Cross-Layer Transcoders Are Incentivized to Learn Unfaithful Circuits

Georg Lange [*1] **Rick Goldstein** [*1] **Kamal Maher** [1] **Katherine Dearstyne** [2]

Abstract

Cross-layer transcoders (CLTs) enable circuit tracing that can extract high-level mechanistic explanations for arbitrary prompts and are emerging as general-purpose infrastructure for mechanistic interpretability. Because these tools operate at a relatively low level, their outputs are often treated as reliable descriptions of what a model is doing, not just predictive approximations. We therefore ask: when are CLT-derived circuits faithful to the model's true internal computation? In a boolean toy model with known ground truth, we show a specific unfaithfulness mode: CLTs can rewrite deep multi-hop circuits into sums of shallow single-hop circuits, yielding explanations that match behavior while obscuring the actual computational pathway. Moreover, we find that widely used sparsity penalties can incentivize this rewrite, pushing CLTs toward unfaithful decompositions. We then provide preliminary evidence that similar discrepancies arise in real language models, where per-layer transcoders and cross-layer transcoders sometimes imply sharply different circuit-level interpretations for the same behavior. Our results clarify a limitation of CLT-based circuit tracing and motivate care in how sparsity and interpretability objectives are chosen.

1. Introduction

In this paper, we explore whether circuits based on Cross-Layer Transcoders (CLTs) (Lindsey et al., 2024) are faithful to ground-truth model computations. We demonstrate that CLTs learn features that skip over computation that occurs inside of models. To be concrete, if features in set A create features in set B, and then these same set B features create features in set C, CLT circuits can incorrectly show that set A features create set C features and that set B features do not create set C features, ultimately skipping B features altogether.

This matters most when researchers are trying to understand *how* a model arrives at an answer, not just that it arrives at the correct answer. For instance, suppose we want to verify that a model answering a math problem actually performs intermediate calculations rather than pattern-matching to memorize answers, or detect whether a model's stated chain-of-thought reasoning actually reflects its internal computation. If CLTs systematically collapse multi-step reasoning into shallow input-output mappings, they cannot help us distinguish genuine reasoning from sophisticated lookup: precisely the distinction that matters for both reliability and existential risk. A model that merely memorizes may fail unpredictably on novel inputs; a model that reasons deceptively may actively conceal its true objectives. Interpretability tools that skip over intermediate computation cannot help us tell the difference.

We demonstrate the "feature skipping" behavior of CLTs on a toy model where ground-truth features are known, showing that the CLT loss function biases models toward this behavior. We then present evidence that this phenomenon also occurs in real language models.

1.1. Background

The goal of ambitious mechanistic interpretability is to reverse engineer an AI model's computation for arbitrary prompts and to understand the model's internal neural circuits that implement its behavior. We do this in two steps: first, we discover which features are represented at any location in the model. Second, we trace the interactions between those features to create an interpretable computational graph.

How? The primary hypothesis in mechanistic interpretability is that features are represented as linear directions in representation space (Nanda, 2023). However, there are more features than there are model dimensions, leading to superposition.

Sparse dictionary learning methods like sparse autoencoders

[*]Equal contribution [1]Independent Researcher [2]University of Notre Dame, Notre Dame, IN, USA. Correspondence to: Georg Lange <mail@georglange.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

(SAEs, Cunningham et al., 2024; Templeton et al., 2024) can disentangle dense representations into a sparse, interpretable feature basis. However, we cannot use SAEs for circuit analysis because SAE features are linear decompositions of activations, while the next MLP applies neuron-wise nonlinearities, so tracing how one SAE feature drives another requires messy, input-dependent Jacobians and quickly loses sparsity and interpretability. Transcoders (Dunefsky et al., 2024), on the other hand, are similar to SAEs but take the MLP input and directly predict the MLP output using a sparse, feature-based intermediate representation, making feature-to-feature interactions inside the MLP easier to track. When we train transcoders for every layer, we can trace circuits (Ameisen et al., 2025) by following feature interactions between these transcoders rather than inside the original network, effectively replacing each MLP with its transcoder and turning the whole system into a "*replacement model*". Crucially, the resulting circuits are technically part of the replacement model, which we simply *assume* are an accurate representation of the base model.

However, it has been observed that similar features activate across multiple layers (Lindsey et al., 2024), which makes circuits very large and complex. It is hypothesized that this is due to cross-layer superposition, where multiple subsequent layers act as one giant MLP and compute a feature in tandem. To learn those features only once, and thus reduce circuit size, Anthropic suggested cross-layer transcoders (CLTs) (Ameisen et al., 2025).

A cross-layer transcoder is a single unified model where each layer has an encoder, but features can write to all downstream layers via separate decoders: a feature encoded at layer ℓ has decoder weights for layers $\ell, \ell+1, \ldots, L$. Crucially, following Anthropic's approach, all encoders and decoders are trained jointly against a combined reconstruction and sparsity loss summed across all layers. The sparsity penalty encourages features to activate rarely, which, combined with cross-layer decoding, pushes features to span multiple layers of computation. Anthropic found this useful for collapsing redundant "amplification chains" into single features and producing cleaner attribution graphs.

Here, we show that this same property can cause **CLTs to skip over intermediate computational steps** that occur in the original model and therefore create an unfaithful representation of the model's computation. We call a circuit representation *faithful* if it captures the actual computational steps the model performs, that is, if the features it identifies as causally relevant match the intermediate computations that would be revealed by ablating the corresponding model components. This extends previous findings that show per-layer transcoders (PLTs) can be unfaithful (Olah et al., 2025).

1.2. Cross-layer Superposition vs Multi-step Circuits

The motivation for using cross-layer transcoders is to represent features that are computed in superposition across multiple subsequent layers only once. But how do we decide whether multiple subsequent layers are computing a single feature in **cross-layer superposition**, versus implementing a **multi-step circuit** that should be represented as two or more distinct features at different layers?

We draw the distinction as follows. In Anthropic's framing, cross-layer superposition occurs when several consecutive layers effectively act like one large MLP (see Figure 1). In this case, there is no meaningful intermediate interaction between layers for that feature and the layers contribute additively and reinforce the same feature direction. By contrast, we say the model is implementing a multi-step circuit when the computation relies on intermediate features that are computed in one layer and then used by a later layer, for example when *layer A* computes a feature that is then read and transformed by *layer B* into a new feature (right-most panel of Figure 1).

A simple diagnostic is whether the intermediate feature lives in a different subspace. If *layer A* writes to a direction that is largely orthogonal to the feature that *layer B* outputs, then layer A is not merely *amplifying* the same feature across layers. Instead, *layer A* is computing a distinct intermediate feature that *layer B* later transforms, which is unambiguous evidence of a multi-step circuit rather than cross-layer superposition.

Here, we show that the same mechanism that enables CLTs to capture cross-layer superposition can also cause them to **approximate a genuine multi-step circuit as a single-step circuit**, producing a circuit that differs from the model's true intermediate computations and is therefore unfaithful.

The remainder of the paper proceeds as follows. Section 2 demonstrates this unfaithfulness in a toy model with known ground truth; Section 3 explains why the CLT loss function incentivizes it and shows the same layerwise asymmetry in real language models; Section 4 gives a language-model case study where CLT- and PLT-derived circuits imply different mechanisms; and Section 5 discusses tradeoffs and mitigations.

2. CLTs Are Not Faithful in a Toy Model

In this section, we construct a toy model computing boolean functions. We manually set weights that implement a two-step algorithm which serves as a ground truth circuit. We then train PLTs and CLTs and show that CLTs learn two parallel single-hop circuits, thus being unfaithful to the ground truth.

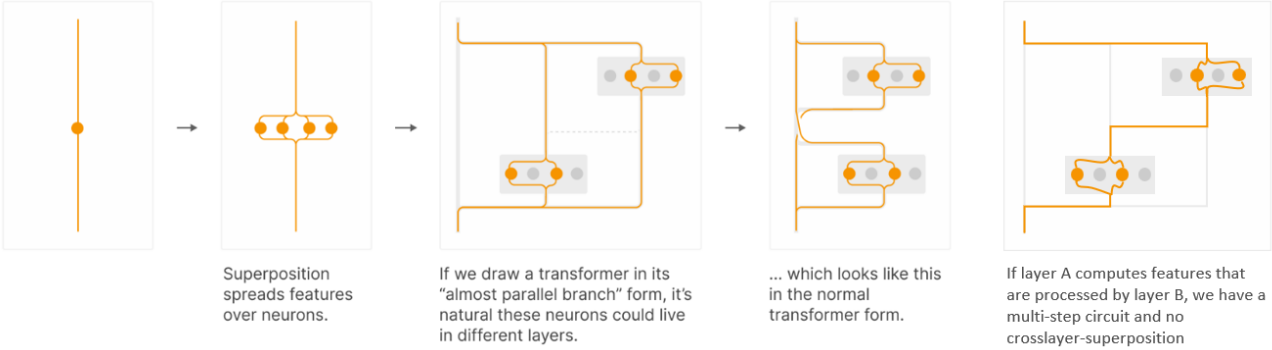


Figure 1. Distinction between cross-layer superposition and multi-step circuits (modified from Lindsey et al., 2024).

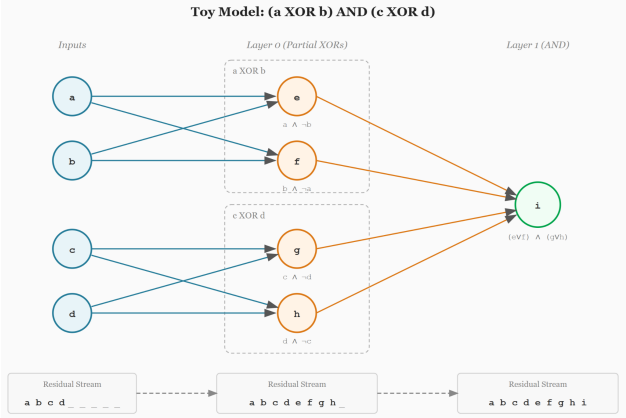


Figure 2. We construct a toy model implementing the boolean function $(a \oplus b) \wedge (c \oplus d)$ using a 2-layer residual MLP.

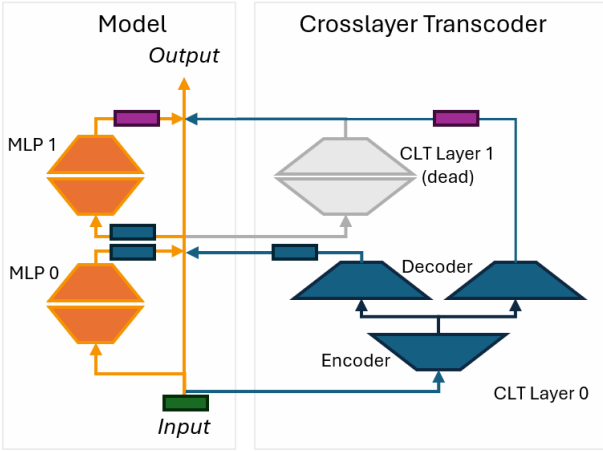


Figure 3. Left: The original ground truth residual MLP that implements a 2-step circuit **A** $\rightarrow$ **B** $\rightarrow$ **C**. Right: The cross-layer transcoder that learned to approximate the original model by learning two one-step circuits **A** $\rightarrow$ **B** (correct) and **A** $\rightarrow$ **C** (incorrect). A feature marked "(dead)" never activates for any input.

2.1. Toy Model Overview

We handcrafted a toy model to compute the boolean function $(a \oplus b) \wedge (c \oplus d)$, with four binary inputs ($+1$ or -1). The model has two MLP-only layers with residual connections.

Layer 0 has four neurons computing the partial products needed for XOR: $\mathrm{ReLU}(a - b - 1)$, $\mathrm{ReLU}(b - a - 1)$, $\mathrm{ReLU}(c - d - 1)$, and $\mathrm{ReLU}(d - c - 1)$. We call these outputs e, f, g, h. Layer 1 computes $i = \mathrm{ReLU}(e + f + g + h - 1)$, which is true if and only if both XORs are true.

Each of these nine values (four inputs, four intermediate features, one output) occupies its own linear subspace in the residual stream.

2.2. Cross-Layer Transcoder Training

We collect MLP input and output activations for all 16 possible inputs and train a JumpReLU cross-layer transcoder with 32 features per layer until convergence following Anthropic's approach (Ameisen et al., 2025). Since our model has two layers, the CLT trains two separate sets of features. The first set encodes the four inputs to the model (the `mlp_in`) and outputs two separate decoders to predict the output of neurons, also shown in Figure 3: one decoder predicts the output of the layer-0 neurons and the other predicts the output of the layer-1 neuron. For the second set of features, the encoder takes in the residual stream after the first MLP (namely the model inputs and the outputs of the first MLP) and decodes to match the output of the second MLP. Importantly, we standardize input and output activations to ensure that reconstruction errors are weighted equally across layers.

Table 1. Core metrics of JumpReLU CLT vs PLT.

Metric	JumpReLU CLT	JumpReLU PLT
Replacement Accuracy	100%	100%
L0 (layer 0)	1.81	1.68
L0 (layer 1)	0.0	0.94
MSE (layer 0)	0.0052	0.0065
MSE (layer 1)	0.0028	0.0033
Alive (layer 0)	8	7
Alive (layer 1)	0	5

2.3. The CLT Skips Intermediate Computation

We trained CLTs with varying sparsity penalties and found perfect reconstruction even with an average L0 loss of ~ 1.0 per layer. We find that with sufficient sparsity pressure and enough features, the CLT learns to not activate any layer-1 features at all but still recovers second layer output perfectly via the first layer's second decoder (see Figure 3 and Table 1). We can directly read off this behavior from the CLT's weights (see Appendix). Both PLT and CLT achieve perfect reconstruction accuracy but the CLT never activates any features in layer 1. Instead, it uses more features in layer 0 and directly constructs layer 1 output, effectively creating a single-hop circuit that directly maps the input to output.

But from how we defined our model, we know that this does not match our model's true internal algorithm. The single neuron in layer 1 is performing important computation on existing intermediate results to determine if the boolean expression is true. If we ablate this neuron, nothing writes to the output channel i, and the network cannot determine the value of the overall boolean expression. The CLT suggests that both MLPs learned features in cross-layer superposition but this is importantly *not* the case here: We can show by ablating the *interaction* between MLP 0 and MLP 1 (which in our case is trivially easy because they occupy their own linear subspace) that this breaks the model. Therefore, the CLT learned an unfaithful circuit.

This alone is all we need to argue that the CLT is not faithful. The post-mlp0 features are responsible for the post-mlp1 feature. Our CLT fails to capture this.

2.4. Other Results

Despite missing the second layer's computation, the CLT still reconstructs the full residual stream accurately. It learns 8 alive features that function as a lookup table: four features (0, 4, 13, 14) correspond to the four input combinations where the overall expression is TRUE, directly setting all intermediate and final outputs. The other four alive features (6, 7, 9, 11) each detect when one XOR is FALSE and zero out the corresponding intermediate features.

While this lookup table correctly computes all outputs, it does not reflect how the model actually performs computation step-by-step. We could implement the same boolean function with a 1-layer MLP using just these four "TRUE-case" features, but that is not the model we trained, which is why we argue the CLT is unfaithful.

3. The CLT's Loss Function Incentivizes this Bias

Our toy model demonstrates that CLTs *can* learn unfaithful representations, but one might wonder whether this actually occurs in practice on real language models. Proving that CLTs learn unfaithful circuits for frontier LLMs is challenging because we don't have ground truth circuits that we could compare against. Instead, in this section, we first make a theoretical argument on how the sparsity objective may bias the CLT towards learning shorter but unfaithful circuits and then test predictions of this hypothesis in GPT-2 and Gemma 2.

3.1. Intuition

Why are CLTs capable of learning unfaithful circuits in the first place? Deep neural computations can often be approximated by much shallower networks, by the universal approximation theorem. A CLT effectively gives us a sparse, giant shallow MLP: early-layer features can directly write to many downstream layers via their decoders, so the replacement model has an easy path to represent deep computations as shallow input-output mappings.

A natural objection is that shallow approximations typically require many more neurons, which should push the CLT toward learning the true intermediate structure instead of shortcutting it. That is true in principle, but it misses a key asymmetry in CLTs: additional decoder vectors are almost free. The CLT needs to encode features starting at layer 0, meaning there are active features that encode and decode in layer 0; **once a feature is already active, giving it additional decoder vectors to multiple later layers does not meaningfully increase the sparsity loss.** As a result, we should expect at least some amount of depth collapse: the CLT can move a significant fraction of computation into early features.

A second objection is that this "free decoder" argument only holds for TopK CLTs, since JumpReLU or ReLU CLTs also penalize decoder norms. But even there, additional decoder vectors remain close to free: because the penalty is applied to the concatenated decoder vector rather than separately per layer, spreading mass across many decoders increases the norm only sublinearly. In practice, using extra decoder vectors costs far less than activating additional features in later layers, so the shortcut representation can still dominate.

Overall, the CLT objective is indifferent to faithfulness: it rewards reconstruction accuracy and sparsity, not preserving intermediate computational steps. **When a shallow shortcut matches reconstruction while paying less sparsity cost than a layered decomposition, the CLT is incentivized to learn the shortcut.**

3.2. Evidence from Real Language Models

We next test predictions of the above theory. First, we show that CLTs shift L0 towards earlier layers compared to PLTs. Second, we show that attribution circuits on natural text are much more dependent on earlier layers.

3.3. Asymmetric L0 Across Layers

We train JumpReLU CLTs and PLTs on gpt2-small and use Anthropic's method (Ameisen et al., 2025) except where specified otherwise. We train on 100M tokens with a batch size of 1000 on the OpenWebText dataset. We collect and shuffle activations from gpt2-small and standardize both input and output activations. We train with 10k features per layer and tune L0 to be around 12–15 active features per layer.

To check that the low L0 in later layers is not simply an artifact of layer-specific activation geometry, we train Per-Layer Transcoders (PLTs) under an otherwise matched setup. Crucially, unlike approaches that train each layer independently, we sum the loss across all layers and train all layers jointly. If a non-uniform L0 profile were driven by the inherent structure of the activations, we would expect the same pattern to appear for PLTs.

Instead, we find that only CLTs show a strong asymmetry: layer 0 has much higher L0 than later layers, often more than 3× higher, with late layers especially sparse (left panel in Figure 7). This pattern is not explained by activation geometry. When we train PLTs jointly, L0 remains similar across layers (right panel). The asymmetry appears only for CLTs, matching our theoretical prediction: early-layer features do disproportionate work because they can write to all downstream layers at little additional cost.

3.4. CLT Circuits Attribute to Earlier Layers

Attribution circuits approximate a model's computation on a given prompt as a graph whose weighted edges capture how much each feature contributes to another (Ameisen et al., 2025). Here, we show that attribution circuits computed with CLTs concentrate almost all of their contribution in the first one or two layers, whereas for PLTs the contribution is spread roughly linearly across layers. For our experiments, we used Gemma2-2b as our base model, the Cross-Layer Transcoder from Hanna & Piotrowski (CLT-HP) (Hanna & Piotrowski, 2025) as the CLT, and the Gemmascope-2b

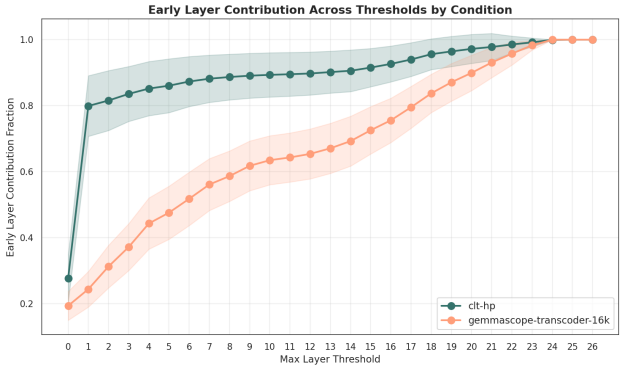


Figure 4. We compute attribution circuits on 45 prompts and compute layerwise cumulative contribution to circuits that compute the largest output logit.

Transcoder (Lieberum et al., 2024) as the PLT. We ran 60 prompts through the replacement model of both the PLT and CLT. These prompts consisted of an equal split of memorized sequences (e.g., MIT license), manually constructed prompts, and random sentences from the internet (e.g., news articles and Wikipedia). From these graphs, we computed the fraction of total contribution to the top output logit (the model's highest-probability prediction) from features in layers 0 through n. Specifically, we collected all positive-weight edges in the attribution circuit and calculated:

$$\mathrm{CumContrib}(n) = \frac{\sum_{l=0}^{n} \sum_{e \in E_l} w_e}{\sum_{l=0}^{L} \sum_{e \in E_l} w_e} \tag{1}$$

where E_l is the set of edges from features in layer l and w_e is the weight of edge e.

By repeating this calculation for increasing values of n (from 0 to the total number of layers), we generated cumulative contribution curves showing what proportion of total attribution comes from early layers at each depth. Figure 4 shows that, for the PLT, all layers contribute comparably, and the cumulative circuit contribution grows roughly linearly with depth. In contrast, the CLT exhibits a sharp jump at layer 1, with nearly all contributions concentrated in layers 0 and 1.

Together, these results are consistent with our theory: CLTs concentrate both sparsity budget and attribution mass in early layers, while PLTs distribute contribution more evenly across depth, suggesting partial depth collapse in CLT replacement circuits. This is only indirect evidence, though, since we do not have ground-truth circuits in real LLMs and cannot directly test faithfulness the way we can in the toy model.

4. Divergent Circuit Interpretations: License Memorization

Finally, we show that "CLTs in the wild" can produce wildly different high-level conclusions. Specifically, we show an "existence proof" of some abstract features that activate on diverse prompts and are steerable but don't exist in the CLT. Then, we investigate mechanisms of memorization and generalization in software licenses and show that interpreting CLTs would lead to different conclusions.

4.1. In the Wild: A Real Feature the CLT Doesn't Learn

We examined verbatim memorization of the MIT license in Gemma-2-2B, specifically the prediction of " sell" after the phrase "publish, distribute, sublicense, and/or." We want to understand whether memorized and generalized circuits interact or collaborate, or whether verbatim memorization elicits completely different circuits compared to generalization.

Using PLTs, we find two important groups of features (a.k.a. supernodes) in late layers at the " or" position: Output features that when active push logits for any form of the word sell (e.g. " sell", " sold", " selling", " sale"; Neuronpedia[1]) and features that activate on lists of present-tense verbs and push logits for present-tense verbs (e.g. " edit", " build", " install", " sell", " fill", " move", " bake"; Neuronpedia[2]). Together, they reinforce one another to push the (memorized) " sell" token to $\sim 100\%$.

The latter features' max-activating examples contain text from diverse contexts and can reliably be activated: We LLM-generate 20 prompts that contain lists of present-tense verbs and 20 controls. 11/20 prompts activated the features at the expected positions while the features fired for none of the controls. Steering those features with modest strength ($-0.7\times$) makes the model predict " selling" instead of " sell" (Figure 5).

Crucially, this works on *arbitrary* text and is not specific to MIT license memorization. We hand-write a new non-existent license text that loosely mirrors some characteristics (specifically, it also contains a list of present tense verbs). On this prompt, the same supernode is active, and steering yields similar results: It pushes the output distribution from present-tense to the participle (i.e. " publishing", " printing", " licensing"; Figure 5).

We investigated CLT circuits[3] (Figure 9) for this prompt but could not find any counterpart to this supernode. It would only be defensible for the CLT to "absorb" this intermediate feature into an upstream MIT license memorization feature if the intermediate feature were itself license specific, meaning it reliably fired only in memorization contexts. But these PLT features are clean, monosemantic, and causally steerable across diverse prompts, including both memorized and non memorized sequences. Since they implement a general present tense verb to participle transformation that applies far beyond the MIT license, collapsing them into a license memorization circuit erases a real intermediate computation that the base model appears to use and should therefore be represented as its own feature.

4.2. How CLTs Lead to the Wrong Story

What is the downstream consequence? We now build and interpret the full circuit graph for the MIT license continuation and show that the CLT and PLT lead to qualitatively different mechanistic stories about what the model is doing.

The CLT graph suggests a lookup-table-like memorization mechanism with little reuse of general language circuitry, while the PLT graph supports a different picture where general present tense verb circuitry (and others) remains intact and memorization appears as a targeted correction on top:

PLT Interpretation: Generalization with minimal memorization (see Figure 8). In the PLT, the next token prediction emerges from two interacting circuits: one promoting "sale-words" and another promoting "present-tense verbs," which together predict "sell" with approximately 100% probability. The present-tense-verb feature directly connects to the embeddings of the present-tense verbs in the list and some software/legal context features. The "sale-words" feature is activated by a larger abstract circuit that first recognizes the "and/or" construction and separately the software-licenses context. There are also direct connections from MIT-license-specific token embeddings to the "say sell-words" feature, which may represent a memorization mechanism. Our interpretation with PLTs is that the model's generalized circuits are always active, with a few lookup-table features that encode the difference between the generalized logits and the actual memorized logits.

CLT Interpretation: Pure memorization (see Figure 9). For the CLT, we also find an important "say sell-words" feature and a feature that fires at exactly all commas in this MIT verb enumeration. However, these features primarily connect directly to layer-0 and layer-1 features. These early features encode either token identity or fire unconditionally (high-frequency features). The CLT interpretation suggests the circuit is completely memorized without generalized circuits that interact and collaborate.

These are not minor differences in circuit topology; rather,

[1] https://www.neuronpedia.org/gemma-2-2b/19-gemmascope-transcoder-16k/6904

[2] https://www.neuronpedia.org/gemma-2-2b/22-gemmascope-transcoder-16k/14442

[3] Interactive graph available on Neuronpedia: [link].

MIT License

"Permission is hereby granted [...] publish, distribute, sublicense, and/or"

Original		Steered **-0.7x**	
Next Token	Probability	Next Token	Probability
sell	1.00	selling	0.69
transfer	0.00	\]	0.04
otherwise	0.00	traveling	0.03
transmit	0.00	licensing	0.03
create	0.00	advertising	0.03

Made-up License

"Anyone receiving this software [...] copy, modify, distribute, and/or"

Original		Steered **-0.7x**	
Next Token	Probability	Next Token	Probability
reproduce	0.14	publishing	0.14
publish	0.09	printing	0.11
transmit	0.09	licensed	0.08
sell	0.08	licensing	0.04
transfer	0.07	publication	0.04

Figure 5. Steering feature L22::14442 [link] and L25::9619 [link] on the MIT license and a similar but made-up license.

they represent fundamentally different theories of how the model performs this task. If the PLT interpretation is correct, the model has learned a generalizable linguistic structure that it applies to memorized content. If the CLT interpretation is correct, the model simply pattern-matches to memorized sequences. Understanding which is true matters for predicting model behavior on novel inputs, assessing robustness, and evaluating the degree to which models have learned generalizable concepts versus surface statistics.

Our theory explains this divergence. The CLT's bias toward early-layer features causes it to represent the computation as a shallow lookup from input tokens to outputs, collapsing away the intermediate abstract features that PLTs surface. The generalized circuits may not be absent from the model; they may simply be absent from the CLT's representation of the model.

5. Discussion

5.1. The Cross-layer Superposition Tradeoff

The original motivation for CLTs was handling cross-layer superposition, where the same concept appears redundantly across layers. Our findings reveal an uncomfortable tradeoff: the same mechanism that efficiently captures persistent features also enables skipping intermediate computations. A feature legitimately persisting from layer 0 to layer 10 can look identical to a shortcut that bypasses layers 1 through 9. This suggests that, under the current CLT architecture, faithfulness and efficient handling of cross-layer superposition may be fundamentally in tension.

This concern started from practice, not theory. We were excited about attribution circuits, used them heavily, and then noticed a consistent mismatch: results that looked robust with PLTs were often not replicable with CLTs, and the two tools sometimes implied very different circuit stories for the same prompts and behaviors. We also found that feature steering with CLTs was often less straightforward than we expected. Because CLTs represent features only once rather than redundantly across layers, we initially hoped steering would become easier, but we frequently observed the opposite. This does not prove unfaithfulness on its own, but it fits the broader pattern that CLT circuits can be harder to interpret and manipulate as if they were directly tracking the model's intermediate computations.

5.2. Detection and Mitigation

Given the systematic bias toward unfaithfulness, practitioners need methods to detect and potentially correct this behavior.

5.2.1. Detection approaches

The most direct signal is asymmetric L0 across layers. If early layers show dramatically higher feature activation than late layers, computation is likely being collapsed into early features. Comparing CLT and PLT L0 distributions on the same model can reveal the extent of this collapse.

Researchers should also compare CLT-derived circuits against causal interventions. If ablating an intermediate neuron disrupts model outputs but the CLT shows no intermediate features contributing, this indicates unfaithfulness. A feature set with L0 of zero for a layer containing clearly active neurons is a particularly strong signal.

More specifically, to *verify* a claim of cross-layer superposition and rule out a genuine multi-step circuit, it is not enough to ablate a neuron in isolation. One must ablate the *interaction between layers* for the putative feature: if the computation truly reflects cross-layer superposition, then breaking the handoff between layers should not destroy the behavior, since the "same feature" is effectively being maintained across layers. If, instead, the behavior depends on

intermediate information computed in one layer and transformed in a later layer, then ablating the inter-layer interaction should break the computation. This kind of targeted inter-layer ablation is the cleanest causal test distinguishing cross-layer superposition from multi-step circuits.

Finally, qualitative comparison between CLT and PLT circuits on the same computation can surface discrepancies warranting investigation, as in our memorization example.

5.2.2. Mitigation approaches

We have not yet systematically tested mitigations, but outline several directions based on our analysis:

Penalizing L0 asymmetry. One could add a regularization term discouraging extreme differences in feature activation across layers. However, this risks forcing spurious features when a layer genuinely performs minimal computation.

Separating decoder norm penalties. The most direct fix would modify the sparsity penalty to sum separate L2 norms for each layer's decoder, rather than computing the norm of the concatenated vector. This removes the geometric advantage we identified: using N decoder vectors would cost $\sim N$ rather than $\sim \sqrt{N}$, matching the cost of separate features at each layer. We have not yet trained CLTs with this modification.

Limiting feature span. Architectural constraints could restrict how many layers a single feature can decode to, for example by using a sliding window of k downstream layers. This sacrifices some ability to represent genuine cross-layer superposition but forces multi-step computation to use multiple features. It would also make CLT training far less computationally intensive: instead of learning decoders for all layer pairs, which scales like n^2 in the number of layers n, you would learn only $n \cdot k$ decoders. In large models this can be a big win, for example $n = 80$ but only allowing decoders to the next $k = 5$ layers.

Auxiliary faithfulness losses. A more principled approach might reward features whose activations correlate with actual neuron activations at each layer, rather than only rewarding output reconstruction. This directly incentivizes tracking the model's internal states.

Local cross-layer transcoders. Another pragmatic mitigation is to restrict each feature's decoder to only write to a local neighborhood of downstream layers, for example the next $k = 5$ layers, instead of allowing writes to all later layers. This does not fundamentally remove the incentive to "collapse" multi-step computation, since a feature can still span multiple layers within its window and you can still chain windows. But it can plausibly fix the common case in practice: many of the problematic absorptions we see involve long range decoding where a single feature effectively jumps over intermediate computations. Enforcing locality makes it harder for the CLT to bypass those steps, and encourages the representation to expose intermediate features rather than compressing them away. This is an 80/20 style mitigation: it may not guarantee faithfulness, but it could substantially reduce the worst failures while also lowering training cost by replacing all pairs decoding with $n \cdot k$ decoders instead of n^2.

We consider the separated decoder norm penalty the most promising near-term experiment, as it directly addresses the cost asymmetry without requiring architectural changes or auxiliary objectives.

5.3. Limitations

Our primary demonstration uses an extremely small toy model with known ground-truth features. While this controlled setting allowed us to definitively identify unfaithfulness, the toy model's 16 possible inputs and simple boolean function may not be representative of real language models. In higher-dimensional settings, lookup-table shortcuts may be infeasible, forcing more compositional representations.

The evidence from Gemma-2-2B is suggestive but not definitive. Without ground truth for what computations the model actually performs, we cannot be certain which interpretation (CLTs or PLTs) is correct. It is possible that the model genuinely implements memorization via shallow lookup tables, and the CLT is faithfully representing this while the PLT invents spurious intermediate structure. Our theory predicts CLT unfaithfulness, but this particular case could be an instance where the CLT happens to be correct.

Finally, our analysis focused on MLPs and did not examine attention layers. Cross-layer transcoders spanning attention computations may exhibit different failure modes, as the computational structure of attention differs significantly from feedforward layers.

5.4. Conclusion

Cross-Layer Transcoders offer computational advantages over per-layer alternatives, particularly in handling cross-layer superposition. However, the same architectural properties that enable these advantages create a systematic bias toward unfaithful representations that collapse intermediate computations into shallow input-output mappings.

Understanding and mitigating this faithfulness problem is important for the field of mechanistic interpretability. We hope this work encourages both caution in interpreting CLT circuits and further research into training procedures that better balance efficiency and faithfulness.

Impact Statement

This work studies the faithfulness of cross-layer transcoders, a tool that is increasingly used as general-purpose infrastructure for mechanistic interpretability. Interpretability methods are attractive precisely because they promise reliable, low-level accounts of how models compute, and such accounts are beginning to inform high-stakes decisions such as auditing models for deception, verifying that reasoning is genuine rather than memorized, and assessing safety before deployment. By identifying a systematic mode in which CLT-derived circuits can match a model's behavior while misrepresenting its internal computation, our results aim to prevent over-reliance on potentially misleading explanations and to encourage interpretability tooling that is faithful enough to support these uses. The main risk we see is the inverse: that surfacing a concrete failure mode could be read as a blanket dismissal of CLT-based interpretability. We do not intend this reading; our goal is to motivate more careful objectives and validation, not to discourage the development of interpretability methods. Beyond these considerations, we do not foresee societal consequences of this work that we feel must be specifically highlighted here.

Author Contributions

Georg Lange conceived the initial idea, designed the experiments, trained the cross-layer transcoders, and ran experiments. Rick Goldstein developed the toy-model experiments. Both wrote the paper together. Kamal Maher and Katherine Dearstyne carried out the attribution experiments in Sections 3.3 and 3.4 and contributed to the analysis in Section 4.

Acknowledgements

We would like to thank Jack Lindsey, Joshua Batson, and Michael Hanna for helpful discussions and feedback on drafts of this paper. Georg Lange was supported by the Foresight Institute. Rick Goldstein was supported by Coefficient Giving. Kamal Maher and Katherine Dearstyne carried out this work as part of the SPAR AI program, with Georg Lange as their mentor.

References

Ameisen, E., Lindsey, J., Pearce, A., Gurnee, W., Turner, N. L., Chen, B., Citro, C., Abrahams, D., Carter, S., Hosmer, B., Marcus, J., Sklar, M., Templeton, A., Bricken, T., McDougall, C., Cunningham, H., Henighan, T., Jermyn, A., Jones, A., Persic, A., Qi, Z., Thompson, T. B., Zimmerman, S., Rivoire, K., Conerly, T., Olah, C., and Batson, J. Circuit tracing: Revealing computational graphs in language models. Transformer Circuits Thread, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/methods.html.

Cunningham, H., Ewart, A., Riggs, L., Huben, R., and Sharkey, L. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.08600.

Dunefsky, J., Chlenski, P., and Nanda, N. Transcoders find interpretable LLM feature circuits. *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.11944.

Hanna, M. and Piotrowski, M. Cross-layer transcoders for Gemma-2-2B (CLT-HP). Neuronpedia, 2025. URL https://www.neuronpedia.org/transcoders-hp.

Lieberum, T., Rajamanoharan, S., Conmy, A., Smith, L., Sonnerat, N., Varma, V., Kramár, J., Dragan, A., Shah, R., and Nanda, N. Gemma scope: Open sparse autoencoders everywhere all at once on Gemma 2, 2024. URL https://arxiv.org/abs/2408.05147.

Lindsey, J., Templeton, A., Marcus, J., Conerly, T., Batson, J., and Olah, C. Sparse crosscoders for cross-layer features and model diffing. Transformer Circuits Thread, 2024. URL https://transformer-circuits.pub/2024/crosscoders/index.html. Published October 25, 2024.

Nanda, N. Actually, othello-GPT has a linear emergent world representation. Blog post, 2023. URL https://www.neelnanda.io/mechanistic-interpretability/othello.

Olah, C., Batson, J., Sklar, M., Lindsey, J., Templeton, A., and Mishra-Sharma, S. A toy model of mechanistic (Un)Faithfulness. Transformer Circuits Thread, 2025. URL https://transformer-circuits.pub/2025/faithfulness-toy-model/index.html. Published August 7, 2025.

Templeton, A., Conerly, T., Marcus, J., Lindsey, J., Bricken, T., Chen, B., Pearce, A., Citro, C., Ameisen, E., Jones, A., Cunningham, H., Turner, N. L., McDougall, C., MacDiarmid, M., Tamkin, A., Durmus, E., Hume, T., Mosconi, F., Freeman, C. D., Sumers, T. R., Rees, E., Batson, J., Jermyn, A., Carter, S., Olah, C., and Henighan, T. Scaling monosemanticity: Extracting interpretable features from Claude 3 Sonnet. Transformer Circuits Thread, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/.

A. Additional Figures Referenced from the Main Text

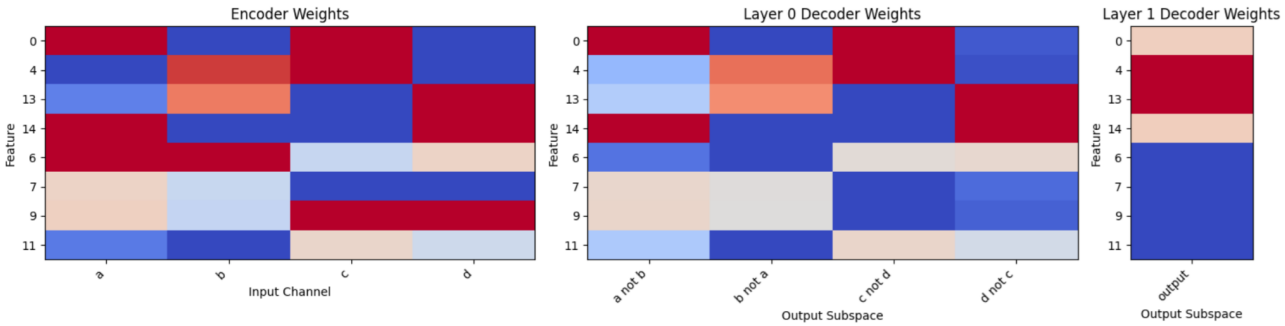


Figure 6. Jump-ReLU CLT features.

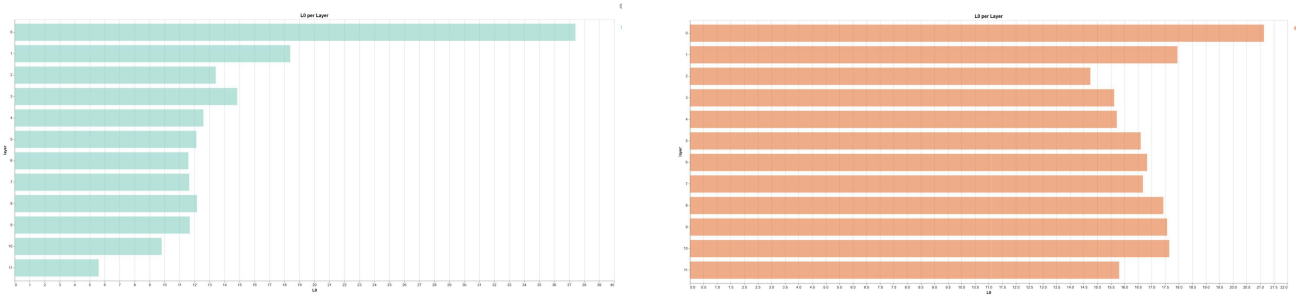


Figure 7. L0 for JumpReLU CLT (left) and PLT (right) trained on GPT2-small and OpenWebText.

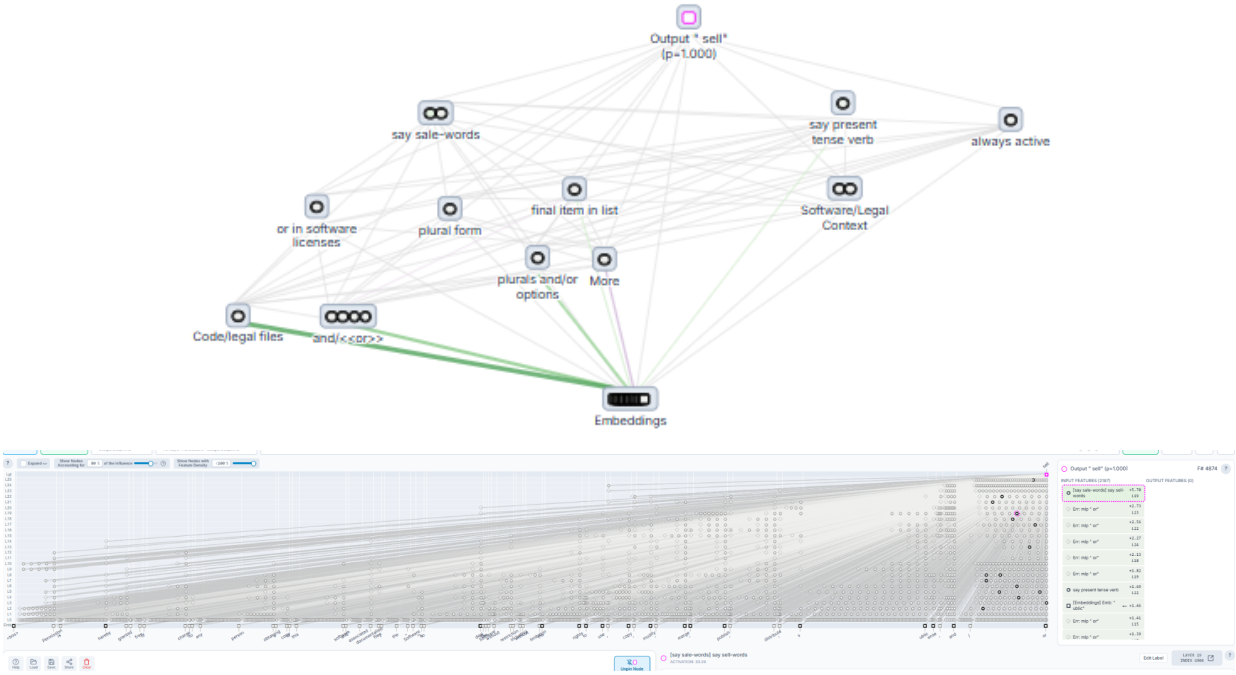


Figure 8. PLT circuit for MIT license memorization.

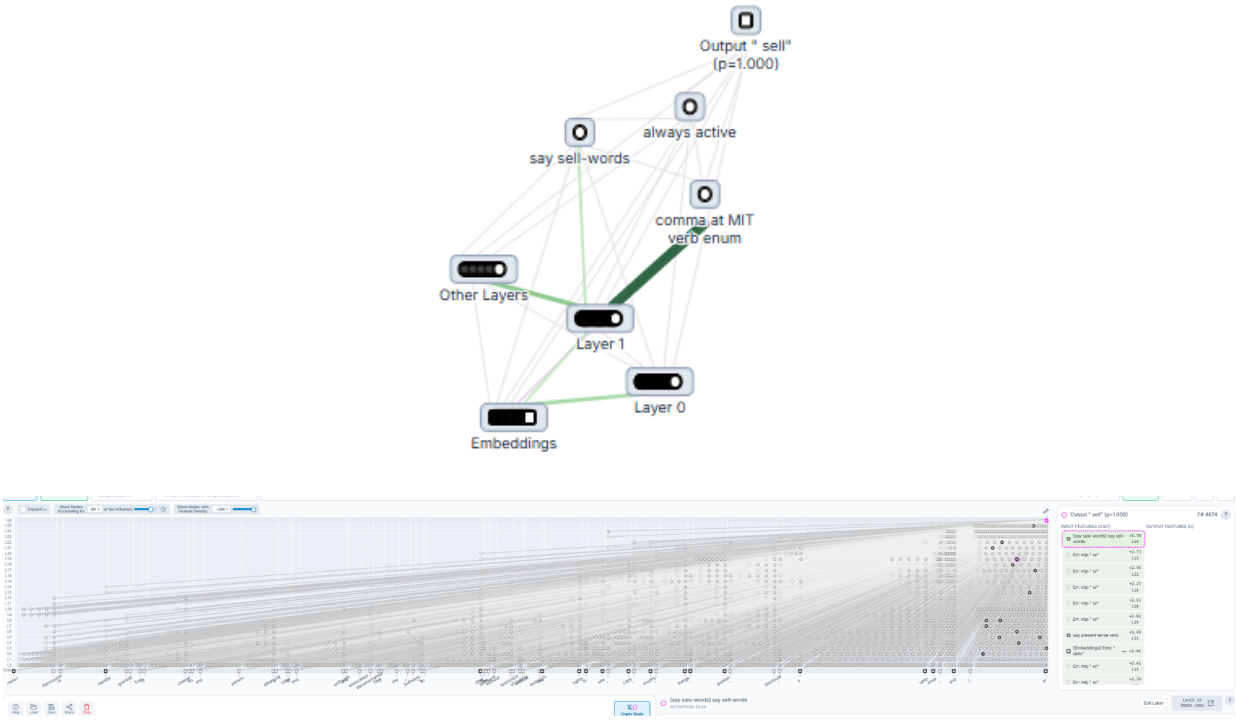


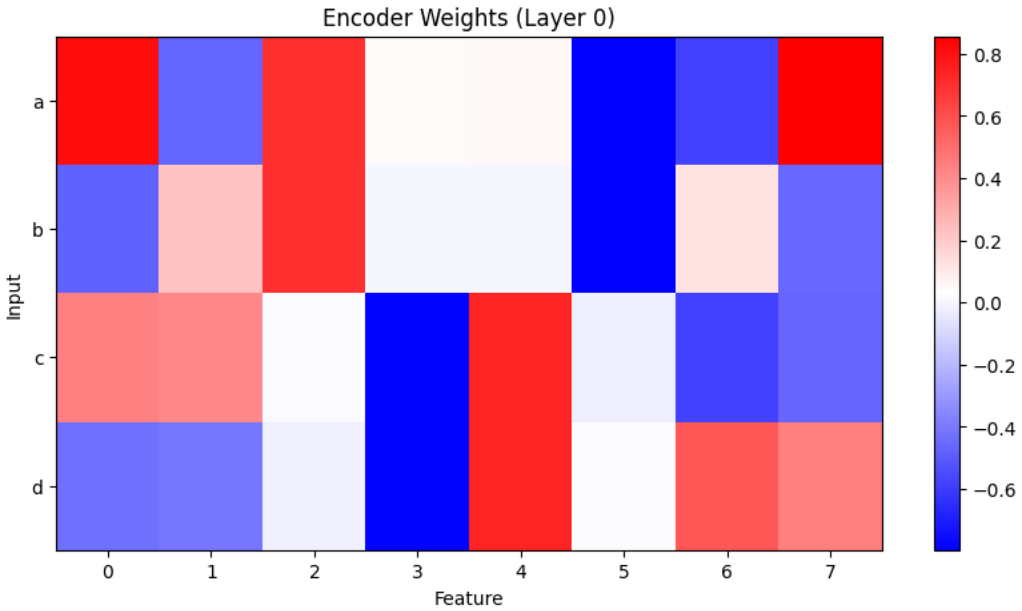
Figure 9. CLT circuit for MIT license memorization.

B. Full Analyses of the Toy-Model Transcoders

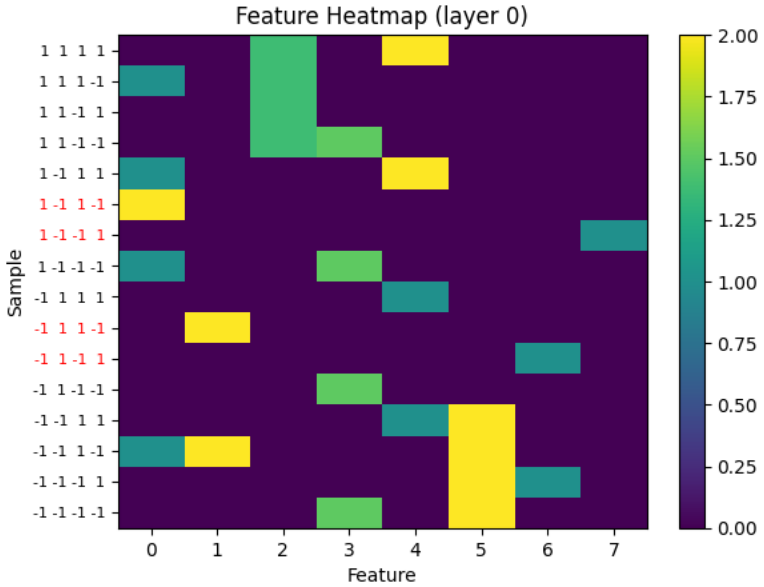
B.1. Full Analysis of JumpReLU CLT (L0=2)

We train a JumpReLU CLT with 32 features and tune the sparsity coefficient to yield an L0 of 2 across both layers. The CLT learns to activate ~ 2 features in layer 0 and none in layer 1. In layer 0, 8 of the 32 features are alive. We only plot weights and activations of living features.

The encoder weights learn different combinations of input features:

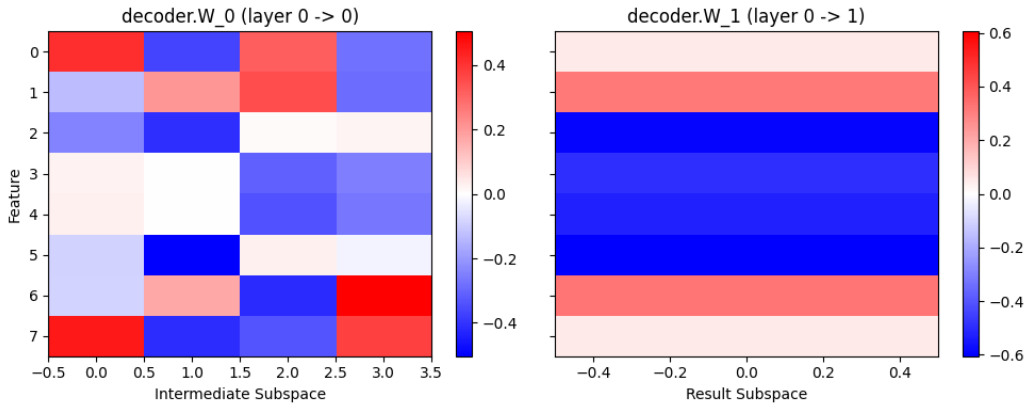


One feature is active for the inputs that should output 1, and two or three features are active for inputs that should output -1.

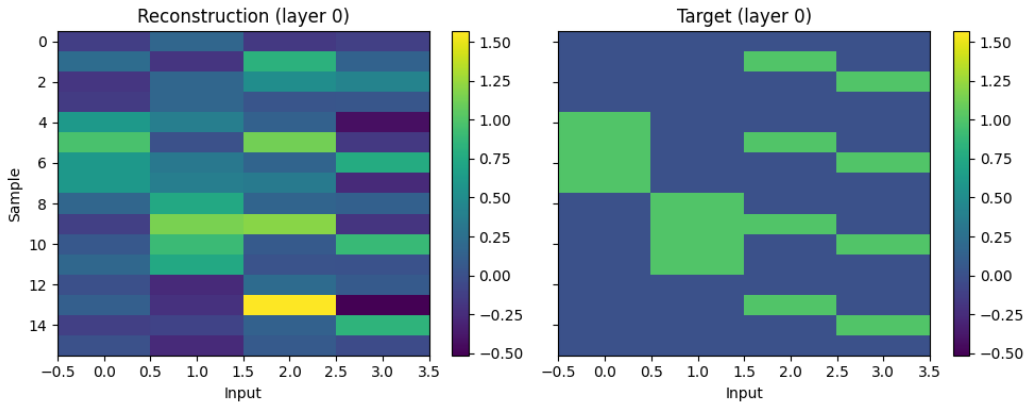


Looking at both layer-0 decoders reveals how these features in combination perfectly recover reconstructions in both layers:

To decode the final result from only layer 0 features, the decoder vector is positive for the 4 features that activate on the correct result and negative for the remainder. Because all except the correct solutions also activate on the features with



negative decoder weights, they effectively cancel out.



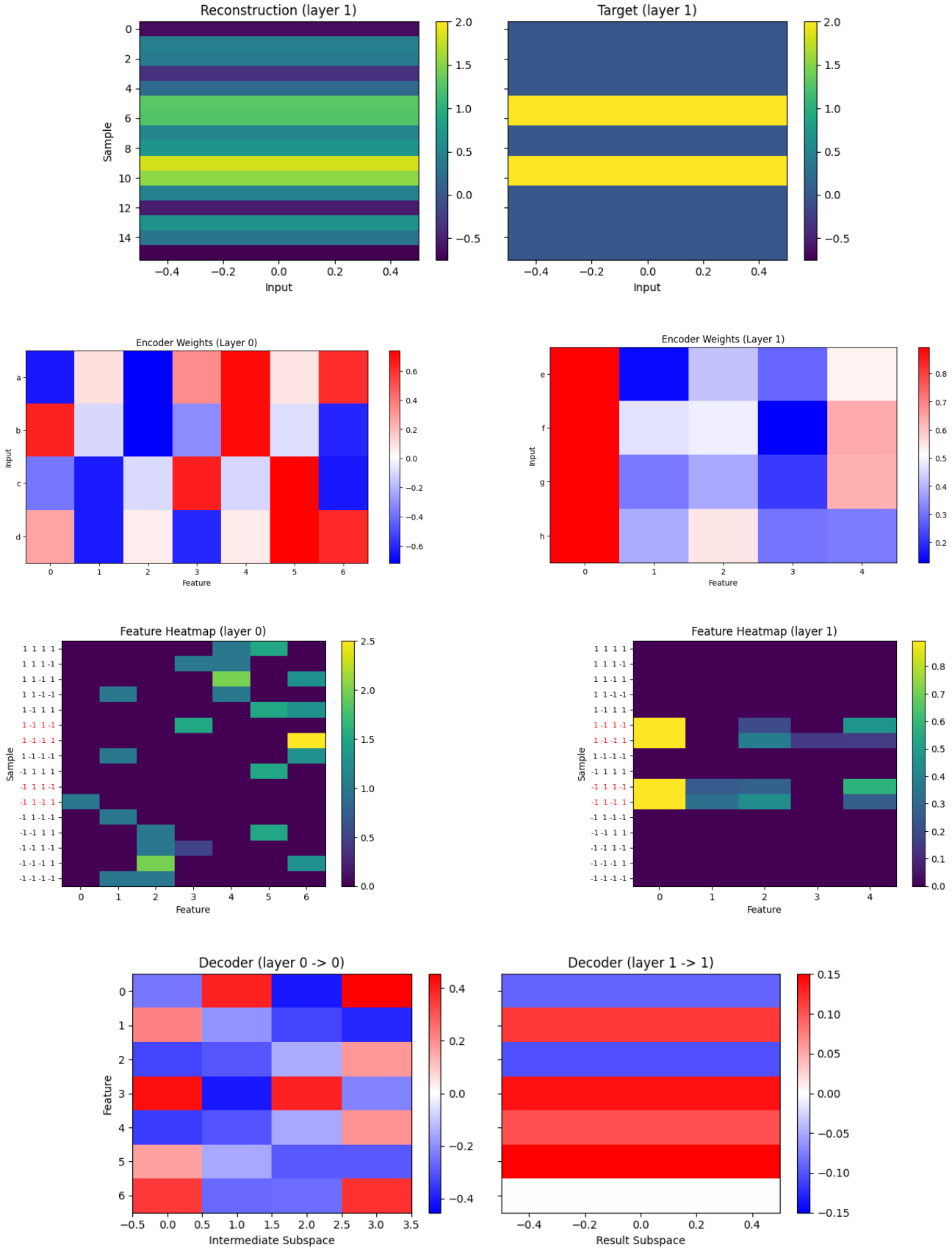
B.2. Full Analysis of JumpReLU PLT (L0=2)

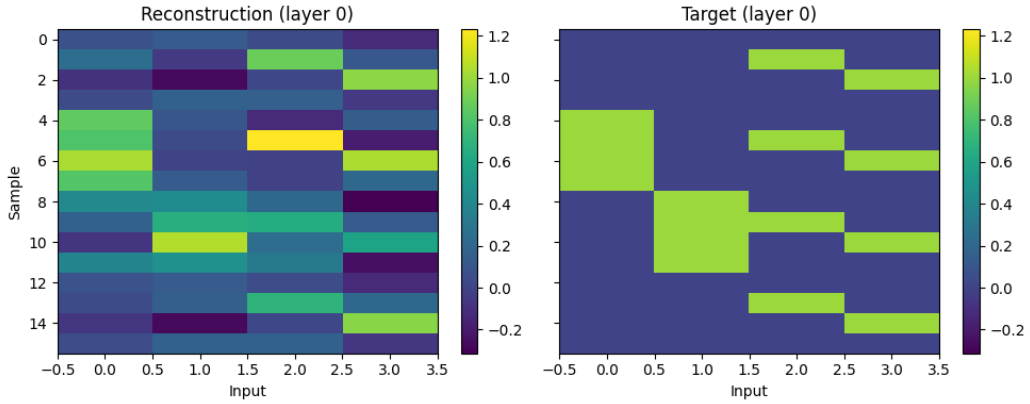
To understand what circuits the PLT learned, we plot weights, activations, and reconstructions again for the entire dataset. We show weights of alive features in both layers. Crucially, the layer-1 encoder learns non-zero weights only for the intermediate subspace $efgh$:

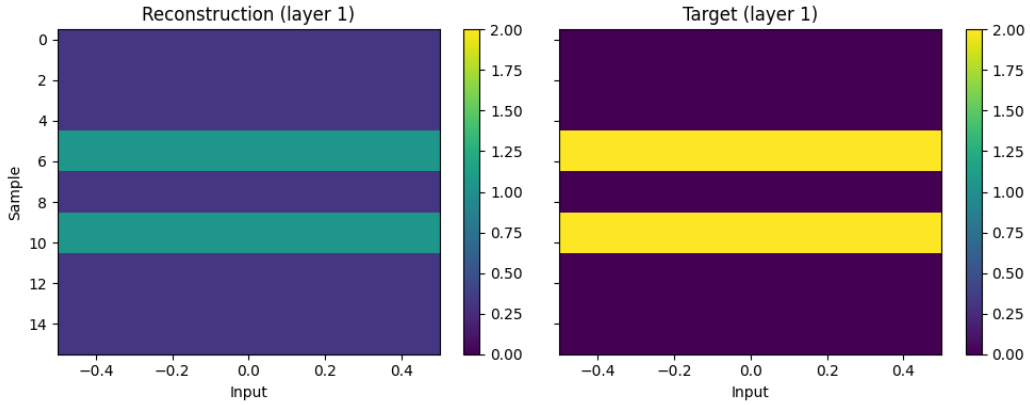
Looking at feature activations reveals that the layer-1 transcoder learned the actual behavior of layer 1: It only activates for the inputs that result in TRUE and doesn't activate for the remaining inputs:

The decoder weights show how those features are transformed into reconstructions:

Here are the reconstructions of the PLT:


Reconstruction (layer 1)
Target (layer 1)
Sample
Input
Encoder Weights (Layer 0)
Encoder Weights (Layer 1)
Feature
Feature Heatmap (layer 0)
Feature Heatmap (layer 1)
Decoder (layer 0 -> 0)
Decoder (layer 1 -> 1)
Intermediate Subspace
Result Subspace


Reconstruction (layer 0)
Target (layer 0)
Sample
Input
Input


Reconstruction (layer 1)
Target (layer 1)
Sample
Input
Input