Individual Parameters in Weight-Sparse Transformers Appear Interpretable

Arnau Marin-Llobet [1] **Stefan Heimersheim**

Abstract

A central goal of mechanistic interpretability is to understand how neural networks work and what each individual component does. Dominant circuit-finding approaches focus on a specific behavior and reverse-engineer the role of components on the associated sub-distribution. However, past work has shown that components can have different functions that are active on different subsets of the input distribution. In this work we ask whether a single weight can be understood globally across the full training distribution by characterizing when it matters (the inputs on which ablating it changes the model's predictions). We introduce an automated LLM pipeline that writes a short, human-readable description of when a weight matters and verifies it on held-out text, crediting a weight only if its description generalizes. Across two sparse and two dense transformers, the fraction of weights that are interpretable (in this sense) is higher in sparse transformers than in dense ones, a gap that widens once unreliable descriptions are discarded. Our results show that a meaningful fraction of a sparse transformer model's weights can be interpreted: 12 to 31% of weights have a single short description that identifies what the weight is used for[*]

1. Introduction

While large language models have rapidly increased in capability, we are still far from understanding how they work. Mechanistic interpretability aims to reverse-engineer neural networks into the algorithms they implement, and the dominant approach does so by isolating circuits responsible for specific behaviors (Ferrando et al., 2024; Olah et al., 2020; Wang et al., 2022; Olsson et al., 2022). A circuit tells us what a set of weights does in service of one task on one distribution, but the same weight can participate in many circuits, contributing differently to each (Elhage et al., 2022b; Janiak et al., 2023).

As an alternative, we ask whether we can understand what an individual weight does on the full training distribution. In standard dense models, this question is difficult to investigate due to superposition: the weight basis is not an interpretable basis of the network's parameters (Gurnee et al., 2023; Bricken et al., 2023; Frankle & Carbin, 2018), with rare exceptions (Yu et al., 2024). Recent work on weight-sparse transformers (Gao et al., 2025) addresses this by training models in which most weights are forced to zero, leaving a small number of surviving weights that organize into compact, readable circuits while the model retains some language modeling capability.

This makes weight-sparse transformers a natural testbed for our question. The zero weights are trivially interpretable since they contribute nothing to the model's behavior; the open question is whether the surviving nonzero weights are interpretable too. These models let us develop weight-based interpretability pipelines today; once the field identifies an interpretable basis in parameter space for dense models (e.g. (Olah et al., 2025; Braun et al., 2025)), we hope to apply these pipelines to frontier language models.

What would it mean to find that a weight is interpretable? We make a specific choice: we try to explain *when* a weight matters; that is, the inputs on which it has a measurable effect on the model's output. This does not capture everything one might want from a full understanding of a weight, but it is concrete, testable, and a natural starting point. For each weight we propose a short, human-readable explanation of when it should be active, and we evaluate the explanation by checking whether it holds on data the explanation was not generated from.

Doing this at scale is hard for two related reasons. The first is *automation*: a sparse transformer has tens of thousands of nonzero weights, and any useful test must run across all of them without human inspection of each one. The second is *measurement*: a useful score must capture two properties at once. When the explanation claims a weight is active, ablating it should reduce model performance on

[1]School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138. Correspondence to: Arnau Marin-Llobet <amarinllobet@seas.harvard.edu>.

Mechanistic Interpretability Workshop at the 43rd International Conference on Machine Learning. Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

[*]Project website: weightpedia.org/individual-parameters-in-sparse-transformers/.

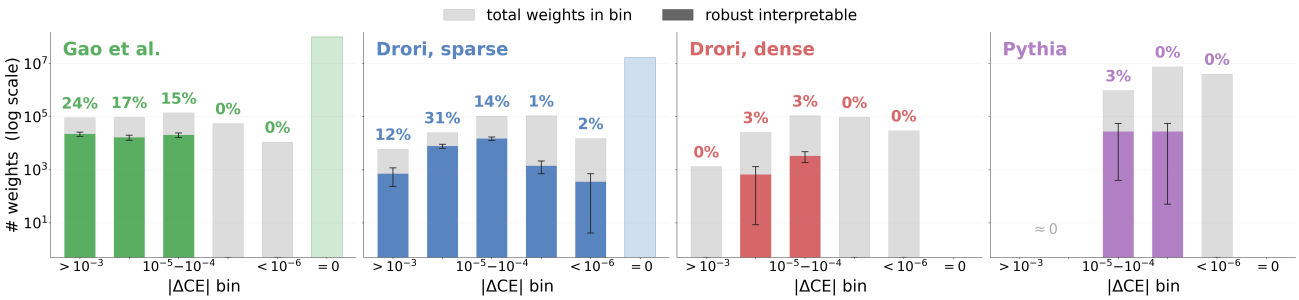

Figure 1. Per-weight causal impact distribution on a held-out distribution. Gray bars: total nonzero weights per $|\Delta\mathrm{CE}|$ bin. Colored bars within each gray bar show the fraction of weights in that bin that pass our interpretability test. The right-most column collects parameters that are exactly zero. Error bars are ± 1 SEM. Pooled interpretability scores across all nonzero weights: Gao et al. (2025) 15%, Drori (2026) sparse 9.6%, Drori (2026) dense 1.5%, Pythia-70m 0.4%.

those inputs; when the explanation claims the weight is inactive, ablating it should leave performance unchanged. Without the first, the explanation does not account for the weight's effect; without the second, the explanation passes trivially by being too broad.

In this paper we introduce an automated, LLM-based pipeline that addresses both challenges. For each nonzero weight we ablate it and measure the resulting Kullback–Leibler (KL) divergence and cross-entropy (CE) shift on a held-out corpus, prompt an LLM to generate candidate Python functions describing the token contexts most affected by the ablation, and score each candidate, taking the best of n candidates as the explanation. We test the pipeline on a weight-sparse coding transformer (Gao et al., 2025), a sparse-and-dense SimpleStories pair (Drori, 2026), and Pythia-70m (Biderman et al., 2023) as a dense pretrained control. We find that the pipeline assigns interpretable predicates to up to 20–30% of nonzero weights in the sparse models that have a high CE impact, compared to 0–3% in the dense baselines when re-evaluated on a held-out corpus. Combined with the trivially interpretable zeros, this means a large fraction of parameters in weight-sparse transformers admit per-weight explanations, while the same is not true of dense models. Our contributions are as follows.

- **A framework for per-weight interpretability.** We define what it means for a single weight to be interpretable in terms of two complementary checks. *Recovery* measures how much of the weight's ablation effect a candidate explanation accounts for. *Inverse* measures the effect recovered by the explanation's negation, which guards against explanations that succeed only by being too broad.
- **An automated LLM pipeline.** Given a weight, the pipeline profiles its ablation effect across a corpus and returns a scored explanation of when the weight matters, with no manual inspection required.
- **Empirical evidence that a substantial fraction of nonzero parameters in weight-sparse transformers are individually interpretable**, far above what is achievable in dense controls, and the gap survives held-out validation.

2. Related work

Circuit decomposition from weight matrices. A line of work going back to the original Circuits Thread (Olah et al., 2020; Elhage et al., 2021) and continued in recent bilinear-MLP analyses (Pearce et al., 2024) examines weight matrices to extract circuits responsible for specific behaviors. The unit of analysis is the circuit; weights enter as ingredients of larger structures rather than as objects of explanation. Approaches in this style identify a behavior, isolate the subnetwork that implements it, and read off the weight matrices that compose it. Recent work shows that this behavior-first localization extends even to broad capabilities in dense LLMs: harmful generation, for instance, depends on a compact, prunable set of weights (Orgad et al., 2026). All of these approaches are powerful when the target behavior is known in advance, but none produces a global account of what the model contains. Our approach inverts the dependency: we ask what each individual weight does without first committing to a behavior, which lets us characterize weights that participate in no named circuit at all, at a scale that hand-curated circuit extraction cannot reach.

Interpretable-by-design and sparse-weight transformers. Several lines of work have produced transformers whose weights admit direct interpretation. Tracr (Lindner et al., 2023) compiles models from a high-level program (RASP; Weiss et al., 2021), so the interpretability of the resulting weights is a property of the compilation pipeline rather than something the model acquired from a training objective. A second line of work modifies the training recipe itself to encourage interpretability, by introducing an activation function (the Softmax Linear Unit) that increases the fraction of MLP neurons responding to a single human-interpretable

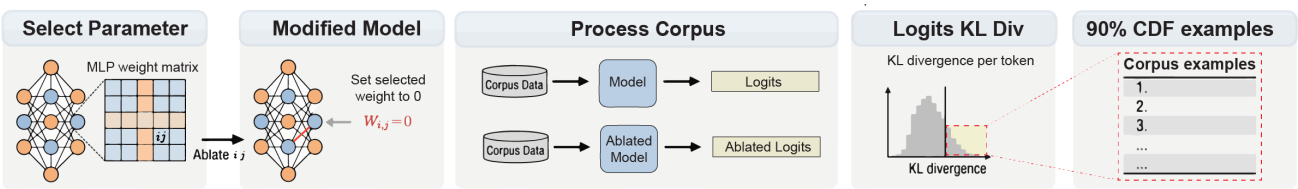

Figure 2. Per-weight interpretability pipeline. Given a target weight $W_{i,j}$ in an MLP layer, we run both the original model and an ablated copy on a corpus, measure the per-token KL divergence between them, and extract the token contexts that account for the top 90% of cumulative KL. These contexts are passed to an LLM, which proposes candidate predicates describing them; the highest-scoring candidate becomes the weight's interpretation.

concept without sacrificing language modeling performance (Elhage et al., 2022a). In our setting, Gao et al. (2025) train transformers in which most weights are forced to zero, and show that the surviving weights organize into compact, readable circuits for specific behaviors. Drori (2026) extends this with the SimpleStories sparse-and-dense pair, which couples a sparse reasoning core to a dense output head and recovers more of the loss-capability frontier than fully sparse training. Our contribution is complementary; the sparse models are our testbeds (Gao et al., 2025; Drori, 2026). Where prior work establishes that sparse training produces interpretable circuits, we ask whether the individual parameters of these models admit per-weight explanations without first choosing a behavior.

Parameter decomposition. A second line of work treats parameters as the unit of analysis but decomposes them into a learned basis of components rather than interpreting the raw entries. APD (Braun et al., 2025), SPD (Bushnaq et al., 2025), and L3D (Chrisman et al., 2025) all follow this strategy. The motivation is that decomposed components may correspond to recognizable functions even when individual weights do not, which our results confirm is the case for dense models. We take a different approach. In sparse models, the raw weights are already a candidate basis for interpretation, and no decomposition step is needed. Working with the parameters as trained avoids the question of whether a learned decomposition reflects the model's computation or the optimization pressure of the decomposition objective. It also yields per-parameter rather than per-component answers, which is the relevant unit when the goal is to identify or edit specific parts of the model.

3. Methods

Our pipeline takes a single nonzero weight as input and returns a Python function that describes when the weight is active, together with quantitative scores measuring how well it explains the weight's effect on the model. The pipeline has three stages, summarized in Figure 2: (i) compute the weight importance location by ablating it and measuring the per-position KL divergence between the original and ablated models; (ii) prompt an LLM to generate candidate predicates from the most-affected token contexts; and (iii) score each candidate by running the original model with the weight gated on or off according to the predicate, and keep the best one. We describe each stage below, give the full algorithm in Algorithm 1, and end with the experimental setup.

3.1. Computing single-weight importance location

To ask whether a single scalar weight w in an MLP layer is interpretable, we first need to know whether and where it matters. We do this by ablation: we run the model twice on the same corpus, once normally and once with w set to zero, and compare the two next-token predictions at every position. The per-position disagreement is the KL divergence between the original and ablated model, which we denote KL_p. The positions with the largest KL_p are the ones the weight influences most, and the cumulative distribution over KL_p tells us how concentrated the weight's effect is. We summarize a weight's overall importance by its cross-entropy impact $\Delta\mathrm{CE}$, the difference between the corpus cross-entropy of the ablated model and that of the original. Figures 3, 4, 8 and 9, show example weights processed this way, with their KL distributions and the token they most affect.

3.2. Generating candidate predicates

Given the ranked positions from §3.1, we want a short, human-readable description of what those positions have in common. We obtain one automatically by selecting the positions accounting for the top 90% of cumulative KL, using a small window of surrounding tokens around each, and passing the resulting set of contexts to an LLM. The LLM is prompted to return a Python function of the form `f(tokens, pos) -> bool` together with a one-line natural-language description of what it detects.

In practice the LLM produces predicates ranging from literal token-set membership (`tokens[pos] in {"200", "201", "202", ...}`) to multi-token context tests (`tokens[pos-1] == '"' and tokens[pos].isalpha()`). The prompt asks for coverage over the affected contexts rather than elegance,

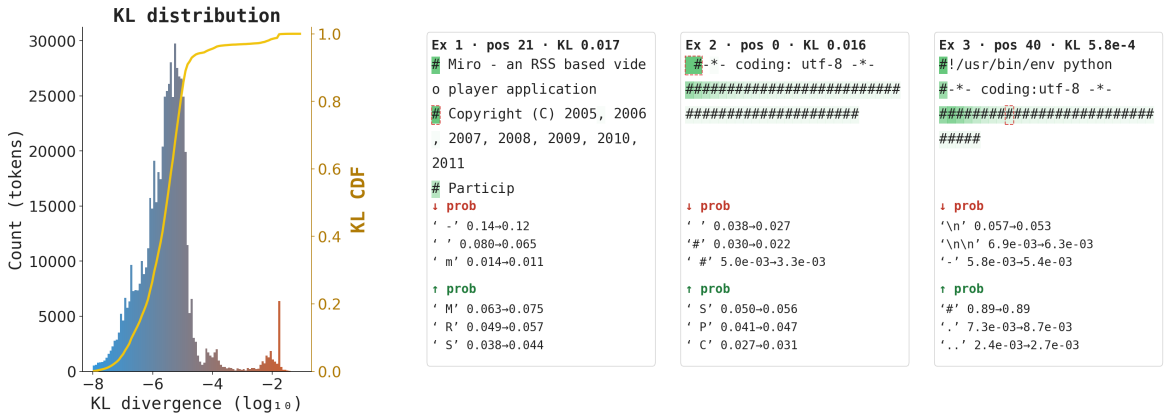

Figure 3. An interpretable weight from Gao et al. (2025) sparse code transformer. Left: distribution of per-token KL divergence over the corpus when this weight is ablated, with the cumulative KL shown in yellow. Right: three token contexts drawn from the top of the KL distribution. Each panel shows the surrounding text, the position of the affected token, and the largest probability shifts (red: probabilities decreased by ablation; green: increased).

Figure 4. An interpretable weight from Drori (2026). The same display as Figure 3; the affected tokens cluster around speech-verb contexts ("called","shouted", "voice", etc.).

and we do not filter or edit the returned code, where we sometimes find minor hallucinations. The context window around each top position is ± 8 tokens. We sample N candidate predicates and choose the best candidate based on the scores detailed in §3.3. We sweep N in Appendix A.4 and find our headline numbers saturate around $N = 100$, and we select it as default candidate budget.

3.3. Scoring a candidate predicate

Given a candidate predicate f for weight w, we want to know whether f actually describes what the weight does. The test is a conditional-zero ablation. We globally ablate w throughout the corpus, then restore its per-position contribution only at the positions indicated by f, and measure the resulting cross-entropy. Concretely, write CE_0 for the cross-entropy of the unmodified model, $\mathrm{CE}_\varnothing$ for the model with w ablated everywhere, and $\Delta\mathrm{CE} = \mathrm{CE}_\varnothing - \mathrm{CE}_0$ for the ablation effect. Let CE_f be the cross-entropy of the conditional-zero model that ablates w everywhere except where f holds, and $\mathrm{CE}_{\neg f}$ the symmetric model that ablates w everywhere except where f does *not* hold. We score the predicate by its *recovery* and *inverse*:

$$\mathrm{recovery}(f) \;=\; 1 \;-\; \frac{\mathrm{CE}_f - \mathrm{CE}_0}{\Delta\mathrm{CE}} \tag{1}$$

$$\mathrm{inverse}(f) \;=\; 1 \;-\; \frac{\mathrm{CE}_{\neg f} - \mathrm{CE}_0}{\Delta\mathrm{CE}}. \tag{2}$$

A recovery of 1 means restoring the weight at the predicted positions fully reproduces the original model, a recovery of 0 means restoring it there has no effect at all, and a recovery > 1 can occur on weights with very small $\Delta\mathrm{CE}$ where the conditional-zero model happens to perform marginally better than the unmodified one — we treat such cases as numerically unreliable and flag them with an asterisk in figures.

Recovery and inverse alone are not enough. A predicate that holds at almost every position trivially recovers most of the ablation effect; in the extreme, the predicate $f \equiv \text{True}$ achieves $\text{recovery} = 1$ and $\text{inverse} = 0$ while saying nothing about the weight. We rule these out with a *coverage gate*. Writing $p(f)$ for the *coverage* of f—the fraction of corpus positions where it fires—we keep a candidate only if it fires on at least one position but no more than half of them, i.e. $0 < p(f) \leq c$ with coverage cap $c = 0.5$. Among the surviving candidates we summarize recovery and inverse by the interpretability *score*,

$$\text{score}(f) \;=\; \min\big(\text{recovery}(f),\; 1 - \text{inverse}(f)\big), \quad (3)$$

and select the highest-scoring candidate f^* among the N returned by the LLM as the weight's interpretation. A weight is called *interpretable at threshold* T if its best predicate clears the score threshold and passes the coverage gate,

$$\text{score}(f^*) \;\geq\; T \qquad \text{and} \qquad p(f^*) \leq c$$

(equivalently $\text{recovery}(f^*) \geq T$ and $\text{inverse}(f^*) \leq 1 - T$). We use $T = 0.75$ and $c = 0.5$ throughout the paper and report sensitivity to T in Appendix A.5. Algorithm 1 gives the full pipeline.

3.4. Models and corpora

We evaluate three training regimes. The weight-sparse code transformer of Gao et al. (2025) is trained at ∼99% weight sparsity on Python code. The SimpleStories sparse-and-dense pair of Drori (2026) consists of a sparse model trained at ∼98% sparsity on the SimpleStories corpus and a dense counterpart with the same architecture but no sparsity constraint. Pythia-70m (Biderman et al., 2023) serves as a dense pretrained control, trained on The Pile (Gao et al., 2020). For each model we evaluate weights on a corpus that approximates its training distribution; see Appendix A.1 for details. For each model we sample nonzero weights uniformly across MLP layers, and report per-figure sample sizes in the corresponding captions. Unless stated otherwise, the auto-interp LLM $\mathcal{A}$ is Gemini 3 Flash, with the candidate budget set as in §3.2. We replicate headline results with Claude Sonnet 4.5, GPT-5, and GPT-4o in Appendix A.5.

3.5. Held-out evaluation

The predicate generation and best-of-n choice both use the same dataset of text, so it might just describe surface patterns of that text rather than the weight itself. To rule this out, we always pick a predicate on one piece of text and *test* it on different text the LLM never saw. Concretely, we take one corpus and cut it into 12 non-overlapping slices from the same distribution. On the first slice we generate candidate predicates and keep the best one, f^* (§3.3). We then rescore that same f^* on the remaining held-out slices, which the LLM never saw. This is the whole test: a predicate is only credited if it keeps working on fresh text, not just on the text that produced it, this is what we call Interp_B. Because a single held-out slice gives just one score per weight (and that score depends on which text we happened to draw) we further use the other remaining $K{=}10$ separate held-out slices. Each weight then has ten held-out scores instead of one, and we summarise them by their average and by how much they move from slice to slice (the standard error), i.e. how noisy the score is. This gives us two measures that we rely on throughout (§4.2). The *robust* rate counts a weight as interpretable only when we are confident its average held-out score clears the bar, not just on a lucky slice. The *reliability* says how much of the spread in scores across a model's weights reflects a real property of the weights, rather than noise from which text we happened to test on. Both are stricter than a single check, and both test the one thing we care about: whether a predicate works off the corpus that produced it.

4. Results

Table 1. Pooled per-weight interpretability rates. Each entry is the percentage of a model's sampled nonzero weights that pass the test (±1 SEM). **Interp**$_A$: the weight's predicate is scored on the same text it was written from (in-sample). **Interp**$_B$: the same predicate, scored instead on separate held-out text. **Robust**: the predicate is re-scored on $K{=}10$ held-out slices, and the weight counts only if its average held-out score is both clearly interpretable and reliably above a "not-interpretable" baseline.

Model	Interp_A	Interp_B	Robust
Gao et al. (2025)	37.3 ± 3.9%	35.3 ± 3.9%	**15.0 ± 1.7%**
Drori (2026), sparse	21.4 ± 1.5%	17.2 ± 1.4%	**9.6 ± 1.1%**
Drori (2026), dense	13.2 ± 1.4%	9.0 ± 1.2%	**1.5 ± 0.6%**
Pythia-70m	12.0 ± 0.9%	5.0 ± 0.6%	**0.4 ± 0.3%**

4.1. Sparse models are significantly more interpretable than comparable dense models

On a per-weight basis, the sparse models are far more interpretable than dense models of comparable size. Throughout we summarise a model by its *robust rate*: the fraction of weights whose interpretation holds up on held-out text, credited only when a weight's average score across the $K{=}10$ held-out slices reliably clears the interpretability bar (§3.5). At the canonical settings ($T = 0.75$, coverage cap $c = 0.5$), the two sparse models sit far above the dense controls — roughly 15% and 10% of their non-zero weights are robustly interpretable, against about 1% for the dense model and essentially none for the pretrained dense control (Table 1). This ordering is not an artifact of one threshold or one judge: it holds within every impact bin below and survives every robustness check we run (Appendices A.4–A.5).

Where do the interpretable weights sit? Binning each model's weights by the size of their causal effect $|\Delta\text{CE}|$

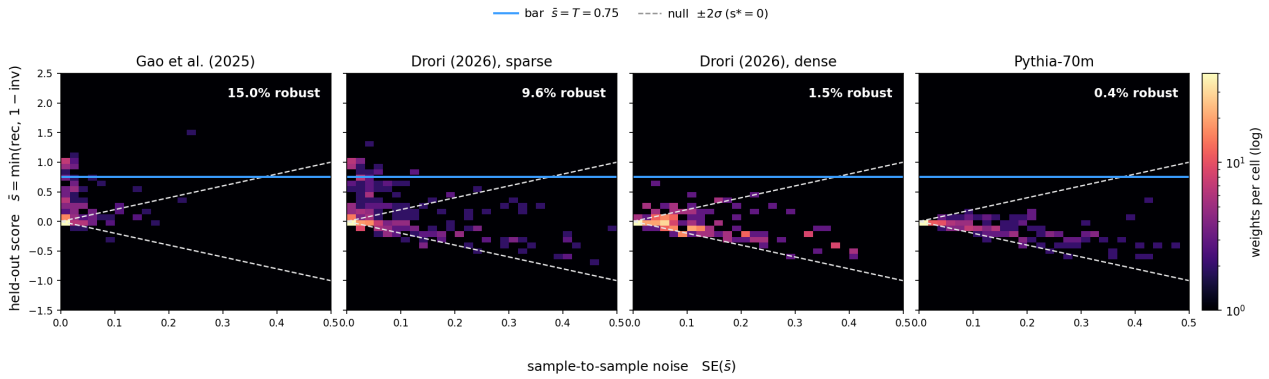

Figure 5. Robust interpretability on held-out data. Each weight is re-scored on $K{=}10$ disjoint held-out slices: the vertical axis is its mean score $\bar{s} = \min(\text{rec},\ 1 - \text{inv})$ and the horizontal axis is how much that score varies between slices (its standard error). Color is the log density of weights, showing the *robustly interpretable* weights above the interpretability bar (blue, $\bar{s} \geq 0.75$) *and* outside the "not interpretable" $\pm 2\sigma$ null cone (dashed). The percentage is the robust rate per model.

(Figure 1) shows two things. The sparse models place a real population of weights in the high-impact range ($|\Delta\text{CE}| \geq 10^{-4}$), and that is exactly where interpretability is densest — roughly a quarter of their high-impact weights are robustly interpretable. The dense models barely populate that range at all, and what little they place there generalises poorly; the pretrained control reaches no high-impact weights in our sample. The right-most column of Figure 1 ($= 0$) counts parameters that are exactly zero by construction, and is excluded from all rates.

What the $|\Delta\text{CE}|$ range means. Per-weight ablation effects span four orders of magnitude in all four models, from above 10^{-3} at the high end to below 10^{-6} at the low end. The lowest bin ($< 10^{-6}$) is at the noise floor of the cross-entropy measurement and contributes essentially nothing to the model's loss; weights there can score high on a predicate by accident, since any predicate's effect on a near-zero ΔCE is dominated by numerical noise rather than the weight's actual function. The coverage gate ($p \leq c = 0.5$ in Algorithm 1) rules out the most flagrant version of this failure mode — predicates that fire almost everywhere and "recover" the ablation by trivially restoring the weight at most positions — but the noise floor remains the reason the lowest CE bin is not where the interpretable population lives. The higher-impact bins are where sparse and dense models part ways: a sparse model places significant probability mass there and most of it is interpretable; a dense model places little mass there and what it does place generalises worse off-corpus.

4.2. Sparse-model scores are properties of the weight, not the corpus

The rates above rest on held-out text, but we can ask a sharper question of the same data: not *how many* weights pass, but *how much* each weight's score depends on which

Gao et al 2025

```
blocks.6.mlp.W_in[690,1203]
''Import or From keywords''
|ΔCE| 4.2e-04   rec 100%   inv 0%   spec 99%
def condition(tokens, pos):
return tokens[pos].strip() in
['import', 'from']
```

Prompt 1
weight ablation:
...olute import,˜div...
regex ablation:
...olute import,˜div...

Drori 2026 (dense)

```
blocks.1.mlp.c_fc.weight[126,75]
''conjunction_suppression_at_boundary''
|ΔCE| 6.1e-05   rec 96%   inv 10%   spec 86%
def condition(tokens, pos):
return tokens[pos] in [',', '.'] and
pos > 2
```

Prompt 1
weight ablation:
...˜can˜rest˜here,"˜she...
regex ablation:
...˜can˜rest˜here,"˜she...

Pythia-70m (dense)

```
blocks.0.mlp.W_in[8,1226]
''technical_and_structured_delimiters''
|ΔCE| 3.3e-05   rec 79%   inv 19%   spec 61%
def condition(tokens, pos):
current_token = tokens[pos]
delimiters = {'\\', '{', '.', ' "', ','}
return current_token in delimiters
```

Prompt 1
weight ablation:
...-collar,˜younger˜and...
regex ablation:
...-collar,˜younger˜and...

Figure 6. Example weights and their LLM-generated predicates. Each row gives a weight's identifier and predicate code, with a high-KL prompt comparing where the weight fires (weight ablation) against where the predicate evaluates true (regex ablation). Rows show, top to bottom, a Gao-sparse, a Drori-dense, and a Pythia-70m weight.

held-out sample it happened to be measured on. If a score reflects the weight, redrawing the sample should barely move it; if it reflects the corpus, it should change.

Re-scoring every weight on $K{=}10$ disjoint held-out splits separates the two. The spread of scores across a model's weights mixes genuine differences between weights with sample-to-sample wobble in each weight's own score; subtracting the wobble leaves the part that survives resampling, whose share of the total we call the *reliability* (Figure 13). It falls steeply from the sparse models to the dense ones: on the sparse models almost all of the spread is real (around

85–90%), so a weight's score is essentially a stable property of that weight; on the pretrained dense control barely half survives (47%), meaning most of what looks interpretable there is decided by which split happened to be drawn.

This instability does more than add noise — it inflates the dense models' apparent interpretability, and removing it makes them collapse. Our headline robust rate keeps a weight only if its held-out score clears the bar *and* is confidently above a "not-interpretable" baseline, not merely on a lucky split (Figure 5). Under this test the sparse models retain a sizeable interpretable population while the dense controls fall to near zero (Table 1): a single held-out check had credited the dense models with a few percent, but almost none of it survives resampling.

Every held-out measure tells the same story (Table 1): the held-out pass rate, the reliability, and the robust rate all give the same ordering, and correcting for sampling noise *widens* the sparse–dense gap rather than narrowing it. The sparse models carry interpretability that belongs to their weights; the dense models' belongs largely to the corpus it was measured on.

4.3. What the predicates look like.

The LLM-generated predicates cluster into a few structural patterns across all four models. The most common form is a single-token lookup: "is the focus token a digit," "is it one of {`th`, `ode`, `b`}," "is it a mid-word lowercase fragment." A smaller fraction conditions on the immediate neighbour: "previous word ends in `-ed`," "next token is an inflectional suffix."[†] Even the longer predicates rarely reach beyond two or three adjacent tokens, and we do not observe predicates that maintain state across a prompt (e.g. "inside an open string literal"). The pattern is consistent with weights that act locally in token-context, even when the higher-level circuits they participate in are not. Figure 6 shows three representative weights (one Gao sparse, one Drori dense, one Pythia) with the predicate code and a few high-KL prompts where the weight fires; predicate length and further class statistics across all four models are in Appendix A.2.

4.4. Threshold sensitivity and pipeline ablations

The interpretable rates reported above depend on a threshold T that is inherent to the metric and on two pipeline hyperparameters — the choice of LLM and the number of candidate predicates N generated per weight. We treat these separately: T is a core property of the interpretability measure, whereas the LLM and N are knobs of our evaluation pipeline that we ablate to confirm the cross-model ordering does not depend on them.

Threshold sensitivity. The interpretable rate depends on the predicate-pass threshold T used to convert per-weight scores into a binary interpretable/not-interpretable judgment. As T varies, absolute rates shift monotonically, but the cross-model ordering (Gao > Drori sparse > Drori dense > Pythia) is preserved across the full sweep we tested (Appendix A.5).

LLM choice and candidate budget. The choice of LLM and the number of LLM calls per weight N are hyperparameters of our pipeline rather than properties of the underlying weights. We re-run the full evaluation under four different LLMs (Gemini 3 Flash, Claude Sonnet 4.5, GPT-5, GPT-4o): absolute rates at $T = 0.75$ differ by up to $\sim$30 percentage points, with stronger LLMs higher, but the rate-vs-T shape and the cross-model rank order are preserved under each LLM (Appendix A.5). Rates also rise with N and saturate by $N \approx 100$ on the sparse models, while the dense baselines stay low across the entire sweep $N \in \{1, 3, 5, 10, 20, 100, 150\}$ (Appendix A.4). The cross-model ordering Gao > Drori sparse > Drori dense > Pythia therefore holds under each LLM and under each candidate budget.

5. Case Studies

To ground the rates of §4 in concrete examples, we work through one MLP neuron of the Gao et al. sparse code transformer. The neuron fires predominantly on numeric tokens at the activation level; applied to its nine nonzero input weights, our pipeline assigns a separately-interpretable predicate to each, and the predicates cluster into multiple digit-count categories — single-digit, two-digit, three-digit, generic and other gates that together compose the activation-level "detects digits" label (Figure 7). The full decomposition is in Appendix A.6, and two further worked examples are in Appendices A.7 (string-closing context dependence) and A.8 (a Drori-sparse speech-verb neuron decomposed into five functional sub-roles).

6. Conclusion

We find that a substantial fraction of nonzero weights in weight-sparse transformers admit individually interpretable explanations of when they are active, whereas the corresponding fraction in dense models is nearly zero. The cross-model ordering is preserved across thresholds, candidate budgets, and choice of auto-interp LLM. The interpretable rate also seems to depend on a weight's causal impact. Weights with larger ablation effects are more likely to admit explanations our pipeline can recover, while weights with very small ablation effects are interpreted less reliably, sug-

[†]These phrases are the auto-interp LLM's own one-line descriptions; each is paired with an executable Python predicate (e.g. `tokens[pos] in {"ed","ing","s","es","ly"}` for the inflectional-suffix example.

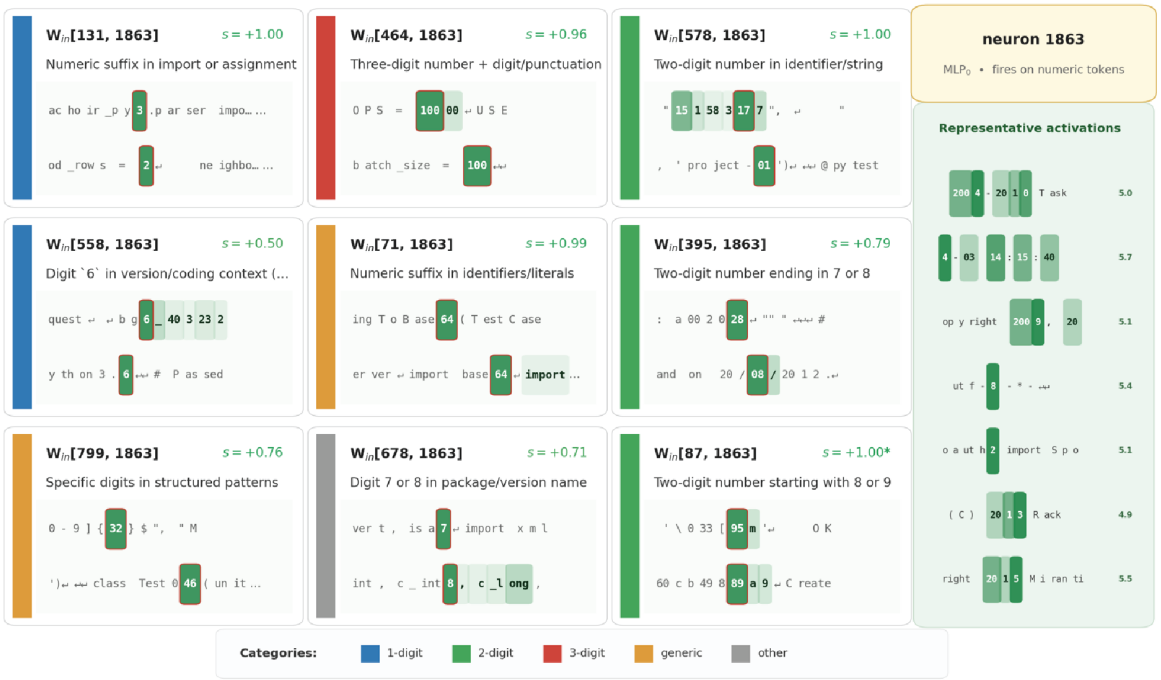

Figure 7. A neuron in the Gao et al. 2025 model that fires on numeric tokens. Each of the neuron's nine nonzero input weights is labelled with the LLM-generated interpretability score. Note: * means that $|\Delta \mathrm{CE}| < 0.00001$.

gesting that low-impact weights either contribute too little signal for the auto-interp LLM to characterize or play roles that the current predicate language cannot express. Three case studies show what the aggregate rates correspond to in practice. We find per-digit-count gates inside a digit-detector neuron, look-back-dependent functions inside the string-closing circuit, and a speech-verb neuron that decomposes at the weight level into five parallel functional roles (speech-verb activators, mental-verb suppressors, a modal suppressor, punctuation gates, and specific-token gates). Combined with the trivial interpretability of the zero weights, the great majority of parameters in these sparse models admit per-weight explanations, while the same is not true of dense models.

Limitations. Our measurements tell us *when* a weight is active, not *what* it computes once active. A function that correctly predicts the inputs on which a weight matters is informative, but it is not a complete account of the weight's role in the model. The numbers we report should be read as evidence that weights in sparse transformers are individually identifiable functional units, not as a complete characterization of what each one does. Another limitation is that our numbers are a lower bound on what the method could in principle find. Our function language is Python predicates over token context, which cannot express semantic features (e.g., whether a span refers to a person) and cannot easily express features that span multiple tokens or require state across the prompt. A weight whose role depends on such structure would be filed as not interpretable here, even if it has a clean account of when it matters that our language simply cannot write down. Finally, our held-out corpus is drawn from the same pretraining distribution as the evaluation corpus; whether these descriptions survive distribution shift is a separate question out of scope in this paper.

Future work. Three extensions follow naturally. First, the function language could be extended beyond token-local tests (for instance to predicates that reference parser state, BPE-segmentation features, or the outputs of learned probes) which should recover weights whose domain of action the current language cannot describe. Second, the fraction of robustly interpretable weights is a quantity one can train against, prune by, or use to compare architectures, in the role that token-level perplexity plays for capability. Third, applying the same method to larger sparse-trained transformers (ideally ones with non-trivial multi-step behavior) will reveal whether per-weight interpretability is a property of these particular small models or of sparsity training itself.

7. Acknowledgements

This work was funded by Pivotal Research. Arnau Marin-Llobet is supported by Coefficient Giving and the RCC-Harvard Fellowship. Thanks to Francisco Ferreira da Silva, Zheng (Zack) Hui, Reilly Haskins, Motahareh Sohrabi, Yonatan Belinkov and Demba Ba for helpful discussions.

References

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. Pythia: A suite for analyzing large language models across training and scaling. In *International conference on machine learning*, pp. 2397–2430. PMLR, 2023.

Braun, D., Bushnaq, L., Heimersheim, S., Mendel, J., and Sharkey, L. Interpretability in parameter space: Minimizing mechanistic description length with attribution-based parameter decomposition. *arXiv preprint arXiv:2501.14926*, 2025.

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askell, A., Lasenby, R., Wu, Y., Kravec, S., Schiefer, N., Maxwell, T., Joseph, N., Hatfield-Dodds, Z., Tamkin, A., Nguyen, K., McLean, B., Burke, J. E., Hume, T., Carter, S., Henighan, T., and Olah, C. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Bushnaq, L., Braun, D., and Sharkey, L. Stochastic parameter decomposition. *arXiv preprint arXiv:2506.20790*, 2025.

Chrisman, B., Bushnaq, L., and Sharkey, L. Identifying sparsely active circuits through local loss landscape decomposition. *arXiv preprint arXiv:2504.00194*, 2025.

Drori, J. Weight-sparse circuits may be interpretable yet unfaithful. https://www.lesswrong.com/posts/sHpZZnRDLg7ccX9aF/weight-sparse-circuits-may-be-interpretable-yet-unfaithful, 2026.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Elhage, N., Hume, T., Olsson, C., Nanda, N., Henighan, T., Johnston, S., ElShowk, S., Joseph, N., DasSarma, N., Mann, B., Hernandez, D., Askell, A., Ndousse, K., Jones, A., Drain, D., Chen, A., Bai, Y., Ganguli, D., Lovitt, L., Hatfield-Dodds, Z., Kernion, J., Conerly, T., Kravec, S., Fort, S., Kadavath, S., Jacobson, J., Tran-Johnson, E., Kaplan, J., Clark, J., Brown, T., McCandlish, S., Amodei, D., and Olah, C. Softmax linear units. *Transformer Circuits Thread*, 2022a. https://transformer-circuits.pub/2022/solu/index.html.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022b.

Ferrando, J., Sarti, G., Bisazza, A., and Costa-Jussà, M. R. A primer on the inner workings of transformer-based language models. *arXiv preprint arXiv:2405.00208*, 2024.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Gao, L., Rajaram, A., Coxon, J., Govande, S. V., Baker, B., and Mossing, D. Weight-sparse transformers have interpretable circuits. *arXiv preprint arXiv:2511.13653*, 2025.

Gurnee, W., Nanda, N., Pauly, M., Harvey, K., Troitskii, D., and Bertsimas, D. Finding neurons in a haystack: Case studies with sparse probing. *arXiv preprint arXiv:2305.01610*, 2023.

Janiak, J., Mathwin, C., and Heimersheim, S. Polysemantic attention head in a 4-layer transformer. https://www.alignmentforum.org/posts/nuJFTS5iiJKT5G5yh/polysemantic-attention-head-in-a-4-layer-transformer, 2023.

Lindner, D., Kramár, J., Farquhar, S., Rahtz, M., McGrath, T., and Mikulik, V. Tracr: Compiled transformers as a laboratory for interpretability. *Advances in Neural Information Processing Systems*, 36:37876–37899, 2023.

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Olah, C., Turner, N. L., and Conerly, T. A toy model of interference weights. Transformer Circuits Thread, July 2025. URL https://transformer-circuits.pub/2025/interference-weights/index.html. Informal note.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Orgad, H., Wei, B., Zheng, K., Wattenberg, M., Henderson, P., Goldfarb-Tarrant, S., and Belinkov, Y. Large language models generate harmful content using a distinct, unified mechanism. *arXiv preprint arXiv:2604.09544*, 2026.

Pearce, M. T., Dooms, T., Rigg, A., Oramas, J. M., and Sharkey, L. Bilinear mlps enable weight-based mechanistic interpretability. *arXiv preprint arXiv:2410.08417*, 2024.

Team, C. Codeparrot dataset. https://huggingface.co/datasets/transformersbook/codeparrot, 2022.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

Weiss, G., Goldberg, Y., and Yahav, E. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021.

Yu, M., Wang, D., Shan, Q., Reed, C. J., and Wan, A. The super weight in large language models. *arXiv preprint arXiv:2411.07191*, 2024.

A. Appendix

A.1. Corpus Selection Additional Details

We use CodeParrot (Team, 2022) as a proxy for the original Python pretraining data of the Gao et al. (2025) model, which is not publicly released, and stream The Pile (Gao et al., 2020) for Pythia-70m, which matches the pretraining distribution but not the exact training sequences. For the Drori (2026) sparse and dense models we use SimpleStories (Drori, 2026), which is the actual training corpus.

To verify that our chosen corpora are reasonable approximations of each model's training distribution, we compare the cross-entropy loss of each model on its evaluation corpus against the loss on deliberately mismatched corpora. Table 2 reports the comparison. The Gao et al. model achieves a CE of 2.34 on CodeParrot, but its loss rises sharply on out-of-distribution text. SimpleStories English yields a CE of 3.70, and French Wikipedia yields 4.50, both substantially higher than the in-distribution CE. Likewise, the Drori sparse model achieves a CE of 2.98 on SimpleStories, well below what either model achieves outside its training domain. We take the size of the in-distribution-versus-out-of-distribution gap as evidence that CodeParrot and SimpleStories are close enough to each model's training distribution to serve as evaluation corpora for our pipeline.

Table 2. Cross-entropy loss of each model on in-distribution and out-of-distribution corpora. The large gap between in- and out-of-distribution CE indicates that our chosen evaluation corpora are reasonable approximations of each model's training distribution.

Model	Corpus	CE Loss	In-dist?
Gao et al. (2025)	CodeParrot (Python)	2.34	Yes
Drori (2026), sparse	SimpleStories	2.98	Yes
Gao et al. (2025)	SimpleStories (English)	3.70	No
Gao et al. (2025)	French Wikipedia	4.50	No

The per-weight ablation effect $|\Delta\mathrm{CE}|$ spans multiple orders of magnitude across all models, from $\sim 10^{-3}$ down to below 10^{-6} (Figure 1).

Figure 8. Another representative MLP weight from Gao et al. (2025) sparse code transformer.

A.2. Distributions of recovery and inverse

Figures 10 and 11 show the distributions of best-of-N recovery and inverse scores across all sampled weights, sorted within each model. These curves give a more complete view of the behavior summarized by the headline interpretability rates. The sparse models have a long tail of weights that pass both the recovery and inverse criteria, while the dense baselines reach the canonical thresholds only on a much smaller slice. Negative inverse values for the sparse models reflect predicates whose negation raises loss above the fully-ablated baseline, which is direct evidence that the predicate is identifying a real, localized effect rather than partitioning a uniform contribution.

Comparing Figure 10 to the held-out version in Figure 11, the gap between the sparse and dense models widens. The sparse

Algorithm 1 Per-weight conditional-zero pipeline.

Input: Model M, weight w, corpus X, LLM $\mathcal{A}$, cap c.
Output: Best predicate f^* and its scores (r^*, v^*, p^*, s^*).

Compute the weight's effect across the corpus. Run M and $M_{w=0}$ on X.
  Compute $\Delta\mathrm{CE} \leftarrow \mathrm{CE}_{\varnothing} - \mathrm{CE}_0$, the corpus-level cross-entropy gap.
  Compute KL_p at every position; keep the top 90% of cumulative KL positions.
Propose candidates. Pass the selected positions to $\mathcal{A}$ and sample K predicates $f_1, \ldots, f_K$.
Score each candidate.
for $k = 1, \ldots, K$ **do**
  $p_k \leftarrow \frac{1}{|X|} \sum_{x \in X} [f_k(x)]$ ▷ *coverage*: fraction of positions where f_k fires
  if $p_k = 0$ **or** $p_k = 1$ **or** $p_k > c$ **then**
    $s_k \leftarrow -\infty$ ▷ trivially empty / broad: ruled out
    continue
  end if
  Build M_{f_k}: ablate w everywhere, restore it where f_k fires.
  Build $M_{\neg f_k}$: ablate w everywhere, restore it where f_k does *not* fire.
  $r_k \leftarrow 1 - \frac{\mathrm{CE}_{f_k} - \mathrm{CE}_0}{\Delta\mathrm{CE}}$ ▷ *recovery*: effect explained by restoring at f_k's firing positions
  $v_k \leftarrow 1 - \frac{\mathrm{CE}_{\neg f_k} - \mathrm{CE}_0}{\Delta\mathrm{CE}}$ ▷ *inverse*: effect explained by the predicate's complement
  $s_k \leftarrow \min(r_k,\ 1 - v_k)$ ▷ *interpretability score*
end for
return $f^* \leftarrow f_{k^*}$ where $k^* = \arg\max_k s_k$, and its scores (r^*, v^*, p^*, s^*).

Interpretability decision (threshold T). A weight is called *interpretable* if

$$r^* \geq 1 - T \quad \textbf{and} \quad v^* \leq T \quad \textbf{and} \quad p^* \leq c.$$

Default settings used: $T = 0.75$ and $c = 0.5$.

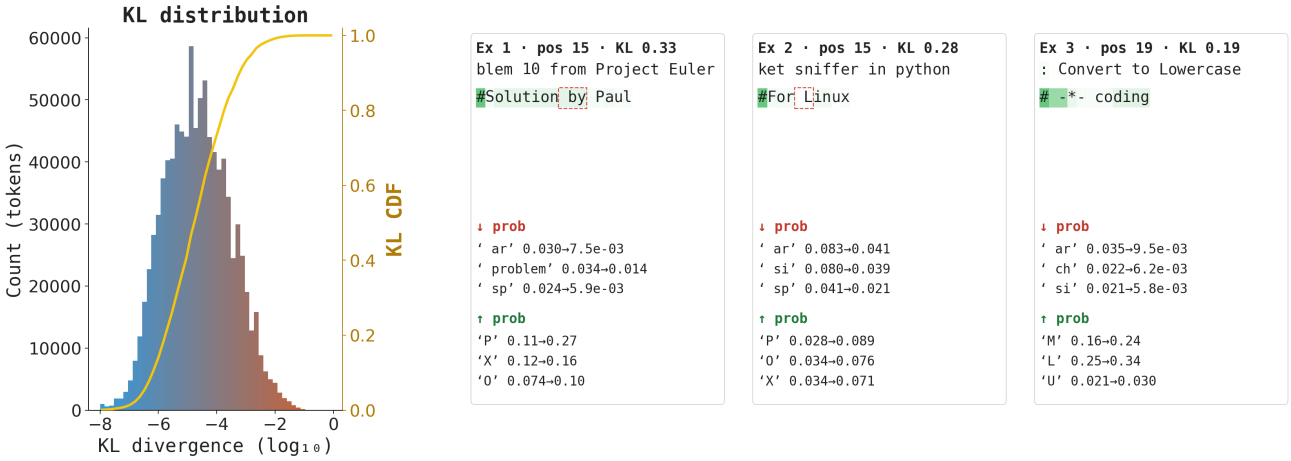

Figure 9. Representative interpretable attention weight from the Gao et al. (2025) sparse code transformer.

models retain most of their best-of-N recovery on the held-out corpus, while the dense models lose more of theirs, which suggests that predicates fitted to sparse-model weights generalize better than those fitted to dense-model weights. We read this as further evidence that the predicates for sparse weights are tracking a real property of the weight rather than overfitting to features of the corpus used to generate them.

Figure 12 reports structural statistics of the predicates themselves. Predicates remain short across all models (median ≤ 4 lines, and only 2 lines on the Drori sparse and dense models), and the predicate language is the same on sparse and dense models. The difference in interpretability rates is therefore not driven by sparse models receiving longer or more elaborate predicates.

A.3. Marginal shape of s_B vs. per-sample consistency

This appendix decomposes the gap and explains why we treat the marginal-shape effect as the dominant driver. Throughout, s_A denotes a predicate's score on the text it was generated from and s_B its score on held-out text the LLM never saw.

Per-sample consistency. Figure 14 fits an ordinary least-squares regression $s_B \sim s_A$ per model and reports the slope with a 95% CI band. A slope of 1 would indicate that the held-out score perfectly tracks the generating-text score; a slope of 0 would indicate that s_A has no predictive content for s_B and a predicate's apparent score is accidental. The fitted slopes order in the same direction as the headline rates.

Random-chance baseline. A useful way to see why the marginal effect dominates is to ask what interpretability scores a random predicate would produce. Consider a predicate that fires on a uniformly random subset of corpus positions with coverage p_{fire}, independent of the weight's actual firing pattern. Restoring the weight at those positions recovers approximately p_{fire} of the ablation effect (rec $\approx p_{\text{fire}}$); restoring at the complement recovers approximately $1 - p_{\text{fire}}$ (inv $\approx 1 - p_{\text{fire}}$); and the score is $s \approx \min(p_{\text{fire}}, p_{\text{fire}}) = p_{\text{fire}}$. Under the gate ($p_{\text{fire}} \leq 0.5$) the random-chance score is bounded above by 0.5, well below the interpretability threshold $s \geq 0.75$. A weight whose predicate scores high on the generating text purely by chance therefore sits near 0.5, and re-scoring on held-out text pulls it toward the random-chance mean rather than near 1. Dense-model weights contribute more mass near 0.5 and less mass near 1, so under held-out re-scoring they regress toward the random-chance mean and lose more than sparse-model weights do. This is why evaluating on held-out text widens the sparse–dense gap rather than narrowing it, and why the widening is mostly attributable to this marginal-shape difference rather than to a per-weight noise difference. The same effect is what the $K{=}10$ resampling in Figure 13 measures directly: subtracting sample-to-sample wobble leaves the dense models with little stable interpretable mass, since most of theirs sat near the random-chance region to begin with.

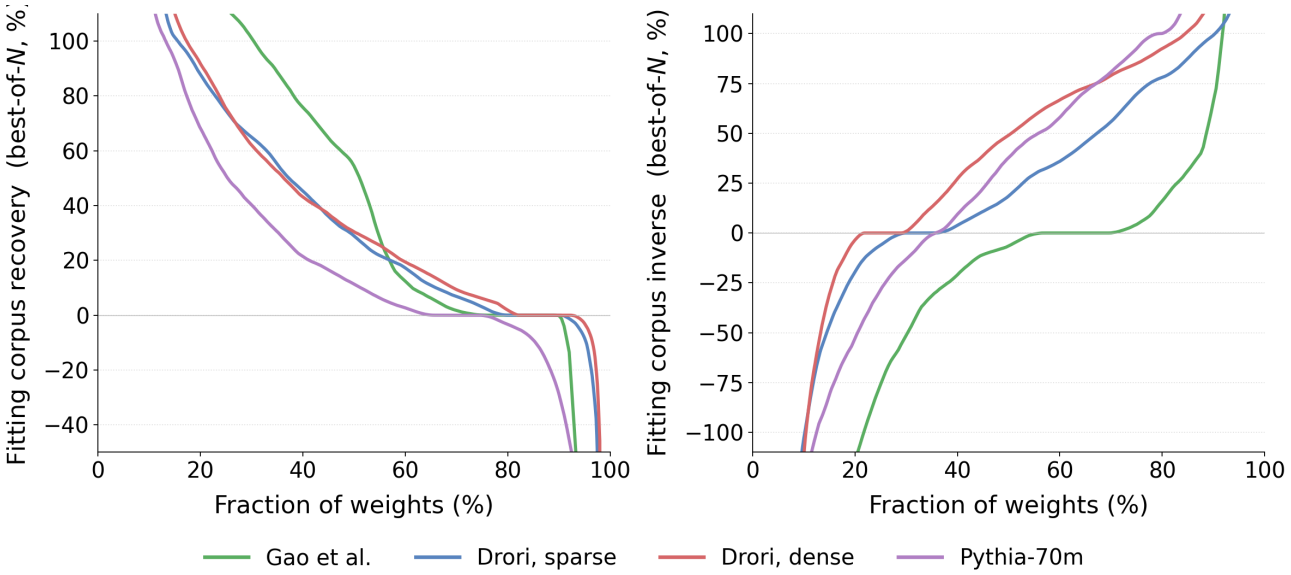

Figure 10. Best-of-N recovery (left) and inverse (right) across all sampled weights, sorted within each model so curves are monotone. Dashed lines mark the canonical thresholds.

A.4. Sensitivity to the candidate budget

A single LLM call is too noisy to call a weight interpretable. Pooling N candidates per weight and selecting the highest-scoring one tightens rates substantially. Sweeping $N \in \{1, 3, 5, 10, 20, 100, 150\}$ (Figure 15) shows interpretability rates rising monotonically with N and saturating before $N = 150$ for the sparse models. We use $N = 100$ as the default because the marginal gain per additional candidate flattens past that point and per-weight LLM cost scales linearly. The cross-model ordering is preserved at every N, so the result is not an artifact of the budget we chose.

A.5. Sensitivity to the auto-interp LLM

To check that the result is not specific to one auto-interp LLM, we re-run the pipeline with Claude Sonnet 4.5, GPT-5, and GPT-4o on the same sampled weights.

Figure 16 reports the interpretable rate at the canonical threshold $T = 0.75$ for each (model, LLM) pair, and the rate-vs-T curves at $N = 5$. The shape of the curve is consistent across LLMs. Absolute rates differ by up to $\sim$30 percentage points, with stronger LLMs higher, but the cross-model ordering (sparse > dense same-architecture > Pythia) is preserved at every T.

A.6. Digit-detection neuron: full decomposition

Per-weight tokens. Table 3 lists, for each of the neuron's nine nonzero input weights, the tokens at the weight's highest-KL positions on the CodeParrot corpus. channel. Table 3 reports the comparison.

A.7. String-closing circuit: context-dependent firing

A weight participating in the quote-closure circuit of Gao et al. 2025 suppresses the model's tendency to predict a closing quote at positions that are still inside an open string. This weight's importance depends on context and cannot be inferred from the focus token alone: when we constrain the predicate language to token-local tests (only the focus token, no look-back), the best predicate the LLM finds reaches an interpretability score of -0.98, because whether a `"` token opens or closes a string is undecidable from the token alone. Allowing a small look-back window (TOKENS[pos$-k$] for $k \leq 8$) takes the score to $+0.83$, well above the interpretability threshold. The weight is interpretable, but only under a predicate language rich enough to look back; the same weight under the token-local language would be filed under "not interpretable," and the circuit would look incomplete from the weights up. We use this finding as the empirical justification for the ± 8-token context window in §3.2.

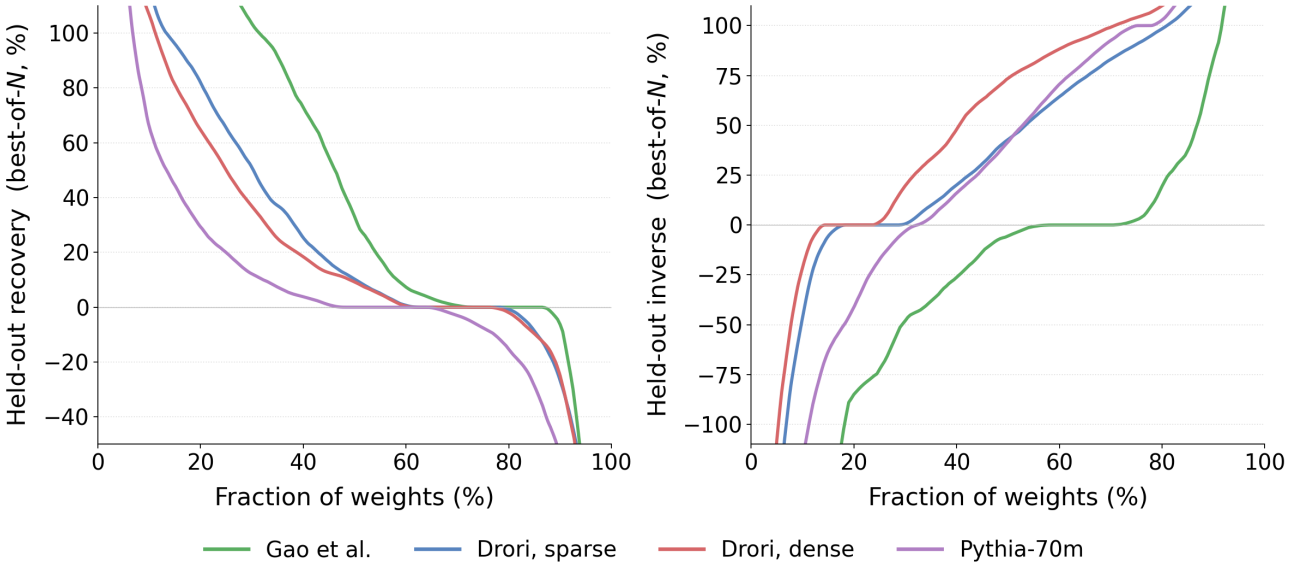

Figure 11. Best-of-N recovery (left) and inverse (right) on the held-out corpus. Same setup as Figure 10, but evaluated on a corpus never seen when generating the candidate predicates.

Table 3. For each input weight of the digit-detection neuron (`neuron 1863`), representative top tokens at the weight's highest-KL positions on the CodeParrot corpus. Tokens are decoded from the tinypython_2k tokenizer. Rows ordered by max KL.

ch (i)	max KL	weight top tokens
464	26.8	`100, 200, 300, 111`
71	22.7	`64, 50, 59, 60`
395	17.1	`28, 08, 27, 35`
799	16.1	`32, 56, 41, 46`
578	13.7	`16, 12, 13, 17`
131	8.0	`4, 2, 3, e`
678	6.0	`7, 8`
87	1.9	`90, 89, 95, f`
558	1.4	`6, 20, 8, 67`

Figure 17 expands the example to the two quote-detector neurons identified by Gao et al. 2025 — neuron 863 (double quotes) and neuron 2790 (single quotes) — with the assigned predicate, interpretability score $s = \min(\text{rec}, 1 - \text{inv})$, and two top-KL example contexts for each of their nonzero input weights, alongside a panel of representative activations of each neuron drawn from the top-90% cumulative-activation band.

A.8. Extended case study: a Drori-sparse speech-verb neuron

This appendix complements the case studies of §5 with a fully worked example on the Drori sparse model: layer-1 MLP pre-GELU neuron 2461. We show the activation-level view first, then decompose it weight-by-weight, and end by stating what the weight view adds over the activation view.

Activation-level view. On the SimpleStories corpus, the pre-GELU activation of neuron 2461 has mean -2.0, std 1.1, and a strong skew: the top positive activations sit between $+3.5$ and $+3.9$, and the top negative activations between -7.2 and -6.4. Figure 18 shows five context snippets per direction, randomly sampled from the top-95% activation band; pill saturation tracks magnitude within each panel. Positive activations (left) are concentrated on speech verbs — *announced, whispered, said, urged, says* — in their natural narrative contexts (*"…," she announced*, *"… " he whispered*). Negative activations (right) are dominated by the copulas `is` and `are`. At the activation level, this is a *speech-verb detector*: it fires up on speech-act verbs and is pushed strongly down on stative-grammar tokens, in line with what neuron-level autointerp would label.

Predicate-length comparison across models

Model	Function length (chars)			Body
	p25	p50	p75	lines
Gao et al. 2025	143	186	244	4
Drori 2026 (sparse)	131	162	227	2
Drori 2026 (dense)	133	164	208	2
Pythia-70m (dense)	166	229	272	4

What the predicates look like. They fall into a small family of structural patterns. Most are lookup-style tests on a *single* token—"is this token one of {`th`, `ode`, `b`}", "is it a digit", "is it a mid-word lowercase fragment". A smaller set conditions on the immediate neighbour (e.g. "previous word ends in *-ed*", "next token is an inflectional suffix"). Even the longer predicates rarely look beyond two or three adjacent tokens: we don't see explanations like "this token is the subject of the sentence" or "we are inside an open string literal" that would require the model to keep state across the prompt.

Figure 12. Structural statistics of the LLM-generated predicates per model: distribution of predicate length (lines of code) and predicate class (single-token lookup, neighbor test, multi-token context test).

Weight-level view. The neuron has 86 nonzero input weights. We rank them by max-KL on a 5,000-sequence SimpleStories sample and profile the top 20, applying the conditional-zero scoring of §3.3 to each individually; the remaining 66 weights have ablation effects below the noise floor of our metric on this corpus and are not analyzed here. Figure 19 shows all 20 cards, grouped by the functional role assigned by their predicate. Five clusters emerge:

- **Speech-verb activators** (positive weights): six channels whose predicates fire on speech verbs — *said, roared, whispered, called, wondered, shouted*. These are the weights that drive the neuron up on the left panel of Figure 18.
- **Mental-verb suppressors** (negative weights): three channels whose predicates fire on *thought, believed, understood, thanks*. These actively pull the neuron down on a near-neighbour class of verbs that would otherwise share representational support with speech verbs.
- **Modal-verb suppressor**: one channel firing on *would*, *could*.
- **Punctuation gates**: seven channels whose predicates fire on clause-boundary commas and periods. These contribute to the neuron's negative tail and explain a large share of the `is`/`are` suppression observed in the activation view: copulas in this corpus tend to sit one or two tokens after a clause-boundary punctuation mark.
- **Specific-token gates**: three channels with narrower predicates — a dedicated *'as'* detector, a *'called out'* pattern, and a physical-action verb gate.

Each card in Figure 19 reports recovery, inverse, and the fraction of corpus positions where the predicate fires; the strongest cards (*e.g.,* ch. 909 *'as'* detector with $\mathrm{rec} = 1.22$, $\mathrm{inv} = -0.22$) sit comfortably above the $T = 0.75$ interpretability bar. Weaker cards (*e.g.,* the modal suppressor on ch. 38 with $\mathrm{rec} = 0.27$, $\mathrm{inv} = 0.73$) do not pass the bar individually, but their predicates remain consistent with the role inferred from neighbouring weights in the same group.

What the weight view adds. At the activation level neuron 2461 reads as a single "speech-verb detector." At the weight level the same neuron implements *multiple* parallel gating decisions: it activates speech verbs, suppresses near-neighbour verb classes (mental verbs, modals) that share representational support with speech verbs, and routes through clause-boundary punctuation patterns that account for its strongest negative activations. These contributions cancel and add into the single scalar that activation-level analyses report, so they are not separable from the activation summary alone — only from the per-weight decomposition.

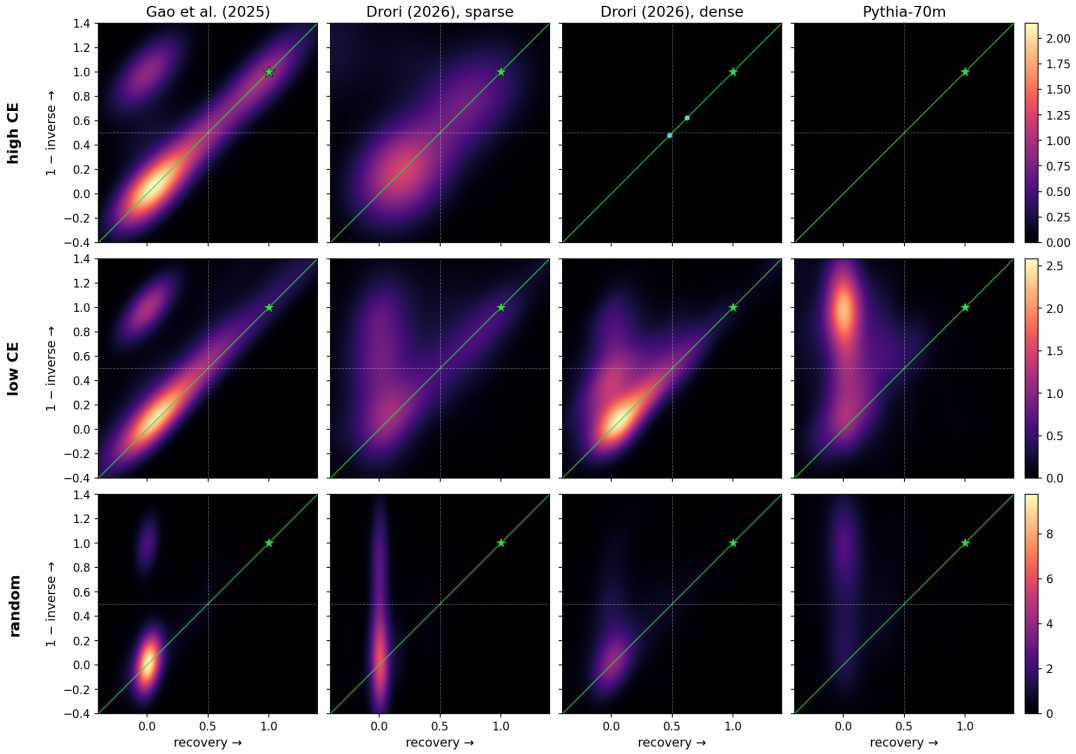

Figure 13. Reliability of per-weight interpretability scores under held-out resampling. Each weight is re-scored on $K{=}10$ disjoint held-out splits; bars show the fraction of each model's cross-weight score variance that survives redrawing the corpus (95% bootstrap CIs). Sparse-model scores are mostly stable properties of the weight; on Pythia, most of the apparent spread is sampling noise.

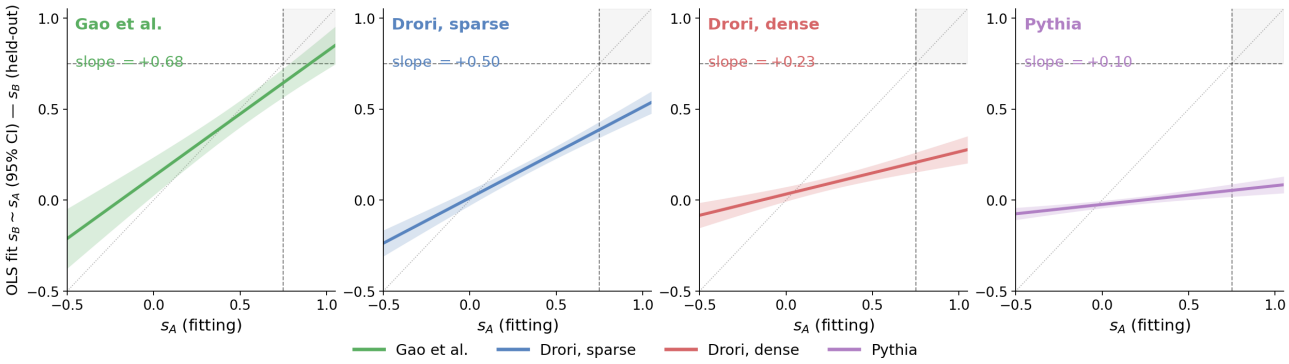

Figure 14. Held-out interpretability score $s_B = \min(\mathrm{rec}_B,\, 1 - \mathrm{inv}_B)$ across models. OLS fit $s_B \sim s_A$ per model with 95% CI; slope annotated.

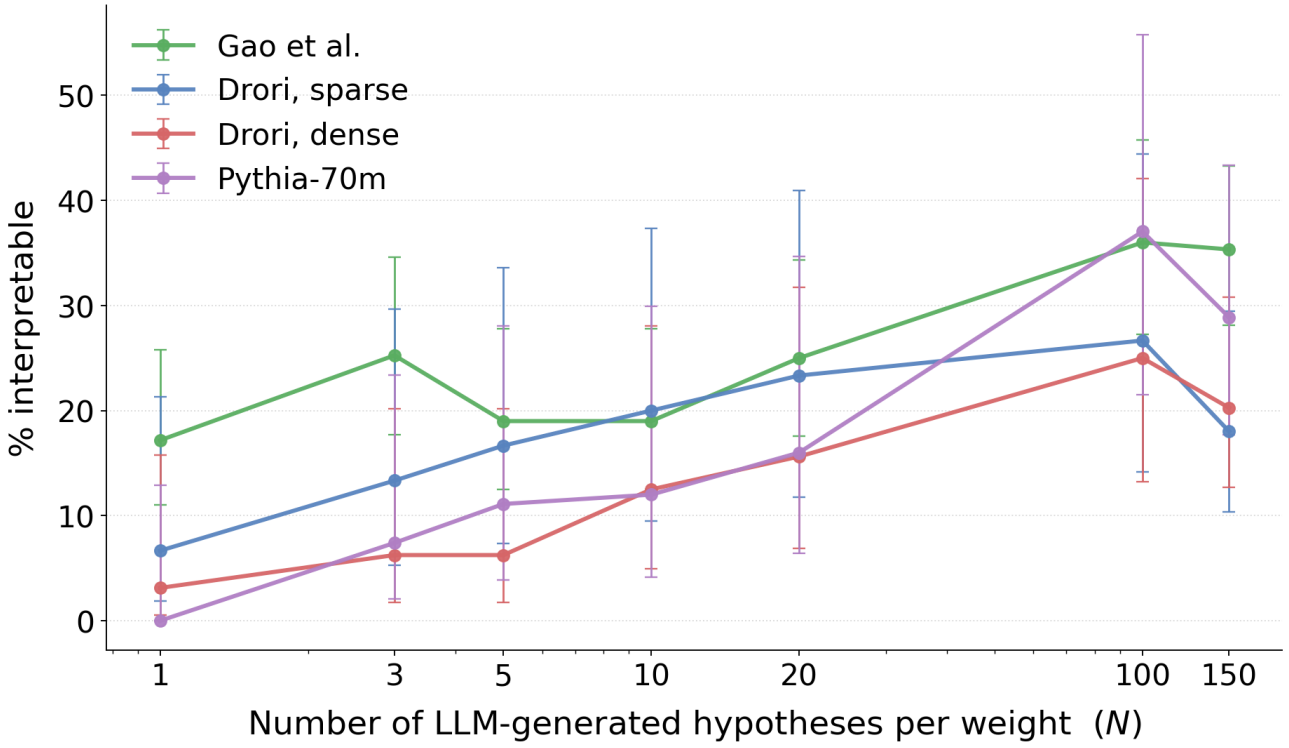

Figure 15. Interpretable rate as a function of the candidate budget N, with 95% Wilson CIs. Single-call N from 1 to 150.

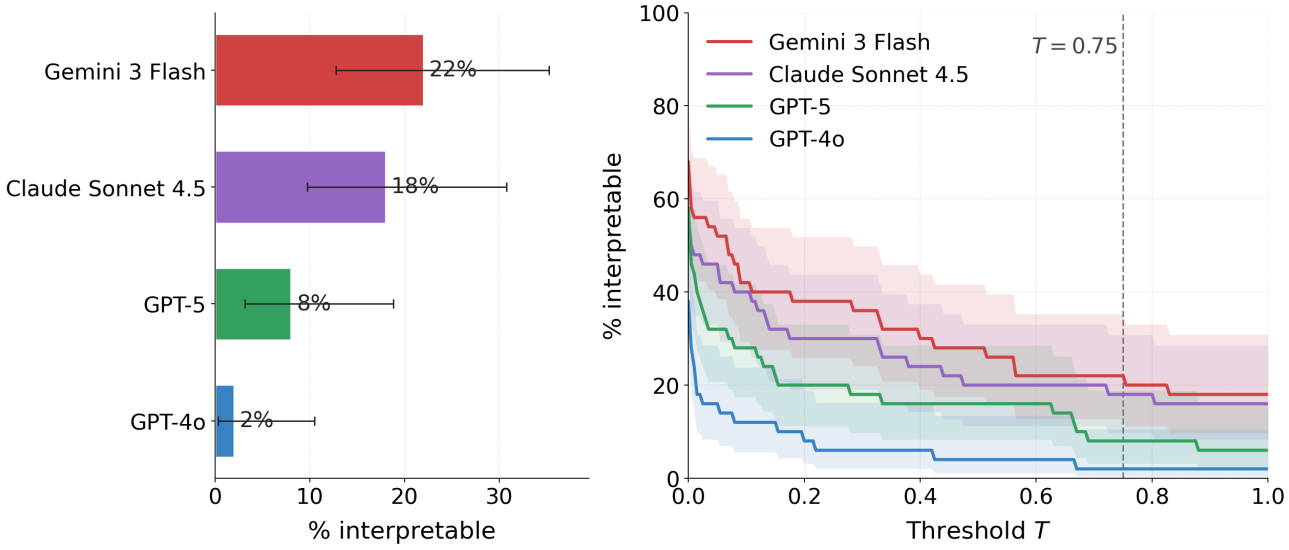

Figure 16. Interpretable rate by auto-interp LLM. **Left:** fraction of sampled weights interpretable at $T = 0.75$, with 95% Wilson CIs, on the same weights evaluated by four different LLMs. **Right:** interpretable rate as a function of threshold T.

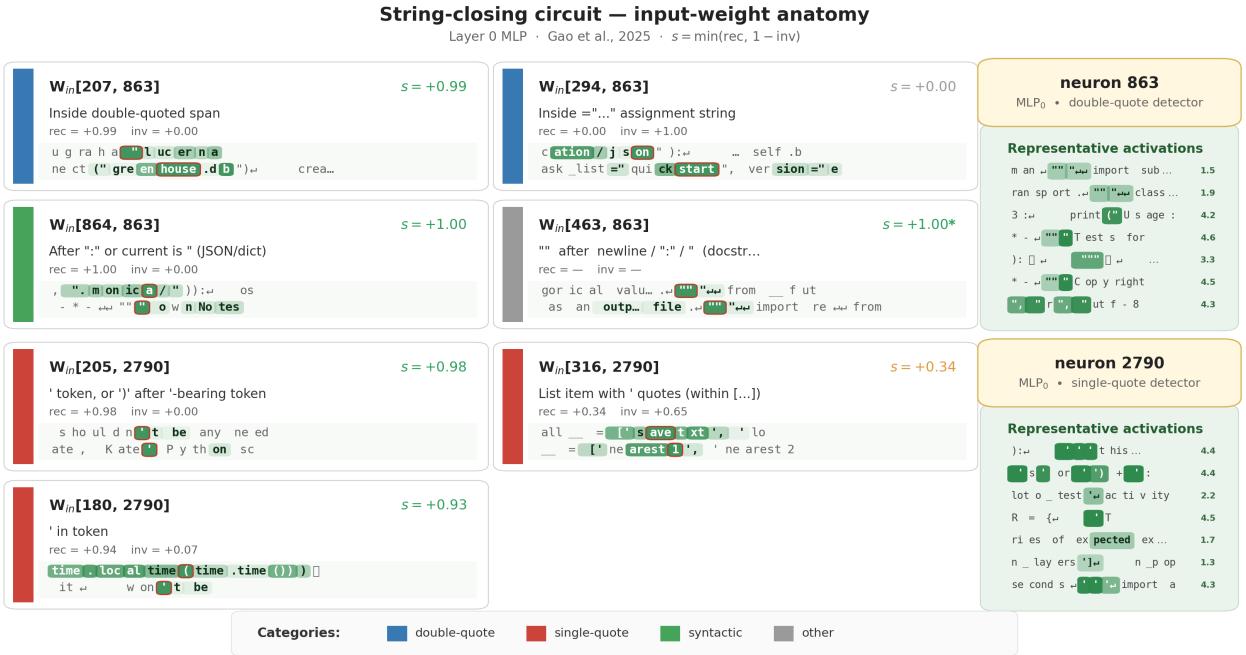

Figure 17. Quote-detector neurons from the string-closing circuit of Gao et al. 2025. **Neuron 863** (top band) fires on double quotes; **neuron 2790** (bottom band) fires on single quotes. Each card shows one nonzero input weight: the predicate, the interpretability score $s = \min(\text{rec}, 1 - \text{inv})$, the recovery and inverse values, and two top-KL example contexts (focus token outlined in red, pill saturation graded by per-token KL within the row). The right panel of each band shows representative activating contexts of the neuron, randomly sampled from the top-90% cumulative-activation band on CodeParrot.
Note: $*$ on the score marks weights whose $|\Delta\text{CE}|$ is at the noise floor.

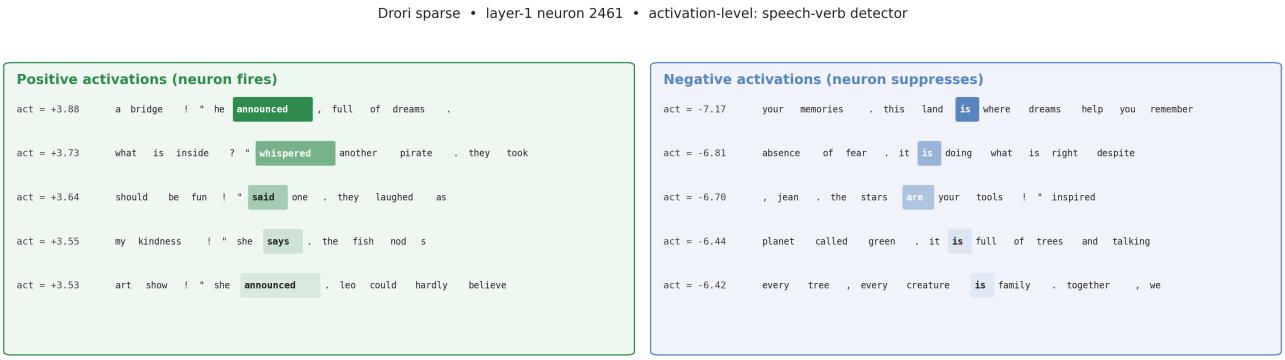

Figure 18. Activation-level view of Drori-sparse layer-1 neuron 2461. **Left:** five contexts where the pre-activation is most positive (random sample from the top-95% positive band). The neuron fires on speech-act verbs. **Right:** five contexts where it is most negative; the neuron is pushed down on copulas (`is`, `are`). Pill saturation tracks $|\text{activation}|$ within each panel.

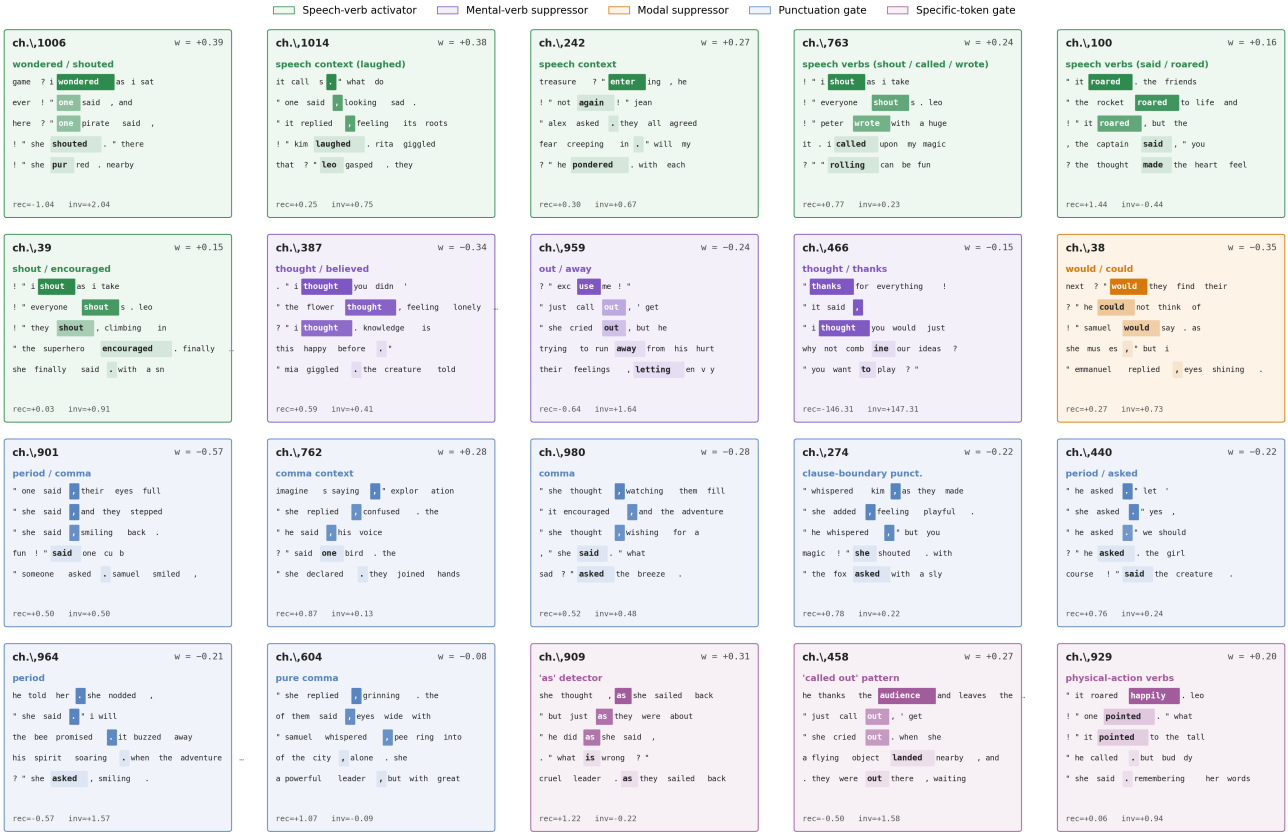

Figure 19. Weight-level decomposition of Drori-sparse layer-1 neuron 2461. The neuron has 86 nonzero input weights; the figure shows the 20 strongest by max-KL on a 5,000-sequence SimpleStories sample, grouped by the functional role assigned by their predicate. Five context snippets per card are sampled from the top-95% cumulative-KL band of that weight; pill saturation tracks the row's KL within the card.