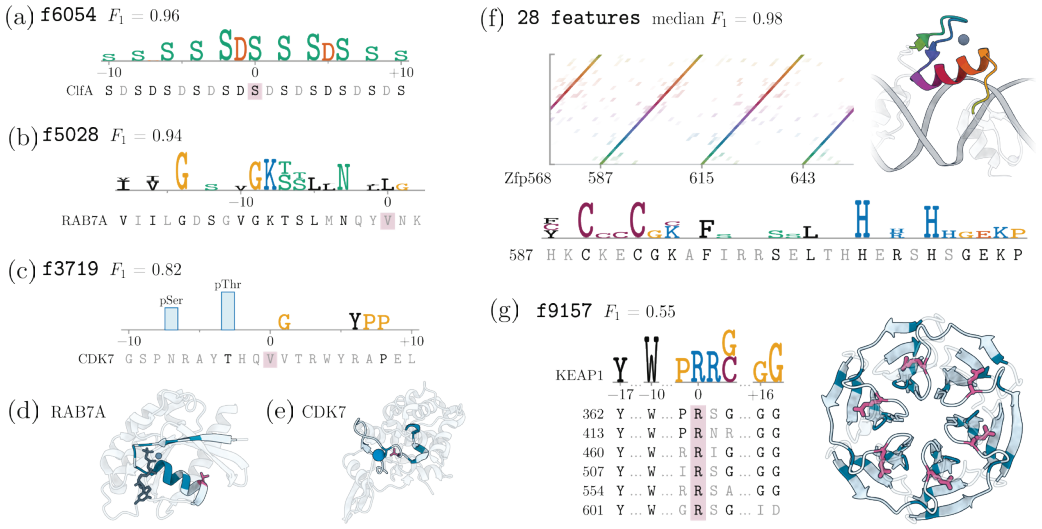


Figure 4: Annotations of protein features from InterPLM (ESM2-650M, layer 33), fit on Swiss-Prot. Residue height is proportional to weight and letters are colored by residue chemistry. Each feature is shown with its F_1 on the test split. (a) A feature detecting the alternating serine–aspartate repeat, shown on the SD-repeat region of clumping factor A (ClfA). (b) A feature reading the P-loop of RAB7A, with its weights reconstructing the motif G-x(4)-G-K-[ST]. (c) A feature with post-translational modifications (from the fit with Swiss-Prot annotations) as its strongest weights, aligned to the sequence of the CDK7 activation loop. (d) The structure of RAB7A (PDB 1T91). The residue where the feature fires is highlighted in pink, and the residues matching the classifier's weights are shown in blue. (e) The structure of CDK7 (PDB 8PYR). The residues are colored the same as in (d), and the phosphate on Thr170 is shown as a blue sphere. (f) The activations of the 28 features along a stretch of Zfp568. The features are sorted, both in color and in row, by the order in which they fire in the zinc finger. The structure on the right shows one of the fingers (residues 587–614) bound to DNA (PDB 5V3M). The logo is shared by the 28 features and is given below. (g) A feature activating on a conserved arginine in the Kelch domain of KEAP1. Each row below the logo is aligned to the arginine where the feature fires. The numbers on the left mark the position of the arginine. The structure on the right (PDB 1ZGK) shows that these arginines occupy the same position in each of the six blades of KEAP1. The arginine in the last blade is a false negative. For all the features besides (c), $\tau = 0.5$. All proteins shown here are from the test split.

5 Beyond text

The construction of Section 2 requires the inputs to have a discrete vocabulary and a context window. Here we annotate the features of InterPLM, an SAE dictionary trained on the protein language model ESM2-650M (Lin et al., 2023; Simon & Zou, 2025). We take the SAE trained on the last layer (layer 33) and fit every feature on Swiss-Prot (The UniProt Consortium, 2025). The only modification is replacing the token vocabulary with residue letters and setting the window to span the full protein (Appendix G). The results of the fit are reported in Table 5.

Similar to our results in Section 4, the annotations span a range of complexity. Many features detect low-complexity regions such as runs of glutamine, asparagine, and lysine. Figure 4(a) shows a feature that detects an alternating serine–aspartate repeat, alongside the disordered region of a protein sequence it fires on. Beyond these, we find multiple features that fire in proximity to established structural motifs, including ankyrin repeats, leucine-rich repeats, TPR, PRR and transmembrane helices. Figure 4(b) shows a feature that reads the P-loop, with its weights reconstructing the motif G-x(4)-G-K-[ST]. The P-loop binds nucleotides through a coordinated Mg^{2+} ion. Figure 4(d) shows the lysine and threonine residues of RAB7A, a small GTPase, coordinating a bound nucleotide and an Mg^{2+} ion. The feature fires seven residues downstream of the motif and its weights at the firing position include hydrophobic residues, presumably sensing the fold surrounding the site.

Swiss-Prot provides human-curated annotations of protein regions, including domains, binding sites, and post-translational modifications. In the second run, we expand the vocabulary to include these annotations on top of residues. The rest of the setup is exactly the same, and the results from the run are shown in Table 5. Two results are notable from this run: the F_1 almost doubles compared to the residue-only run, and the number of features requiring only a single literal to fire increases by $\sim 4\times$. Among features with F_1 above 0.5, 60% have a Swiss-Prot annotation as the highest weight. Figure 4(c) shows an example where the highest weight is a phosphorylated threonine, a post-translational modification that regulates the activity of many proteins, including kinases such as CDK7 (Figure 4(e)). Note that ESM2-650M does not explicitly include post-translational modifications. The feature is presumably detecting correlations in sequence that capture where the modification occurs.

Some features fire periodically, once in every repeat of a motif. Figure 4(f) shows 28 such features in the zinc fingers of Zfp568, one firing at each position of the 28-residue repeat. Zinc fingers are small DNA-binding units, often repeated back to back. They use two cysteines and two histidines to coordinate with a zinc ion, and follow the established motif C-x(2)-C-x(12)-H-x(3)-H. A single finger recognizes about three letters of DNA, so a protein with many fingers can recognize long DNA sequences. Considering the weights of all 28 features together, we find that they reconstruct the exact motif, with each feature seeing it at a different offset.

As a final example, we consider a single feature that periodically fires across the Kelch domain of KEAP1, a ring of six repeated blades that binds NRF2, the main switch of the cell's defense against oxidative stress. The feature peaks once in each blade, always at a conserved arginine (Figure 4(g)). In five of the six blades of KEAP1, the arginine directly contacts the glycine 16 residues downstream. The classifier's top weights include this same pair of glycines.

6 Limitations and discussion

Features may represent something more specific than their descriptions capture (Templeton et al., 2024), and a random draw of negatives might not contain the hard cases that would reveal a feature's specificity (Paulo et al., 2025). Given the imbalance and the formalism of Section 2, a monotone LTF is a simple method of annotation that goes beyond token enrichment and can be run at scale. The annotations are not exact, and limited by the expressivity of the model class. The feature is a Boolean function of its window, so the gap is closable in principle, and richer executable classes inherit the same evaluation.

Circuit analysis explains a feature in terms of the features that feed into it (Marks et al., 2025; Ameisen et al., 2025; Lindsey et al., 2025). The annotation method here offers a complementary view, explaining the same feature in terms of its input. It should be possible to combine the two and follow how phrases are assembled over the course of a forward pass.

The protein features of Section 5 recover functional sites, such as the P-loop and the zinc finger, that recur across the many proteins that carry them. ESM2-650M is not trained on curated annotations or structure, so there must be an optimization pressure that leads it to encode the patterns. Features without a curated annotation may reflect sequence-based correlations that have not yet been characterized.

AI USE STATEMENT

Claude Haiku 4.5 and Claude Opus 5 wrote descriptions and judged them for the evaluations in Section 3. Beyond these experiments, we used generative AI (Claude code) for assistance with coding and generating figures, searching for summaries of related work and polishing the draft. We have reviewed all AI-assisted work and take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

The supplementary material contains the code for Sections 2, 4 and 5. All corpora, models, and dictionaries are public, and the annotations and code will be made available.

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.

Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. Language models can explain neurons in language models, 2023. URL https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html. OpenAI.

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. URL https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 233–240, 2006.

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. URL https://transformer-circuits.pub/2022/toy_model/index.html.

Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pp. 973–978, 2001.

Vitaly Feldman. Hardness of approximate two-level logic minimization and pac learning with membership queries. *Journal of Computer and System Sciences*, 75(1):13–26, January 2009. ISSN 0022-0000. doi: 10.1016/j.jcss.2008.07.007. URL http://dx.doi.org/10.1016/j.jcss.2008.07.007.

Alex Foote, Neel Nanda, Esben Kran, Ioannis Konstas, Shay Cohen, and Fazl Barez. Neuron to graph: Interpreting language model neurons at scale. *arXiv preprint arXiv:2305.19911*, 2023.

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

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. In *International Conference on Learning Representations*, 2025.

Daniel G. Horvitz and Donovan J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952.

Jing Huang, Atticus Geiger, Karel D'Oosterlinck, Zhengxuan Wu, and Christopher Potts. Rigorously assessing natural language explanations of neurons. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 317–331. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.blackboxnlp-1.24.

Adam Karvonen, Can Rager, Johnny Lin, Curt Tigges, Joseph Isaac Bloom, David Chanin, Yeu-Tong Lau, Eoin Farrell, Callum McDougall, Kola Ayonrinde, Demian Till, Matthew Wearden, Arthur Conmy, Samuel Marks, and Neel Nanda. SAEBench: A comprehensive benchmark for sparse autoencoders in language model interpretability. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29223–29264. PMLR, 2025.

Gary King and Langche Zeng. Logistic regression in rare events data. *Political Analysis*, 9(2): 137–163, 2001. doi: 10.1093/oxfordjournals.pan.a004868.

Adam R. Klivans and Rocco A. Servedio. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. *Journal of Computer and System Sciences*, 68(2):303–318, 2004. doi: 10.1016/j.jcss.2003.07.007.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, János Kramár, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 278–300. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.blackboxnlp-1.19.

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, 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. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL `https://transformer-circuits.pub/2025/attribution-graphs/biology.html`.

Samuel Marks, Can Rager, Eric J. 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*, 2025.

H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1222–1230, New York, NY, USA, 2013. Association for Computing Machinery. doi: 10.1145/2487575.2488200.

Jesse Mu and Jacob Andreas. Compositional explanations of neurons. In *Advances in Neural Information Processing Systems 33*, 2020.

Saburo Muroga. *Threshold Logic and Its Applications*. Wiley-Interscience, New York, 1971. ISBN 0471625302.

Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

Gonçalo Paulo, Alex Mallen, Caden Juang, and Nora Belrose. Automatically interpreting millions of features in large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48393–48421. PMLR, 2025.

Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *J. ACM*, 35(4):965–984, October 1988. ISSN 0004-5411. doi: 10.1145/48014.63140. URL https://doi.org/10.1145/48014.63140.

Elana Simon and James Zou. Interplm: discovering interpretable features in protein language models via sparse autoencoders. *Nature Methods*, 22(10):2107–2117, 2025.

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.

The UniProt Consortium. UniProt: the Universal Protein Knowledgebase in 2025. *Nucleic Acids Research*, 53(D1):D609–D617, 2025.

Robert Tibshirani, Jacob Bien, Jerome Friedman, Trevor Hastie, Noah Simon, Jonathan Taylor, and Ryan J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(2):245–266, 2012. doi: 10.1111/j.1467-9868.2011.01004.x.

Berk Ustun and Cynthia Rudin. Learning optimized risk scores. *Journal of Machine Learning Research*, 20(150):1–75, 2019.

Sida Wang and Christopher D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 90–94. Association for Computational Linguistics, 2012.

A Fitting details

A.0 Data, splits, and labels

All experiments run on a 500-million-token sample of the uncopyrighted subset of the Pile (Gao et al., 2020), using the Gemma 2 2B tokenizer. The SAE activations are stored at 64, 128, and 183 slots per token for the layer-19 65k, layer-19 16k, and layer-12 65k SAEs respectively. The corpus is split into contiguous blocks: 25% train, 25% tune, and 50% test. The fits run on the train split, the decision threshold is chosen on the tune split, and the reported scores are measured on the test split. Each feature is fit on a sample of these splits: 4,000 training positives with 40 negatives per positive, capped at 200,000 negatives. The tune and test sets hold 2,000 and 4,000 positives, each with 40 negatives per positive and at least 500,000 and 1,000,000 negatives respectively.

A token's label comes from the feature's activation, and Sections 3 and 4 label differently. In Section 3 the threshold is zero: a token is positive if the feature fires at all, i.e. its activation is strictly above zero. In Section 4 the threshold sits 0.15 of the way between the feature's smallest and largest firing activation, a token is positive at or above it, and firing tokens below it are dropped from the population, counting neither as positives nor negatives. Here the maximum is the largest activation over the corpus, and the minimum is given by the JumpReLU threshold.

	Features	Label threshold	Population
Section 3	1,000 per SAE	> 0 (fires at all)	every token
Section 4	all 147,456	≥ 0.15 of the range	weak firings dropped

A.1 Convexity of the loss function

The loss function in Equation 2 is minimized subject to $w \geq 0$. The intercept b is not constrained or penalized. Because of the positivity of w, the ℓ_1 norm of the weights is the linear function $\sum_{r,v} w_{r,v}$.

Each term in the loss accumulated over the whole dataset is a convex function of s_j, and s_j is affine in (w, b). Convexity is preserved under affine composition and nonnegative sums, so the loss function is a smooth convex function of (w, b), i.e. every local minimum is global. Both smoothness and convexity allow us to use a quasi-Newton method with box bounds (L-BFGS-B).

A.2 Class weight, intercept, and the threshold

We start with a classic result: in a well-specified logistic model, multiplying the positive class weight by $c = |\mathcal{N}|/|\mathcal{P}|$ shifts the intercept by $\log c$ and leaves w unchanged (King & Zeng, 2001; Elkan, 2001). Well-specified means there exist (w^*, b^*) such that $\sigma(w^* \cdot x + b^*) = P(y{=}1 \mid x)$ for every input x, or in other words the true conditional probability itself has the form given by the logistic model. The result now follows by defining $p(x) = \sigma(w^* \cdot x + b^*)$, and considering the weighted loss of a score $s(x)$ (the cost of assigning score s to an input x with positive probability $p(x)$) as:

$$\ell(s(x)) \;=\; -\,c\,p(x)\log\sigma(s(x)) \;-\; (1-p(x))\log\big(1-\sigma(s(x))\big). \tag{4}$$

Taking $\partial\ell/\partial s = 0$ gives

$$\frac{\sigma(s(x))}{1-\sigma(s(x))} \;=\; c\,\frac{p(x)}{1-p(x)} \quad\Longrightarrow\quad s(x) \;=\; \log\frac{p(x)}{1-p(x)} \;+\; \log c. \tag{5}$$

Matching the LHS and RHS and using the definition of $p(x)$, the score that minimizes the loss function is given by $s(x) = w^* \cdot x + (b^* + \log c)$. Hence, the x-dependence is captured by w^*, and the class imbalance only enters through the intercept.

The result above applies to a probabilistic label model, but here we are trying to functionally approximate a Boolean function, where $P(y{=}1|x)$ is either 0 or 1. To reconcile this discrepancy, we first consider the case that the feature is indeed representable through a monotone LTF. Here, we can define a scaling term β such that $\sigma(\beta(w^* \cdot x + b^*)) \to \mathbf{1}[w^* \cdot x + b^* \geq 0]$ as $\beta \to \infty$, recovering the deterministic label exactly in the limit. In this case, the class imbalance is meaningless, i.e. the model achieves zero loss by simply placing all its weight on that literal, regardless of c. However, since we use an ℓ_1 penalty, the weights are finite, which implies that $\sigma(s)$ can take any value between 0 and 1. Hence, the fitted model is inherently probabilistic, and the result above applies. [3]

In practice, given the finite number of samples, we choose θ directly rather than setting the threshold to $b + \log c$. To do so, we note that any form of class imbalance shifts all scores by the same constant, preserving their ordering. This lets us find the optimal threshold regardless of the positive-to-negative ratio the tune set is drawn at. The threshold is chosen by maximizing F_1 on the tune set. Here we exploit the fact that $F_1(\theta)$ only changes when θ crosses a score in the tune set, so the optimum is attained at one of the distinct scores. The scores are sorted and F_1 is computed at every midpoint between consecutive values. The algorithm gives the exact maximizer of F_1 over the tune set at the cost of a single sort, $O(n \log n)$.

A.3 Screening

At $w = 0$, the optimal intercept $b = 0$ (the class weight makes $\sigma(b) = 1/2$). We define $n^+_{r,v}$ and $n^-_{r,v}$ as the number of positives and negatives that contain token v at offset r respectively. The derivative of the loss in $w_{r,v}$ at $w = 0$ takes the form

$$\left.\frac{\partial\mathcal{L}}{\partial w_{r,v}}\right|_{w=0} \;=\; \frac{1}{2}\left(n^-_{r,v} - c\,n^+_{r,v}\right) + \lambda. \tag{6}$$

Hence for the derivative to be negative, the condition is given by:

$$\frac{1}{2}\left(c\,n^+_{r,v} - n^-_{r,v}\right) \;>\; \lambda. \tag{7}$$

Hence, for a literal to have a negative gradient at $w = 0$, it should be enriched among the positives, i.e. appear at a higher rate there than among the negatives, with λ setting the bar.

[3] Since exactly one token is active per offset in our case, adding the same weight to every token at an offset could in principle absorb the class imbalance, but this costs λV more in penalty compared to b, which is unpenalized. The convexity of the loss function guarantees that the optimizer finds this cheaper solution.

A literal could be enriched by coincidence, so we additionally test each literal against a null model. Specifically, we choose a Poisson null with Bonferroni correction: under the null, the label is independent of the literal, and the count of positives containing token v at offset r is Poisson with rate $n_r \cdot q_v$, where q_v is the token's overall rate in the training split and n_r is the number of positives with offset r inside their document. A literal is kept when the probability of a count at least as large as $n^+_{r,v}$ falls below α/d with $\alpha = 0.05$. When no literal clears the bar, the 64 literals with the smallest p-values are kept instead. For a typical feature, the screen cuts the number of literals from $\sim 10^8$ to $\sim 10^3$.

A.4 The regularization path

After the screening step, the loss function is minimized over the surviving literals across a range of λ. The optimization path starts at $\lambda_{\max}$, the smallest λ at which no literal satisfies Equation 7 and all weights are zero. From there, 75 values of λ descend geometrically to $10^{-5} \cdot \lambda_{\max}$, and each solve is warm-started from the previous solution. Each solution along the path is scored by F_1 on the tune set. If the best F_1 sits at the last value, the path is extended a decade at a time, at most three times.

The candidate literals for each solve are chosen by the strong rule (Tibshirani et al., 2012). A literal is a candidate if it already has a nonzero w, or if its gradient at the previous solution falls below the gap between the previous and current λ. The objective over the candidates is solved with L-BFGS-B. After the solve, the gradient is checked for every surviving literal outside the candidate set. Any literal with a negative gradient is added and the solve repeats, until no candidates remain.

B F_1 at low prevalence

In terms of raw counts, $F_1 = 2\,\mathrm{TP}/(2\,\mathrm{TP} + \mathrm{FP} + \mathrm{FN})$, where TP, FP, and FN are the true positives, false positives, and false negatives. For feature i, the counts are $\mathrm{TP} = \mathrm{TPR} \cdot |\mathcal{P}_i|$, $\mathrm{FP} = \mathrm{FPR} \cdot |\mathcal{N}_i|$, and $\mathrm{FN} = (1 - \mathrm{TPR}) \cdot |\mathcal{P}_i|$. Dividing through by $|\mathcal{P}_i| + |\mathcal{N}_i|$ gives the exact form

$$F_1 \;=\; \frac{2\,\mathrm{TPR}\,\pi_i}{(1 + \mathrm{TPR})\,\pi_i + \mathrm{FPR}\,(1 - \pi_i)}. \tag{8}$$

Since $\mathrm{TPR} \le 1$, the first term in the denominator is at most $2\pi_i$. When $\mathrm{FPR} \gg \pi_i$, the denominator is dominated by $\mathrm{FPR}\,(1 - \pi_i) \approx \mathrm{FPR}$, giving Equation 3.

Labeling every token positive has $\mathrm{TPR} = \mathrm{FPR} = 1$, and the exact form gives $F_1 = 2\pi_i/(1 + \pi_i) \approx 2\pi_i$, the baseline of Section 4. Labeling every token negative has $\mathrm{TP} = 0$, so $F_1 = 0$ while the accuracy is $1 - \pi_i$. For the descriptions of Table 2, FPR is 40–200 times π_i, and Equation 3 applies. For the classifier, FPR is at the scale of π_i itself, and its F_1 comes from the counts directly.

C Evaluation constructions

C.1 Detection

The detection prompts, few-shot examples, and system message are taken verbatim from the EleutherAI `sae-auto-interp` repository at commit `65b2a33`, and the window selection is reimplemented against the pinned code. Documents of the test split are tiled into 32-token windows, and a window's activation is the largest stored activation inside it. The explanation is generated from 40 windows sampled by decile of activation rank. The judged set holds 100 activating windows from an independently drawn seeded stream, and 100 windows from documents with no stored firing, shuffled once. In the explainer prompts, runs of tokens strictly above 0.3 of the window maximum are wrapped in marks, as in `Greek relations with Egypt<< dates back to before >>2000 B.<<C>>`. The judge scores five windows per API call. A reply is parsed as the first bracketed list with one integer per window in the call; an unparsed batch scores -1 for each of its windows. The replies fail to parse for 74 of 119,941 batches (0.06%), and 370 of 599,701 labels fall outside $\{0, 1\}$ (0.06%). The score is balanced accuracy over the 200 windows.

C.2 SAEBench

We follow the SAEBench protocol exactly, pinned to commit `8042bb38`. The window selection is reimplemented to match their setup. Documents of the test split are tiled into 128-token sequences. Windows are 21 tokens centered on activation peaks, spaced at least 10 tokens apart and at least 10 tokens from sequence boundaries. The explainer sees 15 windows, 10 top and 5 importance-weighted (weights proportional to the squared activation). The judge sees 14, of which 2 top, 2 importance-weighted, and 10 random with activations zeroed. A window is active when its peak is above 0.01 of the feature's maximum activation. The explanation is capped at 30 tokens. The judge replies with the comma-separated indices of the active windows, as in `1, 7, 8, 14`, capped at 64 tokens with one retry at 320. For 60 of 2,999 features the reply remains unparseable after the retry, and they are excluded from the mean. The score is accuracy over the 14.

	Detection	SAEBench
Window length (tokens)	32	21
Explainer windows	40	15
Judged windows	$100:100$	$4:10$
Marked above	$0.3\times$ window max	—
Active above	—	$0.01\times$ feature max
Windows per judge call	5	14
Explanation cap (tokens)	—	30
Reply cap (tokens)	—	64, one retry at 320

C.3 The marked-token evaluation

Similar to fuzzing (Paulo et al., 2025), the judge reads a 33-token excerpt, clipped at document boundaries, with the sampled token marked at the center. Newlines render as $\hookleftarrow$. The full prompt is shown in Figure 5.

The prompt phrasing was chosen from three variants, piloted on five features of the layer-19 65k SAE. The selection criterion, fixed before the main run, was parse robustness first and lowest false-positive rate second. Variant B won on both and was frozen for all subsequent experiments.

The description's TPR and FPR are measured on 50 firing and 2,000 non-firing tokens, then reweighted to natural prevalence to compute F_1. When a description flags none of its 2,000 negatives, the sample extends to 20,000 to resolve the false-positive rate. This happens for 12 of the 300 features. Wilson 95% intervals on TPR and FPR give bounds on F_1, since F_1 is monotone in both rates. As a control, the same descriptions are judged on a balanced draw of 50 firing and 50 non-firing tokens, giving median balanced accuracies of 0.81, 0.70, and 0.75 for the three SAEs.

The classifier is scored on exact counts over the whole test split. To additionally validate the sampling, we also measure the classifier's TPR and FPR on the same sampled tokens and compute Wilson 95% intervals. The resulting F_1 bounds cover the classifier's exact F_1, for 96–99 of 100 features per SAE.

D The excerpt-level comparison

We repeat the comparison of Table 2 at the excerpt level. The judge labels whole 32-token windows with no marked token, using the same detection prompt. The same 100 features per SAE are judged on 50 activating and 1,000 non-activating excerpts. The judge is Claude Haiku 4.5.

TPR and FPR are measured on the sampled excerpts and reweighted to natural prevalence to compute F_1, as in Appendix C.3. The medians are reported in Table 3.

E The census in full

Section 4 fits a classifier for every feature of the three SAEs; Table 4 reports the results per SAE. Each classifier is scored on the test sample of Appendix A.0 at the threshold fixed at fit time. Since the test sample is stratified, each positive and negative token carries its sampling weight, and precision, recall, and F_1 are computed from the weighted counts (Horvitz & Thompson, 1952).

```
System:
You are an intelligent and meticulous linguistics researcher.
You will be given a certain feature of text, such as ``male
pronouns'' or ``text with negative sentiment''.
You will then be given several text examples. Your task is to
determine which examples possess the feature.
For each example in turn, return 1 if the sentence is correctly
labeled or 0 if the tokens are mislabeled. You must return your
response in a valid Python list. Do not return anything else besides
a Python list.
User:
Feature explanation: {explanation}
Judge only the marked token: label 1 if the explanation applies to
the token inside << >>, otherwise 0.
Text examples:
Example 1:  int USB_RECIP_INTERFACE = 0x01;↩        private<<
static>> final int USB_RT_ACM = UsbConstants.USB_TYPE_
```

Figure 5: The judge prompt of the marked-token evaluation. The sentence in bold is the only text added to the detection prompt; the few-shot examples and system message are unchanged, and the variant was frozen before the main run. The rendered example is a firing token of feature 5119 of the layer-19 65k SAE.

Table 3: Excerpt-level scores at natural prevalence, 100 features per SAE. Each entry is the median over features. Prevalence is the fraction of windows that activate. Both methods are scored on the same sampled excerpts, with F_1 reweighted to natural prevalence. The dollar figure is the judging cost only; the descriptions are those of Table 1 and are not charged again.

		LLM Description ($31.94)			Classifier		
SAE	median prevalence	TPR	FPR	F_1	TPR	FPR	F_1
L19, 65k (L_0=63)	7.8×10^{-3}	0.80	0.13	0.099	0.38	0.005	**0.33**
L19, 16k (L_0=73)	3.9×10^{-2}	0.57	0.14	0.28	0.40	0.018	**0.41**
L12, 65k (L_0=72)	9.8×10^{-3}	0.68	0.15	0.14	0.33	0.006	**0.30**

For the 100 features per SAE of Section 3.2, the classifiers are also scored exactly over every token of the test split. The weighted estimate and the exact count agree: the median absolute difference in F_1 is 0.003–0.005 across the three SAEs.

A feature is skipped when it has fewer than 10 training positives or fewer than 5 tune positives. All 903 unannotated features of Table 4 are skips of this kind.

F Catching phrases

To extract the phrases in Figure 3(b), we consider every feature activation in the tune and test splits, and keep only the true positives. At each true positive, we take the classifier's nonzero-weight literals that are present in the surrounding context, and sort them by weight. The step above is equivalent to ranking the literals by their contribution to the log odds. We then perform a greedy pass, adding literals in order until their summed weight crosses the threshold. This set is the smallest number of literals needed to explain the firing. The phrase is written in offset order.

G Protein

The protein experiments use ESM2-650M (Lin et al., 2023) and the InterPLM SAE trained on its layer-33 hidden state (Simon & Zou, 2025), with dictionary size 10,240 and a mean of 148 features

Table 4: The census: every feature of the three SAEs, fit as in Section 2 and scored on the held-out test split. Medians are over the annotated features; the unannotated remainder had too few firings above the threshold to fit. k^* is the smallest number of literals whose summed weights clear the classifier's threshold. Ratios are of the $(2R{+}1)V \approx 2.6 \times 10^8$ possible literals. The fit time is the wall clock for annotating the full dictionary on a single quad-GH200 node.

	L19, 65k (L_0=63)	L19, 16k (L_0=73)	L12, 65k (L_0=72)
% annotated	99.4%	99.9%	99.2%
Median π_i	7.8×10^{-5}	3.2×10^{-4}	8.9×10^{-5}
Median F_1	0.25	0.29	0.23
Median literals (ratio of possible)	317 (10^{-6})	361 (10^{-6})	251 (10^{-6})
% k^*=1	23%	42%	32%
Approx. fit time	2.8 h	1.1 h	2.8 h

firing per residue. The database is Swiss-Prot (The UniProt Consortium, 2025), which contains 556,222 proteins and 179 million residues. Proteins longer than 1,022 residues are dropped. For our first run, we follow the same procedure as the text experiments in Section 4, but replace tokens with residue letters. The alphabet has 25 symbols: the 20 standard amino acids, selenocysteine (U), pyrrolysine (O), the ambiguity codes B (asparagine or aspartate) and Z (glutamine or glutamate), and the unknown residue X. Here we choose R to be 1,024, since the maximum protein length is 1,022. The SAE activations are stored at 192 slots per residue. The splits and fit budgets of Appendix A.0 are unchanged, with proteins kept whole across splits.

Since the InterPLM SAE uses ReLU rather than JumpReLU, the label threshold is simply a fraction of the feature's maximum activation. We fit each feature at two thresholds, 0.15 and 0.5 of the maximum. Firing residues below the threshold are dropped from the population as in Section 4.

For our second run, we extend the alphabet to include Swiss-Prot annotation labels alongside the residue letters. Here, the only modification is that Swiss-Prot concepts are treated as a token present at every residue the concept covers. We keep concepts with at least 20 occurrences in the database, giving 3,260 concepts and a total alphabet of 3,285. The results from all these runs are shown in Table 5.

Table 5: The protein census: every feature of the InterPLM SAE for layer 33 of ESM2-650M, fit as in Section 2 at two label thresholds and scored on the held-out test split. Medians are over the annotated features. k^* is as defined in Table 4. Ratios are of the $(2R{+}1)V$ possible literals: 51,225 for the residue alphabet and 6,730,965 with Swiss-Prot concepts. The fit time is the wall clock of each run, on the same node as Table 4.

	Sequence only		With concepts	
Label threshold	0.15	0.5	0.15	0.5
% annotated	91.1%	88.3%	91.1%	88.3%
Median π_i	1.3×10^{-3}	5.0×10^{-5}	1.3×10^{-3}	5.0×10^{-5}
Median F_1	0.11	0.14	0.21	0.25
Median literals (ratio of possible)	113 (2×10^{-3})	219 (4×10^{-3})	386 (6×10^{-5})	659 (10^{-4})
% k^*=1	5.9%	0.5%	25.4%	2.4%
Fit time	1.6 h	2.5 h	11.3 h	10.0 h