Neurons as an Interpretable Basis

Curt Tigges, Matthew Kowal, Thomas McGrath, Owen Lewis, Nick Cammarata

Goodfire · November 10, 2026

https://www.goodfire.com/research/neurons-as-an-interpretable-basis

Searchable edition of the online article, including its tables, appendix, references and notes. Figures are images of the online article; interactive figures show their initial state and can be explored there.

Abstract

A companion to Neural Networks Are Already Clean

An intuitive way to interpret a neural network is to study its neurons; that is, the hidden units of its MLP layers. However, they have long appeared too complex and polysemantic for direct study. This view has led much of the field to focus on learned feature spaces over sets of neurons, which makes interpretability more challenging: we first need to transform the model before normal science can begin.

Building on a growing body of literature, our work suggests that neurons are more interpretable than previously assumed. We study the neurons of Qwen3.6-35B-A3B using sparsification methods, attribution tools, and research agents, and find three properties that make neurons interpretable units of computation:

First, they are already functionally sparse. Ablating the 92.2% weakest MoE neuron activations for each forward pass has only modest effects on model capabilities - a 2.9% increase in language model loss.

Second, neurons have relatively few semantic groupings (modes) of computational functions, and we can identify these and describe them concisely. The majority of neurons have either one mode only or a single mode that is overwhelmingly causally significant. In cases where multiple causal modes are present, the number is small and legible to our methods.

Finally, we can use agent-driven experimentation to generate neuron descriptions in natural language, and these descriptions allow us to predict, simulate, and activate the neuron. Providing the description to a judge model allows us to (1) predict whether the neuron will fire in a given passage of text with 95% accuracy vs random text; (2) simulate its token-by-token behavior in high-activation regimes sufficiently to recover 73% of the damage (as a proportion of a ceiling of damage restorable) from the neuron's ablation; and (3) write text that causes the neurons to fire more strongly than anything observed in a 10B-token corpus.

Overall, we argue that neurons are meaningful units of computation that can be effectively used for interpretability in their own right, letting us study the model on its own terms rather than through a learned proxy.

Introduction

In the previous essay in this series, we argued that neurons in deep learning models are more interpretable and cleaner than has often been suggested. In this article, we describe what this means in more detail and share our empirical findings. Throughout, **the term "neuron" refers to an individual computational unit in the intermediate layer of a transformer MLP, whose output is a scalar activation, computed after the nonlinearity (SiLU), as explained in the Background section below.**

Neurons were the first units through which deep networks were interpreted, and early work found them to be closely aligned with recognizable concepts (Le et al., 2012; Zeiler & Fergus, 2014; Bau et al., 2017; Olah et al., 2017; Radford et al., 2017; Goh et al., 2021). However, this story was complicated by observations of

so-called polysemanticity-the idea that individual neurons respond to seemingly unrelated inputs (Nguyen et al., 2016; Olah et al., 2020). Polysemanticity is related to the proposal that more concepts are represented in a network than the number of neurons in the network, an idea canonized in the literature on “superposition” (Elhage et al., 2022; Arora et al., 2018; Park et al., 2023; Costa et al., 2025). This, in turn, led to the development of learned overcomplete feature dictionaries in place of neurons as computational units of analysis (Bricken et al., 2023; Cunningham et al., 2023). Despite these trends in the field, interpretability researchers continued to investigate respects in which neurons may still constitute a meaningful unit of analysis (Geva et al., 2021; Dai et al., 2022; Geva et al., 2022; Gurnee et al., 2024; Arora & Wu, 2025; Dravid et al., 2026; Feucht et al., 2026; Shabalin & Piotrowski, 2026). We continue this line of work, introducing new ways to factor and understand what individual neurons compute.

Whether a neuron is labeled monosemantic or polysemantic can depend on choices about what constitutes a single concept, and responses that initially appear unrelated may share a regularity that becomes apparent with a better understanding of the computations they support (Murphy & Medin, 1985). **We therefore characterize neurons by their computational functions as our primary object of study, considering semantic similarity of the texts that activate them as a secondary question.** We study the circuits in which each neuron participates, tracing how upstream neurons drive its activation and how downstream neurons use its output. This lets us examine how a neuron’s role varies across contexts and assess whether those roles admit a concise description.

A common view is that neurons are an intractable unit of analysis. In the worst case, an investigation could have demonstrated this in three ways: (1) Individual neurons might rarely play identifiable computational roles, so that meaningful units would be clusters of neurons, or directions and manifolds in activation space; (2) Each neuron might take extreme values on a wide and heterogeneous range of inputs that no short description unifies; (3) Or neurons might resist descriptions precise enough to predict when, and how strongly, they fire. We test each possibility directly: we ablate weak activations, determine the semantic range of neurons’ computational roles, and predict and simulate activations from descriptions alone.

Our results provide evidence that neurons are tractable units of analysis. Neurons (1) exert most of their downstream influence in high-activation regimes (above the 97th percentile at the positive extreme and below the 3rd at the negative); (2) have a small number of computational roles, distinguished by their interactions with other components of the network; and (3) can be given concise descriptions that predict and simulate their activations. The three results share a pattern: the stronger a firing, the more the model depends on it and the more precisely we can describe it.

The picture that emerges is not that every neuron means one thing, but that a neuron’s important firings have a small set of computational functions, and a short description captures them. Some neurons are genuinely polysemantic, and some behavior remains unexplained, but so far little of it appears critical for the model’s computation.

We therefore find that neurons are a useful and interpretable basis for understanding LLMs.

Some Definitions

One neuron, one token

Write: The neuron's contribution to the residual stream, $a \cdot w_{router} \cdot W_{down}[:, n]$: its column of the down-projection matrix, $W_{down}[:, n]$, scaled by its expert’s router weight, w_{router}, and the output of its SwiGLU activation function, a.

Activation: The combined scalar $a \cdot w_{router}$.

Pole: As we explain below in more detail, modern neurons’ activation functions can produce a positive or negative value. A pole is the positive or negative extreme of a neuron, and usually (but not always) positive and negative values have unrelated roles.

Firing: A fire is an individual event in which a neuron's activation on a particular token is nonzero (on the positive or negative pole). In an MoE model this requires that the token was routed to the expert containing the neuron. These neurons do not have true “off” values since they are never 0, and in practice most activations are very weak (hence our focus on strong activations).

p97/3, p99/1, etc.: We use this notation as shorthand to refer to the percentile of a neuron’s activation relative to all its activations across a corpus. On this scale, p1, p5, etc. are *negative* activations (as the median activation of neurons in this study is near 0). Thus, p97/3 refers to both positive and negative extremes, p3 ∪ p97.

How neurons connect

Input neurons: Every neuron receives input from a large set of “causally upstream” neurons, which we refer to as inputs. In particular, to serve as input, a neuron from a previous layer must either 1) produce an output vector, written to the residual stream, that is aligned with the target neuron’s gate or up weight matrix; or 2) be identifiable through attribution techniques as having an effect on our target neuron, mediated by attention.

Reader neurons: The inverse of input neurons. A reader neuron is a neuron that responds to whichever neuron we are discussing, either directly or through attention.

Circuit: A computational subgraph of the entire network that is causally sufficient for producing a given output. In this article, we focus on circuits that use individual neurons as nodes.

Subcircuit: A computational function that consists of a set of input neurons and reader neurons where the two sets are connected through a single target neuron. A neuron can host multiple subcircuits and they can exist simultaneously. However, because the members of a subcircuit are highly disjoint from the members of other subcircuits, only one is active at a time.

What a neuron means

Response profile: The pattern of textual features and concepts a neuron fires on across a dataset.

Max-activating examples: Texts that cause a neuron to fire near the top of its activation range; i.e., to output a value that is one of the strongest observed for the neuron.

Computational function: Neurons can receive input from potentially thousands of other upstream neurons, and are read by potentially thousands of downstream neurons. A computational function is a description of the causal process by which a set of identifiable inputs is transformed into a meaningful output; a simple example would be “combines location and cultural cues from upstream neurons to output a signal that Tokyo is being mentioned.”

Mode: A semantic grouping of computational functions (subcircuits) of a neuron. Neurons with more than one mode are referred to as *multimodal*.

Mono- and Polysemanticity: These terms conventionally describe a neuron that responds to a single “concept” or to multiple seemingly unrelated “concepts”, respectively. However, as discussed in the section below (see “*Moving from Polysemanticity to Computational Role*”), these labels admit multiple interpretations, and the apparent semantic coherence of a neuron’s responses does not by itself determine its computational function. We therefore focus on how downstream neurons use its signal.

This is the first in a series of technical articles exploring neuron interpretability. This post introduces our initial findings and sets the frame for further exploration.

Our primary findings are as follows:

- **Language models can operate under considerable neuron sparsity.** Inspired by the foundational work of Arora & Wu (2025), we evaluate several sparsification methods according to (1) KL divergence from the intact model over large datasets of natural text and (2) benchmark performance.

    - By ablating all neuron firings below extreme activations (correcting for the bipolarity of gated neurons, as explained below), we show that **the aggregate causal role of a model's neurons can be attributed largely to the activations in the highest ranges-specifically, the 97th percentile and above and the 3rd percentile and below for negative values, p97/3 in our notation**. This range is precisely where neurons are most interpretable. Up to 92.2% of the weakest absolute neuron activations can be ablated either through top-K or percentile-based filtering with only small effects on loss, KL, and a broad set of performance benchmarks. Further, those negative effects on benchmark performance can be corrected with only 91M tokens of SFT and 9M tokens of RL to re-calibrate reasoning.
- **Individual neuron poles have a small set of distinct computational functions that usually fall into one or a handful of semantic categories, like "beetle and insect-wing anatomy terms" or "Chinese health and wellness vocabulary."** We identify these functions by examining the upstream neurons that drive a firing and the downstream neurons that use its signal, and **call a neuron's coherent groups of functions "modes."** Coherence is defined by LLM judges classifying functions into specific categories and performing held-out example matching.
    - We identify disjoint subcircuits that a neuron participates in and cluster those subcircuits into distinct modes, and find that there is more circuit similarity within modes than between modes.
    - For the majority of neurons, we find either only one mode or a single dominant mode responsible for >=90% of its causal effect across a broad dataset.
    - Modes have degrees of relatedness, and are not always completely unrelated semantically. For example, a neuron may have multiple Python-related coding modes, or it may have a mathematical mode and one related to Japanese honorifics. Usually, even when there are multiple modes, one is far more causally relevant than the others.
- **The computational roles of neurons can be concisely described in natural language and these descriptions can be used to predict, disambiguate, and simulate neurons across a range of tests.** We evaluate neuron descriptions through detection (will a neuron fire in a short token span?), classification (when a neuron with multiple semantic modes fires, can we tell which meaning is operative?), input synthesis (can we create a text that makes the neuron fire strongly?), and simulation (can we replace the neuron's natural activations in a sequence with those based on a description of the neuron?).
    - **In the activation percentiles that our sparsity experiments suggest are most important for the model's primary computations, we can achieve considerable understanding of the neurons' behavior.** As activation strength rises, we are able to make more precise predictions and simulate their effects more accurately.
    - **For percentiles stronger than p97/3, we can detect when a neuron will fire on a short sequence 95% of the time** when comparing the activating text to a random text, and 89% when comparing to a text assigned to the same expert (chance in both cases is 50%). We can also tell a neuron's modes apart, either from the circuit, by observing which downstream neurons a firing moves (its readers), or from the text alone, by giving a language model the mode descriptions.
    - **For percentiles above p99.9, we can simulate neuron behavior well enough to recover 73% of the recovery achievable by a perfect reader. We note that this was achieved without any training of a model to accomplish this task.** Bands p99.9 through the top activations are responsible for 73% of all neuron damage when a neuron is ablated, and thus hold the most causal significance. We ablate these neurons from all tokens in a held-out sequence and measure the KL divergence of the logits. We then replace the activations with simulated values provided by a judge LLM based on the neuron's description. On average, this is sufficient to recover 73% of the KL damage that the perfect-reader ceiling restores.

- **For 94% of neurons, we are able to generate texts that activate the neuron beyond the maximum level found in a 10B corpus** of mixed pretraining and post-training data. Our descriptions capture what activates the neuron effectively, though this is only a test of each neuron's primary mode.

A Perspective from Neuroscience

Neuroscience offers a related debate about the appropriate units of explanation. Studies of single-neuron selectivity have revealed interpretable responses ranging from visual features to abstract concepts (Hubel & Wiesel, 1962; Barlow, 1972; Quiroga et al., 2005). Population approaches examine how neurons jointly represent information and implement computations through their collective activity and dynamics (Saxena & Cunningham, 2019; Ebitz & Hayden, 2021). These perspectives need not conflict: neurons tuned to different orientations can collectively encode orientation while remaining individually somewhat interpretable. A related line of work examines mixed selectivity, in which individual neurons respond to combinations of task variables, enabling flexible computation through population readouts (Rigotti et al., 2013). Neither population coding nor mixed selectivity is equivalent to superposition as studied in artificial networks, and neither implies that individual neurons cannot be meaningfully attributed interpretable functions.

Background: The Structure of a SwiGLU Neuron

Most modern LLMs use gated MLPs (i.e., MLPs with an element-wise gating operation), with SwiGLU specifically being by far the most common gate (Shazeer, 2020; Dauphin et al., 2017; Ramachandran et al., 2017; Elfwing et al., 2018; Hendrycks & Gimpel, 2016; Touvron et al., 2023; Yang et al., 2025). This includes the model of focus used throughout this study, **Qwen 3.6 35B-A3B**, an MoE model with 5 million neurons, 40 layers, and experts composed of 512 neurons each. This model has 256 experts per layer, and at each step of inference 8 experts are chosen per layer by the router (plus one shared expert that is always active).

We chose this model because many architectural details (e.g., SwiGLU, MoE, linear attention) are common in leading large models.

SwiGLU MLPs have both gate and up projection matrices which operate on the input separately. A SiLU activation function is applied to the gate projection, and the result is multiplied with the up projection.

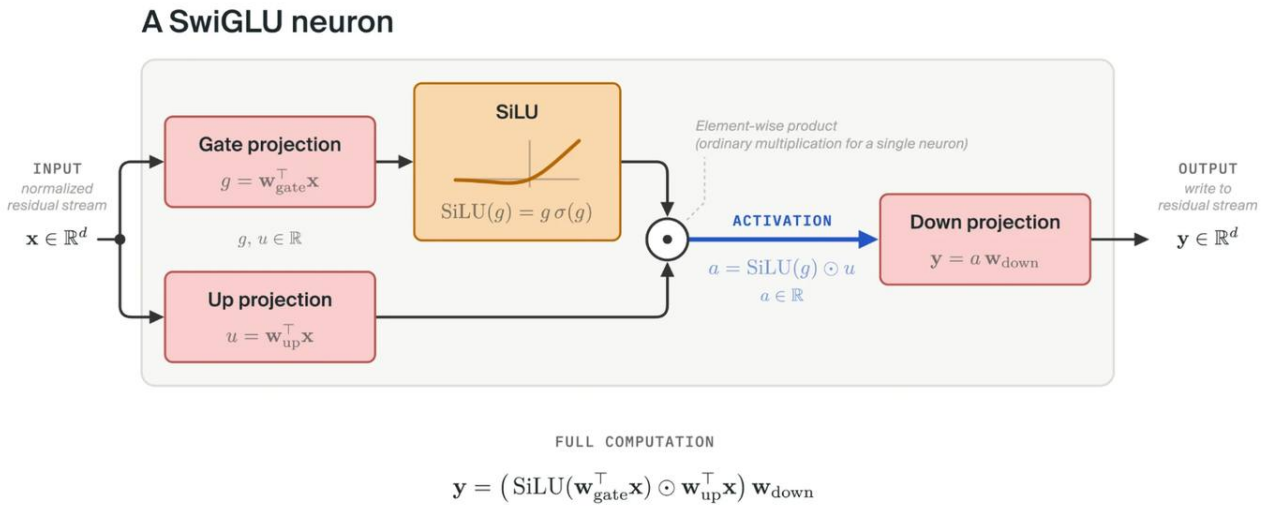


Figure 1: Diagram of a single SwiGLU neuron.

Importantly, SwiGLU allows for both positive and negative activations-that is, the neuron is **bipolar**, and negative activations are as causally relevant as positive ones. We refer to the positive and negative ends of the spectrum as **poles**. As we will see below, the two neuron poles can have completely different meanings,

though sometimes (but rarely) the positive and negative poles will represent contrasting opposites of the same concept. **We thus treat neuron-poles as the unit of analysis when exploring a neuron's meaning.**

The multiplicative input structure (gate ⊙ up) of a SwiGLU neuron allows it to respond selectively to the co-occurrence of learned input patterns detected by the two branches (i.e., an AND operation) (Pearce et al., 2024; Sharkey, 2023; Jayakumar et al., 2020; Veness et al., 2019). However, this can result in a somewhat confusing structure:

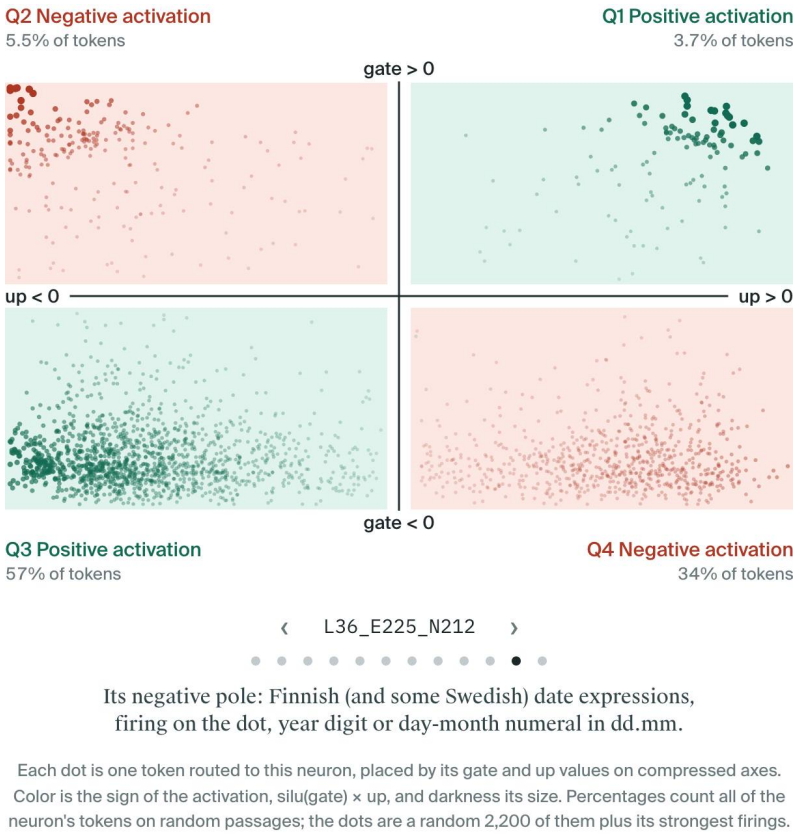


Figure 2: The four SwiGLU quadrants, filled with one neuron's real tokens. Use the arrows to see other neurons.

Most positive activations involve positive activation of both the gate and the up projection, and most negative activations involve positive activation of the gate and negative activations of the up projection. However, positive activations can also occur due to both the gate and up projections being negative (we call these Q3 activations, and later discuss unusual phenomena they express), and negative ones can occur due to negative activation of the gate and positive activation of the up projection (Q4). Q3 and Q4 activations tend to be weaker and harder to interpret. Recall that when we refer to neuron activation values below, we mean the product of the output of the SwiGLU unit and the expert router weight.

A final consequence of the SwiGLU structure is that neurons are never actually off-that is, the neuron output is never zero except in MoEs when the expert in which the neuron resides is not active. This leads to a significant amount of apparent noise in the weaker activations (Fishman et al., 2024; Gurnee et al., 2024).

Neuron Sparsity and Effective Range

The two central questions of this section are:

1. Are individual neurons, instead of distributed directions spread across many neurons, the right basis for analysis of MLP computation? High levels of sparsity in MLP computations would suggest that neurons carry significant parts of the computation on an individual basis.

2. What sorts of neuron activations are meaningful and necessary for computation - do strong activations suffice, or are weak activations indispensable? A highly sparse regime would suggest much higher importance for strong activations.

If the conceptual primitives for transformer computation tend to be spread out among vast arrays of neurons in a distributed code, we should expect high levels of sparsity to sabotage these computations. On the other hand, if neurons tend to be privileged, meaningful units of computation, we should expect these primitives to be more localizable and sparse.

Earlier work has observed that neurons are often quite interpretable and consistent (across activation examples) in their strongest activations, but much more diverse and hard to understand at weaker levels (Szegedy et al., 2014; Morcos et al., 2018; Zhou et al., 2018; Leavitt & Morcos, 2020; Ghorbani & Zou, 2020; Dhamdhere et al., 2018). This motivates the concern: what if weak activations are both indispensable and uninterpretable?

We find that models can still perform surprisingly well even when all weak activations (e.g., between the negative and positive extremes of p10 and p90, or even p3 and p97) are mean-ablated, achieving close to parity with the intact model on a number of non-reasoning benchmarks. More importantly, we test an ablation where every layer retains only the top-320 neurons (out of 4,096 routed neurons) by absolute activation strength for a given step of inference, ablating 92.2% of routed neurons, and show that this model performs comparably to the base model on non-reasoning benchmarks.

For all experiments, we leave the shared expert neurons unchanged, since these neurons are effectively always on and function differently from the expert neurons. In total, the shared experts make up 20,480 of the model's 5.26 million neurons (~0.4%).

For reasoning benchmarks, we observed that the ablated model would often run into reasoning loops and self-check excessively, never completing its answer-though by checking its reasoning output, we were able to determine that the correct answers were present. With a brief fine-tuning step, we corrected the reasoning errors and met or exceeded the original model's performance on a broad set of thinking-based benchmarks for question-answering, code, reasoning, and math. Considering that the fine-tuning tokens used were a small fraction of a typical post-training run, this provides evidence that the fine-tuning primarily corrected reasoning rather than adding new information to the model, and that the majority of the information required for performance on benchmarks was present at the higher activations.

We provide further evidence for sparsity in the Appendix: we preserve specific model predictions with even fewer neurons using the gradient-based attribution method, RelP, and we elaborate on recent findings (Shabalin & Piotrowski, 2026) showing that transcoder replacement models tend to discover neuron-localized features even when not constrained to do so.

Sparsification by Activation Strength

Our first set of neuron sparsity experiments uses neuron activation strengths to determine which neurons to ablate. We consider three methods-mean-ablating (on an individual neuron basis, across all neurons in the model) the central percentile bands of neuron activations, expert top-k filtering, and layer top-K filtering-and measure their effects on loss, KL, and a range of benchmarks. The ablated bands are symmetric in percentile rank around the median: i.e., p1 is a strong negative activation, p99 is a strong positive activation, and weak bands are p10-p90, p25-p75, etc.

Ablation Within Top Percentiles

A natural experiment is to start with the weakest neuron activations and ablate them during inference, measuring the resulting loss and KL divergence. Since SwiGLU neurons are bipolar and neurons have different activation strength distributions, we compute neuron activation strength percentile bins and apply mean ablation to everything between the positive and negative extremes (Li et al., 2023a; Liu et al., 2023a; Mirzadeh et al., 2023; Zhang et al., 2022; Liu et al., 2024; Lee et al., 2024; Jaszczur et al., 2021; Ahmad & Scheinkman, 2019; Makhzani & Frey, 2013; Song et al., 2024; Zhang et al., 2024; Wang et al., 2024; Song et al., 2023; Alizadeh et al., 2023; Zhang et al., 2023). These initial experiments were conducted *without* the router weight scaling, and the percentiles shown likewise do not include this scaling.

Top-K Filtering

Inspired by Shabalin & Piotrowski (2026), we attempted to find a more sparse regime by applying a top-K function (based on raw neuron activations) across each of the experts in the model. We tested top-32 and top-64, which retain 6% and 12% of routed neurons per token respectively. Experts have 512 neurons each. In practice, the top-64 neurons primarily lay at or beyond the p5 and p95 extremes (Sun et al., 2024; Dettmers et al., 2022; Kovaleva et al., 2021; Puccetti et al., 2022; Bondarenko et al., 2023; Olshausen & Field, 2004). As with the ablation within top percentiles, this was performed without the router weight scaling.

Router-Weighted Top-K Filtering

To account for expert router weighting, we apply our activation definition above and score each routed neuron by its router-weighted activation. We then apply a single top-K over all 8 × 512 = 4,096 routed neurons of each token rather than one per expert, so the budget can shift toward experts whose neurons have the strongest activations after router weighting. We keep the top 320 (7.8%) of routed neurons (40 per expert on average, 92.2% sparsity) and zero the rest before the down projection. Here, most activations exceed the p3 and p97 extremes. Like other activation-sparsity methods, the mask is computed from each token's own activations at inference time and needs no training (Liu et al., 2023; Lee et al., 2024; Liu et al., 2024; Wang et al., 2024; Makhzani & Frey, 2013).

Performance Under Ablation

We found that the model was able to operate surprisingly well under high levels of sparsification. Below, we show the sparsified models' loss and KL divergence, as well as performance on a wide range of benchmarks:

Metric / benchmark	Base model	Sparsified models						Baselines (controls)	
		Expert top-K		Layer top-K	Band ablated			Random	Inverse of Top-64
		Top-64	Top-32	Top-320	P25-P75	P10-P90	P5-P95	Random-64	Bottom-448
Web text (8.4M tokens)									
Cells ablated	0%	87.5%	93.8%	92.2%	49.7%	78.6%	88.8%	87.5%	12.5%
Loss	1.886	1.952	2.032	1.940	1.896	1.926	1.977	4.214	4.587
Δ loss	0	+0.066	+0.146	+0.054	+0.010	+0.040	+0.091	+2.328	+2.701
KL	0	0.152	0.263	0.140	0.033	0.088	0.162	2.511	2.885
Classic suite · Δ accuracy vs base model, percentage points (acc; GSM8K strict match)									

Metric / benchmark	Base model	Sparsified models						Baselines (controls)	
		Expert top-K		Layer top-K	Band ablated			Random	Inverse of Top-64
		Top-64	Top-32	Top-320	P25-P75	P10-P90	P5-P95	Random-64	Bottom-448
ARC-Challenge	54.9	+2.0	+1.5	+0.3	−0.7	+1.3	+1.8	−32.8	−37.3
ARC-Easy	81.4	+0.8	+1.5	+0.2	+0.6	+1.7	+2.1	−40.3	−49.1
WinoGrande	73.9	−1.4	−2.0	−2.2	+0.1	+0.2	+0.3	−23.2	−23.6
HellaSwag	57.4	−0.7	−1.1	−1.6	−1.1	−1.3	−1.7	−25.5	−26.9
MMLU	84.2	−1.6	−3.4	−1.2	−0.4	−0.7	−1.8	−60.1	−60.9
GSM8K	96.7	+0.3	−16.7	−1.7	−1.0	−1.3	−9.3	−96.3	−94.0
Thinking evals · Δ accuracy vs base model, percentage points									
SuperGPQA	44.8	−15.2	−17.9	−13.0	+0.9	−8.1	−14.4	−44.5	−44.6
MMLU-Redux	91.5	−3.7	−5.8	−3.0	+1.0	−2.7	−5.0	−89.0	−90.4
MMLU-Pro	78.4	−8.5	−18.9	−8.3	+0.2	−4.3	−13.0	−78.0	−78.0
IFEval	30.9	−0.6	−0.4	+0.4	+0.0	+0.0	+0.4	−15.9	−15.2
MedQA	92.8	−7.8	−16.0	−4.6	+0.8	−2.5	−4.1	−83.9	−80.6
GPQA-Diamond	63.6	−18.2	−25.8	−11.1	+2.5	−10.6	−20.7	−54.0	−55.6
MATH-500	69.8	+7.0	−4.0	+5.8	−3.8	−3.2	−12.8	−69.8	−69.8
AIME 24+25	0.0	+11.7	+3.3	+11.7	+1.7	+3.3	+1.7	+0.0	+0.0
HumanEval+	59.1	+7.9	+3.0	+15.9	−7.3	−20.7	−33.5	−59.1	−59.1
MBPP+	71.7	−20.4	−24.9	−20.9	+5.6	−12.2	−14.8	−71.7	−71.7

Table 1: KL, loss, and benchmark performance under sparsification. Base model benchmark scores are percentages; other benchmark columns show changes in percentage points. Red indicates a decrease; blue indicates an increase. Zero is white. There is a far smaller drop in performance from retaining the top-K neuron activations, either per layer or per expert, as opposed to the complement or random-64; the latter two destroy performance. Ablation based on percentile bands similarly retains a disproportionate amount of performance on the top bands. Note that the P25-P75, P10-P90, etc. percentile bands were computed without router-weighting, unlike the percentile bands referred to throughout the remainder of this text. A thinking token cap of 4,096 was applied to the reasoning evals.

Performance is retained to a surprising degree across benchmarks, especially when compared to controls. The ablated models retained high performance across classic benchmarks like ARC-Challenge, ARC-Easy, WinoGrande, and HellaSwag, improving slightly on some evals and dropping on others. However, we observe drops in reasoning and knowledge-based evals. When we observe rollouts directly, we see that the sparsified model's reasoning tends to loop and self-check excessively. However, when we classified these rollouts with Claude Opus 5.5 agents, we identified that the sparsified model would still mention the facts and

correct answers, usually as the last answer candidate-it would just self-check endlessly afterwards. We further found that reasoning collapse seems related to the double-negative activations we discuss in the next section, since restoring those specific activations restored reasoning.

In order to see if this strong-activation regime could recover full benchmark performance, we follow Shabalin & Piotrowski and perform light fine-tuning on the layer top-320 model and re-measure performance; across benchmarks we were able to see full restoration of function. We discuss methods and results further in the Appendix section Extended Sparsity Results and Discussion.

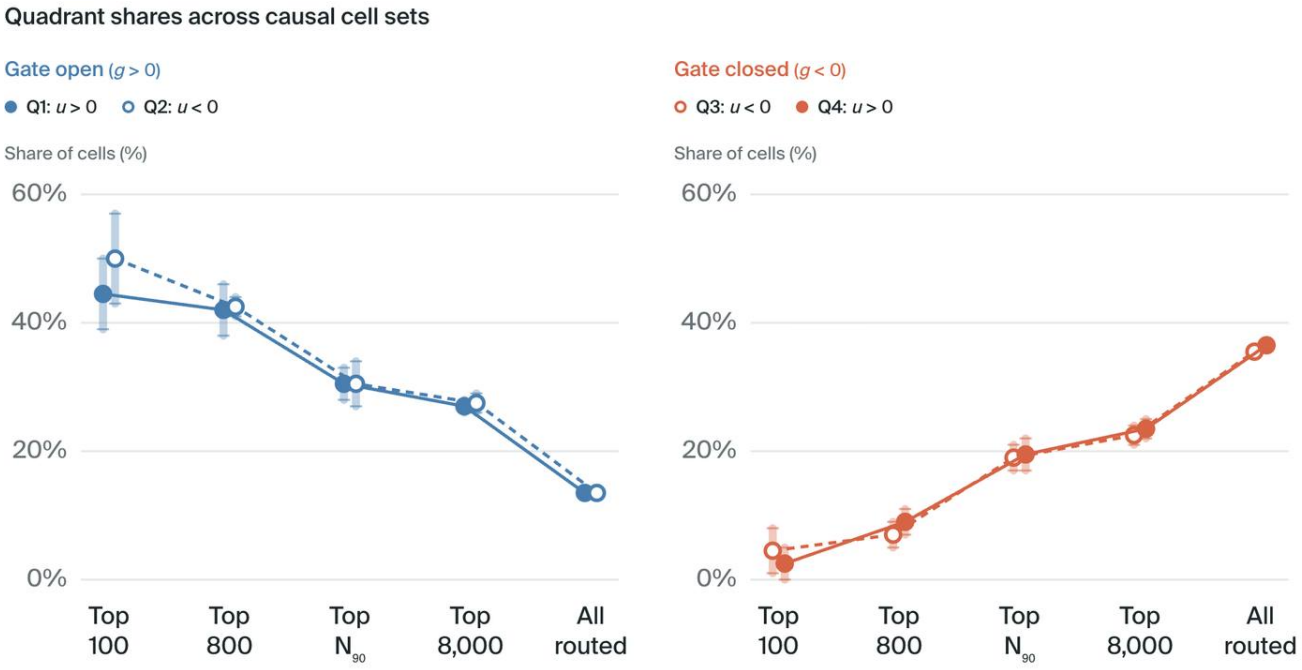
What we suggest, then: the original model is very close to sparse, with most of the computation located in the top activations. This implies that even the original model is likely quite interpretable and causal at the top ranges, but also that it is relatively easy to tip a model over the edge to more complete sparsity. Our subsequent analyses of neurons in the fine-tuned model showed that the neurons retained their original functions, remained interpretable, and had the same numbers of modes.

Principled Selection of a High-Activation Band

Most of the model's functionality is retained after top-320 layerwise filtering, and that lost performance is easily restored with light fine-tuning. The retained router-weighted activations fall almost entirely in the p97/3 band. We therefore use this activation regime for the experiments covered in the remaining sections.

Restoring reasoning with double-negative activations

When analyzing neurons based on activation examples, we had previously observed that when neuron up- and gate-projections were both negative (regime Q3), yielding a positive activation, the activation examples tended to be much less related to other examples from the positive pole. We find that ablating these Q3 activations-approximately one-third of the total neuron activation count across the model-does less damage than the gate-up regimes, except for one specific, strange use case that we describe below.



Cell sets contain the top-ranked cells by RelP. N_{90} keep sets contain 3,200–6,400 cells. Both panels use the same 0–60% scale.

Exact ranges and quadrant definitions >

Exact ranges and quadrant definitions

Ranges are reported as lower-upper percentages; they are not identified as confidence intervals. Midpoints are calculated only to position the chart markers.

Cell set	Q1g > 0, u > 0	Q2g > 0, u < 0	Q3g < 0, u < 0	Q4g < 0, u > 0	Gate closedQ3 + Q4
Top 100	39-50%	43-57%	1-8%	0-5%	1-13%
Top 800	38-46%	41-44%	5-9%	7-11%	13-20%
Top N_{90}	28-33%	27-34%	17-21%	17-22%	33-43%
Top 8,000	26-28%	26-29%	21-24%	22-25%	43-49%
All routed	13-14%	13-14%	35-36%	36-37%	71-72%

The gate-closed column preserves the separately reported Q3 + Q4 range. Its endpoints are not calculated by adding the individual quadrant endpoints.

Figure 3: Share of cells in each gate (g) and up-projection (u) sign quadrant. Whiskers show reported ranges; markers and connecting lines show their midpoints. We see that Q3 and Q4 activations are dramatically underrepresented among the most causal neurons across attribution graphs.

In the course of running benchmark experiments on sparsified models (top-64 and top-32, without additional fine-tuning), we observed that reasoning sometimes became uncalibrated, containing frequent loops and self-checks. In some benchmarks, the model did not reason at all (where the original model did). We discovered that patching Q3 activations back to the model restored both reasoning and accuracy, even to the top-32 sparse model. But this is not the end of the story: reversing the sequence of Q3 activations (tokenwise) also restored reasoning and accuracy. Even patching in the Q3 sequence, token by token, from an entirely different prompt to a target prompt, *also* restored reasoning and accuracy.

The specific experiment we conducted was as follows. First, we take 24 GSM8K problems where the model curtailed its reasoning once all Q3 activations were ablated; reasoning responses went from 1,550 tokens on average to 176 tokens. We then patched in Q3 activations from various regimes and measured the retained accuracy:

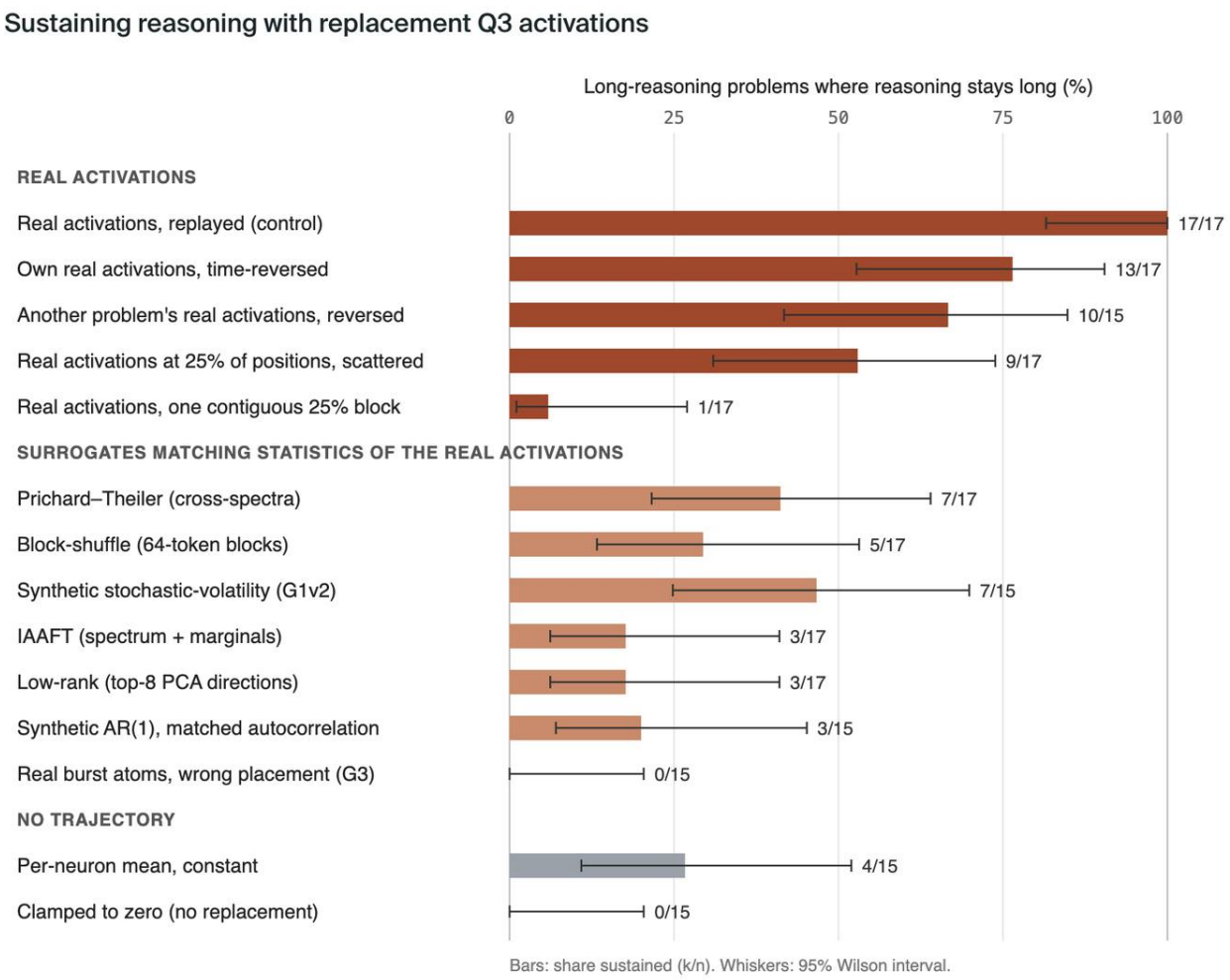


Figure 4: Replacing deep-layer Q3 activations during reasoning. On 24 GSM8K problems (greedy decoding), we replace the Q3 activations of layers 24-39 with each substitute. We count how often reasoning stays long: among the problems where the unpatched model reasons for more than 1,000 tokens (15-17 of the 24), how many patched runs still reach at least 40% of the unpatched length and at least 600 tokens. Real activations keep reasoning long even when played backwards (13/17) or taken from a different problem (10/15). Keeping only a quarter of them, spread across their original positions, still works (9/17). The same quarter as one contiguous block does not (1/17). Surrogates that match the real activations' spectrum, cross-spectra, marginal distributions, low-rank structure or autocorrelation mostly score near the per-neuron mean, a constant with no trajectory (4/15). Whiskers are 95% Wilson intervals.

During generation, we patch the replacement activations in token by token. In the reversed-sequence cases, we simply took another model response, reversed the token order, and patched in the Q3 activations one by one. Importantly, only the real trajectories-whether reversed or patched in from another sequence-led to correct answers.

These results suggest that Q3 activations may support reasoning through aggregate properties of their activity, with recovery depending little on precise token-wise alignment or prompt-specific content. Q3 neurons also appear to operate through a mechanism distinct from that of Q1/Q2 neurons and constitute a smaller, less causally influential subset. We therefore leave a detailed investigation of their role in reasoning to future work.

Sparsity takeaways and the interpretable regime

We now have evidence from several directions suggesting the following:

- The set of neurons required for each step of inference is surprisingly sparse. General sparsification methods like top-K achieve comparable benchmark performance at ~92.2% sparsity; as we discuss in the appendix, specific circuit-finding tools produce even greater sparsity.

- Retained neurons are almost always firing at the positive and negative tails, often p97/3 or stronger.
- Double-negative/Q3 activations seem to consist of a functionally different regime. Though we did not analyze it in detail, we expect that it is likely that it is approachable with further investigation.

From this evidence, we conclude that even if we do not fully understand weak activations, we can nevertheless understand the bulk of what the MLPs are doing if we limit ourselves to examining the tails. Weak activations may not be entirely irrelevant, and may perform a function, but we leave investigation of them to future work.

We further hypothesize that the sheer sparsity of neuron activations in these experiments suggests that neurons constitute a privileged basis, that they are meaningful units of analysis, and that the connections between them can help elucidate their functions.

Now that we have laid the groundwork for investigation of the neurons' positive and negative extremes, the next question is: what does their behavior in these regimes tell us?

Understanding Neuron Computations

Moving from Polysemanticity to Computational Role

Neurons are often thought to be difficult to interpret because a single neuron can respond to several seemingly unrelated concepts. This property, known as polysemanticity, has been described in terms of responses to unrelated latent variables, mixtures of inputs, or multiple semantic concepts (Olah et al., 2020; Bricken et al., 2023; Bills et al., 2023).

These informal definitions do not resolve what makes concepts distinct. Whether a neuron is labeled monosemantic or polysemantic can depend on choices about what constitutes a single concept (Murphy, 2002). As noted in Neural Networks Are Already Clean, humans tend to describe neurons as detectors for groups of nouns. But categories do not always follow this principle. Klindt et al (2026) compared this phenomenon to the New York Times Connections Game, where the objective is for users to group words by non-obvious similarities, and Barsalou, 1983 famously illustrated this with goal-derived categories such as "things to sell at a garage sale". An old GameCube and an unused kayak seem to have little resemblance, but they would both belong to this category. Recognizing a unifying concept also depends on the interpreter's background knowledge (Murphy & Medin, 1985). Responses that initially appear unrelated may share a regularity that becomes apparent only with a better understanding of the computations they support.

An alternative approach is to investigate neurons from a computational perspective first, rather than beginning with semantics. **How do different sets of input neurons make a neuron fire, and how do downstream reader neurons use a neuron's signal?** By identifying distinct computational functions that a neuron performs, we can develop objective, quantitative tools to analyze a neuron; and we can then examine their semantics to map computational functions back to interpretable concepts. This approach makes it possible to demystify neurons even when they perform multiple or confusing computations, because the subcircuits involved remain legible. And as long as these roles are few and concisely describable, we can still understand the neuron by characterizing each role and the contexts in which it operates (Rigotti et al., 2013; Fusi et al., 2016; Quiroga et al., 2005; Barlow, 1972; Marr, 1982). This allows us to retain the neuron as a unit of analysis even when it serves multiple distinct computational roles.

We therefore adopt this approach-first looking at computational function, and only then examining semantics-as our primary investigative procedure. In our next section, we describe precisely how we perform this analysis.

Understanding Neuron Computation via Subcircuits

We intentionally target the high-activation regimes that our sparsity experiments demonstrate to be the most causally important (p97/3 and stronger). As motivated above, our approach here is computational/functional rather than semantic: rather than understanding a neuron in terms of the types of texts that activate it, we instead examine what function the neuron plays in network computation. In particular, a neuron is interpreted in terms of its upstream and downstream connections to other neurons.

We extract various ***subcircuits*** that a neuron participates in as the language model processes input text. We define a subcircuit as a computational subgraph with a set of input and reader neurons that connect through a target neuron, and seek to identify disjoint subcircuits that run through the neuron. A neuron participates in multiple subcircuits, each implementing a unique computation through the gate $\odot$ up multiplication. A neuron always writes to the same direction in the residual stream. However, the same textual context that causes the neuron to fire can also prime different downstream neurons to read the original neuron's signal.

An analogy can be made to the signal processing idea of multiplexing, where multiple signals from different subcircuits are compressed into a single channel (Lee & Messerschmitt, 1988; Mante et al., 2013; Rigotti et al., 2013; Salinas & Thier, 2000; Dreyer et al., 2024; Gandelsman et al., 2024). A single neuron may serve as a shared channel for different computations, with the surrounding context determining how downstream neurons use its signal. This is made particularly easy by the nature of gated MLPs, since upstream neurons can perform a switching function to activate different sets of readers depending on the context through a combination of the gate and up projections. We discuss this in the text on disambiguation below.

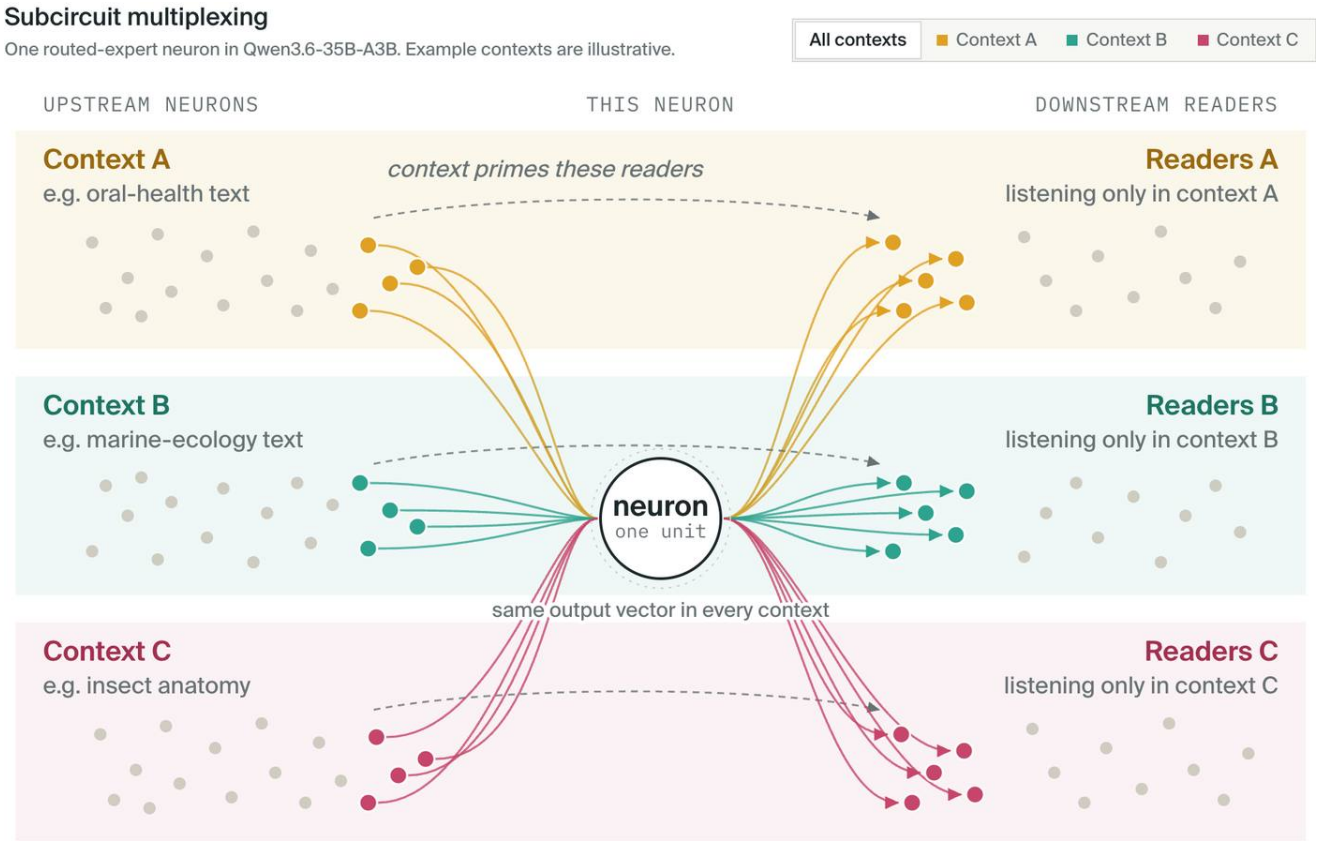


Figure 5: An illustration of how neuron subcircuits work. The neuron output vector is always the same direction, but by priming different downstream reader sets, the neuron can be used for different functions.

Identifying Computational Functions From Subcircuits

To identify these subcircuits, we first ran 10 billion tokens from diverse domains through Qwen 3.6 35B-A3B including data from pretraining, reasoning, chats with users, and math. For each of the model's 5 million neurons, we collected 1,000 examples from each of 14 activation percentile bands, for a total of 14,000 examples per neuron. We then collected an additional 30,000 examples per neuron for a stratified subset of 2,734 neurons. These additional examples were drawn from p95/5 and stronger. As a starting point for neuron descriptions, we used a fine-tuned Qwen3-8b model to produce basic labels for each neuron based on example sentences the neuron fires on.

To identify subcircuits, we first characterize each firing by its upstream inputs. With each layer's RMSNorm scale held at its actual value, the target neuron's gate and up pre-activations decompose exactly into direct contributions from the residual-stream writes of earlier neurons. We keep the inputs that contribute most. To collect downstream readers, we propagate the target neuron's write along the residual stream to every later layer at the same position, through the same frozen RMSNorm. We estimate each reader neuron's change in activation to first order. See the Methods section in the Appendix for details.

We then profile each firing by the inputs that drive or suppress its gate and up projections. For each projection, we subtract each input's mean absolute contribution across the expert's neurons from its absolute contribution to the target neuron, then retain the 400 top inputs for each of the gate and up projections. We separate these inputs into two bins: those that support the sign of the projection's pre-activation and those that oppose it. This gives each firing a profile with four parts: gate drivers, gate suppressors, up drivers, and up suppressors. Keeping these separate lets us distinguish firings where the same input plays different roles-for example, an input neuron could suppress one subcircuit through the gate, or activate another subcircuit through the up projection.

We next compare these profiles across firings of the same pole to identify upstream subcircuits that recur in different contexts. Inputs that appear in nearly every firing provide little information about what distinguishes one firing from another. We therefore weight the profiles with TF-IDF to downweight common inputs, and normalize the four quadrants so that each contributes equally. Since we do not know in advance how many subcircuits a pole has, we use community detection to find groups of firings with similar input profiles. We split examples by document, using 75% of documents for fitting and holding out the remaining 25% for evaluation. We connect each firing in the fitting set to its 15 nearest neighbors by cosine similarity of the input profile, then run the Leiden algorithm, a community detection algorithm. Each held-out firing is assigned to the group whose centroid is nearest in cosine similarity.

This procedure can split a single circuit into several groups. To bring these back together, we merge groups when their top 50 input neurons overlap by at least 30% and their top 50 downstream readers also overlap by at least 30%, a heuristic that we find produces higher-quality subcircuits. Requiring agreement on both sides helps ensure that the merged groups reflect the same underlying circuit.

This yielded a median of 19 subcircuits per pole (with a range between 1 and 51), with low overlap in their inputs and readers. We send each subcircuit, with a description of its inputs and readers, to Claude Opus 5.5 to produce a concise description of the subcircuit.

We see that subcircuits are highly disjoint:

Top-50 overlap between subcircuits of one neuron	Inputs	Readers
Mean over all pairs	5.2%	6.5%
Typical pair, the median	2%	2%
Nearest neighbor, the median	14%	17%

Table 2: Set overlap for input and reader neurons for subcircuits within a neuron. The nearest neighbor is the subcircuit with the most overlap within a neuron.

For each subcircuit, we obtain three different descriptions. The first describes how upstream neuron inputs through the gate and up projections cause the neuron pole to fire. The second describes how the subcircuit appears in text alone. Finally, we also describe the token pattern: what actual tokens in a text does this neuron fire on?

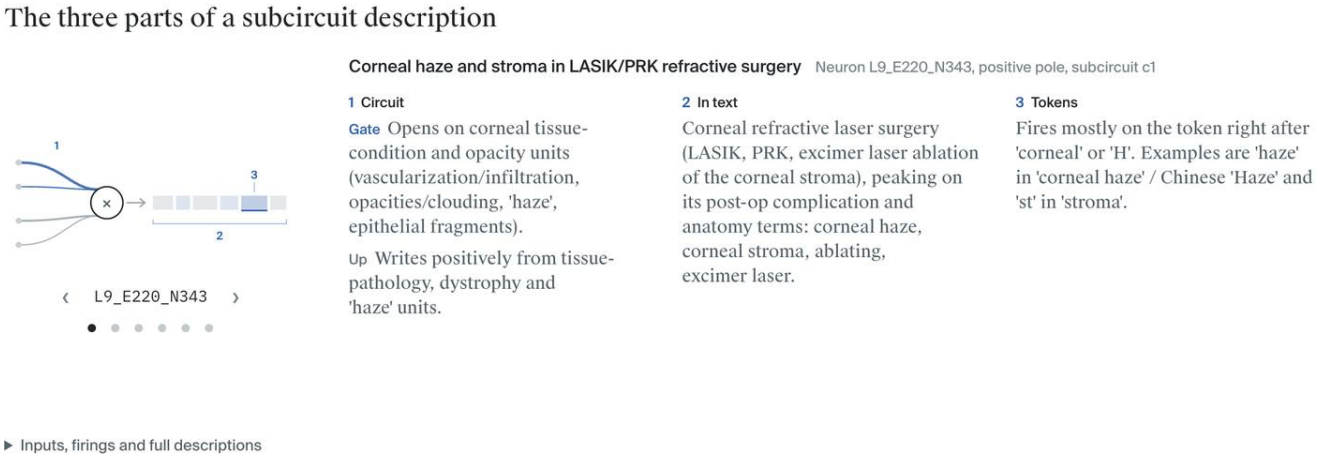


Figure 6: Example of a subcircuit description. Its behavior is described at the circuit level, the text level, and the token pattern level.

Clustering Subcircuits into Modes

While a neuron will participate in many subcircuits, many of these subcircuits can be clustered based on semantically themed groups that we call **modes**. We provide the list of subcircuits and text examples to a Claude Opus 5.5 judge (Zheng et al., 2023; Paulo et al., 2024) and ask it to create a set of categories we call modes into which each subcircuit will be placed. Often, a neuron's subcircuits are related enough that a single mode is assigned, but sometimes the subcircuits demonstrate multiple unrelated modes. We note that mode counts computed in this way are not *definitive* counts of the concepts represented by a neuron, as they are affected to some extent by the descriptive granularity imposed by this procedure.

Broadly, we find that most neurons have between one and a small handful of semantic modes, and most of the damage incurred by ablating a neuron is confined to a single dominant mode. However, in quantifying these intuitions, we do not find that any one metric effectively captures the number of modes and their relatedness within a neuron-pole. Neurons can contain modes with dramatically different levels of relatedness, very high inequality in terms of how causally relevant each mode is, and very different frequency distributions of each mode. We thus provide a variety of metrics and explain the tradeoffs of each, but to put them into context we introduce what we call the *total neuron view.*

The Total Neuron View

To ground our discussion, we provide concrete examples of neuron-poles and their modes in the following format. It allows us in one place to see the different modes within a neuron, their similarity, and how they change over different activation strengths. Within each mode, we can see its firings, and within each firing both the tokens and circuits causally relevant to it.

As described above, these were mapped into subcircuits based on disjoint sets of input neurons, and these subcircuits were then grouped by our LLM judge. In this illustrated neuron, there is one main mode, a secondary one closely related to it, and a much weaker one that's unrelated.

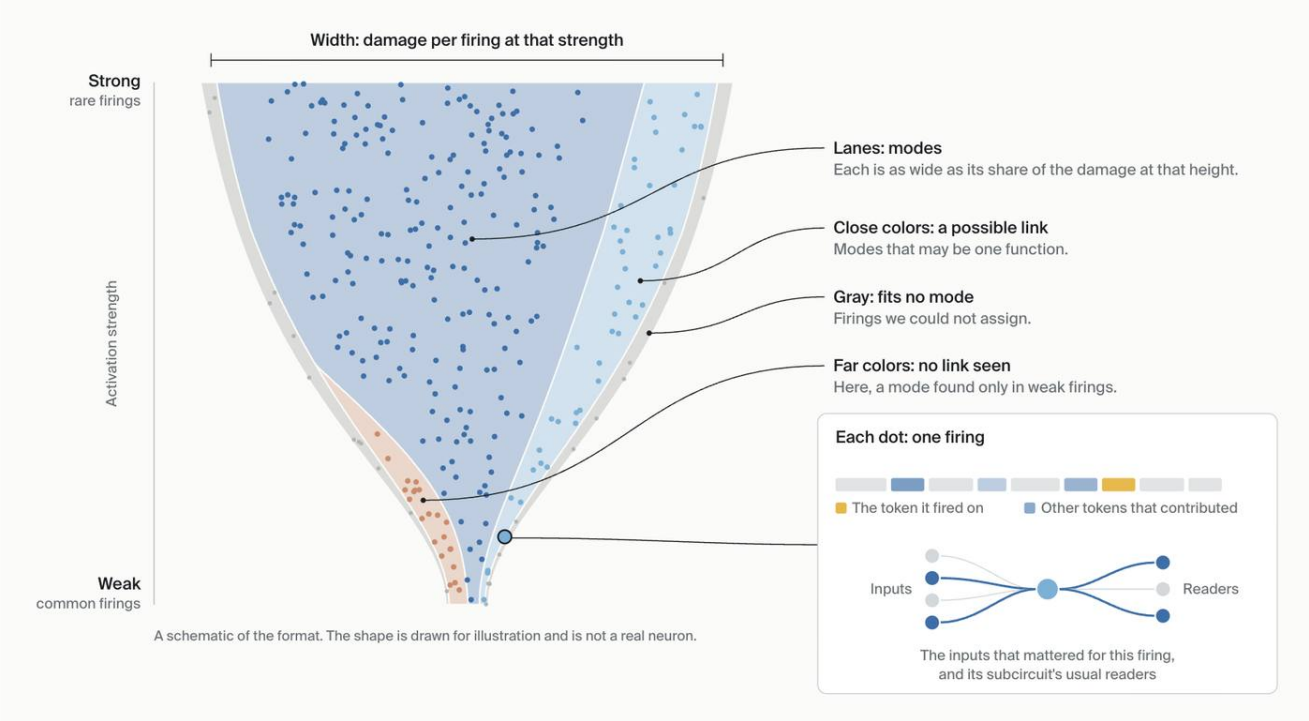


Figure 7: A neuron landscape shows how a neuron's modes and their causal importance vary with activation strength. Height indicates activation strength, from weak, common firings to strong, rare ones, while total width represents average KL damage per firing and each mode's lane reflects its share of that damage. Each dot is one firing; similar colors indicate potentially related modes and gray marks firings not assigned to a mode.

Each dot represents one firing example from our dataset. When looking at a neuron in this way, we can observe multiple things: how related different modes are (based on embeddings of the mode descriptions, as explained below), how causally dominant (how much KL divergence caused by ablation) each mode is, and how they change with activation strength.

By raw count, there are three modes in this illustrative neuron-pole, but this number does not correctly convey what we should expect from this unit, nor how we should use it in interpretability. We thus provide a number of metrics, each of which answers slightly different questions about both this neuron-pole and the population of neurons at large.

- **How many individual modes does the LLM judge assign per neuron-pole?** The median neuron-pole has 1.0 mode, with a mean of 2.82.
- **How many causally important modes does each neuron-pole have?** For each pole, we can weight a mode by its damage: of the total KL damage incurred by ablating the pole, what fraction is due to this mode? In the visualization below, we distinguish poles for which at least 90% of KL damage is confined to their main mode; 69% of poles have such a dominant causal mode.[1]
- **How many highly unrelated modes appear in each neuron-pole?** There is not often a natural place to split a mode, for instance one representing two closely related idioms. In other cases, it does appear to be completely unrelated, like a Colorado and a much rarer Tornado Python framework neuron we will discuss below. While it can be helpful to split a pole into modes, thinking in modes can count these two cases equally. In the plot below, we use color to help express mode similarity, and later show the full distribution of pairwise mode similarities per pole.

To develop an intuitive grasp of how these factors can present in a neuron, we present a selection of neuron profile types below.

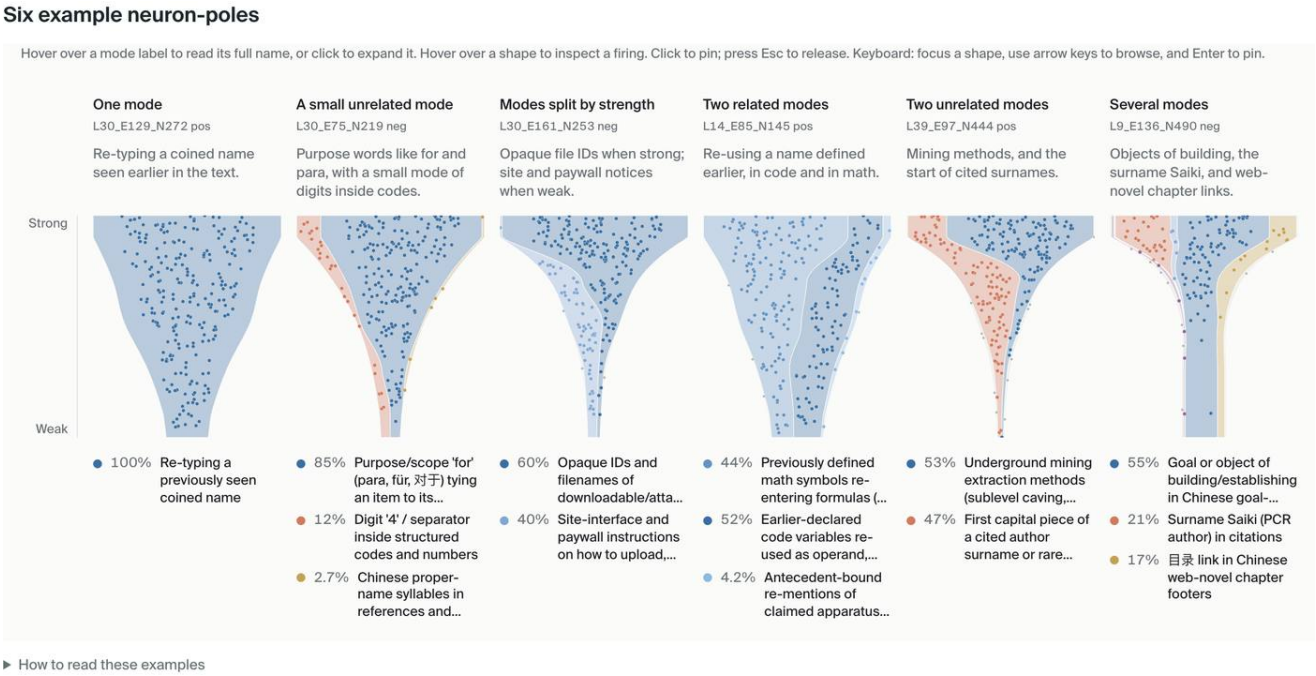


Figure 8: Six example neuron-poles illustrate how modes vary in number, relatedness, and causal importance across activation strengths. Height runs from weak, common firings to strong, rare ones, while width represents average KL damage per firing on a fourth-root scale. Each dot is one firing; similar colors indicate potentially related modes, distinct colors indicate no observed link, and the percentages report each mode's frequency-weighted share of total damage among assigned modes.

How to read these examples

Height runs from weak, common firings to strong, rare firings. Width is average KL damage per firing, compressed using a fourth-root scale so weak firings remain visible. Areas are not damage shares; the printed percentages give frequency-weighted damage shares among assigned modes.

Blue marks the dominant group of linked modes, lighter blue a possible link, and other colors groups with no link seen. Gray is reserved for firings assigned to no mode. Gate-closed firings (negative gate inputs, Q3/Q4) are excluded.

Per-firing input attribution is shown only when the stored text reproduces the original activation within 25%. "Usual readers" are the subcircuit's average readers, not a trace of that individual firing. Blue tokens contributed to the gold one. Estimated to first order from up to 100 tokens of context.

See Appendix "Additional Analyses of Modality Profiles" for a more detailed typology of neuron profiles and their frequencies.

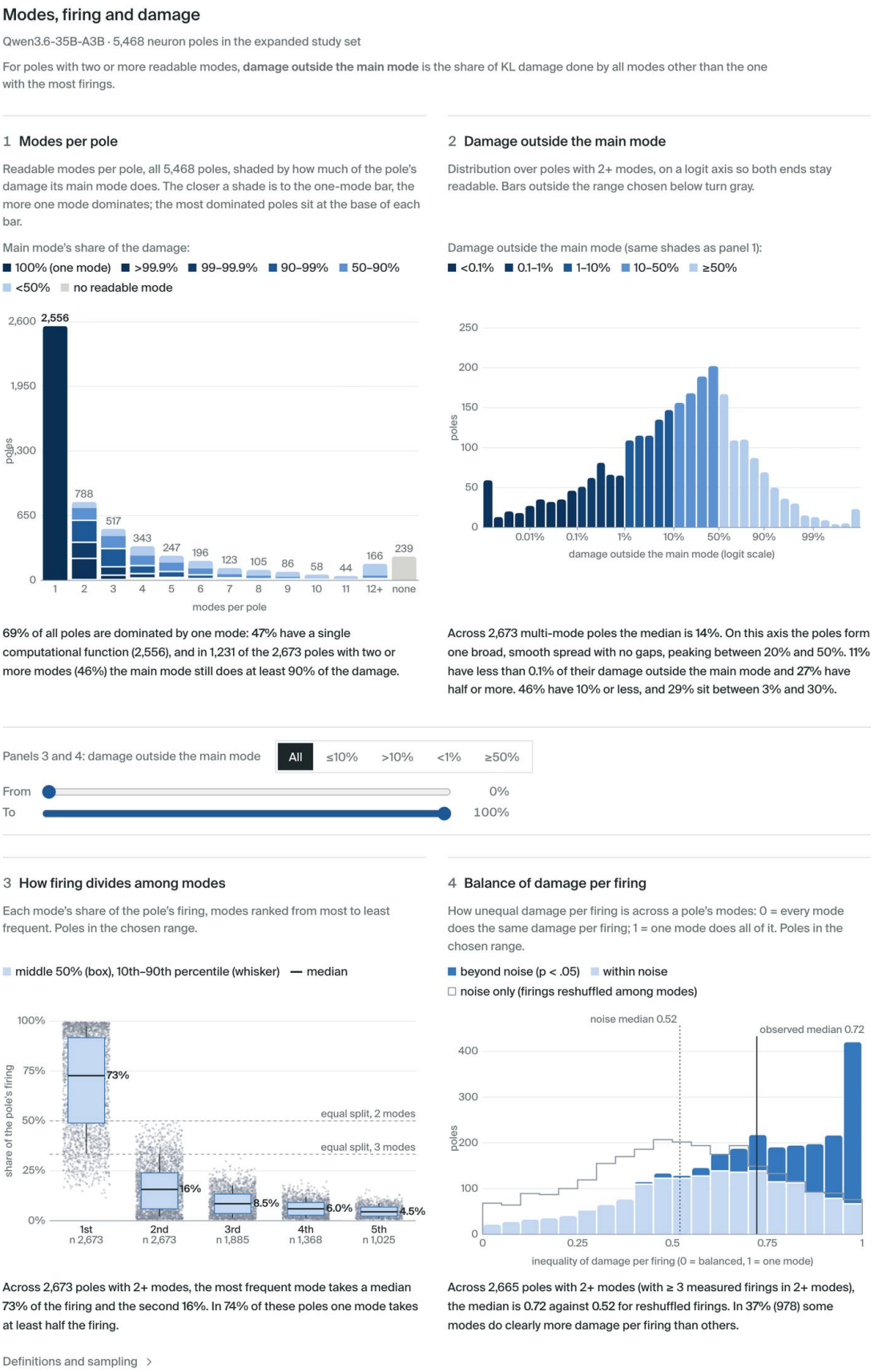
Mode Statistics in Detail

Here, we report the distributions of the metrics outlined above, showing the number of individual modes per neuron-pole, and the fraction of the pole's KL contribution and firing associated with each of its modes.

To measure the KL damage for a pole's modes, we pass the 10B-token corpus through the model with the pole ablated, recording for each token the KL divergence from the intact model's output distribution. Each token, and its associated KL damage, is assigned to a mode according to the subcircuit it activates. We report each mode's fraction of the pole's total KL.

We measure firing fraction similarly, recording each mode's fraction of a pole's total firing for the 10B corpus, as measured by the mode's subcircuit activations.

Below we show the results for all metrics over our entire sample population of 2,734 neurons.



Definitions and sampling

Damage outside the main mode. The main mode is the one with the most firings in the KL sample. A mode's damage is mean KL per measured firing (summed over the firing token and the next 16) times its number of firings in the 5,000-firing sample, which holds 1,000 firings per band from p97 to the extreme band. So damage is weighted by firing count within that band-balanced sample, not by population frequency:

a p97 firing counts the same as an extreme one. Residue and closed-gate groups are excluded. Mono poles have one readable mode, so the number is 0 by definition; they appear in panel 1 only.

Why it can be near 100%. The main mode is chosen by firing count, not by damage. When the most frequent mode is weak and a rarer mode fires hard, almost all the damage sits outside the main mode.

Firing share counts a mode's firings among the pole's real-mode firings in its 5,000-firing sample, 1,000 per band from p97 to the extreme band. It describes the top of the activation range, not population frequency.

Damage per firing (panel 4) is the mean over a mode's measured gate-open firings, from a stratified KL sample of 150-500 firings per pole, using modes with at least 3 measured firings. The noise test reshuffles measured firings among modes, keeping each mode's count, 1,000 times per pole. The outline shows one reshuffle per pole.

Figure 9: Modes, firing and damage across 5,468 neuron poles in the router-weighted five-band sample. Panel 1: The distribution of mode count among neurons. Even when neurons have multiple modes, the mode count is relatively low. Panel 2: The share of KL damage outside the most frequent mode. Panel 3: Firing rate per mode. In general, neurons are dominated by a single mode, sometimes fire on a secondary one, and only rarely fire on tertiary or further modes. Panel 4: Inequality measure for KL damage across modes (without taking into account frequency). Though many multimodal neuron-poles show equally causally-relevant modes, the population is highly weighted towards high levels of inequality, such that many secondary modes have almost no causal effect on the next token.

What do the mode statistics look like in practice? First, a plurality of poles (47%) have only a single mode, regardless of context, and among poles with multiple modes, most have fewer than five. Further, about half of multimodal poles have one dominant mode that accounts for at least 90% of their KL damage. Overall, our findings support the interpretation that a large majority of poles have either exactly one mode or one causally dominant mode.

Mode Relatedness

A factor of variation not captured by the statistics above is semantic similarity between modes within a pole. Often, we observe distinct modes with considerable semantic overlap, such as "Swedish compound common-noun stems" and "Recall of Swedish named entities". To the extent that such similarity exists, it would further support the idea that poles often have consistent meanings.

To assess how related each mode within a neuron is, we embedded each mode description using Qwen-3-Embedding-4B. For each pair of modes within a neuron-pole, we computed the cosine similarity between these embeddings, obtaining a measure of relatedness for each mode pair. We computed the mean, KL-divergence-weighted mean (based on the mean KL divergence from ablation of the neuron in that mode), maximum, and minimum. Below, we show all four, beginning with the maximum. Roughly, the maximum can be interpreted as measuring how many poles might reasonably be said to have fewer than their ostensible number of modes, depending on similarity threshold.

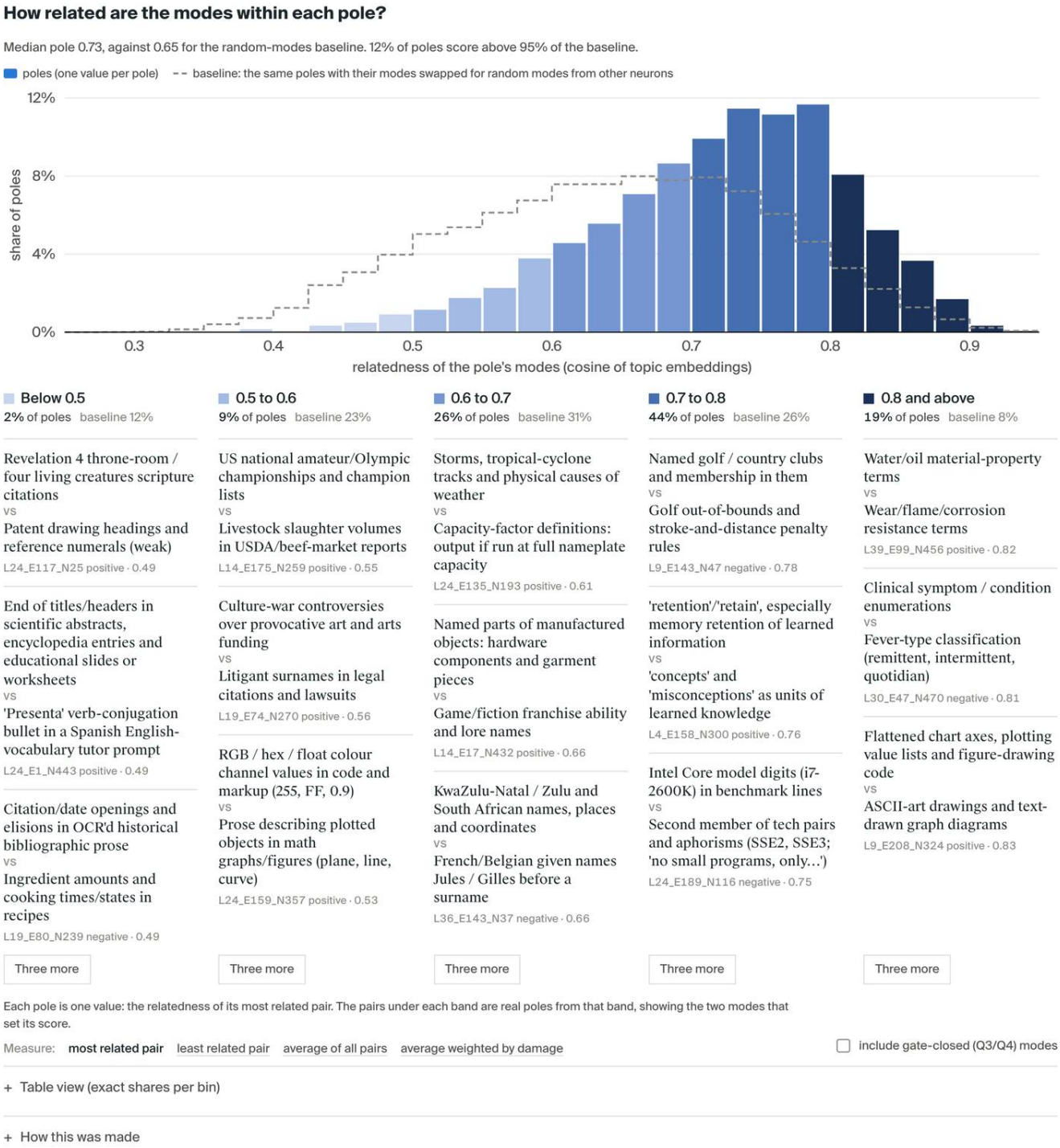


Figure 10: Relatedness of mode descriptions as computed by an embedding model. As can be seen, there is a broad, unimodal distribution of relatedness between different neurons' constituent modes.

We see that the semantic consistency of a neuron can vary widely, but in many cases neurons have multiple modes that are quite related, added evidence of organization in a pole's set of meanings.

How models disambiguate heterogeneous meanings: two circuit case studies

Let us consider Neuron L18_E219_N55, which has two unrelated modes: oral surgery and marine environments. When this neuron fires, it can only write to the residual stream in one direction; there are no different directions for different modes. How does the model know whether this neuron is signaling oral surgery or marine environments?

The architecture of the SwiGLU neuron offers a solution, allowing downstream reader neurons to disambiguate the firing of an upstream multimodal neuron. Consider a neuron n with modes m1 and m2. Often, a reader neuron's up-projection receives input from n, while its gate projection receives inputs from contextual primers associated with, say, m1. Since the reader's firing depends on both its up and gate, it will fire exactly when n fires, *in mode m1*.

In the case of Neuron L18_E219_N55, we can see exactly this mechanism play out. There are two sets of downstream readers-one set for the oral health concept, and another set for the marine environments concept. Both sets read Neuron L18_E219_N55 through their up-projections, but we also identify contextual primers for both the oral and marine modes; these feed readers' gate projection, determining which mode is read.

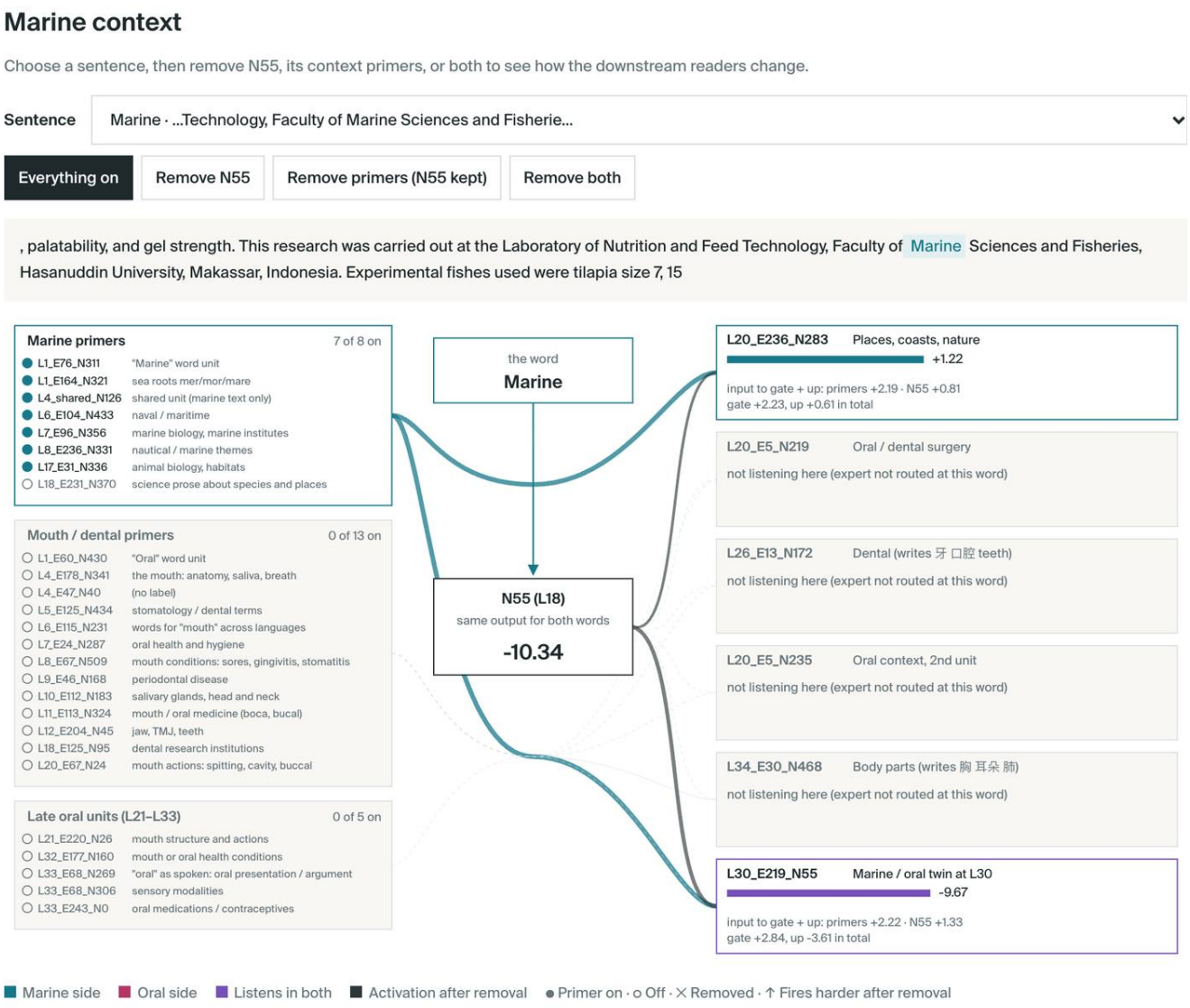


Figure 11: Upstream neurons prime downstream readers to be receptive to L18_E219_N55 in marine or oral health contexts. Use the sentence selector to explore both modes.

In the marine context, the marine primers are active, activating the corresponding reader set. In the oral health context, the opposite happens: oral health neurons fire at both early and late layers, alerting the readers to treat L18_E219_N55's signal as evidence for the oral health concept.

A set of random-direction baselines

It is reasonable to wonder: do random directions yield similarly interpretable activation examples, or are neurons unusually interpretable (Szegedy et al., 2014; Bau et al., 2017; Elhage et al., 2023; Heap et al., 2025; Hewitt & Liang, 2019; Makelov et al., 2023; Zimmermann et al., 2023)? We test this with a variety of random direction types, running them through the same example harvest, subcircuit clustering, and mode detection pipeline that we apply to the neurons themselves. Although random directions have been previously found to be selective for text patterns (Korznikov et al., 2026) we find that the neurons behave differently from random baselines in a number of ways:

- **Neurons recover experts and layer function (as measured by KL recovered vs. complete ablation of the expert or layer)** far more sparsely than random directions; 8 neurons recover as much function as 128-256 random directions.
- **Neurons are wired geometrically to specific downstream readers; random directions are not.** Random directions activate downstream neurons far more diffusely and less specifically.
- **Neurons' meanings narrow as they fire more strongly; random directions do not.** The strongest firings for a neuron are dominated by its primary mode. Random directions remain heterogeneous at all activation ranges.

We provide plots and more detailed data in the appendix. To summarize, however, these results suggest that neurons are a highly meaningful basis for computation relative to random directions (even in the expert basis).

Summary: Neurons as Atomic Units of Computation

Neurons carry out multiple computations through subcircuits, which can be grouped into semantic modes. These modes are usually small in number, indicating that neurons are not intractably polysemantic units of computation, and suggesting that neurons can be described in concise, explicable terms.

Distinct computational roles need not make a neuron an unsuitable unit of analysis even if they are high in number. A neuron participates in different downstream computations - with different semantic interpretations - while remaining an atomic unit of computation. Its write direction is fixed, but the effect of its signal depends on which downstream readers are primed to read it, a property that we can extract through analysis, allowing us to enumerate computational modes. These modes are usually small in number and allow us to readily conduct analysis using neurons as units.

Contextual reuse therefore provides one way for heterogeneous responses to support useful computation. The relevant questions are when downstream consumers can distinguish these uses and what happens when their contexts overlap. Our examples demonstrate mechanisms that support this reuse, but do not establish how prevalent these mechanisms are or how reliably they resolve ambiguity across the model. We leave further analysis of these mechanisms to future work.

We find this to be strikingly good news for interpretability. Though we do not see a perfect one-to-one mapping of meaning to neurons, we *do* see that the number of functions that a neuron carries out is usually relatively small, and can be identified by the text and by looking at which downstream readers are being influenced by the neuron.

Evaluating Neuron Descriptions

We next consider how to evaluate our descriptions of these neurons. A good description should not be merely observational or correlational, but rather should allow us to make causal predictions about the units we are describing, including under intervention. We evaluate these descriptions through tasks that require increasingly precise predictions, from identifying whether a neuron fires to predicting its activation at each token. Our questions are therefore as follows:

- **Input synthesis:** Given a neuron description, can an LLM system create a text that controls the magnitude of a neuron fire, such that it exceeds the maximum found in our 10B text corpus?
- **Detection:** Given a sequence where the neuron fires, and a distractor sequence where it doesn't, can an LLM judge equipped with the neuron description correctly identify the firing example?
- **Classification:** Given a sequence where the neuron is known to activate, and a set of mode descriptions for the neuron, can a blind LLM say, from text alone, which mode is operating? Can it do so for the subcircuit as well?
- **Simulation:** Given a sequence and a neuron description, can a model predict, for each token, what the activation level should be?

As activations increase in magnitude, prediction of neuron behavior becomes easier. When activations are weaker (e.g. p97/3), the neuron often fires on more distantly related topics than those that cause strong fires, which tend to be clearer. However, we are able to make predictions at every quantile within the effective range (p97/3), as follows:

At p97/3, we are able to detect from text whether a neuron should be firing. Activating examples at this quantile are often "blurrier" versions of strong activations. For example, a neuron that fires strongly on discussions of pet microchipping might fire weakly on contexts that discuss tracking wild animals with tags, or on technical documents that describe various types of microchips. Nevertheless, we are able to describe a neuron well enough that we can identify these cases at p97/3.

At p97/3, we are also able to classify which mode is active for a neuron by examining the inputs that made the neuron fire. By looking at what activates the neuron, we can determine which subcircuits and therefore which modes are responding to the text, even when it is not visible from surface features. We can do this with readers as well to a lesser extent.

At p99.9/0.1 and above, we are able to simulate token firing and recover KL divergence caused by neuron ablation. Here, we replace real activations with values that a simulator model (Jev, in this case) provides on a token-by-token basis for an entire sequence, given the description of the neuron. Perfect performance on this task would require in-depth knowledge of attention, inner circuit operations, and movement of information through the network over long contexts, so we do not expect a text description of the neuron to make it possible for a simulator to get a perfect score. Nevertheless, our simulator performs surprisingly well.

Results for all these tasks are summarized in the table below.

Full results

Metric	97th-99th percentile	99th-99.9th percentile	99.9th-99.99th percentile	99.99th+	Top 1,000 by |a| × rw	All bands
DetectionOpus 5.5 picks the firing snippet out of a pair, token marked · 496 poles (chance 0.50)						
All distractors	0.858	0.892	0.927	0.955	0.953	0.914
Same-expert distractors	0.812	0.852	0.901	0.942	0.939	0.885

Metric	97th-99th percentile	99th-99.9th percentile	99.9th-99.99th percentile	99.99th+	Top 1,000 by |a| × rw	All bands
Random-text distractors	0.906	0.932	0.952	0.972	0.970	0.946
Classification · macro-F1Assign each held-out firing to one of the pole's modes · 573 random multi-mode poles, the same firings for every method						
Inputsthe firing's selective gate drivers and opposers, the features its subcircuits are clustered on	0.957	0.972	0.981	0.988	0.990	0.968
Readersthe downstream neurons the firing moves; never used to build the modes	0.827	0.848	0.889	0.938	0.937	0.853
TextJev reads the mode descriptions and the snippet, firing token marked; no examples	0.650	0.703	0.787	0.878	0.881	0.707
Chanceguess modes in proportion to their shares	0.392	0.355	0.415	0.628	0.563	0.290
SimulationAn activation predicted from the description stands in for the neuron inside the model · 486 poles						
Effect restoredshare of ablation damage undone, as a share of what the ceiling restores	Not evaluated	56%	64%	76%	73%†	

Shading runs from chance (pale) to perfect (dark) within each row; grey rows are baselines.

Table 3: *Neuron detection, mode classification, and simulation across activation strengths.* ***Detection*** *measures how accurately an LLM judge identifies the firing snippet from a pair, using same-expert or random-text distractors (chance: 50%).* ***Classification*** *measures how well upstream inputs, downstream readers, or text descriptions identify the active mode on the same held-out firings, scored by macro-F1.* ***Simulation*** *measures how much ablation-induced KL damage is restored by description-predicted activations, normalized to a 10-level oracle ceiling; its "All bands" result pools only the 99.9th-99.99th-percentile, 99.99th-percentile-and-up, and top-1,000 bands.*

Agentic Neuron Science: Generating Descriptions

For generating the neuron descriptions underlying the experiments below, we use a frontier model (typically Claude Fable 5.0 or 5.1, Opus 5.5, or GPT-6 Sol) residing in a harness that allows it to conduct experiments and analyze neurons in detail (Shaham et al., 2024; Schwettmann et al., 2023a; Lu et al., 2024; Gottweis et al., 2025; Singh et al., 2024, Zimmermann et al., 2024, Colin et al., 2024; Marin-Llobet & Ferrando, 2026). We refer to this system as the Neuron Scientist. We provide it with 200 documents where the neuron fired, divided among bands where the strongest neuron fire was in top-N, p99.99, p99.9 and so on down to p75. Activation values within each document are labeled. We then task the Neuron Scientist with producing a description that optimizes a metric like KL recovered through simulation, on a designated training set of texts. It is permitted to conduct a specified number of rounds until it achieves that goal. At each round, the Neuron Scientist develops a hypothesis; tests the hypothesis with experiments (like running texts through a server to collect activations, steering, constructing attribution graphs, or examining the neuron's circuit); submits the description for evaluation against the target metric on a held out dev set of example texts; and receives feedback from the metric on where it failed. It then revises its hypothesis and tries again until it achieves its target or exhausts the designated number of rounds.

At the final step, a supporting agent writes a summary that details the findings and experiments from the research run.

Crucially, the numbers we report are computed on a separate set of test documents not used to optimize the descriptions.

Detection

A faithful description of a neuron should let one identify the inputs it fires on. To test this, we design a detection task. An LLM judge receives the label and description of each of the neuron pole's modes, together with two text snippets from held-out documents, each with one token marked. In one snippet the pole fires on the marked token; in the other, the pole stays below its own 90th-percentile activation on every token. The judge must pick the snippet that makes the pole fire.

We evaluate detection on 500 randomly sampled neuron poles (496 with results), using held-out text from our 10-billion-token corpus with mixed pre-training and post-training data, and report accuracy separately for five activation percentile bands (see Appendix). Claude Opus 5.5 is right 86% of the time for the weakest firings (q97) and 95% for the strongest, against 50% chance. Controls drawn from text routed to the same expert are the harder case, lowering accuracy by 3-5 points, because the expert's domain is present in both snippets.

Share of pairs answered correctly

Opus 5.5 · mode descriptions	p97-99	p99-99.9	p99.9-99.99	p99.99+	Top 1,000	All bands
All distractors	0.858	0.892	0.927	0.955	0.953	0.914
Same-expert distractors	0.812	0.852	0.901	0.942	0.939	0.885
Random-text distractors	0.906	0.932	0.952	0.972	0.970	0.946
Chance	0.50	0.50	0.50	0.50	0.50	0.50

Table 4: Opus 5.5 sees the pole's modes (label and description) and two snippets from held-out documents, the 60 tokens before a token, the token itself (marked) and 8 after; in one the pole fires on the marked token; the other, the distractor, is a snippet where it stays below its own p90 throughout, drawn from text routed to the same expert or from

random text. 496 neuron poles with results from the marked-token detection evaluation; up to 10 held-out firings per band per pole. Mean over poles; All bands pools each pole's pairs.

Mode Classification

A neuron with several modes should affect downstream neurons differently depending on which mode is active. To test whether our modes capture these causal differences, we design a classification task and evaluate how well we can assign a firing example to the correct mode.

Inputs. Each firing is represented by a vector with one entry per upstream unit (earlier-layer neurons and attention components): the size of that unit's direct contribution to the neuron's gate at the firing token, kept for the 400 units that push it hardest towards firing and the 400 that push hardest against. Units that contribute in almost every firing are down-weighted (TF-IDF), and the vector is normalised to unit length. Each subcircuit is summarised by the average of these vectors over its firings in the fitting documents. A held-out firing gets a score for each subcircuit, the cosine similarity of its vector to that average. Each mode takes the score of its best-matching subcircuit, and the firing is assigned to the highest-scoring mode. The subcircuits were clustered on these same features, so this score shows how well the procedure recovers the modes from their own defining features, a near-ceiling reference rather than an independent test. It falls just short of perfect, because the modes come from a graph clustering of firings while this test uses a simpler nearest-average rule, and firings that sit between clusters can land in a neighbouring mode.

Readers. The same procedure in a different feature space: the firing's first-order effect on every downstream routed neuron, meaning how much this neuron's output at that token changes the pre-activation of each later-layer expert neuron the token is routed to. Readers play no part in building the subcircuits, so this is an independent test of whether the modes correspond to distinct circuit behaviour. Layer-39 neurons have no downstream routed readers, so they are excluded.

Text. An NRDM judge (Jev) receives the label and description of each of the pole's modes, together with a snippet of the held-out document: the 60 tokens before the firing token, the token itself (marked) and the 8 tokens after it. It answers a multiple-choice question about which mode explains the marked firing, and returns a probability for each mode. To stop the judge from leaning on mode size, the modes appear in random order under neutral letters, and any sentence describing a mode's size or rank is removed. The judge sees no example firings. The most probable mode is taken as the prediction, and the probabilities are used for separation. Claude Sonnet 5.5 given the same input scores the same. In many neuron-poles adjacent modes are present, often covering features that would co-occur in a text, which makes this a difficult test for the classifier.

Mode classification (macro-F1)

Method	p97-99	p99-99.9	p99.9-99.99	p99.99+	Top 1,000	All bands
Inputs	0.957	0.972	0.981	0.988	0.990	0.968
Readers	0.827	0.848	0.889	0.938	0.937	0.853
Text	0.650	0.703	0.787	0.878	0.881	0.707
Chance	0.392	0.355	0.415	0.628	0.563	0.290

Table 5: Macro-F1, averaged over 573 neuron poles with two or more modes; up to 20 held-out firings per band per pole, with the same firings for every method and no example firings. "All bands" pools each pole's firings across bands.

We find that much of the text-based classification error appears when the highest-firing neuron in the sample arrives via a high-ranked expert. Interestingly, classification based on text suffers, but based in inputs and readers it is relatively stable, suggesting that the text features become less prominent even though the subcircuit itself is still detectable.

Mode classification by expert router rank

Router rank of the expert	Firings	Inputs	Readers	Text (Jev, descriptions only)
1	11,375	0.986	0.943	0.856
2	7,226	0.982	0.923	0.812
3-4	12,250	0.978	0.900	0.764
5-6	11,526	0.975	0.881	0.722
7-8	10,445	0.973	0.865	0.695
Not in the recorded top 8	1,283	0.971	0.852	0.652

Table 6

Simulation

A good explanation should let us predict what a system will do (Hempel & Oppenheim, 1948), and understanding a neural system often means finding a description that is much smaller than the system but still captures how it works (Jonas & Kording, 2017; Lillicrap & Kording, 2019). In interpretability, this idea is known as simulatability, or meta-prediction: an explanation is good if it is short and allows someone - a person or a model - to predict the outputs of the model it explains (Lipton, 2016; Doshi-Velez & Kim, 2017; Ribeiro et al., 2018; Hase & Bansal, 2020; Hase et al., 2020; Colin, Fel et al., 2022; Chen et al., 2023).

This is a difficult task for a text-based description to pass. A simulator must infer many factors from text alone, including the operation of attention and the distribution of input features, often from hundreds of tokens away. A description often reveals whether a neuron will fire, as we saw in the previous experiments, but predicting how *strongly* it will fire is much harder because activation strength depends on how much of each input has accumulated across the context-information rarely apparent in the local text. In our measurements, features of the text explain only about 14% of the variance in firing size among a neuron's firings, against 59% for the activity of its own gate inputs. Consistent with this, giving the simulator a summary of the neuron's input activity and attention at each token raises the share of damage restored by about 10% versus the result we report below. This information varies with each input sequence and is not specified by the neuron's general description.

Although the simulator lacks direct access to the internal signals, it may still capture a substantial part of that effect. To test this, we replace the neuron's actual activations with predictions based on its description and the input text, then measure how much of the damage caused by removing the neuron these predictions repair. We give each neuron pole description to a non-autoregressive decision model (Jev), together with a full document in which a 58-token span is marked. For every token in the span, the model gives a probability for each level from 0 to 9, answering "how strongly does the neuron fire here?". For each of 486 neuron poles we use about 60 held-out documents (and disjoint from those used to write the description), 20 from each of the three strongest activation bands: the 99.9th percentile, the 99.99th percentile, and the pole's top 1,000 activations, where negative poles use the same bands on the negative side. In each span the sampled firing sits at token 45, so the span covers the 45 tokens before it and the 12 after.

We convert Jev's probabilities over levels 0-9 into a single level by rounding their probability-weighted average, then map that level to an activation using a per-neuron lookup table fitted on separate documents. The table assigns each level its mean measured activation, adjusted so that higher levels never correspond to weaker activations, with no added offset or imposed minimum. We substitute these predictions for the neuron's actual activations throughout the marked span, restricted to tokens routed to its expert and activations on the pole being evaluated, including cases where the gate is closed. To measure recovery, we

compare the changes in the model's next-token probability distributions when these activations are replaced by zeros versus simulated values. We measure both changes relative to the original model using KL divergence, summed across the span. Overall recovery is one minus the ratio of mean remaining damage under simulation to mean damage under zero ablation, giving greater weight to neurons whose removal causes more damage.

We compare the simulator with an ideal predictor that knows the true activation at each token and selects the closest value available in the same 10-level lookup table. This ideal predictor recovers 96% of the damage caused by ablation. The simulator achieves 73% of that recovery overall: 56% in the 99.9th-99.99th-percentile band, 64% at the 99.99th percentile and up, and 76% for the strongest 1,000 firings.

We find that we are able to recover a substantial amount of the KL damage via simulation:

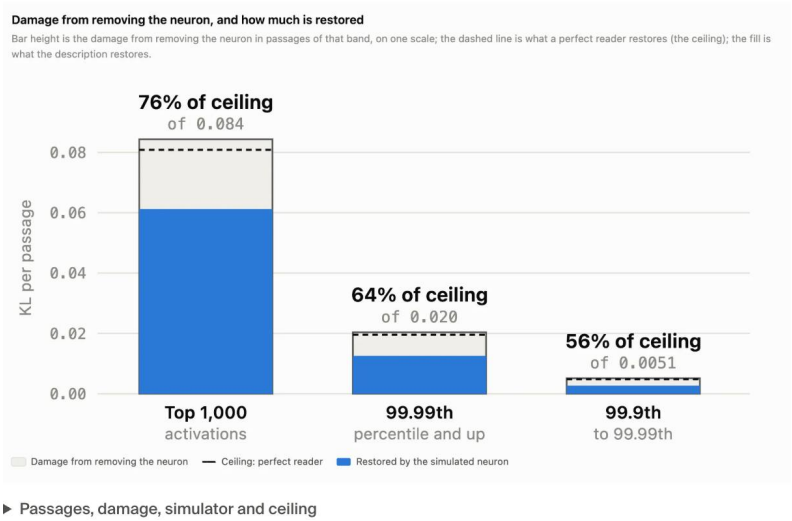


Figure 12: Damage from removing a neuron and the share restored by its simulated activation across the three strongest activation bands. Percentages are relative to a perfect reader with the same 10-level scale and activation table.

Given the high bar that simulation must fulfill, we find the level of simulatability we are able to attain quite significant.

Input Synthesis

A final way to verify our neuron descriptions is to determine whether the description serves as an effective seed for generating texts that cause a maximal neuron activation (Erhan et al., 2009; Olah et al., 2017; Nguyen et al., 2016b; Fel et al., 2023c; Poerner et al., 2018; Thompson et al., 2024; Simonyan et al., 2013; Yosinski et al., 2015; Geirhos et al., 2023; Ghiasi et al., 2022). We attempt this on a random set of 128 neurons in Qwen 3.6 35B-A3B, using Claude Opus and Fable agents to generate the constructed examples.

We implement this procedure as follows. We designate a controller agent, generally a model like Claude Opus 5.5, who directs a set of synthesizer agents. The synthesizer agents' jobs are simply to construct sentences based on the rules that the controller provides them. The controller receives a description of the neuron; no examples are provided. Based on the description alone, the controller agent creates a set of rules for the synthesizer agents, which they use to create candidate sentences. These sentences are then tested against the actual model to see if they cause the neuron to activate. The results are fed back to the controller agent, which then creates a new set of rules for the synthesizers to generate an improved set of candidates. This is repeated for several rounds until it reaches parity with or exceeds the natural corpus skyline, or until the maximum number of rounds is exceeded.

For each neuron, we measure how many rounds it takes to exceed the maximum activation value found in the natural corpus of 10B tokens, and how strongly we are able to make the neuron fire. Intuitively, a better description should result in faster convergence to a maximally exciting text.

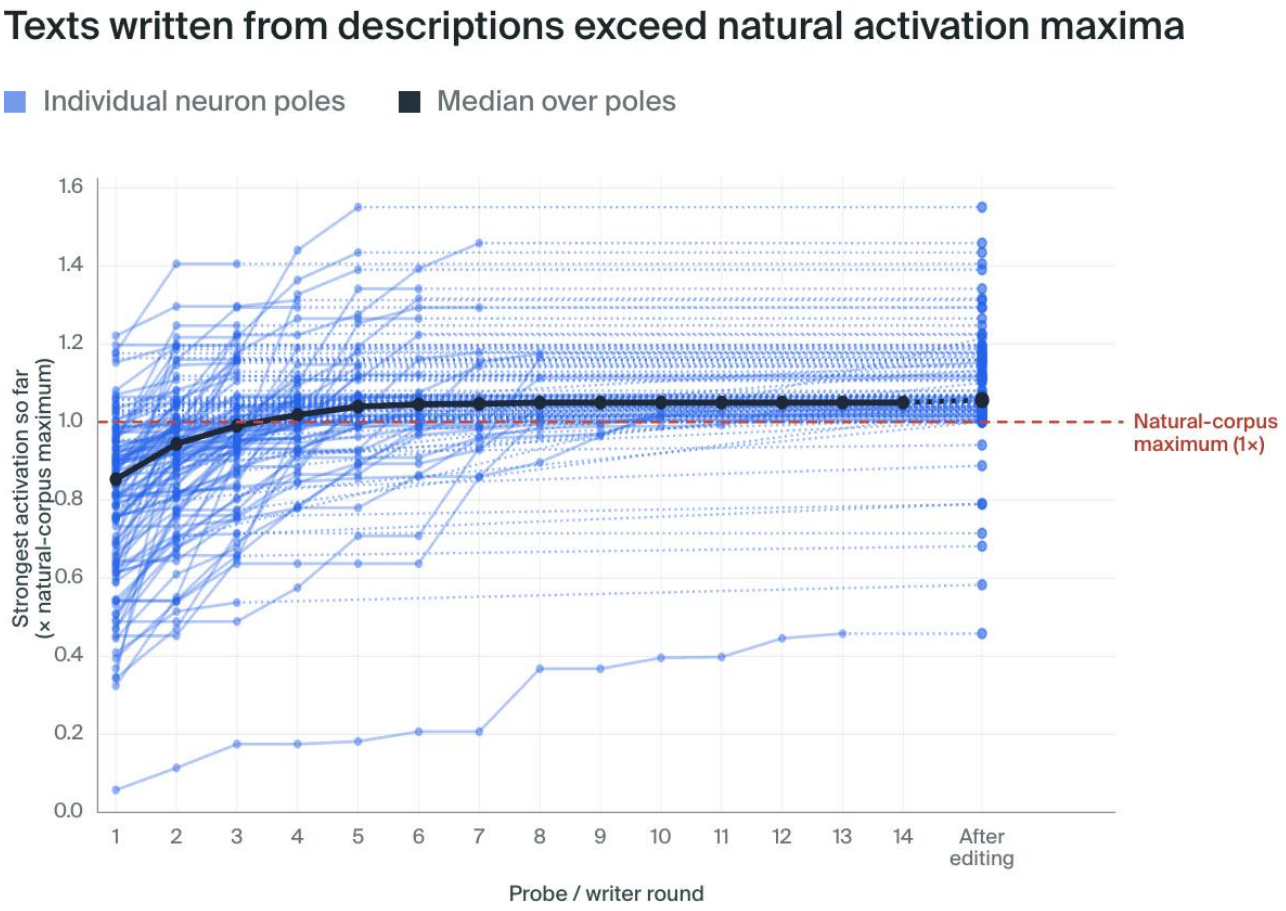


Figure 13: Each thin blue trajectory follows one neuron pole's best activation so far; the dark line is the median. Rounds include controller probe sets and writer rounds, in order, followed by final editing. Writers receive only the neuron's description, never exemplar documents; the controller uses activation feedback to guide them. Of 128 poles (64 positive and 64 negative), 120 exceeded their maximum in a 10-billion-token natural corpus, including 22 only after final editing. Final best: median 1.06×, mean 1.08×, range 0.46-1.55×.

This procedure succeeds in exceeding the natural corpus' maximum 94% of the time. Though this method does not evaluate our ability to capture all of the different activation modes of a neuron, it does demonstrate that we are able to capture a strong sense of what the neuron responds to in its most causal modes through our descriptions.

We show some examples of the generated texts below.

× natural max	
1.55×	f -h | awk 'NR>1 {printf "%s %s\n", $5, $6}' ↵ ↵ echo "Inodes:" ↵ df -i | aw
1.46×	y Pennyquick and her cousin Lady Mid
1.26×	'Sudha Pratibha Nand akumar, 78, of Chennai, passed aw
1.19×	l it keeps going: problems upon problems upon problems upon problems.
1.04×	Τζακάρτα, Μανίλα, Σιγκαπούρη και η Κουάλα Λουμπούρ– κάθε μία συμβάλλε
1.03×	Why?!! Now I will never really know.
1.02×	the miners' old name for it? Answer: methane, which seeps from coal sea
1.02×	不停，黑暗中闪着两只圆圆的眼睛——是貓头鹰。

▶ All 128 neuron poles

Table 7: Each text was written from a neuron's description, with activation feedback but no examples; select a text to see its description. Highlighted is where the neuron fired most; the number compares that to its maximum in 10B tokens of natural text. These eight are chosen to be easy to read.

We consider this a useful validation of the primary mode of each neuron-pole, especially at the higher activation levels, though it does not validate the full range of functions that a neuron can compute.

How we interpret these results

We can synthesize maximally-activating texts across neurons, and predict neuron activations to a significant degree. At p99.9/0.1 and above, we are also able to recover 73% of the perfect-reader ceiling for the difficult task of per-token simulation on a neuron's highest activation bands. Though we see that there is still room to improve these simulation metrics, we can clearly see that increased amounts of intelligence and compute consistently increase our scores. Moreover, there may be a ceiling on what is predictable through text: the description must not only describe what the neuron is detecting from text, but must also to some degree anticipate the intermediate model computations, including attention, that lead to a neuron firing.

We find these results quite compelling nevertheless. Not only can we enumerate a neuron’s modes, but we can also describe its behavior to a meaningful extent! This is far from the expectation we started out with and we think it has strongly positive implications for the field of interpretability.

Example neuron descriptions

So what do the neurons we’ve analyzed look like? We find a vast landscape of fascinating structures among them, encompassing a broad range of computations. We see evidence of universal primitives, collections of convenient heuristics, dataset artifacts, and many more phenomena. Alongside this article we are releasing a dataset of circuit diagrams, activation examples, and Neuron Scientist-authored pages for each of the stratified set of 2,734 representative neurons in Qwen 3.6 35B-A3B. To provide some initial examples:

- **Helping people live more independently** (L13_E133_N30): Responds to writing about technology and support that help older or disabled people communicate, manage daily activities, or live independently. The surrounding context matters: the response brings together information about the kind of writing and the benefits being described.
- **Trying a new approach after getting stuck** (L16_E9_N229): Responds when a problem solver has tried something, failed, and tentatively proposes another approach: “Hmm, this isn’t working. Maybe I should try…”. The response often lands on “Maybe” or “Perhaps”. Looking back with regret, as in “Maybe I should have used…”, produces little response.
- **Registering a moment of self-questioning (**L9_E98_N330): Responds when the model interrupts its reasoning with “Wait”. Increasing its influence can then push the model to settle on an answer too quickly, including accepting a planted wrong answer; pushing it in the opposite direction encourages more self-questioning. Its effect fits a signal that doubt has already been raised.
- **Crossing into the next month or year** (L4_E190_N436): Responds when a date moves into the next month or year relative to an earlier date in the text. It responds at “June” in “31 May - 1 June”, or at the final digit of “2016” after a date in December 2015. The earlier date can be hundreds of word pieces away. Responses are strongest just after the calendar changes; repeating the same month or moving backwards produces little response.
- **Marking a finding as estimated or uncertain** (L14_E2_N240): In research writing, responds to words that qualify a result before naming it: “estimated risk”, “predicted pathways”, “possible interactions”, or “self-reported” measurements. Replacing these with words such as “measured”, “observed”, or “actual” removes the response. Word order also matters: “the software predicted…” produces little response.
- **Locating a place within its former borders** (L24_E230_N452): Responds to the country, empire, or region in a place description when that place used to belong there but no longer does. Examples include “Vilna, Poland” and “Monongalia County, Virginia”. The same country or state name produces little response when it still contains the place, as in “Krakow, Poland” or “Fairfax County, Virginia”.

- **Keeping track of “release” elsewhere in a document** (L36_E110_N224): Its strongest responses occur in documents mentioning “release”, across uses such as a release lever, a mould-release coating, a press release, or a software release date. The response may appear on a space, number, or punctuation mark away from the word itself. Looking only at the immediate phrase misses the connection.
- **Comparing someone’s condition with their age** (L24_E121_N186): Responds to descriptions such as “mature for her age”, “I’m 24 but look 16”, and “wise beyond her years”. The comparison must involve age and an age-related quality, such as maturity, appearance, or health. Replacing age with weight or cost removes the response.

Alongside this writing, we provide a library of neurons for browsing and also release the low-effort labels for all 10 million neuron-poles.

Conclusion

Implications for neuron interpretability

Analyzing neural networks in the neuron basis has often been treated as unworkable. In this view, neurons mix so many unrelated functions, or distribute meaningful computations so diffusely, that studying individual units offers little insight into how the model actually works. Our work stands in contrast to this view, adding to a growing body of work that finds neurons are more interpretable than previously thought. In the model and activation regimes we study, we can interpret most neurons as having one to a handful of computational roles. We can also trace their contributions through sparse circuits.

Alongside Arora & Wu (2025), we find that sparse but strong activations are responsible for the majority of a neuron's causal effect. Removing the weakest activations preserves substantial model function, and brief fine-tuning recovers benchmark performance even with approximately 92% of routed neuron activations removed. Our circuit experiments reveal further sparsity: for specific functions, a handful of neurons per layer can reproduce the effects of the entire MLP layers they occupy. These findings allow us to focus our interpretive efforts on the high-activation regimes where neurons are both most consequential and most readily understood.

Within these regimes, most neuron poles have a single or a small number of coherent, computational roles. Even when neuron poles have multiple roles, we can concisely describe and distinguish them through their upstream inputs and downstream readers. Heterogeneity is less common than previously thought, and less problematic when it does arise: understanding a neuron’s different roles and how the models use them can provide a useful explanation of its computation.

Our agent, the Neuron Scientist, turns these explanations into testable predictions. Its descriptions are precise enough that a simulator can predict neuron activations from the description alone, recovering approximately 73% of the perfect-reader ceiling for KL damage from ablation in the strongest activation regime. These results show that concise descriptions can capture a substantial share of the causal effects of the neuron firings we study.

We approached this work believing that neurons resist direct study and that, as a result, neural networks require some sort of transformation or untangling before we can understand how they work. Instead, we found that models can operate under considerable neuron sparsity; that most neurons have one or a few coherent computational roles; and that we can describe those roles well enough to simulate their firings. We also found that scaling agent-driven investigations reliably improves our explanations. This gives us optimism that continued direct study, with better agents, tools, and methods, can yield significant advances in interpretability.

The ability to devote substantial compute to interpretability in general, and to understanding neurons in particular, is a major change in our field. Agents are now capable enough to conduct experiments, inspect circuits, and test hypotheses (Marin-Llobet & Ferrando, 2026). This lets us conduct detailed investigations of

individual neurons at a scale that would be near impossible to achieve manually. Scaling these investigations could yield superlinear gains: understanding more neurons provides upstream and downstream context, which we have already shown makes interpretation easier. We see substantial room to improve the agent harness, experimental tools, and simulation methods, and to measure how the resulting understanding scales with compute.

Future Work

Much about neurons remains to be understood. We still need to understand how neurons acquire their computational roles during training, how heterogeneous modes arise, and what weak and Q3 activations contribute. We must also establish how far these findings generalize across models and behaviors. But neurons already show substantial promise as a basis for understanding circuits, steering and probe vectors, and higher-level behaviors such as reasoning. The next step is to connect these increasingly precise accounts of individual neurons into explanations of the larger computations they jointly implement.

This work is our first technical post in a series on *native interpretability*: studying the original network via its own units, rather than via a replacement for them, as outlined in “Neural Networks Are Already Clean”. We are especially interested in a few directions for future research: new ways to zoom out from single neurons to the larger computations in which they take part while staying faithful to the network’s own mechanisms; how neurons and circuits form during training; and how attention and neurons work together.

For a more extensive discussion of open questions in neuron interpretability, and what remains to be explained, see the appendix.

Acknowledgements

We are grateful to our friends and colleagues who contributed to this work and provided feedback on early drafts. We would particularly like to thank Aryaman Arora, Leon Bergen, Usha Bhalla, Dan Braun, Lucius Bushnaq, Michael Jae Byun, Oliver Clive-Griffen, Fatih Dinc, William Dorrell, Matt Feiszli, Thomas Fel, Tucker Fross, Atticus Geiger, Tarun Gogineni, Tal Haklay, Dron Hazra, Eric Ho, Thomas Icard, David Klindt, Andrew Lee, Jack Lindsey, Ekdeep Singh Lubana, Jack Merullo, Lee Sharkey, and Connor Watts.

Author Contributions

Curt Tigges: Performed much of the research on the LLM side, including developing sparsification methods, creating methods for identifying subcircuits and modes, designing agents to do neuron research, and conducting wide-scale studies of neuron populations. Identified how neurons’ multiple meanings can be disambiguated downstream. Developed KL-recovery-based simulation and other evaluation methods. Studied thousands of neurons. Wrote much of the post.

Matthew Kowal: Helped run some of the additional experiments for the sparsity and simulation sections. Helped with writing and organizing the post. Studied 10s of neurons.

Thomas McGrath: Provided guidance and ideas for experiments and research directions, provided feedback for writing and early outlines.

Owen Lewis: Helped with project organization, writing, and the design of some experiments and metrics.

Nick Cammarata: Started the project in October 2025. Created the first neuron papers. Built the first agents for describing neurons and circuits. Developed the blind maximum technique and first polysemanticity score measures. Studied thousands of neurons. Improved simulations. Improved the total neuron view and built the mode profiles. Did the Colorado neuron studies and wrote that section. Designed the header. Had a minority role in writing and editing the text more broadly.

Appendix

Additional Results

Extended Sparsity Results and Discussion

Fine-tuning for performance recovery

We performed a brief fine-tuning protocol on the layer top-320 sparse model to see if we could recover performance. Our protocol used only 100M tokens (compared to the multiple trillions used to train the original model), and involved both SFT and RL:

1. **Start:** the masked model. Qwen3.6-35B-A3B with a layerwise top-320 mask, which keeps the 320 strongest of each layer's 4,096 routed neurons per token (ablating 92.2% of routed neurons). We apply the same mask on the model during every training step and all weights are trained. **SFT, as distillation:** The student was trained to match the original model's next-token distributions, using a KL loss against the unmasked clean model as teacher. The data was the unmasked model's own thinking traces on prompts disjoint from every eval, plus 20 percent general web and reasoning text as anchors. We used 91M tokens of this dataset over 1,000 steps. This recovered about 72% of the drop in SuperGPQA, but showed a plateau after further SFT. We thus added RL.
2. **RL with verifiable rewards:** Each round (a cycle of sampling followed by 10 training steps), the current model answered 1,024 knowledge prompts 4 times each: training-split multiple-choice and short-answer questions such as MMLU-auxiliary, MedMCQA and ARC. A sample scored 1 if it finished thinking and its answer matched the gold answer (an exact option letter, or an LLM judge for short answers). Each sample's advantage was its reward relative to its group. The loss was a policy gradient plus a small KL penalty (weight 0.1) toward the clean model, at learning rate 1e-6, 10 steps per round, for 10 rounds. Step 100 was the best checkpoint. It mainly taught the model to commit and stop in time: SuperGPQA reached 45.4 against clean's 44.8, with cap-hits below the original model's.

Why finetune? When we observe rollouts directly, we see that the sparsified model's reasoning tends to loop and self-check excessively. However, when we classified these rollouts with Claude Opus 5.5 agents, we identified that the sparsified model would still mention the facts and correct answers, usually as the last answer candidate-it would just self-check endlessly afterwards. We thus applied the finetuning in order to correct reasoning slip-ups, theorizing that the model might be calibrated to be sensitive to the statistics of the weak activations.

As a control, we also applied the same finetuning regime to bottom-448 (the complement of top-64) and random-64 models, where the filter only allowed those subsets of the neurons through. Fine-tuning the layer top-320 model was successful.

As a result of this fine-tuning, we were able to bring the ablated model to parity and even beyond in some cases. Below are the benchmarks for the original, ablated, and fine-tuned models.

Benchmarks before and after fine-tuning

Accuracy change from the base model, in percentage points. Base-model scores are percentages.

Benchmark	Baseline		Ablated, untrained				Trained under the mask (SFT → RL v1)					Reference: Top-64	
	Base model	Resample	Top-64 Untrained	Top-320 Untrained	Random-64† Untrained	Bottom-448 †Untrained	Top-64†RL v1	Top-320†RL v1	Random-64† RL v1	Bottom-448 †RL v1	Full model† RL v1	Top-64 RL v1 (orig.)	Top-64†RL v3 (RACE)
Routed-exp	-	0%	87.5%	92.2%	87.5%	12.5%	87.5%	92.2%	87.5%	12.5%	0%	87.5%	87.5%

Benchmark	Baseline		Ablated, untrained				Trained under the mask (SFT → RL v1)					Reference: Top-64	
	Base model	Resample	Top-64 Untrained	Top-320 Untrained	Random-64† Untrained	Bottom-448 †Untrained	Top-64†RL v1	Top-320†RL v1	Random-64† RL v1	Bottom-448 †RL v1	Full model† RL v1	Top-64 RL v1 (orig.)	Top-64†RL v3 (RACE)
ert cells ablated													
Classic suiteLog-likelihood, greedy; GSM8K generated 5-shot													
ARC-Challenge · acc_norm	54.7	-	+3.6*	+0.5	−28.8*	−32.8*	+6.0*	+2.0*	−18.3*	−23.1*	+0.1	+6.4*	+5.7*
ARC-Challenge · acc	54.9	-	+2.0	+0.3	−32.8*	−37.3*	+3.1*	−0.5	−22.5*	−25.6*	±0.0	+3.7*	+2.9*
ARC-Easy · acc_norm	72.1	-	+3.7*	+1.9*	−33.2*	−41.1*	+6.0*	+2.8*	−16.3*	−19.4*	+0.1	+6.3*	+6.4*
ARC-Easy · acc	81.4	-	+0.8	+0.2	−40.3*	−49.1*	+2.0*	+0.7	−23.4*	−26.3*	+0.2	+1.8*	+1.9*
WinoGrande · acc	73.9	-	−1.4	−2.2	−23.2*	−23.6*	−1.7	−0.6	−16.6*	−15.9*	±0.0	−1.3	−1.6
HellaSwag · acc_norm	75.8	-	−0.7	−1.8*	−35.2*	−37.7*	−2.2*	−2.3*	−23.7*	−24.4*	−0.2	−1.7*	−1.7*
HellaSwag · acc	57.4	-	−0.7	−1.6*	−25.5*	−26.9*	−1.1*	−1.2*	−17.1*	−20.4*	−0.2	−1.2*	−1.1*
MMLU · acc	84.2	-	−1.6*	−1.2*	−60.1*	−60.9*	−1.6*	−1.0*	−40.5*	−49.8*	+0.2	−1.7*	−1.1*
GSM8K · strict	96.7	-	+0.3	−1.7	−96.3*	−94.0*	±0.0	−0.3	−81.3*	−73.7*	+1.0	−1.3	−0.3
GSM8K · flexible	97.0	-	+0.3	−1.7	−96.7*	−94.3*	−0.3	−0.7	−81.7*	−73.3*	+0.3	−1.3	−1.0
Thinking panel4,096-token cap; a capped trace counts as wrong													
SuperGPQA	44.8	+2.6	−15.2*	−13.0*	−44.5*	−44.6*	−0.5	+0.2	−32.5*	−39.8*	+5.1*	+0.6	+0.7
MMLU-Redux	91.5	+1.0	−3.7*	−3.0*	−89.0*	−90.4*	+1.9*	+0.8	−40.1*	−73.2*	+2.6*	+2.0*	+1.2

Benchmark	Baseline		Ablated, untrained				Trained under the mask (SFT → RL v1)					Reference: Top-64	
	Base model	Resample	Top-64 Untrained	Top-320 Untrained	Random-64† Untrained	Bottom-448† Untrained	Top-64†RL v1	Top-320†RL v1	Random-64† RL v1	Bottom-448† RL v1	Full model† RL v1	Top-64 RL v1 (orig.)	Top-64†RL v3 (RACE)
MMLU-Pro	78.4	−1.1	−8.5*	−8.3*	−78.0*	−78.0*	−1.4	−2.9*	−54.7*	−74.0*	+0.1	−2.5*	−3.4*
IFEval	30.9	−0.6	−0.6	+0.4	−15.9*	−15.2*	+0.2	−1.1	−3.1*	−5.5*	−0.6	+0.2	±0.0
Panel V2Thinking; raw, and on the items both arm and clean finished													
MedQA	92.8	+0.3	−7.8*	−4.6*	−83.9*	−80.6*	−0.4	−1.1	−55.4*	−81.2*	+0.7	−0.7	−0.3
MedQA · both finished	95.9	−0.1	−1.8*	−0.4	−87.6*	−83.5*	−0.6	−1.4*	−47.3*	−83.8*	+0.4	−0.4	±0.0
GPQA-Diamond	63.6	−3.0	−18.2*	−11.1*	−54.0*	−55.6*	+11.6*	+2.0	−44.9*	−49.0*	+9.1*	+9.6*	+6.6
GPQA-Diamond · both finished	88.3	−3.4	−2.5	−1.1	−78.5*	−78.7*	−1.5	±0.0	−56.2*	−67.0*	−1.5	−5.3	−4.0
MATH-500	69.8	+0.8	+7.0*	+5.8*	−69.8*	−69.8*	+0.6	−0.4	−65.6*	−68.4*	+2.8*	−1.6	+2.4
MATH-500 · both finished	99.1	−0.6	+0.6	+0.6	−98.9*	−98.8*	+0.9	+0.6	−69.0*	−87.0*	+0.6	+0.3	+0.9
AIME 24+25	0.0	+5.0	+11.7*	+11.7*	±0.0	±0.0	+5.0	+6.7*	±0.0	±0.0	+1.7	+8.3*	+1.7
AIME 24+25 · both finished	-	-	-	-	-	-	-	-	-	-	-	-	-
HumanEval+	59.1	+0.6	+7.9*	+15.9*	−59.1*	−59.1*	+21.3*	+14.6*	−54.3*	−57.9*	+3.7	+17.1*	+20.7*
HumanEval+ · both finished	85.8	±0.0	±0.0	−0.9	−85.5*	−86.5*	−0.9	+1.9	−78.0*	−83.0*	+1.0	±0.0	+0.9
MBPP+	71.7	+3.7	−20.4*	−20.9*	−71.7*	−71.7*	+13.0*	−1.1	−61.4*	−66.7*	+3.2	±0.0	+8.2*

Benchmark	Baseline		Ablated, untrained				Trained under the mask (SFT → RL v1)					Reference: Top-64	
	Base model	Resample	Top-64 Untrained	Top-320 Untrained	Random-64† Untrained	Bottom-448 †Untrained	Top-64†RL v1	Top-320†RL v1	Random-64† RL v1	Bottom-448 †RL v1	Full model† RL v1	Top-64 RL v1 (orig.)	Top-64†RL v3 (RACE)
MBPP+ · both finished	95.8	±0.0	±0.0	+0.6	−95.7*	−95.9*	+1.1	±0.0	−73.1*	−83.6*	+0.8	±0.0	±0.0

* 95% confidence interval excludes zero (shaded in the online table).

How to read the table and evaluation notes

Resample: base model with a new sampling seed (classic suite is greedy, so none). **Top-64:** only the 64 largest-|activation| cells per routed expert kept. **Random-64:** 64 of the 512 cells per routed-expert slot drawn uniformly at random for every token. **Bottom-448:** all but the 64 largest-|activation| cells kept.

Top-320: layer top-K. At every MoE layer, each token keeps the 320 routed-expert cells with the largest |activation| × routing weight, ranked across all eight routed experts (4,096 cells). This ablates 92.2% of routed cells; the shared expert is untouched. The trained column uses SFT (1,000 steps) followed by RL v1 (100 steps) under the same mask.

Full model: no mask; same recipe (SFT skipped, student = teacher). **Trained columns:** SFT distillation → RL v1 (100 steps) on RL pools with MMLU-aux + ARC replaced by RACE. Panel V2 “both finished” rows compare only items where both the arm and base model produced an answer (the base model finished no AIME item at the 16k cap).

† Measured on 8×H100; the Top-320 RL v1 classic suite was measured on H200. Reference: published Top-64 RL v1 (original pools, H200) and this sweep’s Top-64 RL v3. Intervals: paired bootstrap (generation; 10,000 resamples for Top-320), ±1.96 × paired SEM (classic).

Table 8: Benchmark scores before and after post-training the sparsified model. Red indicates an accuracy decrease; blue indicates an increase. Only changes whose 95% interval excludes zero are shaded. Zero and other results are white; a dash means no result.

What does this tell us? One concern is that we have simply produced a different model: one that is more sparse but that behaves differently from the original model. We believe that despite the fact that the model has been fine-tuned, this provides evidence for the importance of the high-activation regime for the following reasons:

- **The fine-tuning process was quite modest.** Recovery on benchmarks was achieved with only 100M tokens of training data, 9M of which was RL. In the RL step, the model was provided with only 34KB of reward information, as the model was not trained on the correct answers themselves. For perspective: Qwen was originally trained on 34T tokens. Performance recovery from other pruning methods (ShearedLLaMa, Minitron) required billions of tokens of training data.
- **The information was present in untrained rollouts; the observed fine-tuning benefits were primarily conciseness and answer commitment.** The layer top-320 model would output the correct answer before fine-tuning but endlessly self-check. Accuracy between the models before and after finetuning in cases where the model *did* produce an answer stayed flat.
- **Our controls demonstrate that the complement of top-64 is completely non-functional.** Bottom-448 is not able to produce answers, and fine-tuning did not restore it.

- **Examination of the most-changed neurons after fine-tuning reveals that the neurons are still functionally the same.** We ran a token harvest on the model after training, obtaining example texts across percentiles. Semantically, the neurons retained their meanings.

Sparse neuron circuits

We observe even more sparsity than the results above through circuit analysis, an observation previously made by Arora & Wu (2025). A complementary approach obtains interpretable neuron-level circuits by training transformers with sparse weights (Gao et al., 2025). We see this from two primary types of experiments: sparsifying attribution graphs obtained via relevance patching (which we do for single steps of inference and then for continuous autoregressive generation); and also using community detection methods to find a small group of neurons that are both necessary and sufficient for various functions.

RelP circuits

We conduct relevance patching (with some modifications; see Appendix) for a wide range of prompts, obtaining attribution graphs containing hundreds of thousands of neurons, and gradually sparsify the graph in order to measure what percentage of neurons suffice to capture 90% of the logit difference between the top logit and the average vocabulary (Vig et al., 2020; Heimersheim & Nanda, 2024; Zhang & Nanda, 2024; Meng et al., 2022a; Mueller et al., 2024; Mueller et al., 2025; Pearl, 2001):

$$F(N) = \frac{\ell_N - \ell_0}{\ell_{\text{clean}} - \ell_0}$$

A few thousand neurons, 0.2 to 0.8 % of the active routed units, recover 90% of the margin. Random sets of the same size recover at most 0.02 and never restore the top-1. These neurons are almost always activating at their strong tails:

cell set (RelP rank)	in p10-p90	beyond p1 / p99	beyond p0.1 / p99.9	median |act|
top 100	1-6%	82-91%	~52%	2.6-2.9
top 800	5-8%	54-63%	~24%	0.9-1.3
top 3,200	13-18%	37-44%	~14%	0.5-0.6
top 8,000	20-26%	28-35%	~10%	0.34-0.42
uniform null	80%	2%	0.2%	typical neuron's p90 ≈ 0.16

Table 9

Continuous generation under circuit sparsity

The RelP circuits above each explain a single token prediction. To see whether this sparsity holds for autoregressive generation, we generated text from ten prompts across five topics (factual continuations, a story, code, a chat-template reasoning trace and a memorized passage), and computed one RelP graph for every generated token (about 1000 total).

We then measure sparsity in two settings. In the first, we mean-ablate every neuron outside the kept set-at the current token and at every earlier token-so the kept neurons must also reconstruct whatever the earlier text contributed. In the second, the earlier tokens keep their real activations, as they would after a normal forward pass, and only the current token's neurons are kept or ablated.

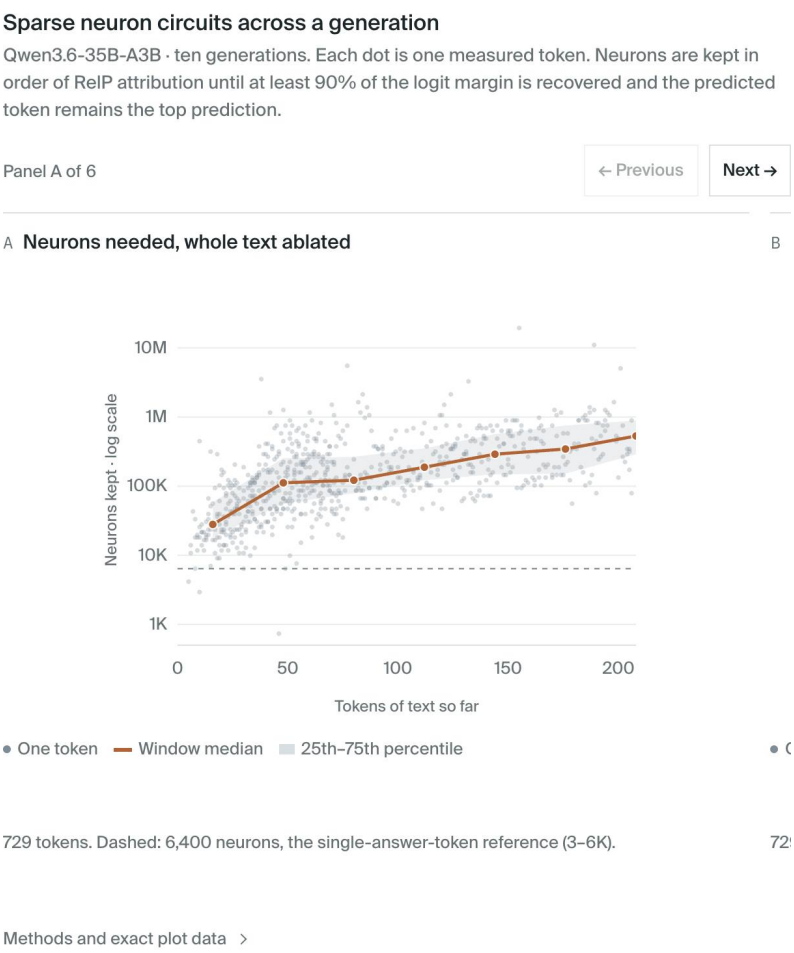


Figure 14: Sparse neuron circuits across a generation. Each point is one generated token from ten generated sequences. Whole text: neurons outside the kept set at the current and all earlier tokens are mean-ablated. Current token: only neurons outside the kept set at the current token are mean-ablated; earlier tokens keep their real activations. Median lines use 32-token windows with at least ten tokens from at least three texts; shaded bands show the middle half of tokens, not confidence intervals. A: Neurons that must be kept to recover 90% of the logit margin when every other neuron in the text, at the current token and all earlier tokens, is mean-ablated; the dashed line marks the single-answer-token result from the previous section. B: The same counts as a share of all active routed neurons in the text so far; dashed lines mark the 0.2-0.8% range for single answer tokens. C: Neurons that must be kept when only the current token's neurons are ablated and earlier tokens keep their real activations. D: KL divergence of the next-token distribution from the original as neurons are kept, with the whole text ablated, one line per range of text length ("none" = every neuron ablated); the dashed line marks 6,400 neurons. E: The same with only the current token ablated. F: Distribution of KL with only the needed neurons kept, for both settings.

When we ablate across the whole sequence, the number of neurons needed grows about linearly with the length of the text (Panel A). As a share of the active routed neurons in the text (Panel B), however, it holds roughly constant at about 1% (median 1.2%), not far from the 0.2 -- 0.8% we see for single answer tokens. When only the current token is ablated (Panel C), a median of ~520 neurons (0.3% of that token's routed units) recovers 90% of the margin, and this does not grow with the length of the text.

These sparse circuits also largely preserve the broader next-token distribution, despite being selected only to recover the target logit difference. A fixed neuron budget becomes less effective as texts get longer when ablating the whole sequence (Panel D), whereas ablating only the current token shows no comparable deterioration with context length (Panel E). Allowing each circuit enough neurons to recover 90% of the logit difference typically keeps the next-token distribution close to the original in both settings, although a minority of tokens show substantial deviations (Panel F).

In summary, when generating a token sequence, the model writes each new token with a few hundred neurons, leaning on what it already computed for the previous tokens. Only when we also ablate the earlier

tokens does the required neuron count grow with length, because every earlier token's small circuit then has to be rebuilt too, which equates to about 1% of all the neurons that originally fired during generation.

Transcoder Evidence for Sparsity

Transcoders approximate MLP computations with sparse replacement features that support circuit analysis (Dunefsky et al., 2024). Further evidence for neurons as a sparse, interpretable basis comes from Shabalin & Piotrowski (2026), who find the following:

- Transcoder features tend to recapitulate neurons (with aligned encoder and decoder directions), and these are the only stable features across different transcoder training runs (Paulo & Belrose, 2025; Leask et al., 2025; Fel et al., 2025; Heap et al., 2025; Chanin et al., 2024; Kantamneni et al., 2025; Wu et al., 2025);
- Top-K MLPs perform better than transcoders (on KL divergence) at matched sparsity after model finetuning

They conclude, overall, that transcoder features add relatively little beyond what can be obtained from neurons.

We take this further and investigate the question: When transcoders learn additional features that are not neuron-aligned, what do these features represent? Using orthogonal matching pursuit (OMP) to determine what neurons explain the features (Pati et al., 1993; Mallat & Zhang, 1993; Olshausen & Field, 1997; Costa et al., 2025, O'Neill et al., 2025), we find that the remaining transcoder features usually end up representing *sparse combinations* of neurons. These are usually composed of 2-64 frequently co-firing neurons and include multiple types:

- Features where the encoder direction is a mixture of several neurons' input directions, but where the decoder direction is aligned with the dominant neuron in the group
- Features where the encoder and decoder are both mixtures of neurons

Additional transcoder features appear to consist of recapitulation of earlier-layer neurons and overfit features for specific textual templates.

1 What each feature is, by layer and width

One row per dictionary. Dead: share of all features with 0 fires in 10M positions. Class shares: percent of the features that fired (each row sums to 100). Neurons with a copy: share of the layer's 131,584 neurons that have at least one neuron-copy feature. Census over 10,000,463 positions (12,800 documents).

	Encoder reads	Decoder writes	What it is
Neuron copy	one neuron's input rows	that same neuron's output column	The transcoder rediscovered a neuron. *L36 f101570 = L36/E116/N363+, nutrient cycling (read 0.91, write 0.97).*
Neuron + read-side condition	that neuron's own input row first (98–99%), plus a correction from 10–60 other neurons' input rows	one neuron's output column	The neuron's own trigger, narrowed by context: it writes exactly what the neuron writes, but only in part of the neuron's range. *L36 f28546 writes like L36/E145/N226– ("th"/"st" after digits) but fires on century markers ("19th century").*
Coalition: neurons fire together	one input slice from each of many neurons	a sparse sum of 2–64 neurons' output columns, and those neurons fire with the feature	A group of neurons the model itself activates together, packaged as one feature. Mostly means what the neurons share (1b). *L19 f37368 = summation-notation neuron + binomial-coefficient neuron → the number inside \binom.*
Combination: neurons do not fire together	as above	a sparse sum of neuron outputs, but those neurons do not fire when the feature does	Matches neurons on paper only: the weights fit a sum of neurons, but the activity does not back it.
Combination: too rare to test	as above	a sparse sum of neuron outputs	Geometrically a coalition like the two rows above, but it fired fewer than 10 times on the 919k-position co-firing ladder, so whether its neurons fire with it could not be tested. Similar in size to the tested coalitions (1c).
No same-layer match	nothing sparse	nothing within 64 neurons reaches cosine 0.7	Explained in section 2: mostly echoes of other layers' neurons or fixed phrases.

Figure 15: What each feature is, by layer and width.

Transcoder feature composition

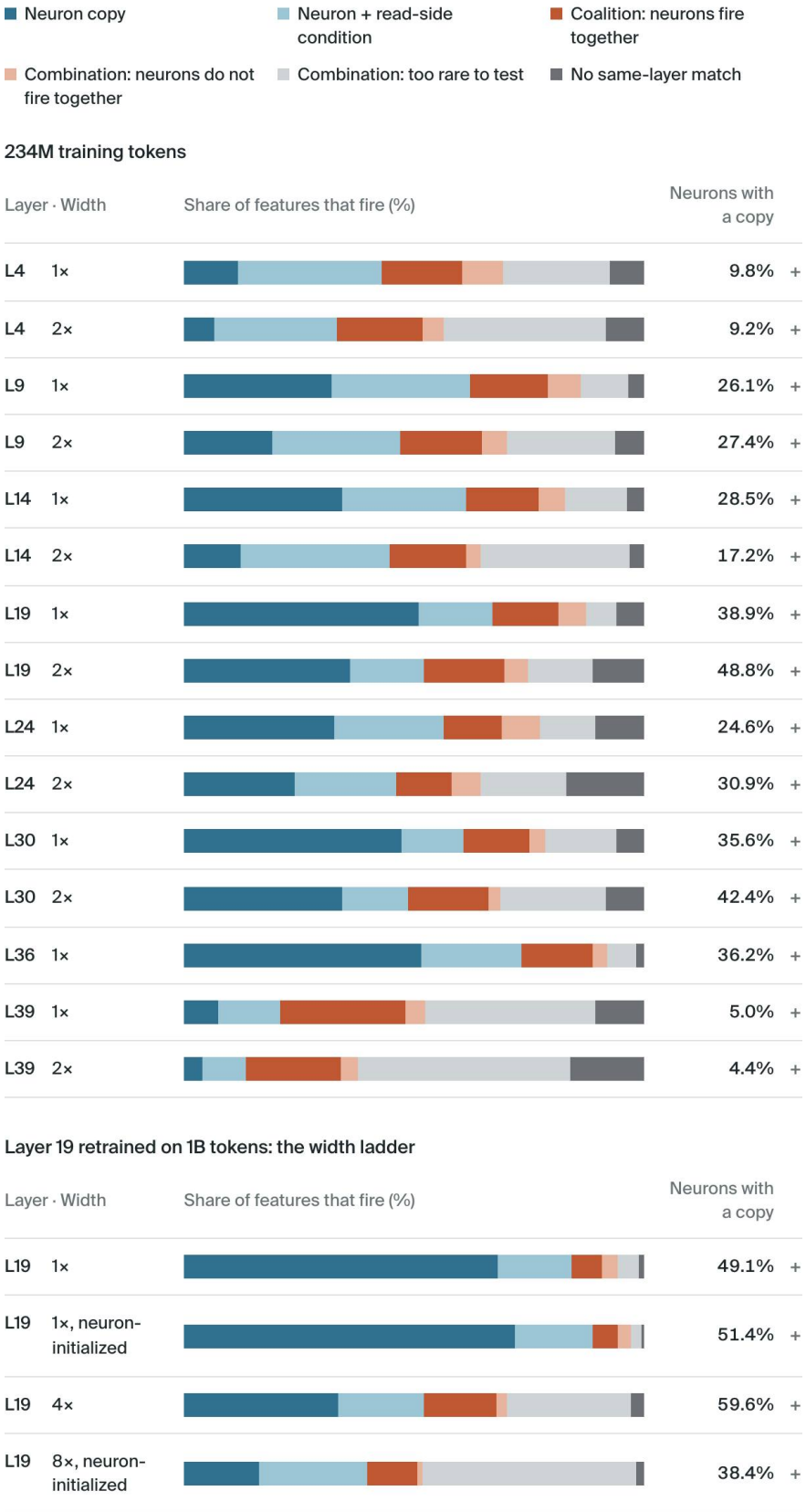


Table 10: Transcoder feature composition.

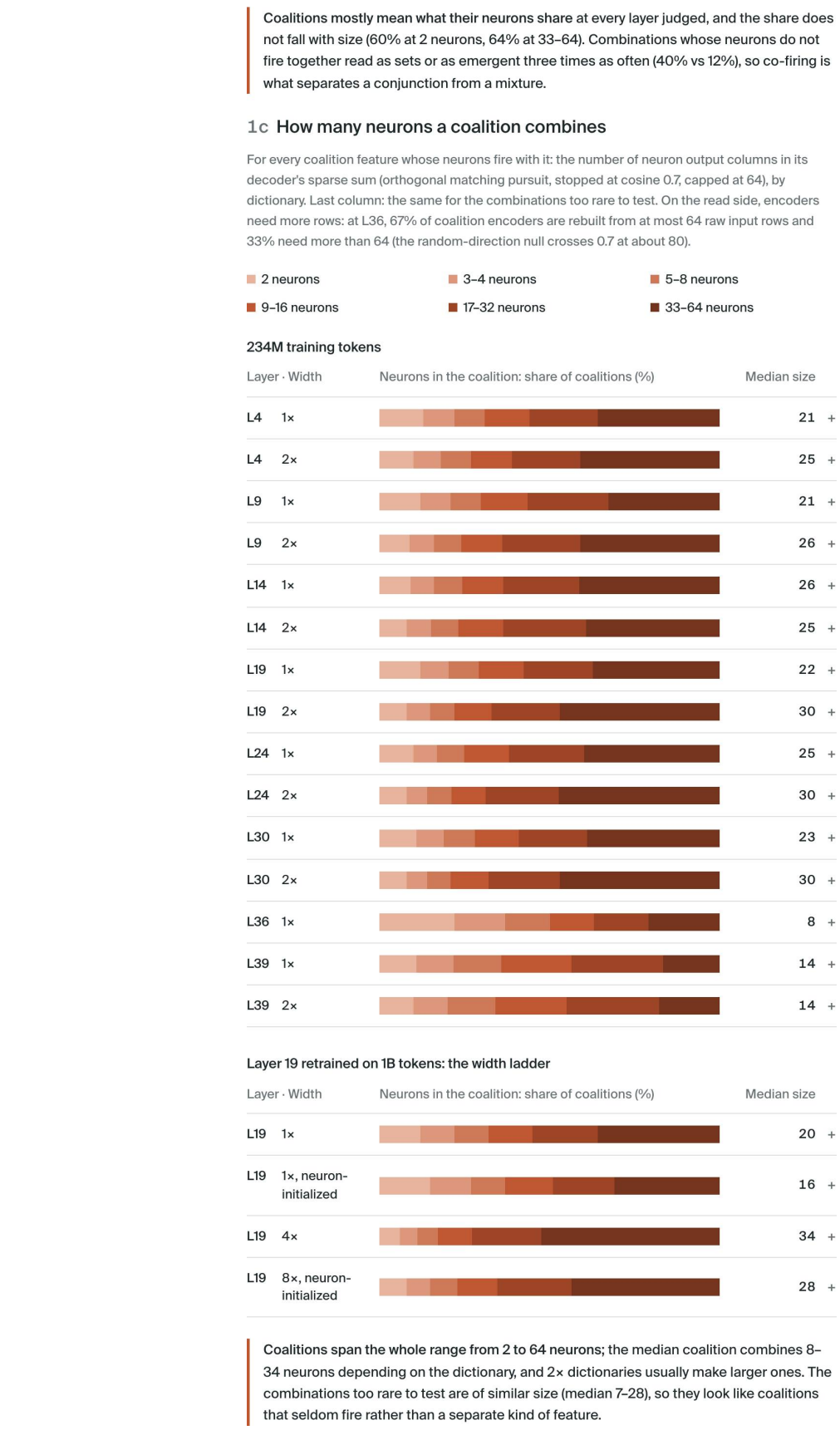


Figure 16: Neuron coalitions by size.

When we examine transcoder features that consist of sparse combinations of neurons, we find that the transcoder feature represents a semantic combination or intersection of the neurons it is related to, rather than novel unrelated features. This suggests that, to the extent that there are features carried by multiple

neurons, they are computed through composition (Olah, 2023; Mu & Andreas, 2020; Smolensky, 1990; Braun et al., 2025; Bushnaq et al., 2025; Braun et al., 2024).

Examples: features next to their neurons

Descriptions are short summaries. Feature descriptions are the exemplar gloss where one was written (coalition, echo and template rows), otherwise the dictionary label. Weights are the decoder's sparse-combination coefficients; the sign picks the neuron's pole (+ positive, − negative). Echo and template rows come from the layer-19 dictionary of section 2.

Kind	Transcoder feature	Constituent neurons (weight)
Neuron copy	L36 f101570 Discussions of nutrient cycling, recycling, and reuse in ecological/agricultural contexts read cos 0.91 · write cos 0.97	L36/E116/N363+ Nutrient cycling, decomposition, and recycling of organic matter in ecosystems, with terms like bacteria, nutrient, organic, decompose.
Neuron + condition	L36 f92499 Permutation formula and reasoning about selecting/arranging distinct items (P(n, r) = n!/(n-r)!) read cos 0.42 · write cos 0.87	L36/E72/N477+ Combinatorial and algebraic formulas using variables n, k, c, r with fractions, exponents, summations, combinations, permutations, factorials, ratios, series.
Neuron + condition	L36 f28546 Ordinal century references like "17th", "19th", "20th" marking historical centuries or time periods read cos 0.41 · write cos 0.83	L36/E145/N226− Ordinal suffixes "th" and "st" after digits in century references, like "19th century" and "21st century".
Coalition: intersection	L19 f37368 Numeric digits in binomial coefficient arguments, like the top slot of \binom{1}{7}, C(n-1, m), C_{23}^{5}, C(4,2). 2 neurons in total	L19/E54/N375+ **+0.60** Mathematical expressions with summations, subscripted variables, superscripts, and LaTeX notation like ∑_{k=1}^n, parentheses, and brackets. L19/E54/N168+ **+0.55** Binomial coefficients and combinatorial notation like "C(n, k)" and "n choose k", with variables n, k, r, x, plus "coefficients", "sorted", "frequency".
Coalition: intersection	L36 f7323 Nouns for measurable attributes (diet, size, habitat, morphology, mineral) used to distinguish or categorize entities, including organisms and minerals. 2 neurons in total	L36/E116/N377− **−0.65** Measurable traits like size, color, mass, or diet analyzed or compared in biological, ecological, experimental contexts, with "distribution," "abundance," "variation," "classification" L36/E116/N13− **−0.57** Distinguishing or confusing similar species by color, wing markings, tail features, size, seasonal plumage; hybridization, separation, identification traits.
Coalition: intersection	L9 f69927 Words for leaders or leadership of political, military, or organizational bodies ("leader," "led," Spanish "lider," Punjabi "agvai"), plus linking prepositions like "of" 3 neurons in total	L9/E33/N395+ **+0.57** Internal structure of communist and revolutionary movements: sections, chapters, groups, affiliations, and integration within international organizations, historical parties, labor movements, ideological factions. L9/E33/N406− **−0.31** The word "leading" meaning a key or prominent role, as in "leading part in," "leading member of," "leading figures," "leading theatres" L9/E33/N426− **−0.21** Mentions of "Central Committee" (or its translations) as a governing body within parties, governments, or institutions, especially in roles and decisions.
Coalition: set (any of)	L36 f111164 The word "grand"/"Grand" in any sense: grand jury, Grand Duke, Grand Prix, grandstand, grand opening, Grandey, grandiflora. 2 neurons in total	L36/E148/N198− **−0.66** The word "grand" in legal terms like "grand jury" and indictment, and in psychology like grandiose, inflated self-image. L36/E148/N127+ **+0.40** "Grand Duke" and "Grand Duchess" noble titles in historical or genealogical texts, with places, dates, family relationships.
Echo	L19 f26 Token "Crest" in "Family Crest Products" headings on genealogy pages with surnames, notables, migration dates.	L14/E219/N15− Heraldry and noble genealogy: coats of arms, crests, bends, armorial bearings, heralds, family branches, titles, cognomen.
Echo	L19 f3530 The word "opposite" in geometry problems, like "sides a, b, c opposite angles A, B, C respectively" in triangle ABC.	L4/E77/N122− Triangle ABC sides a, b, c paired with opposite angles A, B, C, via the phrase "opposite to".

Kind	Transcoder feature	Constituent neurons (weight)
Echo	L19 f1456 Code-golf challenge rules stating that the shortest entry wins: "code-golf", "fewest bytes wins", "lowest byte-count", "shortest".	L9/E137/N294+ Code golf and optimization: minimizing code length or data size, text processing, with words like "characters," "bits," "saved," "shortest," "golf"
Template	L19 f3791 Download notice telling users to ensure they have an application to open this file type before downloading or purchasing.	no same-layer neuron expresses it; a fixed phrase recurs across its examples
Template	L19 f11026 Quirky author-bio blurbs ending scraped blog posts, like "Zombie enthusiast. Subtly charming travel practitioner. Webaholic"	no same-layer neuron expresses it; a fixed phrase recurs across its examples

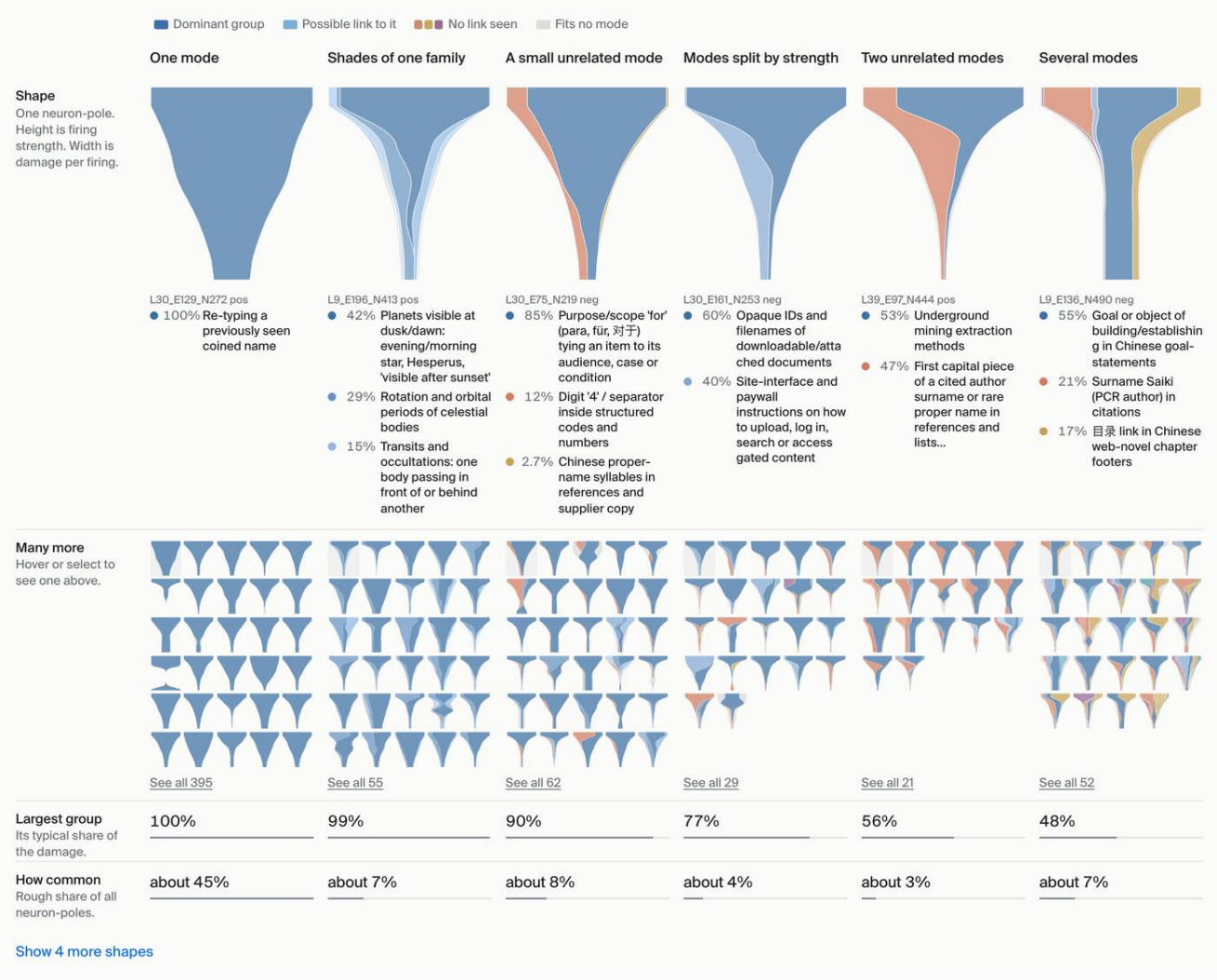
Table 11: Examples of semantic composition. Multiple types of composition occur here, and we also show the neuron-copy features along with the neuron feature modulated by a condition.

Neuron coalitions usually consist of groups within the same expert, and we find that the mixed features' labels generally track the dominant neuron in the group or represent a semantic intersection of those neurons.

Although this does not rule out features spread diffusely across many neurons, the features recovered by transcoders support a picture in which individual neurons are meaningful units of computation, and concepts without a dedicated neuron are represented through combinations of neurons.

Additional Analyses of Modality Profiles

Neuron-pole profiles



▶ How to read these profiles

Figure 17: Neuron-pole profiles.

How to read these profiles

Height runs from weak, common firings to strong, rare firings. Width is average KL damage per firing, compressed using a fourth-root scale so weak firings remain visible. Areas are not damage shares; the printed percentages give frequency-weighted damage shares among assigned modes.

Blue marks the dominant group of linked modes, lighter blue a possible link, and other colors groups with no link seen. Gray is reserved for firings assigned to no mode. Gate-closed firings (negative gate inputs, Q3/Q4) are excluded.

The group boundaries and prevalence estimates are approximate. They use 991 sampled poles, excluding 50 without sufficient gate-open data. Some single-mode examples stand in for others; the displayed shares sum to about 100%.

The browsable selection omits 87 poles flagged for explicit content or interpretation concerns; these remain in the prevalence estimates. Mode labels and link judgments are provisional.

Additional subcircuit and mode comparisons

There is much more similarity between subcircuits in a mode vs. subcircuits between different modes:

Top-50 overlap per pair	Within a mode	Between modes	Ratio	Neurons where within beats between
Inputs	9.8%	2.0%	4.9×	96%
Readers	9.9%	3.1%	3.2×	95%

Table 12

We also see that modes are highly identifiable by the set of neurons reading the target neuron at a given activation, even at the lower percentile bands:

Band	Readers, mode	Chance
q95	0.76	0.33
q99	0.79	0.33
q99.9	0.84	0.31
q99.99	0.88	0.35
extreme 1,000	0.95	0.48

Table 13

Example neuron modes

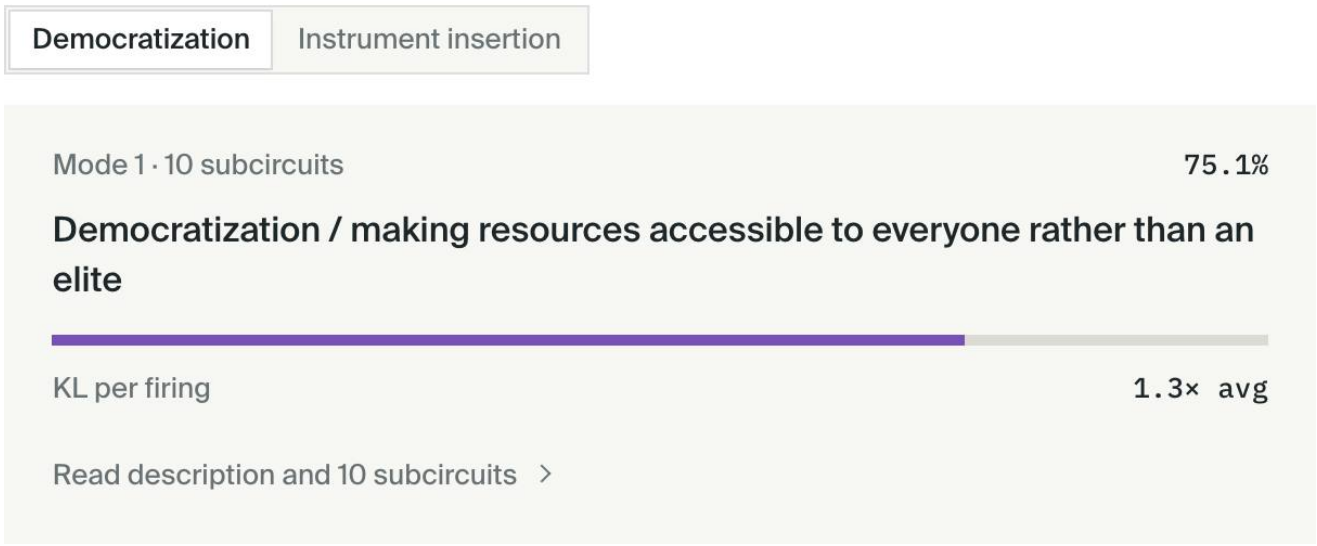


Figure 18: Two modes (of two different neurons), each of which contains a rich cluster of related subcircuits. Each "c" prefix shows a subcircuit label, and percentages on the right indicate how much of the neuron's firing is performed by the subcircuit on our dataset.

Random Baselines for Neuron Circuits

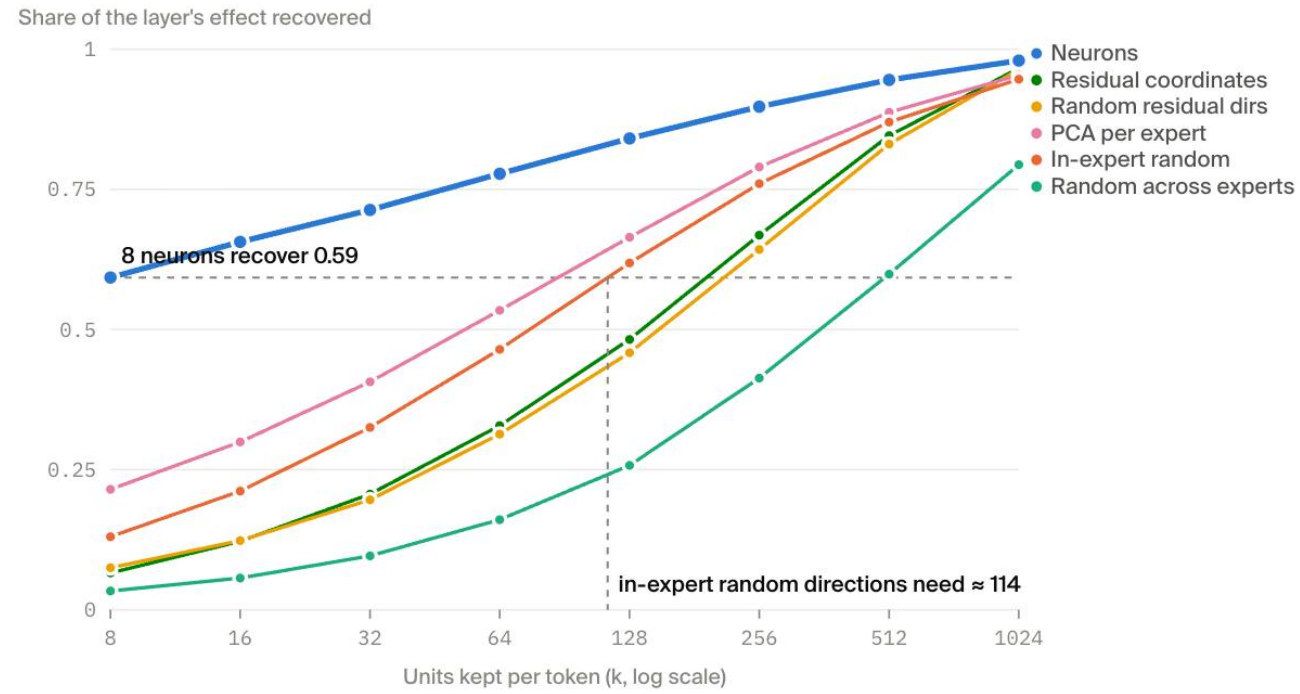
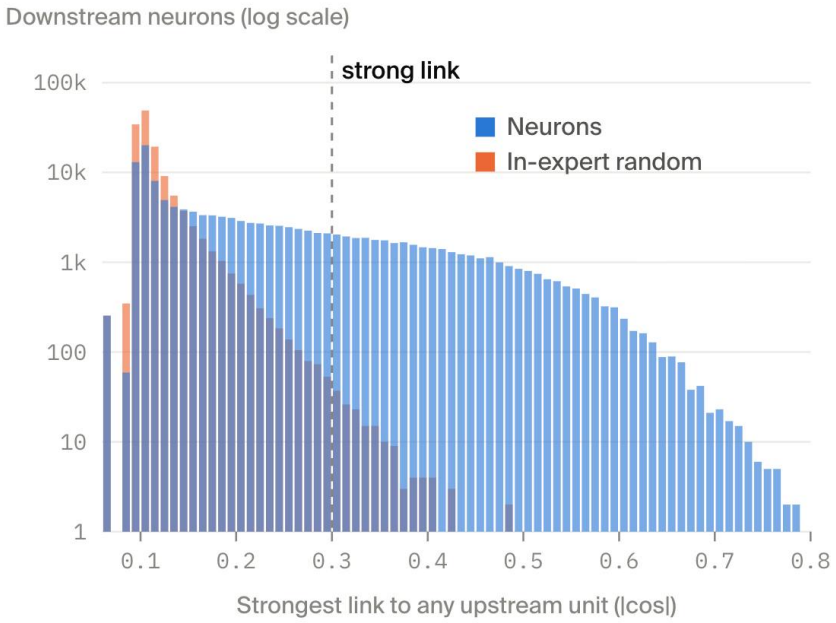


Figure 19: At layers 4, 14, 24 and 36, each token keeps only its top k components of the routed experts' output and the rest are removed; the y-axis is how much of the change in the model's output (KL) that keeps.

Specific neurons are wired to specific neurons; random directions are not



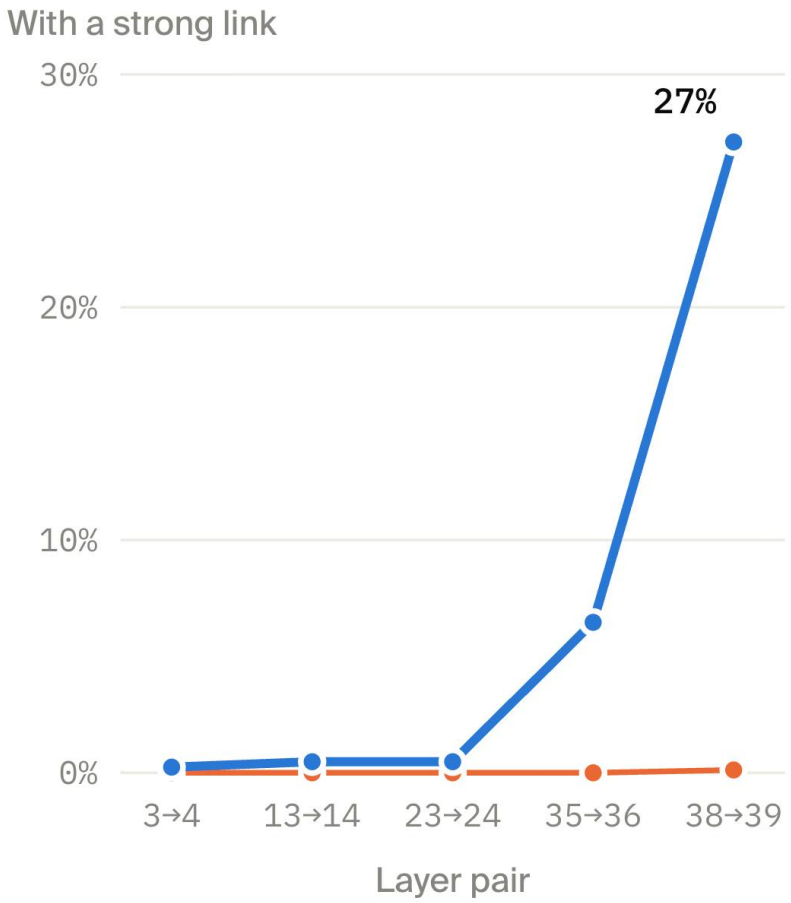


Figure 20: For every downstream neuron, the strongest match between its input (gate) vector and the output vector of any of the 131,072 upstream units. No data is involved, only weights. A strong link has |cos| above 0.3.

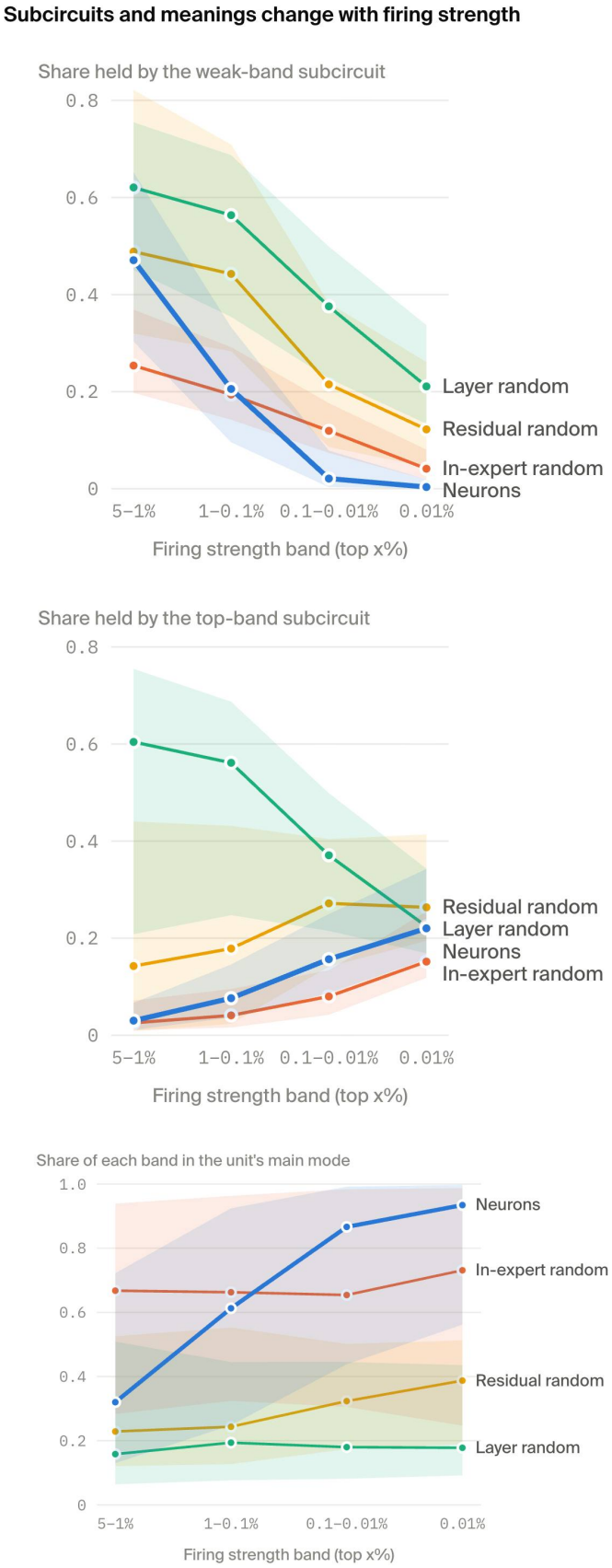


Figure 21: Subcircuits are clusters of a unit's firings that share their upstream drivers (no LLM involved). The final panel groups the same bands by readable mode, using the Opus reader's modes. Lines are medians; shading shows the middle half of units.

Figures 19-21: Random baselines vs. neuron phenomenology. Figure 19: recovery of ablation function for sparse selections of neurons vs. random directions, selected on a greedy basis. Figure 20: selectivity of reader neurons for

neurons vs. random directions. Figure 21: differences in subcircuit and mode distribution through the activation range of a neuron vs. random baselines.

A Colorado neuron with a rare Tornado mode

We examine the most popular counterexample we could find to Neural Networks Are Already Clean: a Colorado neuron in Qwen3-32B (L55/N11703) that also fires on code using Tornado, a Python web framework (Brangus, 2026).

About 98.8% of the neuron's causal effect comes from text about Colorado, other mountainous places, or other locations, decreasing as places become less Colorado-like. Much of this text is only subtly about Colorado: a citation of Lynne Rienner Publishers, which is based in Boulder; a review of South Park; a quote from a researcher whose CU affiliation appeared 150 tokens earlier. For each ambiguous case, we ablated tokens one at a time and traced the circuit driving the activation, rather than judging relatedness from the text alone. In the remaining 1.2%, no single input strongly activates or suppresses the neuron; its activation is composed from thousands of small positive and negative contributions.

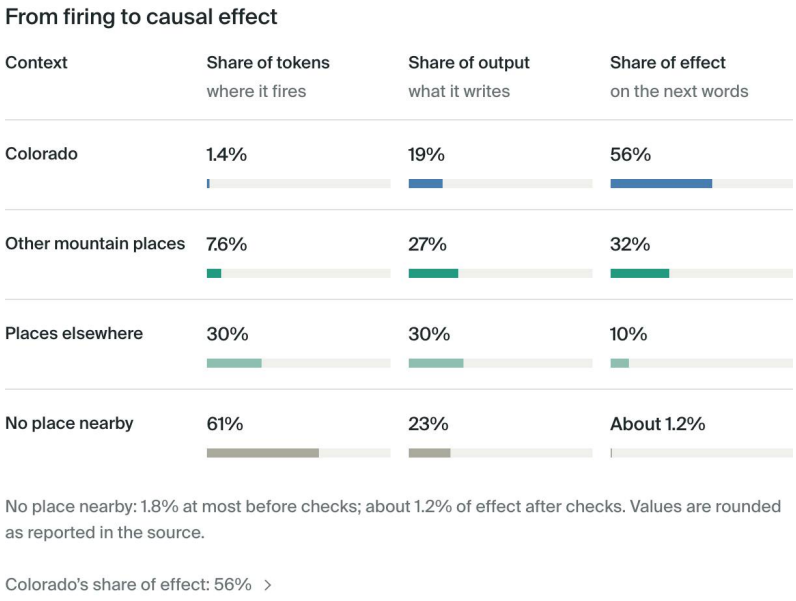


Figure 22: From firing to causal effect.

The Tornado association is causally real. Asked which state Tornado was developed in, Qwen3-32B assigns 30% to Colorado and 0.7% to the correct answer, California (Brangus, 2026). Suppressing the neuron reduces Colorado to 1% and raises California to 4.7%. The neuron's meaning alone predicts this error.

Tornado is not one of a large set of hidden secondary functions. Across 100 million tokens of ordinary text, the neuron fired at ≥30% of its typical Colorado peak 3,812 times. Of these firings, 98.5% were on Colorado or other places, Tornado appeared three times within a single document, and the remaining 1.4% formed 18 small groups of one or two documents each. On a code corpus, 98% of strong firings were on Tornado or code that calls it.

The two computations interact very little. Tornado predictions run mostly through code and web-library neurons. The Colorado neuron appears in 6.6% of their attribution graphs, and zeroing it costs 0.005 nats per token, a twelfth of the cost of Tornado's own top detector. Among the roughly 200 neurons that contribute to predicting Colorado, 94% of sampled firings reflected those neurons' primary functions, and none of the remainder involved code. Colorado and Tornado route 2% of their relevance through shared neurons, comparable to what Colorado shares with unrelated web libraries.

Across the broader circuits behind Colorado and Tornado framework related content, both run mostly through neurons performing their primary functions, with about 5% of firings appearing unrelated. This

estimate fell with each closer inspection (from an initial 10-16%), because subtle relationships are difficult to identify across hundreds of neurons, so we treat it as probably higher than the true amount with substantial uncertainty. The unrelated firings are dispersed across many neurons rather than sharing a common theme. The one recurring overlap is Tornado itself: because layers 32-55 appear to copy layers 8-31, the pathway in which Tornado neurons feed a Colorado neuron at layer 14 recurs at layers 37-38.

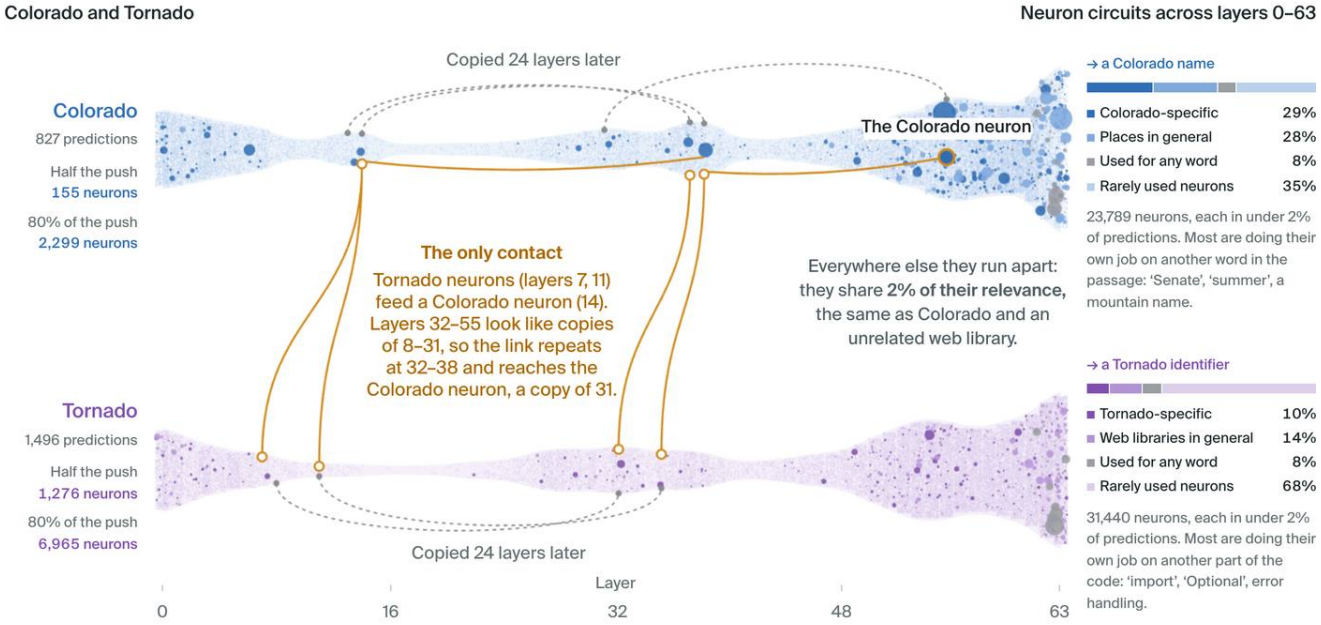


Figure 23: Colorado-Tornado map

We do not know why the Tornado association exists. We suspect Qwen3-32B was initialized by copying layers of Qwen3-14B, since many neurons align semantically 24 layers apart. One hypothesis is that the association served a minor sequence-level function in the earlier copy, such as continuing the second token of a certain kind of sequence, as in the layer 11 neuron. Copied into the more semantic later layers, the association may have been reinforced by appearing in both places rather than trained away. Without Qwen's training data or checkpoints we cannot observe it forming; future studies will focus on models where both are available.

We consider this a benign form of overlap. The two modes are easy to identify and separate: clustering the neuron's firings by their upstream inputs separates Colorado from Tornado perfectly, and the two computations meet only in a narrow pathway around layer 14 and its copy. Tornado also has its own representation among code neurons, so an analysis starting there would reach the overlap quickly. Full separation would be cleaner, but the overlap is narrow and traceable. Studying the original network also has an advantage here: a replacement basis that split Colorado from Tornado would lose the association that predicts the model's error. Because the modes are easily separable, analysis can move between the two views as needed.

Methods: Subcircuit and Mode Methods

Finding Neuron Subcircuits

To identify subcircuits, we group different firings of the same neuron according to which upstream inputs produce them, then use downstream effects to help decide which groups belong together. We collected firing examples by running 10 billion tokens from diverse domains through Qwen 3.6 35B-A3B, including data from pretraining, reasoning, chats with users, and math. For each of the model's 5 million neurons, we collected 1,000 examples from each of 14 activation percentile bands, for a total of 14,000 examples per neuron. We then collected an additional 30,000 examples per neuron for a stratified subset of 2,734 neurons. These additional examples were drawn from above the 95th activation percentile (p95) for positive poles and below the 5th percentile (p5) for negative poles. As a starting point for neuron descriptions, we used an internal version of Qwen3-8B that we fine-tuned to produce basic labels for each neuron based on example sentences the neuron fires on.

Setup. Consider a target neuron n in routed expert e of layer L. At token position t, with residual stream h_t, its activation is

$$a_t = \mathrm{SiLU}(z_t^g) z_t^u, z_t^\ell = w_\ell^\top \mathrm{RMSNorm}(h_t), \ell \in \{g, u\},$$

where z^g and z^u are the gate and up pre-activations. We call each projection ℓ a lane. A firing f is one example (x,t) from the neuron's examples; we write F_n for the set of firings of one pole of n.

Input attributions. To identify subcircuits, the first step is to characterize each firing by the upstream inputs that produce it. For each firing, we use first-order Taylor attribution to find the input neurons most responsible for the target neuron's activation (see Binder et al., 2016 for related Taylor-based attribution methods). We linearize the RMSNorm at its actual scale, $v_t = w_t \odot (1+\gamma_L)/\mathrm{rms}(h_t)$. Each pre-activation then decomposes exactly into contributions from upstream components j, whose writes to the residual stream are d_j:

$$z_f^\ell = \sum_j c_j^\ell(f), c_j^\ell(f) = v_\ell^\top d_j(f).$$

Many inputs contribute strongly to many neurons in the expert, rather than specifically to the target neuron, so we measure each input against the expert's average neuron. Let $\mu_j^\ell(f)$ be the mean of $|c_{j\to i}^\ell(f)|$ over the 512 neurons i of expert e. For each lane we keep the 400 inputs with the largest excess $|c_j^\ell(f)| - \mu_j^\ell(f)$.

Four-part profile. Let $s_\ell(f) = \mathrm{sign}(z_f^\ell)$ be the sign of lane ℓ's pre-activation at the firing. We split each kept input's contribution into the part that supports this sign and the part that opposes it, each net of the expert baseline:

$$\delta_j^\ell(f) = [s_\ell(f) c_j^\ell(f) - \mu_j^\ell(f)]_+, \omega_j^\ell(f) = [-s_\ell(f) c_j^\ell(f) - \mu_j^\ell(f)]_+,$$

where $[x]_+ = \max(x, 0)$. This gives each firing a profile with four parts: gate drivers δ^g, gate suppressors ω^g, up drivers δ^u, and up suppressors ω^u. Keeping these separate lets us distinguish firings where the same input plays different roles.

Readers. To collect downstream readers, we take the target neuron's write at the firing, $a_f \rho_f W_{\mathrm{down}}[:, n]$, where ρ_f is the router weight. We compute its first-order effect $r_k(f)$ on the activation of every neuron k in later layers at the same position, again through each reader layer's linearized RMSNorm. We keep the 1,000 readers with the largest $|r_k(f)|$; for the last layer, the readers are output logits.

Weighting. We next compare these profiles across firings of the same neuron to identify upstream subcircuits that recur in different contexts. Inputs that appear in nearly every firing provide little information about what distinguishes one firing from another. We therefore weight each block $B \in \{\delta^g, \omega^g, \delta^u, \omega^u\}$ with TF-IDF over the neuron's firings and then L2-normalize it:

$$\tilde{B}_j(f) = \frac{B_j(f)\,\mathrm{idf}_j}{\|B(f) \odot \mathrm{idf}\,\|_2}, \mathrm{idf}_j = \log\frac{1+|\mathcal{F}_n|}{1+df_j} + 1,$$

where df_j is the number of firings in which input j appears. The four normalized blocks are concatenated so that each contributes equally:

$$\varphi(f) = \frac{1}{2}[\tilde{\delta}^g(f)\|\tilde{\omega}^g(f)\|\tilde{\delta}^u(f)\|\tilde{\omega}^u(f)].$$

The factor 1/2 normalizes $\varphi(f)$ to unit length when all four blocks are non-empty.

Community detection. Since we do not know in advance how many subcircuits a neuron has, we use community detection to find groups of firings with similar input profiles. We split firings by document: 75% of documents form the fitting set $\mathcal{F}n\mathrm{fit}$, and the remaining 25% are held out for evaluation. We build a graph on the fitting set by connecting each firing to its k=15 nearest neighbors $N_{15}(f)$ by cosine similarity, with edge weights

$$A_{ff'} = \cos(\varphi(f), \varphi(f')), f' \in \mathcal{N}_{15}(f).$$

The graph is made undirected by summing reciprocal edges. We then run the Leiden algorithm (Traag et al., 2019) to find the partition σ that maximizes the Reichardt-Bornholdt configuration objective (Reichardt & Bornholdt, 2006) at resolution γ=1:

$$Q(\sigma) = \sum_{f,f'}(A_{ff'} - \gamma\frac{k_f k_{f'}}{2m})\mathbf{1}[\sigma(f) = \sigma(f')].$$

Here k_f is the weighted degree of f and m is the total edge weight. Communities with fewer than 15 fitting firings are left unassigned. Each held-out firing is assigned to the community whose centroid is nearest in cosine similarity.

Merging. This procedure can split a single circuit into several groups. To bring these back together, we describe each group C by two sets of 50 neurons:

- Its top 50 input neurons I_C, by mean TF-IDF driver weight $\tilde{\delta}^g$ and $\tilde{\delta}^u$ over its firings.
- Its top 50 readers R_C, by mean TF-IDF reader weight.

We link two groups when both overlaps reach a threshold τ:

$$\frac{|I_C \cap I_{C'}|}{50} \geq \tau \quad \text{and} \quad \frac{|R_C \cap R_{C'}|}{50} \geq \tau, \tau = 0.3.$$

The final subcircuits are the connected components of this link graph. Requiring agreement on both inputs and readers helps ensure that merged groups reflect the same underlying circuit.

This yielded a median of 19 subcircuits per pole (with a range between 1 and 51), with low overlap in their inputs and readers. We send each subcircuit, with a description of its inputs and readers, to Claude Opus 5.5 to produce a concise description of the subcircuit.

Modes

Claude Opus 5.5 produces three descriptions of each subcircuit:

- A **semantic reading** of the content the neuron tracks, based only on the text of six firings.
- A **token-pattern description**, based on the same six firings.
- A **functional description**, based on short labels for the most selective inputs and readers, in the form “combines X and Y to signal Z, used by readers that do W.”

The semantic reading receives no circuit evidence, allowing us to compare it with that evidence later. Readings must identify specific content rather than token form. Descriptions broad enough to fit a random neuron are rewritten.

We group subcircuits into modes in two Opus passes. The first groups subcircuits with the same meaning and classifies the pole as monosemantic if one specific underlying meaning accounts for all identifiable meanings, or polysemantic otherwise.

The second pass reviews the groups using longer contexts. Opus receives each mode's label, subcircuit readings, KL damage, overlap with other modes, and up to three full-length exemplars. It first looks for document-level context that could connect the modes, then judges whether each pair reflects one mechanism or two. We applied this review to all 2,673 multi-mode poles in the random sample and used the resulting groups as our reported modes.

We measure the effective number of modes using the inverse Simpson index:

$$N_{\text{eff}} = \frac{1}{\sum_i p_i^2}.$$

Reproducibility and simulation

We also checked whether the groupings were stable across repeated runs and different fitting documents. We repeated the Opus grouping three times on the first 219 poles. The runs agreed on whether a pole had one mode or several 96% of the time, with a median adjusted Rand index of 0.95. Clustering two halves of the fitting documents separately recovered 68% of subcircuits, compared with 1% under a shuffled null.

Methods: Evaluation Methods

We evaluate neuron descriptions at three levels: (i) *detecting* text that makes a pole fire, (ii) *classifying* the mode or subcircuit responsible for a firing, and (iii) *simulating* token-level activations well enough to reproduce the neuron's effect on the model.

Data splits and exclusions

We use the document splits and activation bands described in the preceding section. Descriptions, subcircuit centroids, and retrieved examples use only the discovery and validation folds and we evaluate on a separate held-out test set.

To check for repeated contexts across folds, we compare the 16-token sequence ending at each firing and the 13-token sequence beginning at it. We count two contexts as repeated if either sequence matches exactly. Approximately 2.6% of firings repeat a context. Removing test-set firings whose context also appears in a fitting fold changes the headline results by at most 0.002.

Each pole also has a firing threshold, θ, which defines the activation unit used in simulation. For the pole included in the census, θ is the census threshold. For the opposite pole, θ is 0.65 times that pole's own p99.9.

Detection setup

The *detection* task tests whether the mode descriptions help a judge distinguish text containing a held-out firing from a control sequence in which the neuron stays below q90.

Each firing sequence contains the sampled firing token, the 60 preceding tokens, and the eight following tokens. We construct two types of control sequences with the same window: same-expert controls use a token routed to the neuron's expert, while random controls use a randomly selected corpus position. The

sampled token is marked in both the firing sequence and its control, so the judge knows which token to assess. We randomize the order of the firing sequence and its control.

For this evaluation, we use the mode descriptions produced before the long-context review, excluding residue and gate-closed groups from the list shown to the judge. Each mode is presented with its label and description only; the judge sees no example firings.

The judge evaluates pairs of sequences in batches of 10, selecting the sequence containing the firing and reporting its confidence in each choice. We calculate accuracy separately for each pole within each activation band, then report the mean across poles.

Description variants and context limits

To understand what limits detection performance, we test whether more detailed descriptions or combinations of text methods help, and examine whether the short sequences preserve the context that makes the neuron fire.

Adding more detail to the descriptions made little difference on a 40-pole development benchmark. The mode list alone achieved a detection AUROC of 0.909. Including each subcircuit's label, semantic reading, and token pattern gave 0.914; adding examples from the fitting folds gave 0.911.

On 40 fresh poles, an ensemble of text methods reached a detection AUROC of 0.938. The "ensemble" combined Opus predictions from descriptions, Opus predictions from 10 retrieved fitting examples, refined descriptions, and an embedding classifier. It was evaluated against same-expert controls, with an AUROC of 0.838 at q95.

Alternatively, we found that shorter sequences can omit context that determines whether the neuron fires. When we rerun the underlying model on the sequence alone, approximately 15% of exemplars stop firing, while some controls begin firing. Providing 250 tokens of actual preceding context improves detection against same-expert controls:

The short sequence alone gives an AUROC of 0.927; adding 250 preceding tokens gives 0.978.

With the additional context, AUROC reaches 0.943 at q95. These results measure the underlying model's behavior when rerun with different amounts of context; the text judge receives only the short sequence.

Reader-based classification

The *classification* task tests whether we can identify which mode was active by looking at how a firing affects downstream neurons.

For each firing, we retain the 1,000 downstream readers with the largest absolute attribution values. We represent the firing as a TF-IDF vector over these readers and normalize it to unit length.

We compute each subcircuit's centroid from its fitting-fold firings. To classify a held-out firing, we calculate its cosine similarity to every centroid and apply a softmax with temperature 0.05. Each mode receives the highest score among its subcircuits, and we predict the mode with the highest score.

Readers are excluded from the initial clustering features but are used when merging clusters, using only fitting-fold firings. The evaluation therefore tests whether groups initially defined by their inputs also have distinct downstream effects, while retaining some dependence on readers during fitting.

The main classification evaluation includes only firings assigned to real modes. We exclude layer 39, which has no downstream routed readers, and test-set firings whose context also appears in a fitting fold.

We calculate two metrics separately for each activation band:

- **Accuracy:** the median fraction of correct predictions across poles. For mode classification, the baseline always predicts the pole's most frequent mode. For subcircuit classification, chance accuracy is the sum of squared subcircuit proportions.
- **Separation:** each mode's one-versus-rest AUROC, summarized as the median across pole-mode pairs. This measures whether the mode scores its own firings above other modes' firings, even when it does not receive the highest score overall. Chance AUROC is 0.50.

Additional classification tests

On the 40-pole benchmarks, Opus receives the mode descriptions, eight retrieved fitting examples, and a test sequence. At q95, marking the firing token brings its accuracy close to the reader classifier's, although readers achieve better separation between modes.

With the firing token marked, Opus achieves 0.825 accuracy and 0.944 separation AUROC at q95, compared with 0.824 and 0.973 for the reader-based classifier. Without marking the firing token, Opus reaches 0.74 accuracy.

We also test a stricter setting in which residue and gate-closed groups compete as possible answers. Assigning a real-mode firing to either group counts as an error. For text classification in this setting, Jev receives subcircuit readings and two examples per subcircuit, with the firing token unmarked in the test sequence.

Reader-based accuracy ranges from 0.78 at q95 to 0.96 in the extreme band for modes, and from 0.66 to 0.86 for subcircuits. Jev's mode accuracy ranges from 0.40 to 0.83, compared with chance baselines of 0.36 and 0.60.

Under this stricter evaluation, both reader-based and text classification improve for stronger firings.

Token-level simulation

Simulation is a much harder test of our understanding of a neuron than detection or classification, requiring us to predict the neuron's activation at every token accurately enough to reproduce its effect on the model. To do this, we get Claude Sonnet 5.5 to convert each description into a fixed JSON format describing what makes the pole fire, its token pattern, and its activation levels. Jev (jev-1.14) uses this description to assign each token a probability distribution over 10 activation levels, with level 0 representing silence.

We express activations relative to a threshold, θ, using the census threshold for the pole included in the census and 0.65 times the opposite pole's own p99.9 for that pole. To convert the predicted levels into activations on this scale, we fit a level table for each pole on windows from the description side of the split. Each token's level is its expected level under Jev's distribution, rounded. Each level's value is the mean true activation of the description-side tokens assigned to it, made non-decreasing across levels with an isotonic fit. The table never sees the scored windows. Every token routed to the neuron's expert counts, including tokens where the gate is closed.

We score 486 poles with up to 20 held-out windows per band per pole. The focal token of each window is a firing drawn from a router-weighted band (activation × routing weight). The top band is instead the pole's top 1,000 firings by activation. Each window contains the focal token, 45 preceding tokens, and 12 following tokens, preceded by another 768 tokens of actual context. During these runs, we keep routing fixed to the clean model's routing and replace the neuron's activation at all tokens with either the predicted value for simulation or zero for ablation. The damage a window shows is the KL divergence of the ablated model from the clean model. Simulation restores the share of that damage it removes. We compare this with a ceiling that gives every token the table value nearest its true activation, the best any reading of the same 10 levels could do. Simulation restores 72.7% of the ceiling.

Recovery score

For each window, we sum the divergence between the clean and patched next-token distributions, then compare the weighted totals for simulation and ablation:

$$K_m(w) = \sum_{t \in w} D_{KL}\left(p^{clean}_{w,t} \| p^{m}_{w,t}\right)$$

$$R = 1 - \frac{\sum_w \alpha_w K_{sim}(w)}{\sum_w \alpha_w K_{abl}(w)}$$

Here, m denotes simulation or ablation, $p_{w,t}$ is the next-token distribution at position t in window w, and α_w is that window's sampling weight. A recovery score of 1 means simulation reproduces the clean model's predictions; 0 means it performs as well as removing the neuron, and a negative score means it performs worse.

We report both an unweighted average across poles and an average weighted by each pole's ablation damage. The former describes performance on the average pole; the latter measures the fraction of aggregate ablation damage recovered.

Evaluation windows

All evaluation windows are held out from the description writer. We use three sampling schemes:

- **Activation bands:** 20 windows per band-S95, S99, S99.9, S99.99, and Top-centered on sampled firings. Top contains the pole's 1,000 strongest firings. We either weight the bands equally or weight them by the proportion of routed tokens in their activation intervals.
- **Threshold-relative bands:** 20 windows per band defined in units of θ, again using either equal weights or weights based on band frequency.
- **Random text:** 40 randomly selected corpus windows, used to measure the damage caused when the simulator predicts firings that should not occur.

Open Questions & Future Work

Memorization

Text memorization seems to occur through something like this: A non-memorized, generated subsequence S has a typical circuit C that codes for the next token T_s through a combination of neuron connections and attention. If S is a part of a memorized text M, there is a certain probability that neurons coding for the next token T_m are among the potential downstream recipients of the part of the circuit that recognizes S. However, in order for them to activate sufficiently to produce T_m requires that specific contextual clues (such as the beginning of the passage) accumulate in attention and be used to boost the probability of the circuit for T_m over T_s. This is our current hypothesis, derived from observations and ablations, but we will deliver a much more complete account soon.

Composition (and interactions with manifolds)

Our transcoder and clustering experiments already suggest that neurons have frequent co-firing partners and that together they code for something like the semantic intersection of the individual parts. We intend to explore to what degree this occurs and what interactions it has with residual stream manifolds. However, in principle this would allow for the encoding of a vast array of continuous interpolations and combinations without requiring diffuse features spread geometrically over many neurons.

Neuron evolution over training

Neurons often display odd or disconnected modes. How were these acquired over the training process? We often observe secondary modes related to math and coding; might these be acquired through post-training, and if so, what conditions allow them to develop this way?

Detection

Can we develop better detectors and activation monitors in the neuron- or neuron-based circuit basis? Probes allow us to detect representations in the residual stream, but we may sometimes wish to detect when certain mechanisms are carried out inside the model.

What else the network is doing with weak activations

We've seen that weak activations are generally far less causally relevant to model operation than the stronger, more interpretable activations. However, there is a band of neuron operation that cannot be ablated but where the neuron's function is less or only occasionally apparent. Initial experiments suggest that this is a transition zone where the model still provides a certain probability to the feature, but we intend to more fully characterize this in future work.

Attention Mediation (Elhage et al., 2021; Gurnee et al., 2024; Xiao et al., 2023, Doshi et al., 2026)

A certain percentage of neurons were initially quite difficult to interpret until we discovered that their primary downstream readers (and in some cases, inputs) were attention layers. We intend to characterize how these function and to what degree it exists as a phenomenon.

N-Gram Detection (Voita et al., 2024; Chang & Bergen, 2025; Gurnee et al., 2023)

A prominent question in the literature has been: if neurons were monosemantic, how could they detect bigrams and other n-grams, the potential number of which far exceeds neuron counts in many models? Models may use a number of different strategies for this, but we've found that in at least some cases the semantic intersection/composition of neurons suffices to do this. We intend to dive deeper into this question in future work.

References

1. Le et al. (2012). Building high-level features using large scale unsupervised learning.
2. Zeiler & Fergus (2014). Visualizing and Understanding Convolutional Networks.
3. Bau et al. (2017). Network Dissection: Quantifying Interpretability of Deep Visual Representations.
4. Olah et al. (2017). Feature Visualization.
5. Radford et al. (2017). Learning to Generate Reviews and Discovering Sentiment.
6. Goh et al. (2021). Multimodal Neurons in Artificial Neural Networks.
7. Nguyen et al. (2016). Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks.
8. Olah et al. (2020). Zoom In: An Introduction to Circuits.
9. Elhage et al. (2022). Toy Models of Superposition.
10. Arora et al. (2018). Linear Algebraic Structure of Word Senses, with Applications to Polysemy.
11. Park et al. (2023). The Linear Representation Hypothesis and the Geometry of Large Language Models.
12. Bricken et al. (2023). Towards Monosemanticity: Decomposing Language Models With Dictionary Learning.
13. Cunningham et al. (2023). Sparse Autoencoders Find Highly Interpretable Features in Language Models.
14. Geva et al. (2021). Transformer Feed-Forward Layers Are Key-Value Memories.
15. Dai et al. (2022). Knowledge Neurons in Pretrained Transformers.
16. Geva et al. (2022). Transformer Feed-Forward Layers Build Predictions by Promoting Concepts in the Vocabulary Space.
17. Gurnee et al. (2024). Universal Neurons in GPT2 Language Models.
18. Arora & Wu (2025). Language Model Circuits Are Sparse in the Neuron Basis.
19. Dravid et al. (2026). Neuron Populations Exhibit Divergent Selectivity with Scale.
20. Feucht et al. (2026). Arithmetic in the Wild: Llama uses Base-10 Addition to Reason About Cyclic Concepts.
21. Shabalin & Piotrowski (2026). Comparing Sparsified MLPs and Transcoders: Many Transcoder Features Are Neurons in Disguise.
22. Hubel & Wiesel (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.
23. Barlow (1972). Single Units and Sensation: A Neuron Doctrine for Perceptual Psychology?
24. Quiroga et al. (2005). Invariant visual representation by single neurons in the human brain.
25. Saxena & Cunningham (2019). Towards the neural population doctrine.
26. Ebitz & Hayden (2021). The population doctrine in cognitive neuroscience.
27. Rigotti et al. (2013). The importance of mixed selectivity in complex cognitive tasks.
28. Shazeer (2020). GLU Variants Improve Transformer.
29. Dauphin et al. (2017). Language Modeling with Gated Convolutional Networks.
30. Ramachandran et al. (2017). Searching for Activation Functions.

31. Elfwing et al. (2018). Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning.
32. Hendrycks & Gimpel (2016). Gaussian Error Linear Units (GELUs).
33. Touvron et al. (2023). LLaMA: Open and Efficient Foundation Language Models.
34. Yang et al. (2025). Qwen3 Technical Report.
35. Pearce et al. (2024). Bilinear MLPs enable weight-based mechanistic interpretability.
36. Sharkey (2023). A technical note on bilinear layers for interpretability.
37. Jayakumar et al. (2020). Multiplicative Interactions and Where to Find Them.
38. Veness et al. (2019). Gated Linear Networks.
39. Fishman et al. (2024). Scaling FP8 training to trillion-token LLMs.
40. Szegedy et al. (2014). Intriguing properties of neural networks.
41. Morcos et al. (2018). On the importance of single directions for generalization.
42. Zhou et al. (2018). Revisiting the Importance of Individual Units in CNNs via Ablation.
43. Leavitt & Morcos (2020). Selectivity considered harmful: evaluating the causal impact of class selectivity in DNNs.
44. Ghorbani & Zou (2020). Neuron Shapley: Discovering the Responsible Neurons.
45. Dhamdhere et al. (2018). How Important Is a Neuron?
46. Li et al. (2023a). The Lazy Neuron Phenomenon: On Emergence of Activation Sparsity in Transformers.
47. Liu et al. (2023a). Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time.
48. Mirzadeh et al. (2023). ReLU Strikes Back: Exploiting Activation Sparsity in Large Language Models.
49. Zhang et al. (2022). MoEfication: Transformer Feed-forward Layers are Mixtures of Experts.
50. Liu et al. (2024). Training-Free Activation Sparsity in Large Language Models.
51. Lee et al. (2024). CATS: Contextually-Aware Thresholding for Sparsity in Large Language Models.
52. Jaszczur et al. (2021). Sparse is Enough in Scaling Transformers.
53. Ahmad & Scheinkman (2019). How Can We Be So Dense? The Benefits of Using Highly Sparse Representations.
54. Makhzani & Frey (2013). k-Sparse Autoencoders.
55. Song et al. (2024). ProSparse: Introducing and Enhancing Intrinsic Activation Sparsity within Large Language Models.
56. Zhang et al. (2024). ReLU² Wins: Discovering Efficient Activation Functions for Sparse LLMs.
57. Wang et al. (2024). Q-Sparse: All Large Language Models can be Fully Sparsely-Activated.
58. Song et al. (2023). PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU.
59. Alizadeh et al. (2023). LLM in a flash: Efficient Large Language Model Inference with Limited Memory.
60. Zhang et al. (2023). Emergent Modularity in Pre-trained Transformers.
61. Sun et al. (2024). Massive Activations in Large Language Models.
62. Dettmers et al. (2022). LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale.
63. Kovaleva et al. (2021). BERT Busters: Outlier Dimensions that Disrupt Transformers.
64. Puccetti et al. (2022). Outliers Dimensions that Disrupt Transformers Are Driven by Frequency.

65. Bondarenko et al. (2023). Quantizable Transformers: Removing Outliers by Helping Attention Heads Do Nothing.
66. Olshausen & Field (2004). Sparse coding of sensory inputs.
67. Bills et al. (2023). Language models can explain neurons in language models.
68. Murphy (2002). The Big Book of Concepts.
69. Rosch et al. (1976). Basic objects in natural categories.
70. Murphy & Medin (1985). The role of theories in conceptual coherence.
71. Sloutsky & Deng (2019). Categories, concepts, and conceptual development.
72. Barsalou (1983). Ad hoc categories.
73. Klindt et al. (2026). A unifying framework from neural superposition to sparse interpretable codes.
74. Vig et al. (2020). Causal Mediation Analysis for Interpreting Neural NLP: The Case of Gender Bias.
75. Geiger et al. (2025). Causal Abstraction: A Theoretical Foundation for Mechanistic Interpretability.
76. Mueller et al. (2025). The Quest for the Right Mediator: Surveying Mechanistic Interpretability Through the Lens of Causal Mediation Analysis.
77. Fusi et al. (2016). Why neurons mix: high dimensionality for higher cognition.
78. Marr (1982). Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.
79. Lee & Messerschmitt (1988). Multiplexing.
80. Mante et al. (2013). Context-dependent computation by recurrent dynamics in prefrontal cortex.
81. Salinas & Thier (2000). Gain modulation: a major computational principle of the central nervous system.
82. Dreyer et al. (2024). PURE: Turning Polysemantic Neurons Into Pure Features by Identifying Relevant Circuits.
83. Gandelsman et al. (2024). Interpreting the Second-Order Effects of Neurons in CLIP.
84. Binder et al. (2016). Layer-wise Relevance Propagation for Neural Networks with Local Renormalization Layers.
85. Traag et al. (2019). From Louvain to Leiden: guaranteeing well-connected communities.
86. Zheng et al. (2023). Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.
87. Paulo et al. (2024). Automatically Interpreting Millions of Features in Large Language Models.
88. Hill (1973). Diversity and Evenness: A Unifying Notation and Its Consequences.
89. Jost (2006). Entropy and diversity.
90. Simpson (1949). Measurement of Diversity.
91. Elhage et al. (2023). Privileged Bases in the Transformer Residual Stream.
92. Heap et al. (2025). Automated Interpretability Metrics Do Not Distinguish Trained and Random Transformers.
93. Hewitt & Liang (2019). Designing and Interpreting Probes with Control Tasks.
94. Makelov et al. (2023). Is This the Subspace You Are Looking for? An Interpretability Illusion for Subspace Activation Patching.
95. Zimmermann et al. (2023). Scale Alone Does not Improve Mechanistic Interpretability in Vision Models.
96. Korznikov et al. (2026). Sanity Checks for Sparse Autoencoders: Do SAEs Beat Random Baselines?
97. Shaham et al. (2024). A Multimodal Automated Interpretability Agent.

98. Schwettmann et al. (2023a). FIND: A Function Description Benchmark for Evaluating Interpretability Methods.
99. Lu et al. (2024). The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery.
100. Gottweis et al. (2025). Accelerating scientific discovery with Co-Scientist.
101. Singh et al. (2024). Rethinking Interpretability in the Era of Large Language Models.
102. Zimmermann et al. (2024). Measuring Per-Unit Interpretability at Scale Without Humans.
103. Colin et al. (2024). Local vs distributed representations: What is the right basis for interpretability?
104. Marin-Llobet, A., & Ferrando, J. (2026). Automated Interpretability and Feature Discovery in Language Models with Agents. arXiv:2605.01555v1. doi:10.48550/arXiv.2605.01555.
105. Erhan et al. (2009). Visualizing Higher-Layer Features of a Deep Network.
106. Nguyen et al. (2016b). Synthesizing the preferred inputs for neurons in neural networks via deep generator networks.
107. Fel et al. (2023c). Unlocking Feature Visualization for Deeper Networks with MAgnitude Constrained Optimization.
108. Poerner et al. (2018). Interpretable Textual Neuron Representations for NLP.
109. Thompson et al. (2024). Fluent dreaming for language models.
110. Simonyan et al. (2013). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.
111. Yosinski et al. (2015). Understanding Neural Networks Through Deep Visualization.
112. Geirhos et al. (2023). Don't trust your eyes: on the (un)reliability of feature visualizations.
113. Ghiasi et al. (2022). Plug-In Inversion: Model-Agnostic Inversion for Vision with Data Augmentations.
114. Hempel & Oppenheim (1948). Studies in the Logic of Explanation.
115. Jonas & Kording (2017). Could a Neuroscientist Understand a Microprocessor?
116. Lillicrap & Kording (2019). What does it mean to understand a neural network?
117. Lipton (2016). The Mythos of Model Interpretability.
118. Doshi-Velez & Kim (2017). Towards A Rigorous Science of Interpretable Machine Learning.
119. Ribeiro et al. (2018). Anchors: High-Precision Model-Agnostic Explanations.
120. Hase & Bansal (2020). Evaluating Explainable AI: Which Algorithmic Explanations Help Users Predict Model Behavior?
121. Hase et al. (2020). Leakage-Adjusted Simulatability: Can Models Generate Non-Trivial Explanations of Their Behavior in Natural Language?
122. Colin, Fel et al. (2022). What I Cannot Predict, I Do Not Understand: A Human-Centered Evaluation Framework for Explainability Methods.
123. Chen et al. (2023). Do Models Explain Themselves? Counterfactual Simulatability of Natural Language Explanations.
124. Gao et al. (2025). Weight-sparse transformers have interpretable circuits.
125. Heimersheim & Nanda (2024). How to use and interpret activation patching.
126. Zhang & Nanda (2024). Towards Best Practices of Activation Patching in Language Models: Metrics and Methods.
127. Meng et al. (2022a). Locating and Editing Factual Associations in GPT.

128. Mueller et al. (2025). MIB: A Mechanistic Interpretability Benchmark.
129. Pearl (2001). Direct and Indirect Effects.
130. Dunefsky et al. (2024). Transcoders Find Interpretable LLM Feature Circuits.
131. Paulo & Belrose (2025). Sparse Autoencoders Trained on the Same Data Learn Different Features.
132. Leask et al. (2025). Sparse Autoencoders Do Not Find Canonical Units of Analysis.
133. Fel et al. (2025). Archetypal SAE: Adaptive and Stable Dictionary Learning for Concept Extraction in Large Vision Models.
134. Chanin et al. (2024). A is for Absorption: Studying Feature Splitting and Absorption in Sparse Autoencoders.
135. Kantamneni et al. (2025). Are Sparse Autoencoders Useful? A Case Study in Sparse Probing.
136. Wu et al. (2025). AxBench: Steering LLMs? Even Simple Baselines Outperform Sparse Autoencoders.
137. Pati et al. (1993). Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition.
138. Mallat & Zhang (1993). Matching pursuits with time-frequency dictionaries.
139. Olshausen & Field (1997). Sparse coding with an overcomplete basis set: A strategy employed by V1?
140. Costa et al. (2025). From Flat to Hierarchical: Extracting Sparse Representations with Matching Pursuit.
141. O'Neill et al. (2025). Compute Optimal Inference and Provable Amortisation Gap in Sparse Autoencoders.
142. Olah (2023). Distributed Representations: Composition & Superposition.
143. Mu & Andreas (2020). Compositional Explanations of Neurons.
144. Smolensky (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems.
145. Braun et al. (2025). Interpretability in Parameter Space: Minimizing Mechanistic Description Length with Attribution-based Parameter Decomposition.
146. Bushnaq et al. (2025). Stochastic Parameter Decomposition.
147. Braun et al. (2024). Identifying Functionally Important Features with End-to-End Sparse Dictionary Learning.
148. Brangus, 2026.
149. Jafari et al. (2025). RelP: Faithful and Efficient Circuit Discovery in Language Models via Relevance Patching.
150. Elhage et al. (2021). A Mathematical Framework for Transformer Circuits.
151. Xiao et al. (2023). Efficient Streaming Language Models with Attention Sinks.
152. Doshi et al. (2026). Bi-Orthogonal Factor Decomposition for Vision Transformers.
153. Voita et al. (2024). Neurons in Large Language Models: Dead, N-gram, Positional.
154. Chang & Bergen (2025). Bigram Subnetworks: Mapping to Next Tokens in Transformer Language Models.
155. Gurnee et al. (2023). Finding Neurons in a Haystack: Case Studies with Sparse Probing.
156. Reichardt & Bornholdt (2006). Statistical Mechanics of Community Detection.

Citation

Download BibTeX

Tigges, C., Kowal, M., McGrath, T., Lewis, O., and Cammarata, N., 2026. Neurons as an Interpretable Basis. Goodfire.

BibTeX:

```
@article{tigges2026neurons,

author = {Tigges, Curt and Kowal, Matthew and McGrath, Thomas and Lewis, Owen and Cammarata, Nick},

title = {Neurons as an Interpretable Basis},

publisher = {Goodfire},

year = {2026},

url = {https://www.goodfire.com/research/neurons-as-an-interpretable-basis},

month = {nov},

day = {10}

}
```

Footnotes

1. **How many causally important modes show up in the population?** Because we sample the same number of firings from each activation band, rarer high-activation firings are over-represented. To recover population frequency, we weight each firing by w = (the band's share of the population) / (the number of firings kept from that band), and replace KL and firing counts with their w-weighted versions before computing shares. This gives us a mean of 1.56 and a median of 1.04.