The parameters in weight-sparse transformers are interpretable

Anonymous Author(s)
Affiliation
Address
email

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. Past work has shown however, that components can have different functions that are active on different subsets of the input distribution. In this work we test whether it is possible to understand individual weights globally, on the full training distribution. We focus on weight-sparse transformers in which we expect individual weights to be more interpretable than dense models. Here, we introduce introduce an automated LLM-pipeline that produces a short, human-readable account of when a given weight matters, verifies this account on held-out data, and applies it at scale to compare two weight-sparse transformers against two dense models. Empirically, we find that a significant percentage of nonzero weights on sparse transformers are interpretable ($17 - 35\%$), compared to $5 - 9\%$ on dense models. Our results are a proof of concept that a substantial fraction of language model weights can be interpretable, and confirms that the weights of sparse models are more interpretable that those of dense models.

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 [1–4]. 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 [5, 6].

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 [7–9], with rare exceptions [10]. Recent work on weight-sparse transformers [11] 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 its 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. [12, 13]), 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

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

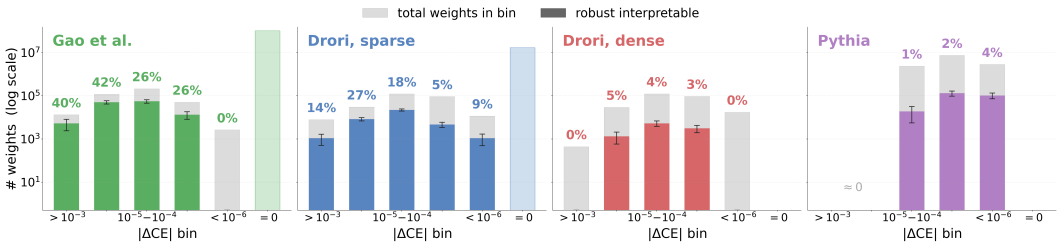

Figure 1: Per-weight causal impact distribution on a held-out distribution. Gray bars: total nonzero weights per $|\Delta\mathrm{CE}|$ bin. Colored bars within each gray bar show the fraction of weights in that bin that pass our interpretability test. The right-most column collects parameters that are exactly zero. **Pooled interpretability scores across all nonzero weights:** Gao et al. (2025) 35.3%, Drori (2026) sparse 17.2%, Drori (2026) dense 9.0%, Pythia-70m 5.0%.

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 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 [11], a sparse-and-dense SimpleStories pair [14], and Pythia-70m [15] as a dense pretrained control. We find that the pipeline assigns interpretable predicates to 17–35% of nonzero weights in the sparse models, compared to 5–9% in the dense baselines, and the gap persists when predicates are 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 [2, 16] and continued in recent bilinear-MLP [17] analyses examines weight matrices to extract circuits responsible for specific behaviors. The unit of analysis is the circuit; weights enter

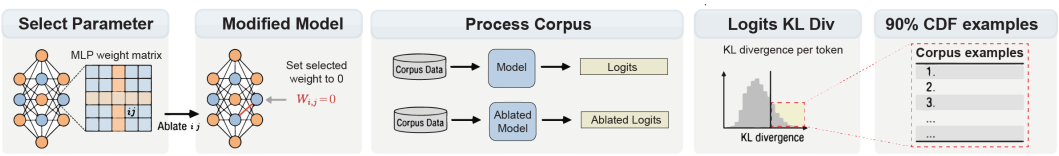

Figure 2: Per-weight interpretability pipeline. Given a target weight $W_{i,j}$ in an MLP layer, we run both the original model and an ablated copy on a corpus, measure the per-token KL divergence between them, and extract the token contexts that account for the top 90% of cumulative KL. These contexts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

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. This is powerful when the target behavior is known in advance, but it does not produce 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 [18] compiles models from a high-level program (RASP; 19), 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 concept without sacrificing language modeling performance [20]. In our setting, Gao et al. 2025 [11] 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 [14] extend 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 [11, 14]. 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 [13], SPD [21], and L3D [22] 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.

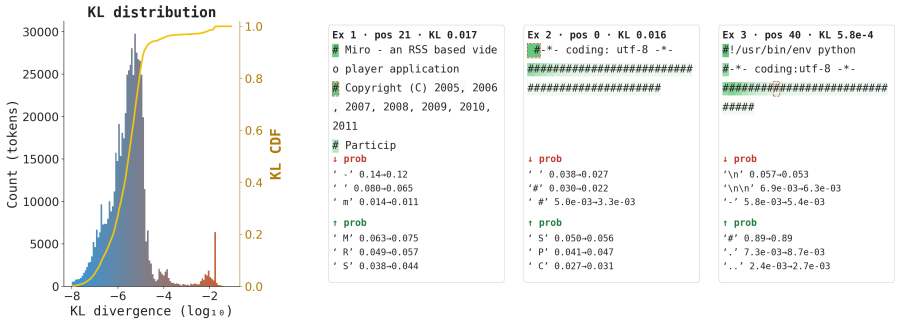

Figure 3: An interpretable weight from the Gao et al. (2025) sparse code transformer. Left: distribution of per-token KL divergence over the corpus when this weight is ablated, with the cumulative KL shown in yellow. Right: three token contexts drawn from the top of the KL distribution. Each panel shows 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 the sparse SimpleStories model. The same display as Figure 3; the affected tokens cluster around speech-verb contexts ("called","shouted", "voice", etc.).

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, and A1 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` that fires on those positions.

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, 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*:

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

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, and the symmetric trivial predicate ($f \equiv \mathrm{True}$) achieves $\mathrm{recovery} = 1$ and $\mathrm{inverse} = 0$ without saying anything about the weight. We rule these out with a *coverage gate*. Define the coverage of f as the fraction of corpus positions where it fires, $p(f) = \frac{1}{|X|}\sum_{x \in X} \mathbb{1}[f(x)]$, and reject any candidate with $p(f) \in \{0, 1\}$ or $p(f) > c$, where $c = 0.5$ is the coverage cap. Among gated candidates, we summarize recovery and inverse by the predicate's interpretability *score*,

$$\mathrm{score}(f) \;=\; \min\big(\mathrm{recovery}(f),\; 1 - \mathrm{inverse}(f)\big), \tag{2}$$

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 satisfies $\mathrm{recovery}(f^*) \geq 1 - T$, $\mathrm{inverse}(f^*) \leq T$, and $p(f^*) \leq c$ — equivalently $\mathrm{score}(f^*) \geq 1 - T$ together with the coverage gate. We use $T = 0.25$ and $c = 0.5$ throughout the paper and report sensitivity to T in Appendix A.5. Algorithm 1 gives the full pipeline; it returns the best predicate f^* together with the four scores (r^*, v^*, p^*, s^*) — recovery, inverse, coverage, and the combined score.

3.4 Models and corpora

We evaluate three training regimes. The weight-sparse code transformer of Gao et al. [11] is trained at ∼99% weight sparsity on Python code. The SimpleStories sparse-and-dense pair of Drori et al. [14] 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 [15] serves as a dense pretrained control, trained on The Pile [23]. 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

A predicate that passes the criterion on the corpus the LLM saw to generate the function could still be describing surface regularities of that corpus rather than the weight. To test these, we create two corpora A and B from the same pretraining distribution (different random seeds and no overlapping sequences), generate and select the best predicate f^* on corpus A exactly as in §3.3, and then re-evaluate f^* on corpus B — re-computing recovery, inverse, and coverage on B, with the LLM never seeing any of B. This yields three quantities per weight:

- Interp_A: fraction passing the criterion on the fitting corpus (the natural interpretable *score*).
- Interp_B: fraction passing the criterion on the held-out corpus.

Algorithm 1 Per-weight conditional-zero pipeline.

Input: Model M, target weight w, corpus X, auto-interp LLM $\mathcal{A}$, coverage cap c (default 0.5).
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**
  $p_k \leftarrow \dfrac{1}{|X|} \sum_{x \in X} \mathbb{1}[f_k(x)]$ ▷ *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 / trivially 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.
  $r_k \leftarrow 1 - \dfrac{\mathrm{CE}_{f_k} - \mathrm{CE}_0}{\Delta\mathrm{CE}}$ ▷ *recovery*: effect explained by restoring at f_k's firing positions
  $v_k \leftarrow 1 - \dfrac{\mathrm{CE}_{\neg f_k} - \mathrm{CE}_0}{\Delta\mathrm{CE}}$ ▷ *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* iff

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

Default settings used throughout the paper: $T = 0.25$ and $c = 0.5$.

We treat Interp_B as the headline number throughout the paper, since it directly tests whether a predicate generalizes off the corpus that produced it. We additionally report Interp_A to surface the over-fitting gap and to enable comparison with prior work that does not split corpora.

4 Results

4.1 Sparse models are significantly more interpretable than comparable dense models

The sparse models of Gao et al. (2025) and Drori (2026) are substantially more interpretable, on a per-weight basis, than comparable dense models. We report the fraction of weights that pass our interpretability test in Figure 1 and Table 1. Pooled across all nonzero weights at the canonical settings $T = 0.25$ and coverage cap $c = 0.5$, the held-out interpretability rate Interp_B (§3.5) is 35.3% on Gao, 17.2% on Drori sparse, 9.0% on Drori dense, and 5.0% on Pythia. The sparse–dense gap holds within each $|\Delta\mathrm{CE}|$ bin and persists under every robustness check we run (paragraphs below; Appendices A.5–A.4).

To understand the distribution of interpretable weights in more detail, we bin each model's nonzero weights by the magnitude of their ablation effect $|\Delta\mathrm{CE}|$ and compute Interp_B within each bin (Figure 1). Two patterns are visible. First, the sparse models concentrate a substantial population of weights in the higher-impact bins ($|\Delta\mathrm{CE}| \geq 10^{-4}$), where the interpretable rate is also the highest — 42% for Gao and 27% for Drori sparse in that bin. Second, the dense controls have a much lighter head in the same range and a correspondingly lower interpretable rate everywhere; on Pythia, no weight in our sample reaches $|\Delta\mathrm{CE}| \geq 10^{-4}$ at all. The right-most column of Figure 1 ($= 0$) tallies parameters that are exactly zero by construction; we exclude them from the interpretability rates above.

What the $|\Delta\mathrm{CE}|$ 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 $\Delta\mathrm{CE}$ 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 generalizes worse out-of-sample.

Table 1: Pooled interpretability rates at the canonical settings (§3.3). Reported as $\%\pm$ SEM. The headline metric is Interp_B (held-out, bold); Interp_A is the in-sample rate and is reported for reference. The final column $r(s_A, s_B)$ is the Pearson correlation between each weight's per-weight interpretability score s computed on the in-sample corpus (s_A) and on the disjoint held-out corpus (s_B), measured across all nonzero weights in the model; higher values indicate that per-weight interpretability scores are more consistent across data samples.

Model	Interp_A	Interp_B	$r(s_A, s_B)$
Gao et al. (2025)	$37.3 \pm 3.9\%$	**$35.3 \pm 3.9\%$**	$+0.525$
Drori (2026), sparse	$21.4 \pm 1.5\%$	**$17.2 \pm 1.4\%$**	$+0.252$
Drori (2026), dense	$13.2 \pm 1.4\%$	**$9.0 \pm 1.2\%$**	$+0.160$
Pythia-70m	$12.0 \pm 0.9\%$	**$5.0 \pm 0.6\%$**	$+0.100$

4.2 Held-out validation widens the sparse–dense gap.

The in-sample rate Interp_A (the fraction of weights whose best predicate passes on the corpus the LLM saw) understates the difference between sparse and dense models. Drori dense reaches 13.2% on Interp_A — not far below Drori sparse at 21.4%. Re-evaluating each predicate on a disjoint held-out corpus drawn from the same distribution (§3.5) widens the gap: Interp_B drops to 9.0% on Drori dense versus 17.2% on Drori sparse (Table 1). Equivalently, the fraction of in-sample predicates that survive held-out re-scoring ($\mathrm{Interp}_B/\mathrm{Interp}_A$) is 95% on Gao, 80% on Drori sparse, 68% on Drori dense, and 42% on Pythia. Sparse predicates therefore generalize better off-corpus than dense ones, consistent with sparse-model predicates tracking a property of the weight rather than a property of the corpus they were generated from.

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 5 shows three representative weights (one Gao sparse, one Drori dense, one Pythia) with the predicate code and three 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.

Figure 5: Example weights and their LLM-generated predicates. Each row shows a single weight's identifier, the predicate code, and three high-KL prompts comparing the weight's actual firing positions against where the predicate evaluates true. **Top:** a Gao-sparse weight that fires on the tokens `import` and `from`. **Middle:** a Drori-dense weight active around clause-boundary punctuation. **Bottom:** a Pythia-70m weight whose predicate fires on structural delimiters but whose causal impact ($|\Delta\text{CE}| = 3.3 \times 10^{-5}$) sits near the noise floor.

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). $T = 0.25$ is the canonical operating point used throughout this section.

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 $T = 0.25$ 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 aggregate rates of §4 in concrete examples, we work through one MLP neuron of the Gao et al. 2025 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 6). 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 small. The cross-model ordering is preserved across thresholds, candidate budgets, and choice of auto-interp LLM. The interpretable rate also depends 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, suggesting that low-impact weights either contribute too little signal for the auto-interp LLM to characterize or play roles that the current

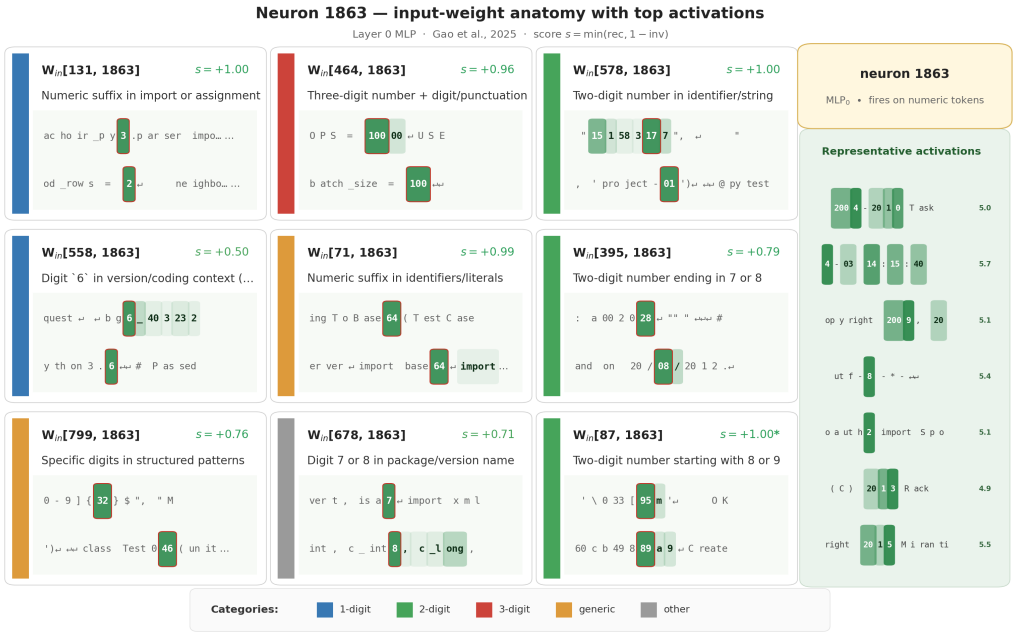

Figure 6: A neuron in the Gao et al. 2025 model that fires on numeric tokens. Each of the neuron's nine nonzero input weights is labelled with the LLM-generated interpretability score. *Note: * means that* $|\Delta\mathrm{CE}|$ *is tiny.*

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. The most important limitation is conceptual. 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 the 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. Our ablations indicate that the candidate budget $N = 100$ is sufficient under our current language, so the binding constraint is the language itself rather than the search budget. Finally, the held-out corpus is from the same pretraining distribution as the evaluation corpus. Whether explanations generalize across distribution shift is a separate question we do not address.

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 outputs of learned probes. This should recover weights whose roles 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 same role token-level perplexity plays for capability. Third, applying the same method to larger sparse-trained transformers, ideally those with non-trivial multi-step behavior, will show whether per-weight interpretability is a property of these particular small models or a property of sparsity training itself.

References

[1] Javier Ferrando, Gabriele Sarti, Arianna Bisazza, and Marta R Costa-Jussà. A primer on the inner workings of transformer-based language models. *arXiv preprint arXiv:2405.00208*, 2024.

[2] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

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

[4] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

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

[6] Adam Jermyn, Chris Olah, and Tom Henighan. Polysemantic attention head in a 4-Layer Transformer. `https://www.alignmentforum.org/posts/nuJFTS5iiJKT5G5yh/polysemantic-attention-head-in-a-4-layer-transformer`, Nov 2023. Accessed: 2024-05-26.

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

[8] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[9] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[10] Mengxia Yu, De Wang, Qi Shan, Colorado J Reed, and Alvin Wan. The super weight in large language models. *arXiv preprint arXiv:2411.07191*, 2024.

[11] Leo Gao, Achyuta Rajaram, Jacob Coxon, Soham V Govande, Bowen Baker, and Dan Mossing. Weight-sparse transformers have interpretable circuits. *arXiv preprint arXiv:2511.13653*, 2025.

[12] Chris Olah, Nicholas L. Turner, and Tom Conerly. A toy model of interference weights. Transformer Circuits Thread, July 2025. Informal note.

[13] Dan Braun, Lucius Bushnaq, Stefan Heimersheim, Jake Mendel, and Lee Sharkey. Interpretability in parameter space: Minimizing mechanistic description length with attribution-based parameter decomposition. *arXiv preprint arXiv:2501.14926*, 2025.

[14] Jacob Drori. Weight-sparse circuits may be interpretable yet unfaithful. LessWrong, February 2026. Accessed: 2026-04-29.

[15] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International conference on machine learning*, pages 2397–2430. PMLR, 2023.

[16] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

[17] Michael T Pearce, Thomas Dooms, Alice Rigg, Jose M Oramas, and Lee Sharkey. Bilinear mlps enable weight-based mechanistic interpretability. *arXiv preprint arXiv:2410.08417*, 2024.

[18] David Lindner, János Kramár, Sebastian Farquhar, Matthew Rahtz, Tom McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. *Advances in Neural Information Processing Systems*, 36:37876–37899, 2023.

[19] Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pages 11080–11090. PMLR, 2021.

[20] Nelson Elhage, Tristan Hume, Catherine Olsson, Neel Nanda, Tom Henighan, Scott Johnston, Sheer ElShowk, Nicholas Joseph, Nova DasSarma, Ben Mann, Danny Hernandez, Amanda Askell, Kamal Ndousse, Andy Jones, Dawn Drain, Anna Chen, Yuntao Bai, Deep Ganguli, Liane Lovitt, Zac Hatfield-Dodds, Jackson Kernion, Tom Conerly, Shauna Kravec, Stanislav Fort, Saurav Kadavath, Josh Jacobson, Eli Tran-Johnson, Jared Kaplan, Jack Clark, Tom Brown, Sam McCandlish, Dario Amodei, and Christopher Olah. Softmax linear units. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/solu/index.html.

[21] Lucius Bushnaq, Dan Braun, and Lee Sharkey. Stochastic parameter decomposition. *arXiv preprint arXiv:2506.20790*, 2025.

[22] Brianna Chrisman, Lucius Bushnaq, and Lee Sharkey. Identifying sparsely active circuits through local loss landscape decomposition. *arXiv preprint arXiv:2504.00194*, 2025.

[23] 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*, abs/2101.00027, 2020.

[24] CodeParrot Team. Codeparrot dataset. `https://huggingface.co/datasets/transformersbook/codeparrot`, 2022.

A Appendix

A.1 Additional details about corpus selection

We use CodeParrot [24] 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 [23] 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 [14], 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 A1 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. [1].

Table A1: 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 $|\Delta\text{CE}|$ spans multiple orders of magnitude across all models, from $\sim 10^{-3}$ down to below 10^{-6} (Figure 1).

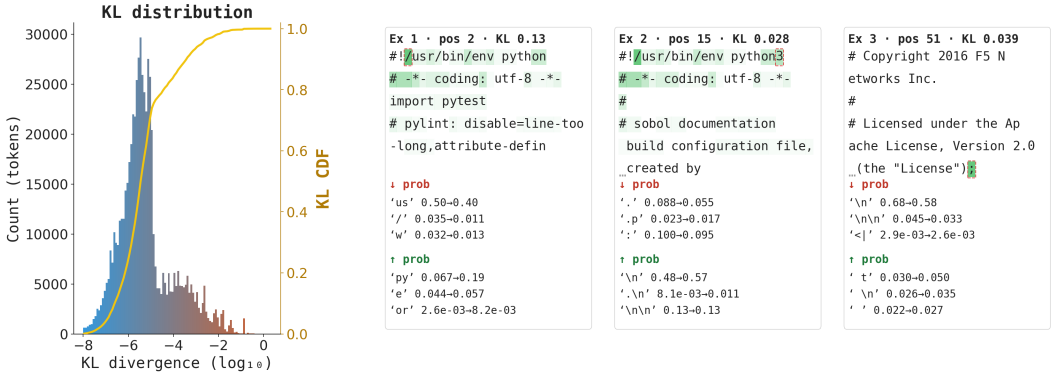

Figure A1: Another representative weight from the Gao et al. (2025) sparse code transformer.

A.2 Distributions of recovery and inverse

Figures A2 and A3 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.

[1] Relevant code and representative resources are made available at anonymous121512.github.io/

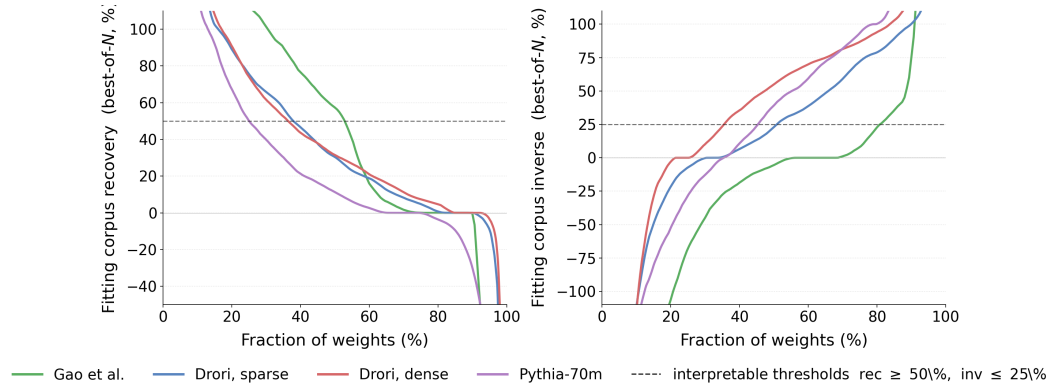

Figure A2: Best-of-N recovery (left) and inverse (right) across all sampled weights, sorted within each model so curves are monotone. Dashed lines mark the canonical thresholds (recovery ≥ 0.5, inverse ≤ 0.25).

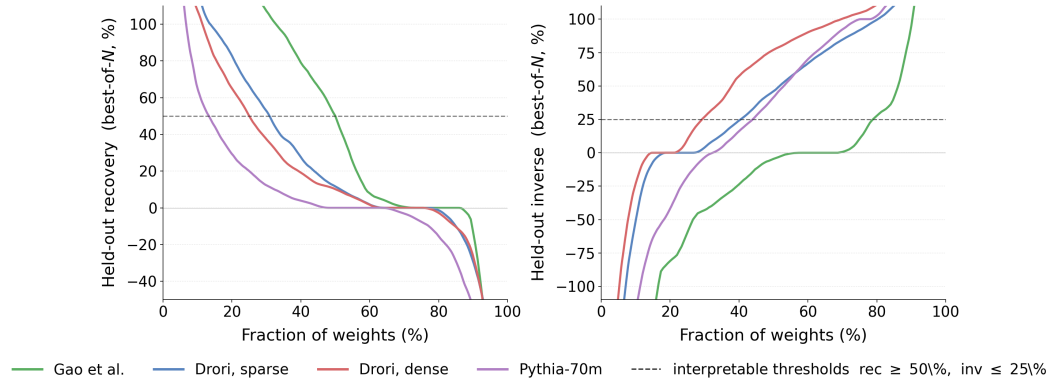

Figure A3: Best-of-N recovery (left) and inverse (right) on the held-out corpus. Same setup as Figure A2, but evaluated on a corpus never seen when generating the candidate predicates.

Comparing Figure A2 to the held-out version in Figure A3, the gap between the sparse and dense models widens. The sparse 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 A4 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.

Per-sample consistency. Figure A5 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 in-sample score; a slope of 0 would indicate that s_A has no predictive content for s_B and the predicate's apparent score on A is accidental. The fitted slopes order in the same direction as the headline rates: Gao $+0.68$, Drori sparse $+0.50$, Drori dense $+0.23$, Pythia $+0.10$.

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 A4: 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).

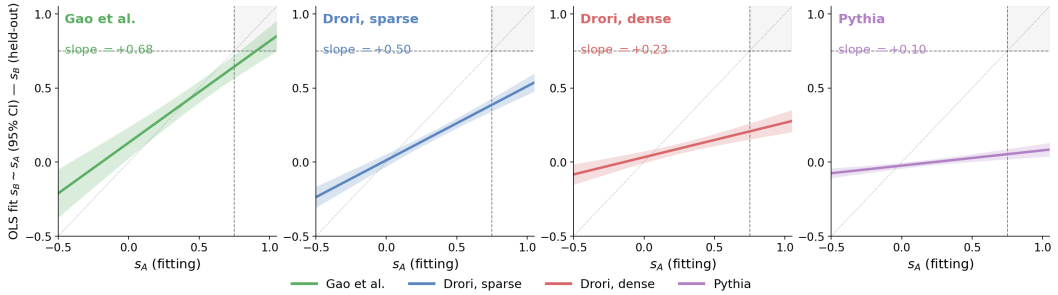

Figure A5: Held-out interpretability score $s_B = \min(\mathrm{rec}_B, 1 - \mathrm{inv}_B)$ across models. OLS fit $s_B \sim s_A$ per model with 95% CI; slope annotated.

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 A purely by chance therefore lives near 0.5 on the random-chance axis and re-scoring on B 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 on A, so when they are re-scored on B they regress toward the random-chance mean and lose more in the joint criterion than sparse-model weights do. This is why $\mathrm{Interp}_{A \cap B}$ widens the sparse–dense gap relative to Interp_A, and why the widening is mostly attributable to the marginal-shape difference rather than to a per-weight noise difference.

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 A6) 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.

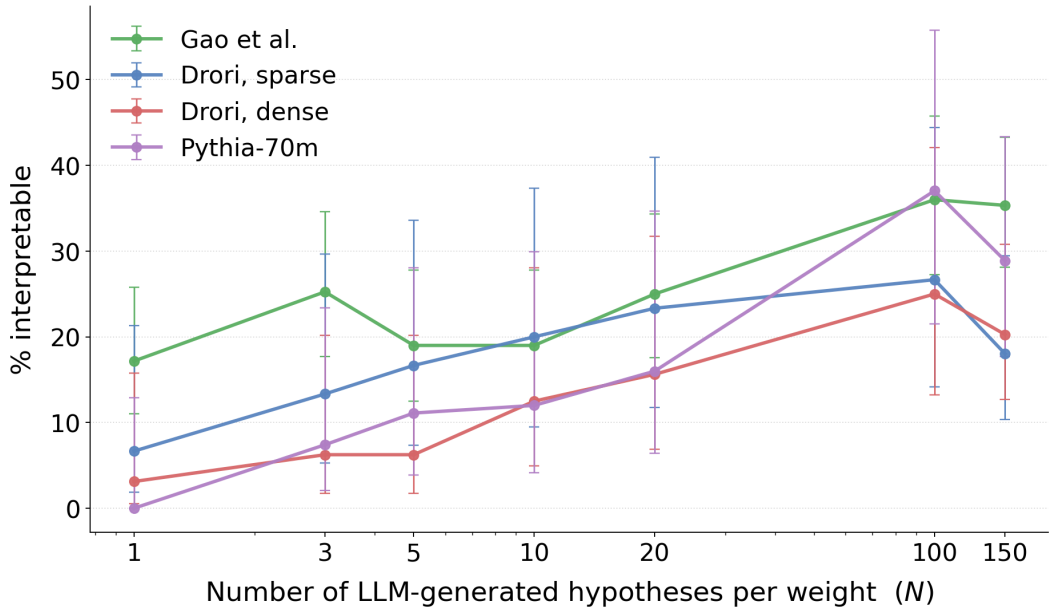

Figure A6: Interpretable rate as a function of the candidate budget N, with 95% Wilson CIs. Single-call N from 1 to 150.

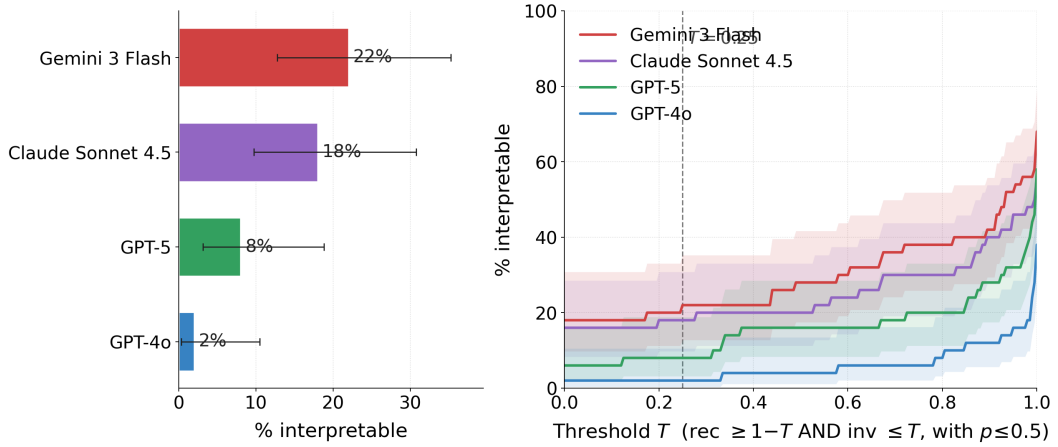

Figure A7: Interpretable rate by auto-interp LLM. **Left:** fraction of sampled weights interpretable at $T = 0.25$, with 95% Wilson CIs, on the same weights evaluated by four different LLMs. **Right:** interpretable rate as a function of threshold T.

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 A7 reports the interpretable rate at the canonical threshold $T = 0.25$ 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 ∼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

This appendix expands the digit-neuron case study of §5. We identify an MLP neuron in the Gao et al. 2025 sparse code transformer that fires predominantly on numeric tokens; at the activation

level the neuron looks monosemantic and its role is naturally read off as "detects digits." Gao et al. note that not all nodes in their models are exactly monosemantic on the pretraining distribution (their Figure 39), and we observe the same on our evaluation corpus — not every node admits a clean single-concept label at the activation level. We ask what our pipeline returns when applied one level deeper, to the individual W_{in} entries that connect the neuron to its upstream residual channels.

Per-weight predicates. The neuron has nine nonzero input weights. Our pipeline assigns a separately-interpretable predicate to each, and the predicates cluster into three digit-count groups: a single-digit detector (1 weight), two-digit detectors (5 weights), and a three-digit detector (1 weight); two further weights receive "generic" or "other" numeric-context predicates that do not split cleanly by digit count. The strongest weights in each group have high recovery and low inverse, and the digit-count gates together account for the bulk of the neuron's output (Figure 6 in §5). The activation-level "detects digits" label is therefore carrying information about *which* digit count is being detected: the same neuron's individual parameters partition the digit-detection role into per-count gates that the activation-level summary collapses into a single label.

Comparison with the upstream-channel view. For each of the input weights, we recorded two things on the same CodeParrot corpus: the tokens at the highest-KL positions when the weight is ablated, and the tokens at the highest-|activation| positions of the weight's upstream residual channel. Table A2 reports the comparison.

Table A2: For each input weight of the digit-detection neuron (`neuron 1863`), representative top tokens at the weight's highest-KL positions (left) and top tokens at the upstream channel's highest-activation positions (right), drawn from the same CodeParrot corpus. Tokens are decoded from the tinypython_2k tokenizer. Rows ordered by max KL.

ch (i)	max KL	weight top tokens	upstream-channel top tokens
464	26.8	`100, 200, 300, 111`	`110, 200, 120, 100`
71	22.7	`64, 50, 59, 60`	*non-printable byte*, `_len`
395	17.1	`28, 08, 27, 35`	`39, 28, 35`
799	16.1	`32, 56, 41, 46`	`ran`
578	13.7	`16, 12, 13, 17`	`tern`
131	8.0	`4, 2, 3, e`	`whi`
678	6.0	`7, 8`	`!=, raise`
87	1.9	`90, 89, 95, f`	`_map, _list`

For two of the input weights (channels 464 and 395) the upstream channel and the weight pick out broadly the same tokens (multi-digit numbers in both columns). For the remaining six, the upstream channel's top-activating tokens are unrelated to digits (`whi`, `ran`, `tern`, `!=`, `raise`, `_map`, `_list`, plus one channel dominated by a non-printable byte fragment). The weight's effect on numeric tokens does not appear in the channel's top activations: a description of channel 131 as "fires on `whi`" would not predict that ablating $w_{131,1863}$ disrupts predictions on the digit `4`. We report this as an observation on a single neuron in a single sparse model; whether it generalizes is left to future work.

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 A8 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,

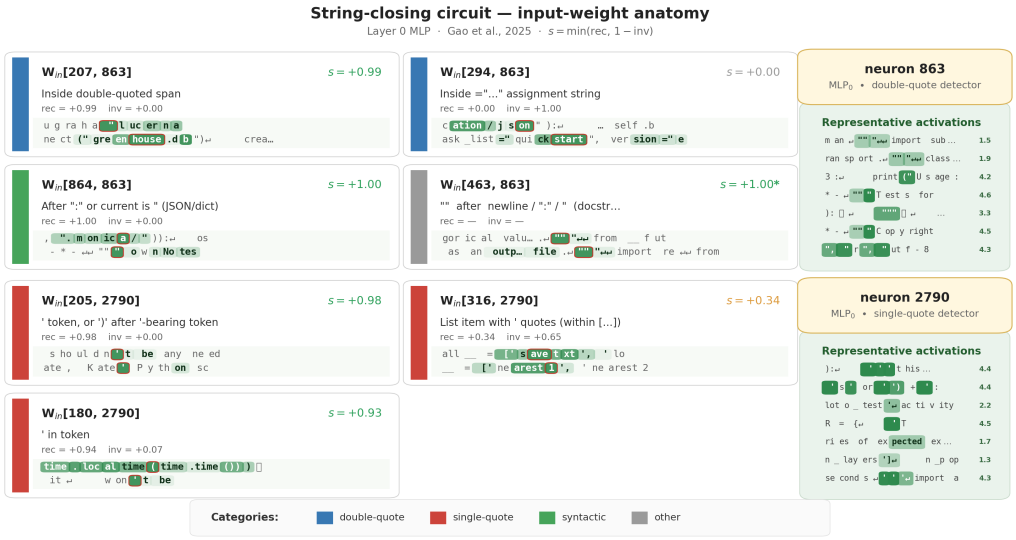

Figure A8: Quote-detector neurons from the string-closing circuit of Gao et al. 2025. **Neuron 863** (top band) fires on double quotes; **neuron 2790** (bottom band) fires on single quotes. Each card shows one nonzero input weight: the predicate, the interpretability score $s = \min(\text{rec}, 1 - \text{inv})$, 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, so the rec/inv ratios are not numerically reliable.

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 A9 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.

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 A10 shows all 20 cards, grouped by the functional role assigned by their predicate. Five clusters emerge:

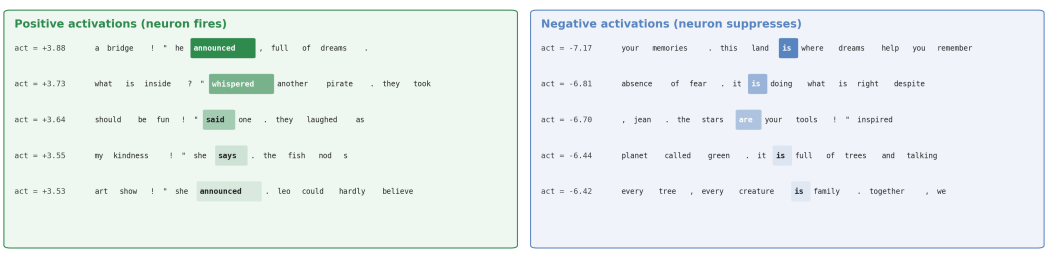

Figure A9: Activation-level view of Drori-sparse layer-1 neuron 2461. **Left:** five contexts where the pre-activation is most positive (random sample from the top-95% positive band). The neuron fires on speech-act verbs. **Right:** five contexts where it is most negative; the neuron is pushed down on copulas (`is`, `are`). Pill saturation tracks $|\text{activation}|$ within each panel.

- **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 A9.
- **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 A10 reports recovery, inverse, and the fraction of corpus positions where the predicate fires; the strongest cards (*e.g.,* ch. 909 *'as'* detector with $\text{rec} = 1.22$, $\text{inv} = -0.22$) sit comfortably above the $T = 0.25$ interpretability bar. Weaker cards (*e.g.,* the modal suppressor on ch. 38 with $\text{rec} = 0.27$, $\text{inv} = 0.73$) 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 at least *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.

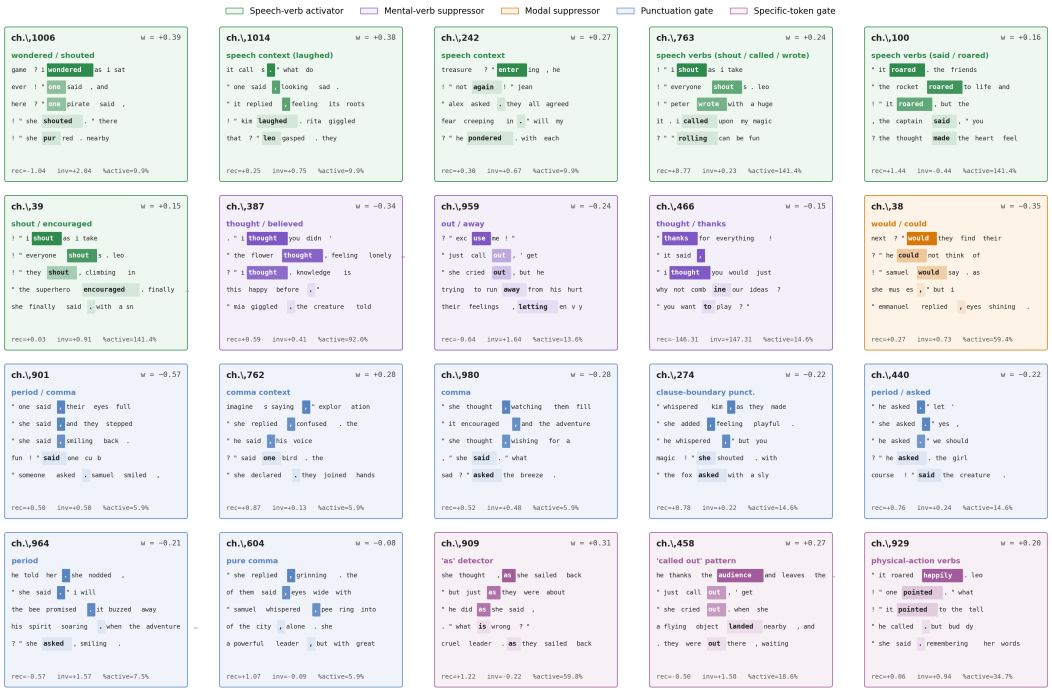

Figure A10: Weight-level decomposition of Drori-sparse layer-1 neuron 2461. The neuron has 86 nonzero input weights; the figure shows the 20 strongest by max-KL on a 5,000-sequence SimpleStories sample, grouped by the functional role assigned by their predicate. Five context snippets per card are sampled from the top-95% cumulative-KL band of that weight; pill saturation tracks the row's KL within the card. Recovery, inverse, and % active come from the conditional-zero scoring of §3.3.