A GENERAL MODEL DIFFING FRAMEWORK FOR UNDERSTANDING POST-TRAINING: ATTRIBUTING BEHAVIORAL DIFFERENCES TO PARAMETER UPDATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training dramatically changes the behavior of large language models, yet the mechanism behind these changes is hard to interpret because it originates from parameter updates of enormous scale and dimensionality. Prior model diffing works in the representation space and cannot trace behavioral differences back to the parameter updates that cause them. We propose a circuit-level model diffing framework that attributes the difference between two models within a single attribution graph. Running the base model and the post-trained model on the same input, we isolate the direct linear effect of the parameter difference at each block and convert it into an activation difference. We model these activation differences as nodes in an attribution graph built with a cross-layer transcoder, so that every path runs causally from a parameter-induced activation difference through interpretable features to the logits on which the two models disagree. Clustering these circuits across inputs yields a shared circuit that characterizes the behavioral difference. Applying the framework to a base model and the reasoning model obtained by reinforcement learning, we find that shallow-layer parameter updates on public tokens activate shared reasoning features in the middle-to-deep layers, which then exert a long-horizon influence across the generated response to drive specific reasoning behaviors. Extensive experiments show that activating only these few localized activation differences recovers most of the reasoning model's accuracy on in-distribution and out-of-distribution tasks, and steering the identified features elicits reasoning behavior.

1 INTRODUCTION

Post-training dramatically shifts large language model (LLM) behavior and enables capabilities such as instruction following (Ouyang et al., 2022; Wei et al., 2021; Zhou et al., 2023), reasoning (Guo et al., 2025; Lightman et al., 2024; Zelikman et al., 2022), and safety alignment (Bai et al., 2022a;b; Touvron et al., 2023) through remarkably few gradient updates (Zhou et al., 2023; Aghajanyan et al., 2021; Hu et al.). Yet the mechanisms by which models acquire these capabilities remain poorly understood (Bereska & Gavves; Allen-Zhu & Li, 2023; Jain et al., 2024) because the resulting parameter adjustments are massively scaled, high-dimensional, and practically infeasible to interpret directly. Model diffing (Lindsey et al., 2024) is an analytical approach that circumvents this problem, characterizing training-induced changes by contrasting the internal representations of different models (Kornblith et al., 2019; Lindsey et al., 2024; Minder et al., 2025), finding, for instance, that refusal behavior is mediated by a single direction in activation space (Arditi et al., 2024). However, such representation-level analyses cannot be traced back to the underlying parameters. Altered representations are merely intermediate effects that obscure the root cause. Because training algorithms exclusively operate by modifying weights, a genuine mechanistic understanding requires tracing behaviors directly back to the specific parameter updates that physically create them.

Recent progress in mechanistic interpretability decomposes model activations into interpretable features. However, merely identifying present or altered features fails to explain behaviors because behaviors emerge from causal interactions rather than isolated components. A genuine mechanistic explanation therefore requires a circuit-level view of this causal wiring. Circuit tracing (Ameisen et al., 2025; Lindsey et al., 2025) recovers these interactions by organizing them into sparse attribu-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

tion graphs. Instead of only extracting features, it replaces each MLP with a cross-layer transcoder (CLT), a sparse dictionary that reconstructs the MLP output from interpretable features acting across layers. This yields a compact feature set and makes cross-layer computation fully traceable. Inspired by this line of work, we extend circuit tracing from analyzing a single model to comparing pairs of models, and propose a novel circuit-level model diffing framework.

We run forward passes on both the base and post-trained models to isolate the linear effect of parameter differences on component activations, thereby converting the parameter difference into an activation difference. Because this linear formulation matches standard attribution graph assumptions, we can trace logits directly back to these parameter-induced changes. By using the CLT to decompose MLP layers into features, we build a unified graph where every causal path starts from an activation difference, propagates through interpretable features, and terminates at a logit difference. This renders the behavioral shift as a traceable structure. We then cluster these per-input circuits across multiple samples to extract shared structures that representatively characterize the behavioral differences. In addition, we scale the framework to a sequence length of over five hundred tokens, far beyond the few dozen tokens used in prior work (Ameisen et al., 2025; Zhao et al., 2025).

Our framework applies to any base model and any of its post-trained variants (e.g., safety-aligned, reasoning-enhanced), regardless of the post-training method (e.g., SFT, RLHF, DPO). In this paper, we specifically apply it to analyze the difference between a base model and a reasoning model obtained through reinforcement learning. This sheds light on an important question in the community (Guo et al., 2025; Yue et al., 2025; Chu et al., 2025): *how reinforcement learning endows large language models with reasoning ability*. We obtain several nontrivial findings. On public tokens within the prompt template, activation differences induced by shallow-layer parameter updates serve as primary triggers. These highly localized changes activate a critical set of shared reasoning features located in the middle-to-deep layers. These shared features then exert a sustained and long-horizon influence across the entire generated response to drive specific reasoning behaviors such as problem decomposition, step-by-step thinking, and self-doubt. Extensive experiments validate our framework. Selectively activating the tiny fraction of parameter differences localized by our method allows the base model to nearly recover the reasoning model's accuracy on both in-distribution and out-of-distribution tasks. Furthermore, steering the identified shared features successfully elicits explicit reasoning patterns even on simple questions normally answered directly. Our contributions:

- We propose a general model diffing framework modeling parameter differences as activation nodes within a single attribution graph. Unlike prior approaches comparing separate circuits, our framework causally traces differences from parameter updates to logit changes within one graph, yielding a faithful, localized circuit-level explanation applicable to any post-training method, including reinforcement learning and supervised fine-tuning.
- Applying the framework to a base model and the reasoning model obtained by reinforcement learning, we find that on public tokens, shallow-layer parameter changes activate shared features in the middle-to-deep layers that represent specific reasoning patterns. These features then exert long-horizon effects to drive diverse reasoning behaviors in the generated response. Extensive experiments confirm this, as activating only these few localized changes recovers most of the reasoning model's performance and steering the reasoning features induces pronounced reasoning behavior.
- To understand genuine long-CoT scenarios, we scale the framework to a sequence length of over five hundred tokens, far beyond those handled by prior work, making our method applicable to real-world scenarios.

2 Related Work

Model Diffing. Model diffing (Lindsey et al., 2024) analyzes post-training changes across weight (Ilharco et al.; Ortiz-Jimenez et al., 2023), representation (Kornblith et al., 2019; Arditi et al., 2024), feature (Lindsey et al., 2024; Minder et al., 2025) and circuit spaces (Prakash et al., 2024). Feature space methods only identify feature set changes and existing circuit approaches trace models separately using specific discovery techniques (Wang et al., a; Conmy et al., 2023; Marks et al., 2025) or cross layer transcoders (Ameisen et al., 2025) to compare the resulting static circuits. We instead reconstruct paths by modeling parameter variations as nodes within a single shared attribution graph. This yields complete causal paths from the update to the changed output and directly explains the

differences between models in one circuit diagram. Furthermore, while prior tracing methods target small models and short sequences (Zhao et al., 2025; He et al., 2026), our pruning scales attribution graphs to the long chain-of-thought trajectories of a real-world reasoning model.

Mechanism Interpretation of LLMs Reasoning. Existing work interprets transformer mechanisms (Chen & Zou; Chen et al., 2024; Li et al., 2026; Zhang et al.) and reasoning capabilities (Chen et al., 2026; Cheng et al., 2025; Zhou et al., 2025). Because natural language reasoning often misrepresents underlying computations (Turpin et al., 2023; Lanham et al., 2023; Arcuschin et al.; Chen et al., 2025), researchers apply causal and circuit analyses (Stolfo et al., 2023; Nanda et al.; Brinkmann et al., 2024) to reveal reasoning processes (Lindsey et al., 2025) and learned look ahead (Jenner et al., 2024). Recent studies on reasoning models identify sparse features (Galichin et al., 2026), guide behaviors (Venhoff et al.) and measure causally related steps (Bogdan et al.). While these works analyze fixed models or synthetic settings, we provide a circuit level description of how reinforcement learning reshapes the inherited reasoning circuits.

3 PRELIMINARIES

Our framework builds on circuit tracing (Ameisen et al., 2025; Lindsey et al., 2025), a pipeline that explains a model's prediction in three steps. First, a cross-layer transcoder (CLT) is trained to replace the model's MLP layers with a dictionary of sparse, interpretable features. Second, substituting the CLT into the model and freezing the remaining nonlinearities yields a local replacement model, a surrogate that is linear in the feature activations on that input. Third, this linearity allows the contribution of every component to every downstream component to be computed exactly, and these contributions are assembled into an attribution graph, from which circuits are read off.

Cross-layer transcoders. A CLT (Dunefsky et al., 2024; Ameisen et al., 2025) replaces the MLP layers with a dictionary of F sparse features. At layer l and position t, it reads the MLP input $\mathbf{x}_{\text{mlp}}^{(l,t)} \in \mathbb{R}^d$ and computes the activation $a_f^{(l,t)}$ of each feature f as the inner product with an encoder direction $\mathbf{w}_{\text{enc}}^{(l,f)}$ followed by a sparsifying activation function σ such as JumpReLU. Each active feature then writes a decoder vector $\mathbf{w}_{\text{dec}}^{(l\to l',f)} \in \mathbb{R}^d$, scaled by its activation, into the MLP output of every layer $l' \geq l$. The sum of these decoder contributions reconstructs the MLP output $\mathbf{y}_{\text{mlp}}^{(l,t)}$, and what remains is the reconstruction error $\boldsymbol{\epsilon}^{(l,t)}$. In this work, we use the CLT detailed in Appendix B.1.

Circuit trace. For a given prompt, Ameisen et al. (2025) substitute the CLT for the MLP layers and add the errors back as constants, giving a local replacement model that reproduces the original output exactly. Its computation is summarized by an attribution graph $G = (V, E)$ (Dunefsky et al., 2024; Ameisen et al., 2025; Lindsey et al., 2025), whose nodes V are active CLT features, reconstruction errors, prompt tokens, and the next-token logits we trace back from. With the attention patterns and LayerNorm scales frozen, every node reads a linear function of the upstream nodes, and the weight of an edge in E is the direct linear effect of its source node on its target node. Section 4 gives the complete form of the graph, with the derivation in Appendix A.

4 ATTRIBUTE BEHAVIORAL DIFF TO PARAMETER UPDATES

Our goal is to understand the general behavioral difference that post-training induces between a base model M_{base} and its post-trained counterpart M_{post}. The challenge is that the parameter difference between them is massive and not directly interpretable. We address this with a three-step pipeline (Figure 1). First, for a given prompt, we translate the parameter difference into localized Δ-activations (Section 4.1). Second, we build a diff graph that traces the causal paths from the Δ-activations to the logits on which the two models disagree (Section 4.2). Third, we cluster the resulting per-prompt circuits across prompts to obtain a shared circuit that attributes the general behavioral difference to the parameter updates (Section 4.3).

4.1 TRANSLATE Δ PARAMETERS INTO Δ ACTIVATIONS

The key idea is to run every block, such as an attention or MLP block, twice on the same input, once with the parameters θ_{post} of M_{post} and once with parameters θ_{base} of M_{base}. Since the input is

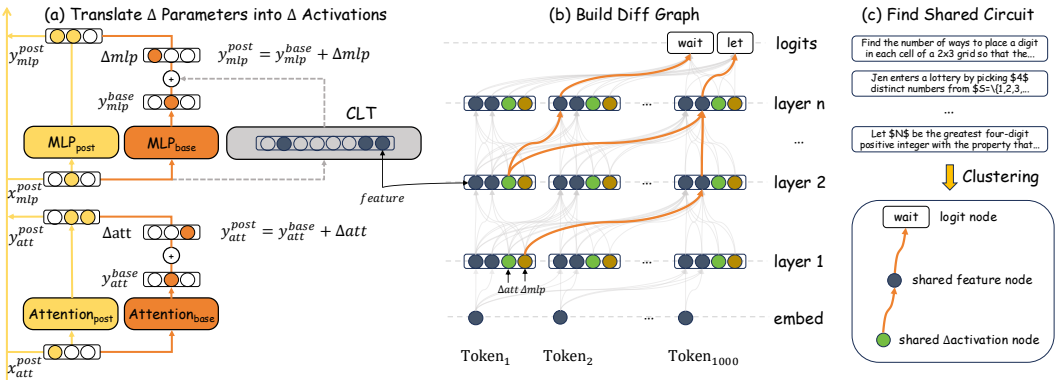


Figure 1: Our model diffing pipeline. **(a)** By fixing the input, the high-dimensional parameter differences between M_{base} and M_{post} are converted into localized Δ-activations. **(b)** These Δ-activation are incorporated into an attribution graph built with a cross-layer transcoder (CLT), allowing us to trace the causal paths from the Δ-activations through interpretable features to the changed logits. **(c)** We cluster per-input circuits to discover a shared circuit that characterizes the behavioral shift.

identical and only the parameters differ, the difference between the two outputs isolates the effect of the parameter update at that block, translating $\Delta\theta = \theta_{\text{post}} - \theta_{\text{base}}$ into a set of localized Δ-activations.

Concretely, given a prompt, we run M_{post} on the prompt together with its generated response and cache the input of every block. Let $\mathbf{x}_{\text{mlp}}^{(l,t)}$ be the cached input of the MLP block at layer l and position t. We pass this input through the MLP block under both sets of parameters and define the Δ-activation as the difference of the two outputs,

$$\Delta\mathbf{y}_{\text{mlp}}^{(l,t)} = \underbrace{\text{MLP}^{(l)}\left(\mathbf{x}_{\text{mlp}}^{(l,t)}; \theta_{\text{post}}\right)}_{\mathbf{y}_{\text{mlp}}^{(l,t)}} - \underbrace{\text{MLP}^{(l)}\left(\mathbf{x}_{\text{mlp}}^{(l,t)}; \theta_{\text{base}}\right)}_{\mathbf{y}_{\text{mlp}}^{(l,t),\,\text{base}}}. \tag{1}$$

As the actual M_{post} MLP output, the first term decomposes as $\mathbf{y}_{\text{mlp}}^{(l,t)} = \mathbf{y}_{\text{mlp}}^{(l,t),\,\text{base}} + \Delta\mathbf{y}_{\text{mlp}}^{(l,t)}$. The attention Δ-activation $\Delta\mathbf{y}_{\text{att}}^{(l,t)}$ is defined analogously using cached inputs $\mathbf{x}_{\text{att}}^{(l,s)}$ at positions $s \le t$.

Intuitively, this translation turns each block's parameter update into a vector that is written additively into the residual stream, similar to feature and error nodes of an attribution graph. Its contribution to any downstream feature or logit is therefore a direct linear effect, which is exactly how the edges of an attribution graph are defined (Section 3). Each Δ-activation can thus enter the graph as a node of its own, which makes the effect of the parameter update at every layer and position computable.

We focus exclusively on Attention and MLP Δ-activations, as updates to non-transformer parameters (e.g., embeddings, LM head) negligibly affect post-training behavior (Appendix B.8).

4.2 Build the Diff Graph

The diff graph is a directed acyclic graph in which every path starts at a Δ-activation and ends at a logit on which the two models disagree, reaching it either directly or through a chain of one or more interpretable CLT features that may span multiple layers and token positions. We specify the graph by its source nodes, edges, and target nodes below.

Source nodes. A standard attribution graph (Section 3) contains token, feature, error, and logit nodes, and traces a logit back to all of its sources. We add a Δ-activation node for every $\Delta\mathbf{y}_{\text{mlp}}^{(l,t)}$ and $\Delta\mathbf{y}_{\text{att}}^{(l,t)}$. The key difference is that the diff graph focuses on the subgraph whose sources are these Δ-activation nodes, since they are where the parameter update enters the computation.

Edges. An edge $u \to v$ carries the direct linear effect of a source node u on a target node v, that is, the part of the value of v that comes from u without passing through any other node. Every source node writes a vector $\mathbf{o}_u$ into the residual stream, and every target node reads the residual

stream along a direction $\mathbf{r}_v$, which is the encoder row $\mathbf{w}_{\text{enc}}^{(l,f)}$ for a feature node, since σ acts as the identity on active features, and the unembedding direction for a logit node. The edge weight is $A_{v,u} = \mathbf{r}_v^\top J_{u\to v}\, \mathbf{o}_u$, where $J_{u\to v}$ is the linear map of the frozen model from where u writes to where v reads. Concretely, for an active feature (l, f) at position t with read direction $\mathbf{w} = \mathbf{w}_{\text{enc}}^{(l,f)}$, the activation decomposes as

$$
\begin{aligned}
a_f^{(l,t)} \;=\; & \underbrace{\mathbf{x}_{\text{embed}}^{(t)} \cdot \mathbf{w}}_{\text{token}} \;+\; \underbrace{\textstyle\sum_{l''\le l'<l,\, f'} a_{f'}^{(l'',t)}\, \mathbf{w}_{\text{dec}}^{(l''\to l',f')} \cdot \mathbf{w}}_{\text{features}} \;+\; \underbrace{\textstyle\sum_{l'<l} \boldsymbol{\epsilon}^{(l',t)} \cdot \mathbf{w}}_{\text{errors}} \\
& +\; \underbrace{\textstyle\sum_{l'<l} \Delta\mathbf{y}_{\text{mlp}}^{(l',t)} \cdot \mathbf{w}}_{\text{MLP }\Delta\text{-activations}} \;+\; \underbrace{\textstyle\sum_{l'\le l} \Delta\mathbf{y}_{\text{att}}^{(l',t)} \cdot \mathbf{w}}_{\text{attention }\Delta\text{-activations}} \;+\; \underbrace{\textstyle\sum_{l'\le l,\, h,\, s\le t} \alpha_{t\leftarrow s}^{(l',h)} \left(W_{\text{OV}}^{(l',h)}\, \mathbf{x}_{\text{att}}^{(l',s)}\right) \cdot \mathbf{w}}_{\text{base attention}},
\end{aligned}
\tag{2}
$$

where $\alpha_{t\leftarrow s}^{(l',h)}$ and $W_{\text{OV}}^{(l',h)}$ are the attention pattern and the OV matrix of M_{base}. Since the attention pattern is frozen, the base attention term is linear in the residual streams $\mathbf{x}_{\text{att}}^{(l',s)}$ at positions $s \le t$, which expand recursively in the same way until only source nodes remain. This recursion is what $J_{u\to v}$ accumulates. The same holds for logits, and Appendix A gives the full derivation.

Target nodes. Let $z_{1:T}$ be the prompt together with the M_{post}'s response, which we also feed to M_{base}. A diff position t occurs when the models predict different next tokens $\arg\max_v M_{\text{post}}(v \mid z_{\le t}) \neq \arg\max_v M_{\text{base}}(v \mid z_{\le t})$. At these positions we create a logit node for every token in the union of their top-K predictions. These logits are where the behavior of the two models diverges, so tracing them back to the Δ-activation nodes explains the behavioral difference.

4.3 Find Shared Circuits

A single prompt yields an overly large and specific diff graph. We seek a sparse and general circuit to describe the behavior of the model rather than one particular sequence. We obtain sparsity by influence-based pruning and generality by clustering across prompts.

Influence. Following Ameisen et al. (2025), we take the absolute values of the edge weights and normalize the incoming edges of every node to sum to one. On the resulting weighted DAG, the influence of every feature node and every Δ-activation node on the target logits can be computed efficiently, as detailed in Appendix B.4.

Pruning. For a model with L layers and a sequence of T tokens, the diff graph contains about $\ell_0 \times T \times L$ feature nodes, where ℓ_0 is the average number of active features per layer and position, and $2 \times T \times L$ Δ-activation nodes. Every node can receive edges from all nodes at earlier positions and lower layers, and computing the incoming edges of a single node requires one backward pass through the frozen model. Building the full graph therefore becomes prohibitive in both computation and memory as T grows, which is why existing work traces only short prompts. We scale tracing to around $T = 1000$ tokens with three rounds of pruning:

1. **Adaptive tracing.** Following Ameisen et al. (2025), we start from the target nodes, repeatedly re-estimate the influence of every untraced feature from the edges computed so far, and trace only the features with top influence. We verify that tracing only a small fraction of the features does not change the influence ranking of the Δ-activation nodes in Appendix D.
2. **Δ-activation pruning.** Although every layer and position carries two Δ-activation nodes, we keep only those with high influence on the targets. In Section 5 we verify that these few high-influence Δ-activations dominate the behavioral change of the model.
3. **Path pruning.** Among all paths from the retained Δ-activation nodes to the target logits, we keep the k paths with the highest product of normalized edge weights, found by dynamic programming (Appendix B.4).

Appendix D provides implementation details and experiments for scaling tracing length.

Clustering across prompts. A general behavior does not depend on any particular prompt. For example, M_{post} may exhibit reasoning behavior across a dataset of math problems, and we seek the mechanism behind this behavior rather than behind a single solution. We therefore extract a per-prompt circuit, the pruned diff graph, for every prompt in the dataset of interest, and cluster

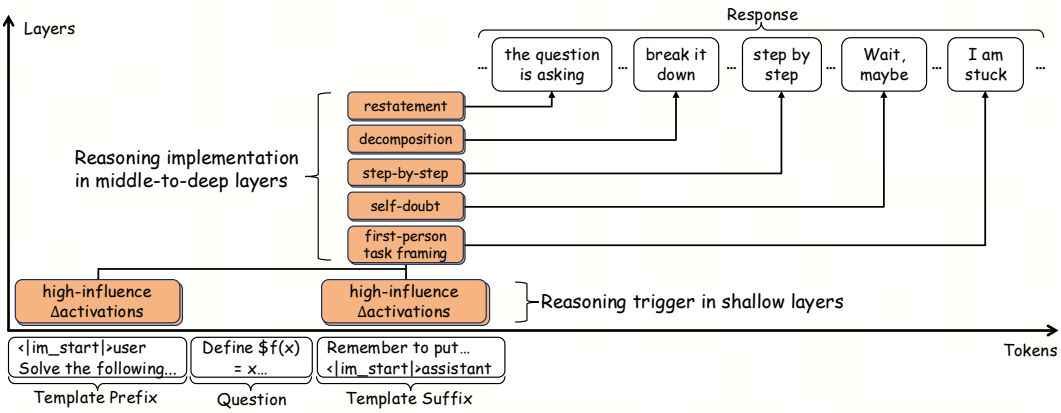


Figure 2: The hierarchical structure of the shared reasoning circuit. High-influence Δ-activations on public tokens (prefix and suffix) act as reasoning triggers in shallow layers. These activate shared features in the middle-to-deep layers that represent specific reasoning patterns (such as self-doubt), which then exert long-horizon effects to drive diverse reasoning behaviors in the generated response.

their nodes across prompts. Since sequences differ in length and content, absolute positions are not comparable, so we employ a bucketing strategy to map them to fixed intervals. We keep the key Δ-activation sources and the key intermediate features that appear in the circuits of all prompts, and further compare the length distributions of the key paths. The nodes shared by all sequences form the shared circuit, which we take as the explanation of the general behavioral difference between M_{base} and M_{post}. Appendices B.5–B.7 provide implementation details and experimental results.

5 Experiments

We validate our method across different post-training paradigms. Section 5.1 applies the framework to reasoning model to analyze how reasoning capabilities emerge. Section 5.2 tests whether these circuit-level principles hold under supervised fine-tuning, identifying a shared public token trigger. Section 5.3 quantitatively validates the identified causal differences and framework feasibility.

5.1 How Reinforcement Learning Enhances Reasoning Capabilities

Models and Datasets. We use Qwen3-8B-Base (Team, 2025) as M_{base}. For M_{post}, we perform standard GRPO (Shao et al., 2024) training using the DAPO-Math-17k (Yu et al., 2026), utilizing the verl (Sheng et al., 2025) framework (See Appendix B.2 for details). Our experiments are based on four datasets: GSM8K for grade-school math (Cobbe et al., 2021), MATH-500 for advanced competition-level problems (Lightman et al., 2024), AIME24 for challenging Olympiad-level mathematics (Codeforces) and MMLU-pro for cross-subject world knowledge (Wang et al., 2024).

To understand the emergence of reasoning, we trace individual circuits across all AIME24 samples to extract their shared circuit (Section 4). Results show this shared circuit acts as the reasoning backbone, driving diverse downstream reasoning patterns.

Sources on public tokens are significantly more important, activating shared features that also occur in public tokens. Any complete sequence divides into four segments: a template prefix, a specific question, a template suffix, and an answer (Figure 2). We term the constant prefixes and suffixes across samples as "public tokens". Figure 3 shows that public token sources are significantly more important, while the question and answer regions are relatively unimportant. Furthermore, shared features also concentrate heavily on these public tokens (Figure 4) and are primarily triggered by public token Δ-activations (Figure 12a). This aligns with our modeling, where all feature nodes are causally activated only by sources.

Shared features can trigger specific reasoning behaviors and exert long-horizon influence. Unlike general features (e.g., specific vocabulary), shared features exhibit unique properties. Steering them individually spontaneously elicits explicit reasoning patterns (such as task decomposition and

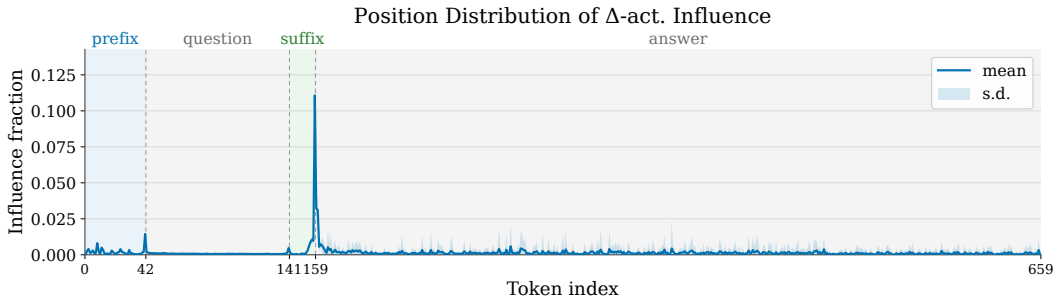


Figure 3: Position distribution of Δ-activation influence across the sequence, indicating important Δ-activations are highly concentrated on public tokens.

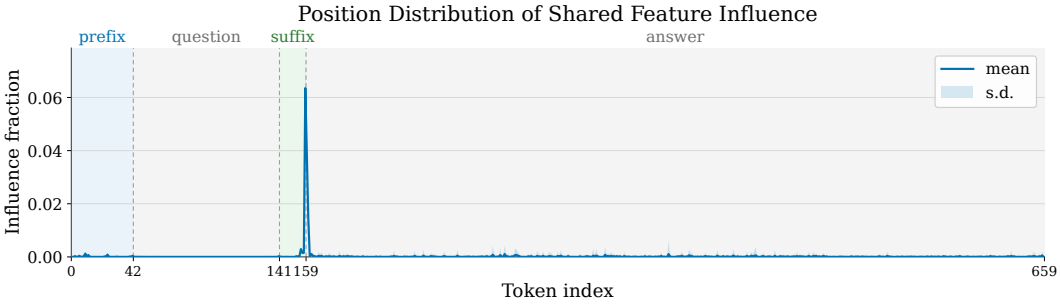


Figure 4: Position distribution of shared feature influence across the sequence, indicating important shared features are highly concentrated on public tokens.

self-doubt), even on simple questions normally answered directly. LLM-as-a-judge evaluations confirm that simultaneously steering these features significantly increases the occurrence of such behaviors (Figure 5). Furthermore, these features exert a profound long-horizon influence on the logits. Although they are primarily activated in the public token rather than in the answer, the lengthy generative answers constantly exhibit their dictated reasoning behavior. See Appendix C for all feature steering and explanation details.

Shared features representing reasoning are concentrated in middle-to-deep layers and triggered by shallow-layer Δ-activations. Figure 11a reveals a bimodal depth distribution for shared features: layer 0 features encode common syntactic or structural elements (e.g., saying prepositions), while middle-to-deep features directly represent specific reasoning behaviors. Figure 13a confirms these are primarily activated by Δ-activations heavily localized in the shallowest layers. Thus, the reasoning causal topology proceeds as follows: high-influence Δ-activations in the shallow layers of public tokens activate middle-to-deep shared reasoning features on those same tokens, which finally exert a sustained, long-horizon influence on logit generation throughout the entire response.

5.2 Consistency Across Post-training Methods

Having established how the framework explains RL-induced reasoning, we next investigate whether the same circuit-level mechanism applies to supervised fine-tuning (SFT).

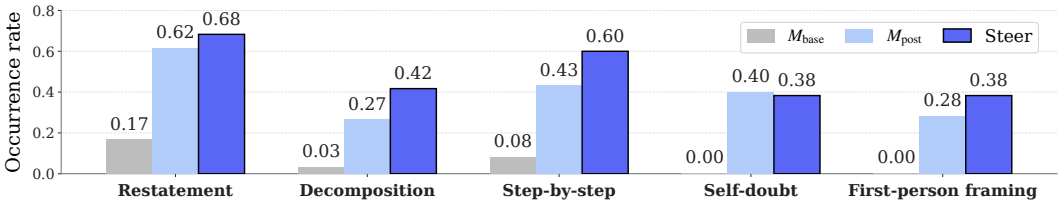


Figure 5: The occurrence rate of different reasoning behaviors. Steer represents steering all shared features based on M_{base}, which significantly improves the occurrence rate of each specific behavior.

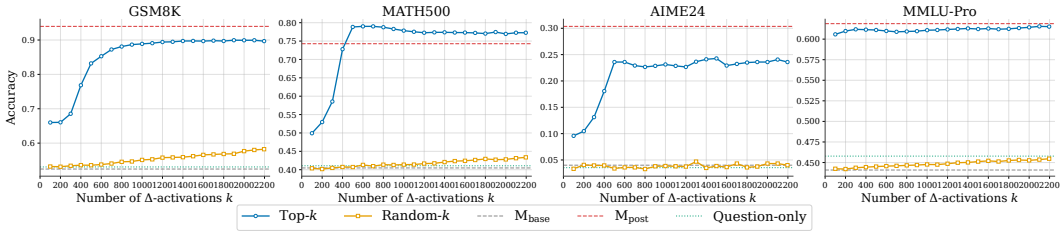


Figure 6: Performance recovery by selectively activating Δ-activations. Activating the top-k Δ-activations (blue line) rapidly recovers M_{post}'s accuracy across all datasets. Activating random nodes (orange line) or only question tokens (green line) yields negligible improvements over M_{base}. This proves our framework accurately localizes the sparse, critical parameter changes driving reasoning. All accuracies are reported as mean@64, sampled with a temperature of 1.0.

5.2.1 Generalization to Supervised Fine-tuning

We perform SFT on a synthetic Dolly-15K dataset (Conover et al., 2023) where instructions remain in English and answers are translated to uppercase French (details in Appendix B.3). On the held-out set, the SFT model produces French (94.0%) and uppercase (96.0%) answers, whereas the base model satisfies neither, confirming SFT successfully induces the target behavior.

Applying our framework to this SFT pair produces circuits mirroring the RL pair's positional properties. Δ-activation influence heavily concentrates on public tokens in the template prefix and suffix, leaving little for the question or answer regions (Figure 9). Shared SFT features and their upstream Δ-activations similarly localize to these public tokens (Figure 10, Figure 12). Thus, public token importance is not exclusive to mathematical reasoning.

5.2.2 A Shared Public Token Trigger Across RL and SFT

Despite differing in data, objective, and optimization procedure, both RL and SFT experiments yield the same conclusion: important Δ-activations concentrate on public tokens. This holds for overall influence (Figure 9) and specifically for shared circuit sources (Figure 12). In both settings, question tokens contribute substantially less influence than the fixed template.

This localization provides a unified interpretation of post-training differences. Δ-activations on public tokens, arising from parameter updates at fixed template positions, act as the triggers of the shared circuit. This likely occurs because, unlike varying questions and answers, the prompt template remains invariant across all post-training samples, leading the model to learn it as a reliable trigger for the new behavior. They activate shared features and initiate downstream computations that produce the changed behavior. Specifically, this trigger initiates long-horizon reasoning patterns for RL, and the uppercase French convention for SFT. In both settings, the main causal signal driving the behavioral shift enters the shared circuit through these public tokens.

Our findings align with prior work showing prompt templates dictate fine-tuning retention (Lyu et al., 2024) and that narrow fine-tuning leaves steerable activation traces independent of input text (Minder et al., 2026). Building on the premise that prompt templates carry critical information, we provide a specific circuit-level characterization: across both RL and SFT, the dominant signal localizes to public tokens of the shared template, acting as the shared circuit's source. By converting parameter updates into Δ-activations and tracing their causal paths, we validate these public token Δ-activations as a universal trigger for behavioral changes. This extends prior observations by mechanistically explaining how public tokens drive post-training shifts.

5.3 Validating Causal Exactness and Graph Faithfulness

5.3.1 Can Δ-activations Restore Model Performance?

We now verify that the identified Δ-activations are correct. Specifically, if we activate only the important Δ-activations on M_{base}, can the model recover M_{post}'s accuracy on real-world datasets?

(We intervene on Δ-activations rather than features, as feature steering alone is mechanistically insufficient (Appendix E.1)). We investigate this across in-distribution and out-of-distribution tasks.

Since high-importance sources concentrate on public tokens, we evaluate performance by incrementally activating these Δ-activations on top of M_{base} by influence (Figure 6). We compare five settings: M_{base} (baseline) and M_{post} (the target we aim to approach). Top-k activates the most important public Δ-activation nodes, while Random-k activates k random Δ-activation nodes across the entire sequence for fair comparison. These top-k nodes are derived exclusively from the AIME24 shared circuit and directly transferred to all datasets. Question-only activates all nodes within the question. Its performance remains close to M_{base}, confirming that question tokens do not drive reasoning, matching Figure 9. See Appendix B.7 for details.

Our framework correctly identifies the effective Δ-activation nodes. Across datasets, Top-k significantly approaches or even outperforms M_{post}, notably restoring MMLU-Pro performance almost perfectly with fewer than 100 nodes. Conversely, activating random nodes yields slow recovery, and activating all question nodes fails entirely, despite outnumbering public token Δ-activations. This confirms the framework successfully isolates the critical Δ-activations.

Performance can be restored with extremely sparse Δ-activation nodes. The Top-k curve shows M_{post} can be recovered or surpassed using only the top 600 Δ-activation nodes. For a 500-token sequence in a 36-layer model (over 18,000 total Δ-activation nodes), this constitutes less than 5% of all nodes. This demonstrates that post-training induces sparse yet highly effective changes.

5.3.2 Does Adding Δ-activation Node Affect Graph Interpretability?

We empirically verify that Δ-activations preserve graph interpretability, meaning that influence stems from valid sources rather than errors. We evaluate this using the completeness and replacement scores from Ameisen et al. (2025). Completeness measures the proportion of non-error input edges weighted by logit influence. Replacement measures the proportion of paths from embeddings or Δ-activations that reach logits entirely through non-error nodes, weighted by path strength.

Table 1: Graph scores on different datasets.

Graph Score	GSM8k	MATH-500	AIME24	MMLU-pro
Replacement				
w/o Δ-act.	0.5852	0.5864	0.5788	0.5771
w/ Δ-act.	0.6300	0.6326	0.6247	0.6197
Completeness	0.8991	0.8999	0.8978	0.8956

Table 1 reports graph scores across the four datasets. Replacement (w/o Δ-act.) is the standard score originating only from embeddings, ignoring model differences. Replacement (w/ Δ-act.) incorporates both embeddings and Δ-activations. Our average replacement (w/ Δ-act.) and completeness scores are 0.627 and 0.898, respectively. While these exceed the 0.61 and 0.80 reported by Ameisen et al. (2025), although direct comparison is precluded due to differing settings, such as cross-layer transcoder parameters.

6 Conclusion

In this work, we introduce a general circuit-level model diffing framework to mechanistically interpret the behavioral shifts induced by post-training. By converting massive parameter updates into localized activation differences within a cross-layer attribution graph, we trace causal paths directly from parameter changes to altered predictions. We apply this framework to analyze how reinforcement learning enhances the reasoning capabilities of large language models. Our findings reveal a highly structured, hierarchical causal topology: high-influence shallow-layer Δ-activations on public tokens trigger middle-to-deep shared reasoning features residing at the same public token positions, which then exert a long-horizon influence to drive diverse reasoning behaviors across the generated response. Furthermore, by comparing reinforcement learning with supervised fine-tuning, we demonstrate that this circuit-level mechanism and the critical reliance on public token triggers are remarkably consistent across different post-training paradigms, confirming that public token localization is a universal principle governing behavioral shifts. Extensive quantitative evaluations confirm our localization's exactness, demonstrating that activating a minute fraction of identified Δ-activations is sufficient to recover the reasoning model's accuracy. By scaling circuit tracing to sequences exceeding five hundred tokens, our approach moves beyond restricted toy settings, offering a scalable and faithful tool for dissecting the internal mechanisms of post-training paradigms.

AI USE STATEMENT

In this work, we used generative AI tools for supporting qualitative and thematic data analysis (e.g., utilizing LLM-as-a-judge for behavioral evaluations), interpreting results (e.g., generating natural language explanations of learned features), and generating synthetic datasets (e.g., constructing the synthetic Dolly-15K dataset for SFT). We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, or implementing methods, and assisting with translation, and cleaning and reformatting datasets are not applicable to this work. Additionally, we used generative AI tools for polishing the research paper to improve readability. We have reviewed all AI-assisted work. Specifically, we manually reviewed all LLM-generated feature explanations against their corresponding activation contexts to guarantee factual correctness and interpretability accuracy. Furthermore, all AI-assisted text revisions and drafted captions were meticulously verified and modified by the authors to ensure scientific rigor and strict alignment with our empirical findings. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our work we have meticulously documented all critical components of our proposed framework within the paper and its appendices. The complete mathematical derivations and theoretical justifications for integrating parameter differences into the attribution graph are thoroughly presented in Appendix. We also provide comprehensive descriptions of the experimental configurations alongside the sources and data processing steps for all evaluated datasets in the experiments section and corresponding appendices. Furthermore we will publicly release the complete source code upon the official publication of this manuscript to guarantee that our scalable attribution pipeline and all empirical results can be fully replicated by the research community.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: long papers)*, pp. 7319–7328, 2021.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation. *arXiv preprint arXiv:2309.14402*, 2023.

Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, et al. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 6:16318–16352, 2025.

Iván Arcuschin, Jett Janiak, Robert Krzyzanowski, Senthooran Rajamanoharan, Neel Nanda, and Arthur Conmy. Chain-of-thought reasoning in the wild is not always faithful. In *Workshop on Reasoning and Planning for Large Language Models*.

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. *Advances in Neural Information Processing Systems*, 37:136037–136083, 2024.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022a.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022b.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1533–1544, 2013.

Leonard Bereska and Stratis Gavves. Mechanistic interpretability for ai safety-a review. *Transactions on Machine Learning Research*.

Paul C Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. Thought anchors: Which llm reasoning steps matter? In *Mechanistic Interpretability Workshop at NeurIPS 2025*.

Jannik Brinkmann, Abhay Sheshadri, Victor Levoso, Paul Swoboda, and Christian Bartelt. A mechanistic analysis of a transformer trained on a symbolic multi-step reasoning task. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4082–4102, 2024.

Xingwu Chen and Difan Zou. What can transformer learn with varying depth? case studies on sequence learning tasks. In *Forty-first International Conference on Machine Learning*.

Xingwu Chen, Lei Zhao, and Difan Zou. How transformers utilize multi-head attention in in-context learning? a case study on sparse linear regression. *Advances in Neural Information Processing Systems*, 37:119573–119613, 2024.

Xingwu Chen, Tianle Li, and Difan Zou. Reshaping reasoning in LLMs: A theoretical analysis of RL training dynamics through pattern selection. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=2OO399hRD6.

Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, et al. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*, 2025.

Sitao Cheng, Xunjian Yin, Ruiwen Zhou, Yuxuan Li, Xinyi Wang, Liangming Pan, William Yang Wang, and Victor Zhong. From atomic to composite: Reinforcement learning enables generalization in complementary reasoning. *arXiv preprint arXiv:2512.01970*, 2025.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. In *International Conference on Machine Learning*, pp. 10818–10838. PMLR, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

MAA Codeforces. American invitational mathematics examination-aime 2024, 2024.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems*, 36:16318–16352, 2023.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. Free dolly: Introducing the world's first truly open instruction-tuned llm, 2023. URL https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm.

Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable llm feature circuits. *Advances in Neural Information Processing Systems*, 37:24375–24410, 2024.

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

Andrey V Galichin, Alexey Dontsov, Polina Druzhinina, Anton Razzhigaev, Oleg Rogov, Elena Tutubalina, and Ivan Oseledets. I have covered all the bases here: Interpreting reasoning features in large language models via sparse autoencoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30771–30779, 2026.

Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *International Conference on Learning Representations*, volume 2025, pp. 26721–26754, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Abir Harrasse, Florent Draye, Punya Syon Pandey, Zhijing Jin, and Bernhard Schölkopf. Tracing multilingual representations in llms with cross-layer transcoders. *arXiv preprint arXiv:2511.10840*, 2025.

Yicheng He, Zheng Zhao, Zhou Kaiyu, Bryan Dai, Jie Fu, and Yonghui Yang. Codecircuit: Toward inferring llm-generated code correctness via attribution graphs. *arXiv preprint arXiv:2602.07080*, 2026.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 523–533, 2014.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*.

Samyak Jain, Robert Kirk, Ekdeep Singh Lubana, Robert Dick, Hidenori Tanaka, Tim Rocktäschel, Edward Grefenstette, and David Krueger. Mechanistically analyzing the effects of fine-tuning on procedurally defined tasks. In *International Conference on Learning Representations*, volume 2024, pp. 9944–10015, 2024.

Erik Jenner, Shreyas Kapur, Vasil Georgiev, Cameron Allen, Scott Emmons, and Stuart Russell. Evidence of learned look-ahead in a chess-playing neural network. *Advances in Neural Information Processing Systems*, 37:31410–31437, 2024.

Marek Kadlčík, Michal Štefánik, Ondrej Sotolár, and Vlastimil Martinek. Calc-x and calcformers: Empowering arithmetical chain-of-thought through interaction with symbolic systems. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12101–12108, 2023.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pp. 1152–1157, 2016.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pp. 3519–3529. PMlR, 2019.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

Tianle Li, Chenyang Zhang, Xingwu Chen, Yuan Cao, and Difan Zou. On the robustness of transformers against context hijacking for linear classification. *Advances in Neural Information Processing Systems*, 38:69373–69411, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Jack Lindsey, Adly Templeton, Jonathan Marcus, Thomas Conerly, Joshua Batson, and Christopher Olah. Sparse crosscoders for cross-layer features and model diffing. *Transformer Circuits Thread*, 2024.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

Kaifeng Lyu, Haoyu Zhao, Xinran Gu, Dingli Yu, Anirudh Goyal, and Sanjeev Arora. Keeping llms aligned after fine-tuning: The crucial role of prompt templates. *Advances in Neural Information Processing Systems*, 37:118603–118631, 2024.

Samuel Marks, Can Rager, Eric Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. In *International Conference on Learning Representations*, volume 2025, pp. 23888–23923, 2025.

Julian Minder, Clément Dumas, Bilal Chughtai, and Neel Nanda. Robustly identifying concepts introduced during chat fine-tuning using crosscoders. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*, 2025.

Julian Minder, Clément Dumas, Stewart Slocum, Helena Casademunt, Cameron Holmes, Robert West, and Neel Nanda. Narrow finetuning leaves clearly readable traces in activation differences. In *International Conference on Learning Representations*, volume 2026, pp. 35340–35406, 2026.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems*, 36:66727–66754, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Nikhil Prakash, Tamar Shaham, Tal Haklay, Yonatan Belinkov, and David Bau. Fine-tuning enhances existing mechanisms: A case study on entity tracking. In *International Conference on Learning Representations*, volume 2024, pp. 8057–8082, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Alessandro Stolfo, Yonatan Belinkov, and Mrinmaya Sachan. A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7035–7052, 2023.

Alessandro Stolfo, Vidhisha Balachandran, Safoora Yousefi, Eric Horvitz, and Besmira Nushi. Improving instruction-following in language models through activation steering. In *International Conference on Learning Representations*, volume 2025, pp. 55790–55823, 2025.

Nishant Subramani, Nivedita Suresh, and Matthew E Peters. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 566–581, 2022.

Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 407–416, 2024.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, 2019.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, et al. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *arXiv preprint arXiv:2605.29358*, 2026.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*, 2023.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

Constantin Venhoff, Iván Arcuschin, Philip Torr, Arthur Conmy, and Neel Nanda. Understanding reasoning in thinking language models via steering vectors. In *Workshop on Reasoning and Planning for Large Language Models*.

Constantin Venhoff, Iván Arcuschin, Philip Torr, Arthur Conmy, and Neel Nanda. Base models know how to reason, thinking models learn when, 2026. URL https://arxiv.org/abs/2510.07364.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. In *The Eleventh International Conference on Learning Representations*, a.

Xu Wang, Yan Hu, Benyou Wang, and Difan Zou. Does higher interpretability imply better utility? a pairwise analysis on sparse autoencoders. In *Socially Responsible and Trustworthy Foundation Models at NeurIPS 2025*, b.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Jake Ward, Chuqiao Lin, Constantin Venhoff, and Neel Nanda. Reasoning-finetuning repurposes latent representations in base models, 2025. URL https://arxiv.org/abs/2507.12638.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 57654–57689. Curran Associates, Inc., 2025. doi: 10.52202/085713-1933. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/537d5aa768c2d534016a4d06f87bc8fb-Paper-Conference.pdf.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Chenyang Zhang, Qingyue Zhao, Quanquan Gu, and Yuan Cao. Transformers trained via gradient descent can provably learn a class of teacher models. In *The Fourteenth International Conference on Learning Representations*.

Zheng Zhao, Yeskendir Koishekenov, Xianjun Yang, Naila Murray, and Nicola Cancedda. Verifying chain-of-thought reasoning via its computational graph. *arXiv preprint arXiv:2510.09312*, 2025.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021, 2023.

Yufa Zhou, Yixiao Wang, Xunjian Yin, Shuyan Zhou, and Anru R Zhang. The geometry of reasoning: Flowing logics in representation space. *arXiv preprint arXiv:2510.09782*, 2025.

A Derivations

This appendix gives the complete attribution derivation used in Section 4. It attributes feature activation or logit to embedding, attention block, MLP block, and two Δ-activations. Appendix A.1 defines the notation and conventions, Appendix A.4 performs the attribution, and Appendix A.5 justifies treating a Δ-activation as a direct linear edge.

A.1 Notation and Conventions

Table 2 lists the symbols used below. We adopt two conventions. First, a quantity written without a base superscript denotes the result of the M_{post} forward propagation, and only quantities carrying a base superscript are computed with M_{base}'s parameters. A bar, as in $\bar{\alpha}$ and $\bar{W}_{\text{OV}}$, represent a fixed circuit of a base model that is frozen by stop-gradient and treated as a constant in the attribution, which allows us avoid repeating the base superscript. Second, we omit LayerNorm from the residual stream, following common practice in attribution (Elhage et al., 2021; Ameisen et al., 2025; Lindsey et al., 2025; Wang et al., a; Dunefsky et al., 2024; Syed et al., 2024; Conmy et al., 2023; Marks et al., 2025). For a fixed forward propagation, the LayerNorm scale is a scalar determined by the activation value at that point. Freezing it by stop-gradient absorbs it into the adjacent linear weights without altering the additive linear structure. Every linear and additive statement below holds in the sense that this scale is frozen.

Table 2: Notation used in the derivations. Here d is the residual stream width, and F the CLT dictionary size.

Symbol	Space	Meaning
$t,\ s$	index	target and source token positions
$l,\ l',\ l''$	index	target, shallower, and source layers, $l'' \le l' < l$
$(\cdot)^{\text{base}},\ (\cdot)^{\text{post}}$		computed with M_{base} or M_{post} parameters, no base superscript means post
$\mathbf{x}_{\text{embed}}^{(t)}$	$\mathbb{R}^d$	token embedding at position t
$\mathbf{x}_{\text{att}}^{(l,t)},\ \mathbf{x}_{\text{mlp}}^{(l,t)}$	$\mathbb{R}^d$	residual stream into the attention and MLP blocks of layer l
$\mathbf{y}_{\text{att}}^{(l,t)},\ \mathbf{y}_{\text{mlp}}^{(l,t)}$	$\mathbb{R}^d$	attention and MLP block outputs of layer l
$\text{Att}^{(l)},\ \text{MLP}^{(l)}$	$\mathbb{R}^d \to \mathbb{R}^d$	attention and MLP block maps of layer l
$W_{\text{enc}}^{(l)},\ \mathbf{w}_{\text{enc}}^{(l,f)}$	$\mathbb{R}^{F\times d},\ \mathbb{R}^d$	layer-l encoder matrix and feature-f encoder direction
$\mathbf{w}_{\text{dec}}^{(l''\to l',f')}$	$\mathbb{R}^d$	CLT decoder of feature (l'', f') into layer l'
$a_f^{(l,t)}$	$\mathbb{R}$	activation of feature f at (l,t)
$\boldsymbol{\epsilon}^{(l,t)}$	$\mathbb{R}^d$	CLT reconstruction error at (l,t)
$\bar{\alpha}_{t\leftarrow s}^{(l',h)}$	$\mathbb{R}$	frozen base attention score of head h, $s \to t$
$\bar{W}_{\text{OV}}^{(l',h)}$	$\mathbb{R}^{d\times d}$	frozen base OV circuit of head h at layer l'
$\mathbf{w}_{\text{logit}}^{(v)}$	$\mathbb{R}^d$	mean-subtracted unembedding direction of token v
$\Delta\mathbf{y}_{\text{mlp}}^{(l,t)},\ \Delta\mathbf{y}_{\text{att}}^{(l,t)}$	$\mathbb{R}^d$	MLP and attention Δ-activations (post minus base outputs)

A.2 Feature Activation

The cross-layer transcoder (CLT) encodes the MLP input and reads feature f by projecting onto its encoder direction,

$$\mathbf{a}^{(l,t)} = \sigma\Big(\mathbf{x}_{\text{mlp}}^{(l,t)} W_{\text{enc}}^{(l)}\Big), \tag{3}$$

$$a_f^{(l,t)} = \sigma\Big(\mathbf{x}_{\text{mlp}}^{(l,t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}\Big) \overset{\text{active}}{=} \mathbf{x}_{\text{mlp}}^{(l,t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}. \tag{4}$$

For an active feature the argument of σ is positive, the JumpReLU is locally linear, and the activation becomes a linear function of $\mathbf{x}_{\text{mlp}}^{(l,t)}$. This linearity is the basis of the additive attribution below.

A.3 Decomposing the MLP Input

The MLP input at (l,t) is the sum of the embedding, all attention outputs up to layer l, and all MLP outputs below layer l,

$$\mathbf{x}_{\text{mlp}}^{(l,t)} = \mathbf{x}_{\text{embed}}^{(t)} + \sum_{l'\le l} \mathbf{y}_{\text{att}}^{(l',t)} + \sum_{l'<l} \mathbf{y}_{\text{mlp}}^{(l',t)}. \tag{5}$$

Each block output splits into a base part and a Δ-activation:

$$\mathbf{y}_{\text{att}}^{(l',t)} = \text{Att}^{(l'),\text{base}}(\mathbf{x}_{\text{att}}^{(l',t)}) + \Delta\mathbf{y}_{\text{att}}^{(l',t)}, \qquad \mathbf{y}_{\text{mlp}}^{(l',t)} = \text{MLP}^{(l'),\text{base}}(\mathbf{x}_{\text{mlp}}^{(l',t)}) + \Delta\mathbf{y}_{\text{mlp}}^{(l',t)}. \tag{6}$$

We first decompose the output of the MLP. CLT is trained on M_{base}, and for any input, it reconstructs the base MLP output as a sparse sum of shallower feature decoders plus an error,

$$\mathrm{MLP}^{(l'),\text{base}}(\mathbf{x}_{\text{mlp}}^{(l',t)}) = \sum_{l''\le l'} \sum_{f'} a_{f'}^{(l'',t)}\, \mathbf{w}_{\text{dec}}^{(l''\to l',f')} + \boldsymbol{\epsilon}^{(l',t)}. \tag{7}$$

So the MLP parameter change is carried by $\Delta\mathbf{y}_{\text{mlp}}^{(l',t)} = \mathrm{MLP}^{(l'),\text{post}}(\mathbf{x}_{\text{mlp}}^{(l',t)}) - \mathrm{MLP}^{(l'),\text{base}}(\mathbf{x}_{\text{mlp}}^{(l',t)})$.

We emphasize that the CLT functions strictly as a feature-level decomposition of the M_{base} MLP output, independent of the input's origin. In Formula 7, the CLT is exclusively utilized to decompose the output of the M_{base} MLP block. Even though the input to this MLP block originates from the M_{post} forward pass, the CLT faithfully reconstructs how the M_{base} parameters process that exact input. Consequently, this decomposition remains mathematically consistent and lossless, and this does not introduce distribution shift issues.

We now decompose the output of attention blocks. We freeze the base attention score to $\bar{\alpha}_{t\leftarrow s}^{(l',h)} := \mathrm{stopgrad}(\alpha_{t\leftarrow s}^{(l',h),\text{base}})$ and the base OV circuit to $\bar{W}_{\text{OV}}^{(l',h)}$ (Elhage et al., 2021; Ameisen et al., 2025; Lindsey et al., 2025; Syed et al., 2024; Conmy et al., 2023; Wang et al., a; Marks et al., 2025). The base attention output is then a fixed linear position-mixing operator on the residual stream,

$$\mathrm{Att}^{(l'),\text{base}}(\mathbf{x}_{\text{att}}^{(l',t)}) = \sum_{h} \sum_{s\le t} \bar{\alpha}_{t\leftarrow s}^{(l',h)}\, \mathbf{x}_{\text{att}}^{(l',s)}\, \bar{W}_{\text{OV}}^{(l',h)}. \tag{8}$$

Freezing $\bar{\alpha}$ cuts the query and key path by stop-gradient, so attention is linear in the residual stream.

So the attention parameter change is carried by $\Delta\mathbf{y}_{\text{att}}^{(l',t)} = \mathrm{Att}^{(l'),\text{post}}(\mathbf{x}_{\text{att}}^{(l',t)}) - \mathrm{Att}^{(l'),\text{base}}(\mathbf{x}_{\text{att}}^{(l',t)})$.

A.4 Attribution

The attribution below holds for any linear readout of the residual stream. A feature reads with its encoder direction and a logit reads with the unembedding direction,

$$a_f^{(l,t)} = \mathbf{x}_{\text{mlp}}^{(l,t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}, \qquad \text{logit}_v^{(t)} = \mathbf{x}_{\text{final}}^{(t)} \cdot \mathbf{w}_{\text{logit}}^{(v)}. \tag{9}$$

Every term derived for feature f carries over to logit v by the replacement $\mathbf{w}_{\text{enc}}^{(l,f)} \to \mathbf{w}_{\text{logit}}^{(v)}$, so we derive the feature case. Projecting the residual stream equation 5 onto the encoder direction and distributing over the sum gives the embedding, attention, and MLP contributions,

$$a_f^{(l,t)} = \underbrace{\mathbf{x}_{\text{embed}}^{(t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}}_{\text{embedding}} + \sum_{l'\le l} \underbrace{\mathbf{y}_{\text{att}}^{(l',t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}}_{\text{attention}} + \sum_{l'<l} \underbrace{\mathbf{y}_{\text{mlp}}^{(l',t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}}_{\text{MLP}}. \tag{10}$$

The embedding term $\mathbf{x}_{\text{embed}}^{(t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}$ is the projection of the token embedding onto the encoder direction.

MLP contribution. Writing each MLP output as base plus Δ-activation and expanding the base part with the CLT equation 7 gives the feature-to-feature edges, the CLT error, and the MLP Δ-activation,

$$\sum_{l'<l} \mathbf{y}_{\text{mlp}}^{(l',t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)} = \sum_{l'<l} \Big(\underbrace{\sum_{l''\le l'} \sum_{f'} a_{f'}^{(l'',t)} \big(\mathbf{w}_{\text{dec}}^{(l''\to l',f')} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}\big)}_{\text{feature } (l'',f') \text{ contributes to feature } (l,f) \text{ via MLP } l'} + \underbrace{\boldsymbol{\epsilon}^{(l',t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}}_{\text{CLT error contribution}} + \underbrace{\Delta\mathbf{y}_{\text{mlp}}^{(l',t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}}_{\text{MLP diff contribution}} \Big) \tag{11}$$

The first term is the feature-to-feature edge. The source feature (l'', f') writes through its decoder $\mathbf{w}_{\text{dec}}^{(l''\to l',f')}$ into the base MLP output of layer l' and is read by feature (l, f). The second term is the contribution of the CLT error node. The last term is the contribution of the MLP Δ-activation node.

Attention contribution. Writing each attention output as base plus Δ-activation and expanding the base part over heads with equation 8 gives the attention edges and the attention Δ-activation,

$$\sum_{l' \le l} \mathbf{y}_{\text{att}}^{(l',t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)} = \sum_{l' \le l} \Big(\underbrace{\sum_{h} \sum_{s \le t} \bar{\alpha}_{t \leftarrow s}^{(l',h)} \big(\mathbf{x}_{\text{att}}^{(l',s)} \bar{W}_{\text{OV}}^{(l',h)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)} \big)}_{\mathbf{x}_{\text{att}}^{(l',s)} \text{ contributes to feature } (l,f) \text{ via attention head } (l',h)} + \underbrace{\Delta \mathbf{y}_{\text{att}}^{(l',t)} \cdot \mathbf{w}_{\text{enc}}^{(l,f)}}_{\text{attention diff contribution}} \Big) \tag{12}$$

The first term carries the source residual $\mathbf{x}_{\text{att}}^{(l',s)}$ through head (l', h) to position t and reads it with $\mathbf{w}_{\text{enc}}^{(l,f)}$. The second term is the contribution of the attention Δ-activation node, which merges the part produced at t with the part carried from earlier positions $s < t$. The source residual $\mathbf{x}_{\text{att}}^{(l',s)}$ is itself a residual stream and unrolls again by equation 5, so its MLP contributions become further feature-to-feature edges and its attention contributions expand again into base head transport plus Δ-activation, until the recursion terminates at the embeddings.

Completeness. Collecting the embedding, MLP, and attention contributions writes $a_f^{(l,t)}$ as an exact sum over feature nodes, error nodes, token embeddings, and the two Δ-activation nodes. The equality is exact, so the two Δ-activation nodes enter as additional additive sources that do not remove or rescale any existing contribution. Completeness is therefore preserved, which is the precise sense in which the Δ-activation node is admissible in the attribution graph.

A.5 Linearity of the Δ-activation Edge

Splitting each block output into a base term and a Δ-activation is an exact identity for any definition of the Δ-activation as post minus base. The attribution above, however, treats both the base term and the Δ-activation as linear edges, while the softmax in attention and the activation in the MLP are nonlinear. Approximating these inherently nonlinear interactions as local linear edges aligns with a widely adopted practice in mechanistic interpretability (Wang et al., a; Conmy et al., 2023; Syed et al., 2024; Dunefsky et al., 2024; Marks et al., 2025; Ameisen et al., 2025; Lindsey et al., 2025). We now argue that treating a Δ-activation as a direct linear edge is justified, using attention as the carrier and carrying the conclusion over to the MLP.

Two versions of the delta. The measured delta is what we compute. We freeze the base attention score and OV circuit and subtract the base linear term from the post forward pass,

$$\Delta \mathbf{y}_{\text{att}}^{\text{meas},(l',t)} = \text{Attn}^{(l'),\text{post}}(\mathbf{x}_{\text{att}}^{(l',t)}) - \sum_{h} \sum_{s \le t} \bar{\alpha}_{t \leftarrow s}^{(l',h)} \, \mathbf{x}_{\text{att}}^{(l',s)} \, \bar{W}_{\text{OV}}^{(l',h)}. \tag{13}$$

The exact delta is the true change in the block output that the parameter difference $\Delta\theta$ produces, written in closed form. It is the ground truth against which we check the measured delta. The softmax nonlinearity acts only through the attention score, so when the parameter change is confined to W_{OV} and leaves the score unchanged, freezing $\bar{\alpha}$ introduces no approximation and the attention output is exactly linear in W_{OV},

$$\Delta \mathbf{y}_{\text{att}}^{\text{exact},(l',t)} = \sum_{h} \sum_{s \le t} \bar{\alpha}_{t \leftarrow s}^{(l',h)} \, \mathbf{x}_{\text{att}}^{(l',s)} \, \Delta W_{\text{OV}}^{(l',h)}, \qquad \Delta W_{\text{OV}}^{(l',h)} = W_{\text{OV}}^{(l',h),\text{post}} - W_{\text{OV}}^{(l',h),\text{base}}. \tag{14}$$

This exact delta is produced entirely by the parameter difference $\Delta\theta$ acting through the OV circuit, and it is fully interpretable.

When they agree. When only W_{OV} changes, the two deltas are identically equal, and the linear treatment needs no assumption. Under real fine-tuning the query and key also change, so $\alpha^{\text{post}} \neq \bar{\alpha}$ and the two deltas differ by exactly the softmax nonlinearity that freezing $\bar{\alpha}$ drops.

Carrying over to the MLP. The same distinction holds for the MLP. Its last linear map is the down projection W_{down}, the counterpart of W_{OV}. Confining the parameter change to W_{down} makes the block output exactly linear in the changed parameter, so the resulting Δ-activation is exact. Our method instead reads the Δ-activation off the whole MLP block, which folds the change in the

remaining projections through the activation nonlinearity into $\Delta \mathbf{y}_{\text{mlp}}$. As with attention, this is a linearization whose gap we check empirically.

Quantifying the gap. Rather than bound this gap analytically, we check it empirically. Restricting each Δ-activation to the exactly linear map, W_{OV} for attention and W_{down} for the MLP, gives a delta with no approximation that we treat as ground truth. Our method instead reads each Δ-activation off the whole block. What attribution uses is not the exact magnitude of a Δ-activation but which Δ-activation nodes drive the logits, so we compare the two definitions by the importance ranking they induce over the Δ-activation nodes. A high rank correlation means the whole-block delta preserves the causal ordering of the exactly linear delta, so the dropped nonlinearity does not matter for attribution. We run this comparison in detail in Appendix A.6 and find the rankings agree strongly, which supports treating each Δ-activation as a direct linear edge.

A.6 Empirical Validation of the Linearity Assumption

We validate the linear treatment of the Δ-activation by comparing four ways of forming the two Δ-activations and checking that the importance ordering of the Δ-activation nodes is stable across them.

Configurations and Metric. For a block, the exact form replaces only its last linear map on a fixed input and takes the difference of that map's outputs, which is exactly linear in the changed parameter and carries no approximation. The whole-block form replaces the entire block and takes the difference of the block outputs, the definition used in the main method. The exact map is W_{OV} for attention and W_{down} for the MLP. Crossing the two choices for the two blocks gives the four configurations detailed in Table 3, including both blocks being exact linear (Exact Baseline), only the attention block being linear (Attention Exact), only the MLP block being linear (MLP Exact), and both blocks being frozen nonlinear (Our Method). To evaluate stability, we rank the Δ-activation nodes by their influence on the logits and compute the Spearman correlation between the baseline ranking and the rankings produced by the alternative configurations across the AIME24 dataset. We form three rankings, one over the attention Δ-activation nodes, one over the MLP Δ-activation nodes, and one over both together.

Table 3: Configurations of the two Δ-activations and their Spearman correlation of node influence rankings compared to the Exact Baseline. The high correlation achieved by our whole-block method demonstrates that approximating the block-level difference as a direct linear edge reliably preserves the causal ordering of nodes, empirically justifying our linear treatment.

Configuration	Attention Δ-act.	MLP Δ-act.	Attention	MLP	Both
Exact Baseline	W_{OV} only	W_{down} only	-	-	-
Attention Exact	W_{OV} only	whole block	0.9953	0.8961	0.8785
MLP Exact	whole block	W_{down} only	0.9733	0.9992	0.9867
Our Method	whole block	whole block	0.9682	0.8956	0.8918

Results and analysis. Table 3 presents the average Spearman correlation of each configuration relative to the baseline across three ranking scopes (Attention nodes, MLP nodes, and Both combined). Across all comparisons, the Spearman correlation remains highly stable, ranging from roughly 0.87 to over 0.99. When attention remains precisely linear (Attention Exact), the MLP Δ-activations ordering is highly correlated with the baseline, and the attention ordering remains almost entirely unchanged. A symmetrical conclusion holds for MLP Exact. Most importantly, Our Method, using the whole block for both attention and the MLP, recovers almost the identical influence ordering as the exactly linear baseline. The nonlinearity dropped in both blocks (the softmax in attention and the activation in the MLP) shifts the magnitude of a Δ-activation only slightly and does not alter which nodes causally drive the logits. Because our framework utilizes the Δ-activation nodes via this causal ordering, its stability confirms that the whole-block Δ-activation adopted throughout the paper is an empirically supported approach rather than a limiting assumption.

B EXPERIMENT DETAILS

B.1 CROSS-LAYER TRANSCODER.

We use a cross-layer transcoder (CLT) (Ameisen et al., 2025) trained on all 36 layers of Qwen3-8B-Base (Yang et al., 2025). Each layer has 32,768 latents, whose encoder reads the MLP input of that layer and whose decoders write to the MLP outputs of that layer and all later layers. Sparsity is enforced by a single TopK selection over all layers: for each token, every latent is scored by its rectified pre-activation times the norm of its concatenated decoder weights, and the 360 highest-scoring latents across all layers are kept, i.e. 10 per layer on average. The number of active latents per token is thus fixed, while their distribution over layers is learned. The CLT is trained to minimize the squared reconstruction error of the MLP outputs summed over layers, with MLP inputs and outputs normalized to an expected norm of $\sqrt{n}$, plus an auxiliary loss on dead latents following Gao et al. (2025). We train on activations from 3.2B tokens drawn from a mixture of pre-training, reasoning, and instruction-following data in the Olmo 3 corpus (Olmo et al., 2025), using AdamW with a peak learning rate of 8×10^{-5} and an effective batch size of 32,768 tokens. On held-out data, the CLT reaches a fraction of variance unexplained (FVU) of 27.1% averaged over layers, with 3.2% of its latents dead.

B.2 RL SETUP FOR THE REASONING SCENARIO

For the reinforcement learning in the reasoning scenario, we configure the GRPO algorithm (Shao et al., 2024) with a training batch size of 512, a learning rate of 1×10^{-6}, a maximum prompt length of 2048, and a rollout group size of 8. The training process runs for 100 steps, with a maximum response length set to 20480 during generation. When evaluating the accuracy in Section 5.3.1, we use a sampling temperature of 1.0 and report all results as mean@64. Additionally, the prompt templates and the answer parsing methods are completely identical to those used in (Yu et al., 2026).

B.3 SYNTHETIC SFT DATASET AND TRAINING SETUP

We construct a synthetic supervised fine-tuning (SFT) dataset from the Dolly-15K training set (Conover et al., 2023). For every example, we preserve the original instruction exactly as written in English, translate the original answer into French, and capitalize every character in the translated answer. The resulting task therefore keeps the question in English while requiring an uppercase French response.

Example. A representative training example is shown below. The English text represents the original answer.

Instruction	What are the words of House Buckwell?
English	"Pride and Purpose"
SFT	"FIERTÉ ET DESSEIN"

Training and evaluation configuration. We randomly sample 10,288 examples from the Dolly-15K training set for SFT. We train for 80 steps with a learning rate of 1×10^{-5} and a batch size of 256. For evaluation, we randomly sample 50 examples as a held-out test set, with no overlap with the training examples. The resulting post-trained model is denoted by M_{post}, while the original Qwen3-8B-Base model is denoted by M_{base}.

We use LLM-as-a-judge to determine the occurrence rate of behaviors. On the held-out test set, the base model satisfies neither target behavior: its French and uppercase response rates are both 0.000. After SFT, the model produces French responses for 94.0% of the examples and uppercase responses for 96.0%. These results show that the SFT model learns the requested French-uppercase response format.

B.4 FIND CIRCUITS

To extract a difference circuit for one prompt, we treat the causal structure between the Δ-activation nodes and the logit nodes as a multi-source, multi-sink pathfinding problem on a DAG. The sources are the Δ-activation nodes and the sinks are the logit nodes. Incoming influences at each node are converted into shares

$$w_{v,u} \;=\; \frac{|A_{v,u}|}{\sum_{u'} |A_{v,u'}|}\,, \tag{15}$$

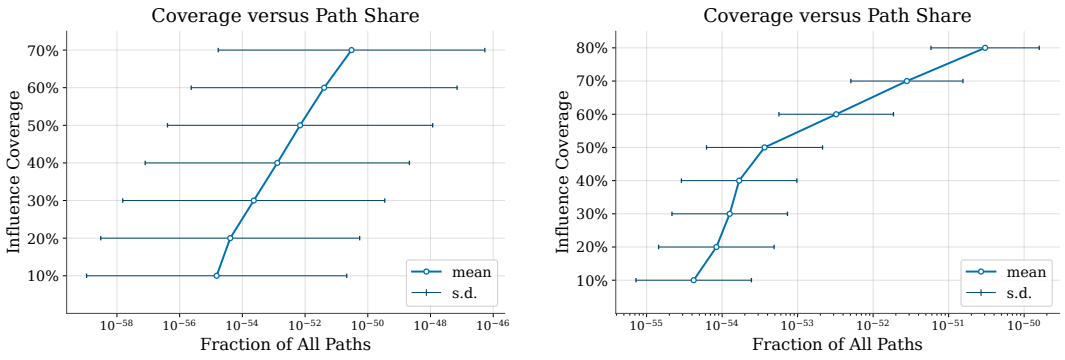
so that every edge weight is the fraction of the target explained by that source. A stronger path represents stronger causality, and we rank all source-to-sink paths jointly under the max-product objective

$$\pi^{\star} \;=\; \operatorname*{argmax}_{\pi:\,\Delta\text{-act} \rightsquigarrow \text{logit}} \;\prod_{(u\to v)\in\pi} w_{v,u}. \tag{16}$$

Because the graph is a DAG, this ranking is obtained exactly by reverse topological dynamic programming. Let $\mathrm{Best}_K(u)$ be the K highest-scoring suffixes from node u to the logits. Logit nodes are initialised by their sink weights, and every earlier node is updated from its successors:

$$\mathrm{Best}_K(u) \;=\; \mathrm{TopK}\big\{\, w_{v,u}\,\mathrm{Best}_K(v)[r] \;\big|\; u \to v,\; r < K \,\big\}. \tag{17}$$

A second merge over the outgoing edges of the Δ-activation nodes yields the global ranking of complete paths. We keep the highest-scoring paths until their influence sums to a prescribed fraction of the total Δ-activation to logit mass. Each path is a human-understandable chain from a Δ-activation through interpretable features to a changed prediction. The union of the retained paths is the difference circuit for the prompt.



(a) Synthetic SFT Case

(b) RL Reasoning Case

Figure 7: Cumulative influence coverage versus the fraction of retained paths. Both distributions exhibit a strict Pareto pattern. (a) In the synthetic supervised fine-tuning case, retaining a path fraction approaching 10^{-50} captures 70% of the causal mass. (b) In the reinforcement learning reasoning case, an exceptionally sparse subset (approximately 10^{-51}) accounts for 80% of the overall causal mass. Retaining additional paths beyond these respective thresholds yields heavily diminishing returns and becomes computationally prohibitive.

In this work, we set the prescribed threshold to retain the top paths that cumulatively account for a target fraction of the total influence mass. Because our framework traces exceptionally long sequences, the resulting directed acyclic graph becomes enormously complex and contains roughly 10^{55} distinct paths. Retaining and processing every possible path is therefore computationally intractable.

Fortunately, the distribution of path quality strictly follows a Pareto principle. As illustrated in Figure 7, we dynamically adjust the retention threshold based on the specific scenario to balance fidelity and computational feasibility. For the synthetic supervised fine-tuning case (Figure 7a), we set the threshold to 70% because capturing this mass already requires a substantial path fraction approaching 10^{-50}, and targeting a higher coverage would cause an intractable computational explosion. Conversely, for the reinforcement learning reasoning case (Figure 7b), retaining an exceptionally sparse fraction of approximately 10^{-51} of all possible paths is sufficient to reach an 80% cumulative

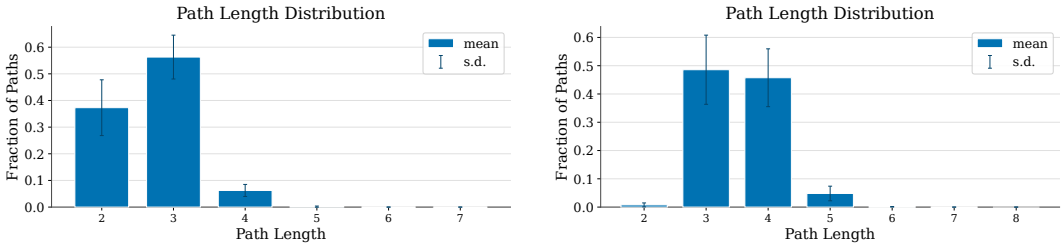
influence threshold. This indicates that capturing the top subset of paths effectively restores the vast majority of the causal mass and makes exhaustive path retention unnecessary. We halt the retention process at these specific thresholds because incorporating additional paths yields strict diminishing returns and prohibitive computational costs.

B.5 Path Length Distribution of Pruned Graphs

To understand the structural complexity of the causal graphs obtained after the pruning procedure (detailed in Appendix B.4), we analyze the distribution of path lengths within these subgraphs. In our formulation, a complete causal path consists of a source node (the Δ-activation), intermediate feature nodes, and a sink node (the final logit). Following this definition, for example, a path of length 3 indicates that there is exactly one intermediate feature node between the source and the sink.

As illustrated in Figure 8, the paths within the pruned graphs are remarkably short across different post-training paradigms. In the RL reasoning case (Figure 8a), the distribution is heavily dominated by paths of length 3, meaning the causal chains are highly direct. Similarly, in the synthetic SFT case (Figure 8b), the vast majority of paths have lengths of 3 or 4, with length 3 being the most frequent.

The observation that the average path length is consistently short (typically centering around 3) provides strong empirical support for our analysis of the model's reasoning topology. It confirms that the causal chain of reasoning is relatively direct rather than deeply recursive: the highly important Δ-activations located in the shallow layers serve as direct triggers for the shared reasoning feature nodes in the middle-to-deep layers, which in turn directly exert their influence on the final logit generation.



(a) RL Reasoning Case (b) Synthetic SFT Case

Figure 8: Path length distribution of the pruned subgraphs. A path comprises the source (Δ-activation), intermediate feature nodes, and the final sink (logit). A path length of 3 indicates the presence of exactly one intermediate feature node. In both the (a) RL reasoning case and the (b) synthetic SFT case, the paths are remarkably short, with the vast majority centering around a length of 3. This short average path length aligns with our topological finding that shallow Δ-activations directly trigger middle-to-deep shared features, which then directly influence the logits.

B.6 Hierarchical Structure of Shared Circuits

In Section 5.1, we analyze a series of topological properties of shared circuits in reasoning scenarios. Now, for each conclusion we have claimed, we will provide detailed experimental results and analysis.

Δ-activations on public tokens are significantly more important. Any complete sequence can be divided into four segments: a template prefix, a specific question, a template suffix, and a specific answer (Figure 2). Across different samples, the template prefixes and suffixes remain constant, which we refer to as "public tokens". As illustrated in Figure 9, the average importance of Δ-activations is overwhelmingly concentrated on these public tokens, while the specific question and answer regions exhibit near-zero importance. This indicates that parameter changes on the public template tokens serve as the primary drivers initiating the model's reasoning behavior, rather than the specific content of the question itself.

For all results of position distribution in this section, including Figure 9, we use the following method for bucketing calculation and visualization. Take the RL reasoning case as an example. First, all results are the mean of all samples in AIME24. For each sample, we set the maximum response length to 500. Therefore, for different samples, they have the same prefix (42 tokens in total), different questions (average length of 99), the same suffix (18 tokens in total), and answers with different content but the same length (500 tokens in total). We use a bucketing strategy to calculate the average influence of each token in a sequence. Each segment is considered a bucket, with bucket sizes of 42, 99, 18, and 500 respectively. We can then map the token influence of each segment of each sample onto buckets of fixed length to calculate the average influence at a specific position within a segment.

Shared features are also mainly concentrated on public tokens. Beyond the Δ-activations, the shared features within the mined reasoning circuits exhibit a remarkably similar positional bias. Figure 10 demonstrates that the influence of these shared features is significantly localized to the public tokens of each sample.

Shared features representing abstract behaviors are concentrated in the middle-to-deep layers. Analyzing the layer-wise distribution reveals that these shared features are not evenly distributed across the network depth. As shown in Figure 11, the shared features present a distinct bimodal distribution, concentrating at two ends: one portion is localized in layer 0, while the other portion is concentrated in the middle-to-deep layers. The features in layer 0 consistently correspond to common syntactic or structural elements, such as "saying a certain preposition". In contrast, the features concentrated in the middle-to-deep layers represent the genuine abstract behaviors. For example, in the RL reasoning case, these features explicitly govern the reasoning behaviors, acting as a central processing hub for the circuit.

The Δ-activations that trigger the shared features are located on the public tokens and in the shallow layers in the reasoning case. Recall that for each individual sample, we obtain a sparse subgraph via the pruning procedure detailed in Appendix B.4. The shared features are extracted by clustering across these sample-specific subgraphs. And we also compute the average influence distribution of the Δ-activations on the pruned subgraph of each sample. As the origin points of the shared circuit, these Δ-activations act as the initial triggers that activate the subsequent shared features. Figure 12a and Figure 13a confirm that in the RL reasoning case, these triggering Δ-activations are heavily localized to both the public tokens (spatially) and the shallow layers of the model, particularly layer 0 (depth-wise). This pinpoints the exact origin of the post-training reasoning shift: shallow-layer parameter updates operating directly on prompt templates.

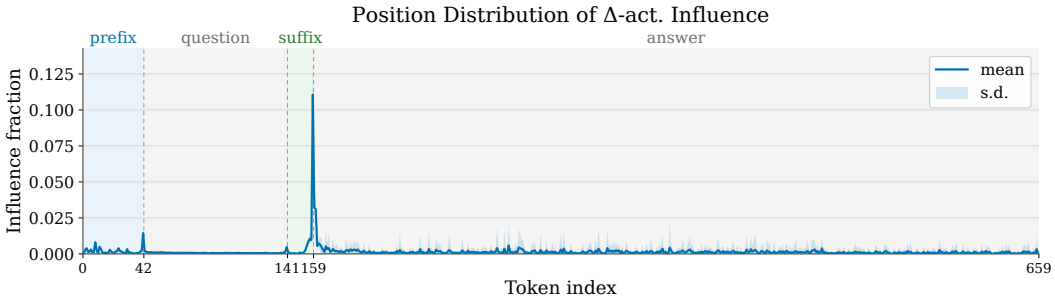
B.7 Implementation Details of Selectively Activating Δ-activations

In Section 5.3.1, we attempt to restore model performance by selectively activating certain Δ-activations. Now we explain specifically how we activate Δ-activations.

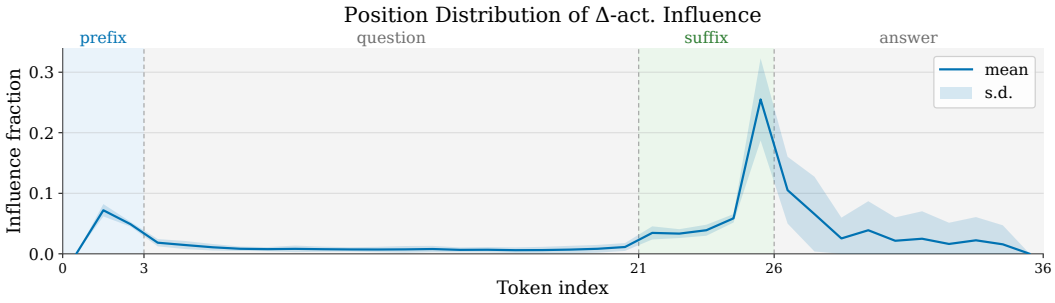
In our framework, model differences are causally attributed to Δ-activation nodes, which serve as the exclusive source nodes for causal paths (including those within the shared circuit). Because these Δ-activation nodes directly stem from parameter updates ($\Delta\theta$), we implement selective activation via targeted parameter replacement.

Specifically, when evaluating model performance on M_{base} while selectively activating a specific Δ-activation node at a given layer l and position p, we dynamically and precisely replace the parameters of the corresponding sub-layer block rather than the entire transformer layer. If the node is an MLP Δ-activation, we replace only the parameters of the MLP block at layer l with those from M_{post} when auto-regression reaches position p. Conversely, if the node is an Attention Δ-activation, we replace only the parameters of the Attention block at layer l with those from M_{post}. For all other blocks, layers, and sequence positions, the model strictly retains the original parameters of M_{base}. Consequently, in Section 5.3.1, selecting the Top-k or Random-k configurations equates to performing this precise block-level parameter substitution for exactly k selected nodes (which can be any combination of MLP and Attention Δ-activations) at their respective layers and token positions.

We highlight that this direct parameter replacement provides a mathematically grounded and exact mechanism. By the surgical insertion of Δ-parameters at the designated granularity of specific

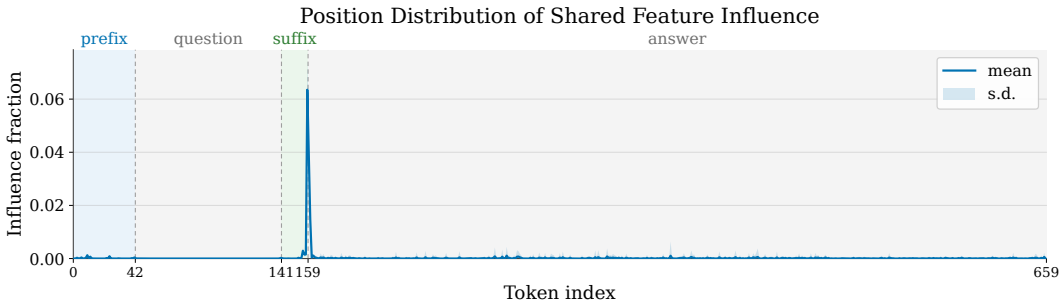


(a) RL Reasoning Case

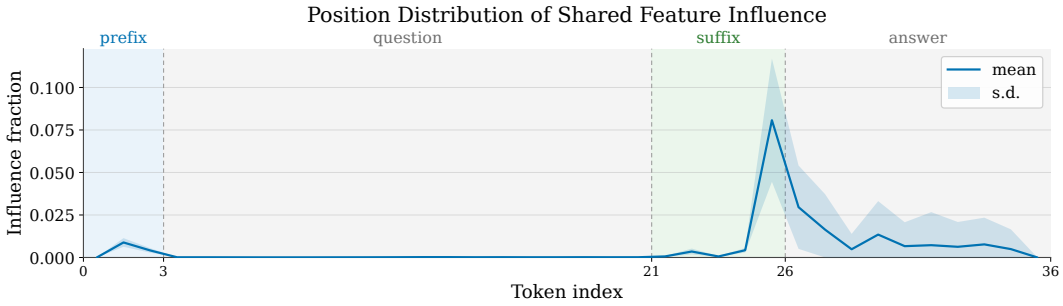


(b) Synthetic SFT Case

Figure 9: Position distribution of Δ-activation influence across the sequence. The results are calculated and visualized by binning into four segments (the same applies below). In both the (a) RL reasoning case (averaged over AIME24 samples) and the (b) synthetic SFT case (averaged over Dolly15k samples), the mean importance is highly concentrated on public tokens within the template prefix and suffix, while the question and answer regions show near-zero importance. This confirms that Δ-activations on public tokens are significantly more important across different post-training paradigms.



(a) RL Reasoning Case



(b) Synthetic SFT Case

Figure 10: Position distribution of shared feature influence across the sequence. In both the (a) RL reasoning case and the (b) synthetic SFT case, shared features are highly concentrated on public tokens within the template prefix and suffix.

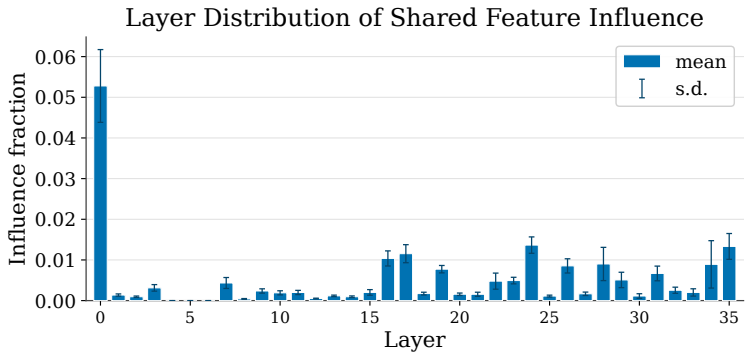
network blocks and tokens, we can precisely isolate the causal contribution of specific nodes without introducing estimation bias or arbitrary scaling artifacts, ensuring that the evaluated performance recovery accurately reflects the true impact of the identified circuit components.

B.8 Impact of Non-Transformer Parameters During Post-Training

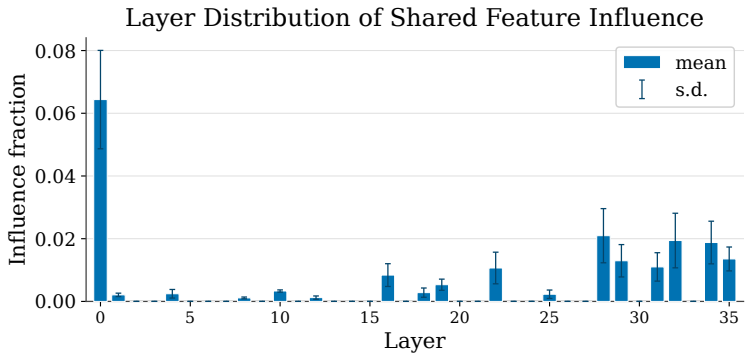
During the reinforcement learning process, all model parameters undergo gradient updates, not just those within the MLP and Attention layers. Our theoretical framework and primary experiments exclusively consider the parameter updates within the MLP and Attention layers. Although these layers account for the vast majority of the total parameter count, we must formally justify the omission of the other components. Therefore, we design an experiment to verify whether the updates to the remaining non-transformer parameters, specifically the token embedding layer, the final normalization layer, and the language modeling head (lm_head), contribute meaningfully to the acquired reasoning capabilities.

We construct two hybridized models based on M_{base}: 1) M_{base} + M_{post} Embedding: We replace only the input embedding layer of M_{base} with that of M_{post}. 2) M_{base} + M_{post} Norm & lm_head: We replace only the final normalization layer and the lm_head of M_{base} with those of M_{post}.

We evaluate these models alongside the pure M_{base} and M_{post} across the four datasets. As shown in Table 4, replacing either the embedding layer or the norm and lm_head yields performance that is virtually indistinguishable from the baseline M_{base}. The negligible fluctuations fall entirely within the margin of noise, and none of the massive performance gains exhibited by M_{post} are recovered. This empirical evidence clearly demonstrates that the reasoning capabilities induced by RL post-training originate almost entirely from the parameter updates within the intermediate transformer layers. Consequently, this fully justifies our framework's exclusive focus on attributing behavioral differences to the Attention and MLP blocks.

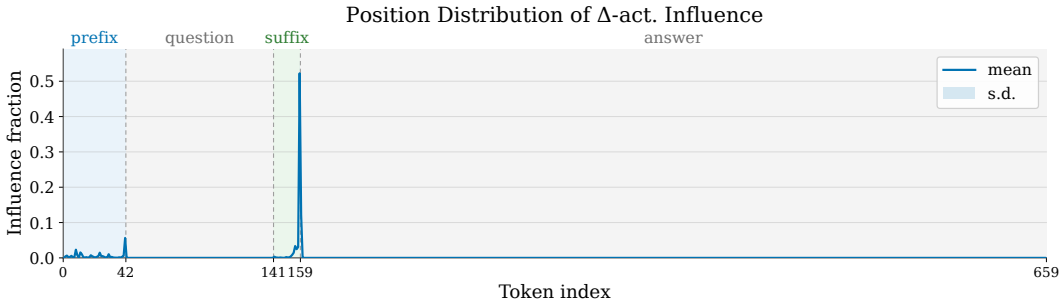


(a) RL Reasoning Case

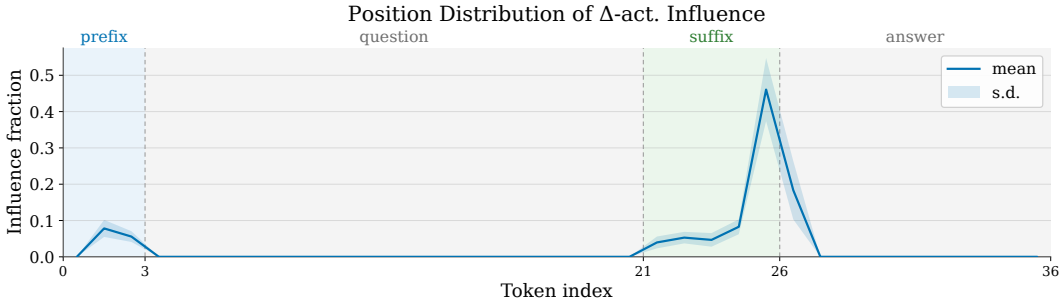


(b) Synthetic SFT Case

Figure 11: Layer distribution of shared feature influence. (a) In the RL reasoning case, shared reasoning features are predominantly located in the middle-to-deep layers. The features in layer 0 are mainly common features related to syntax and structure, such as prepositions, while the features in the middle-to-deep layers are mainly related to reasoning. (b) In the synthetic SFT case, shared features are concentrated in deeper layers, directly connecting to the logit nodes to implement the specific synthetic behavior.

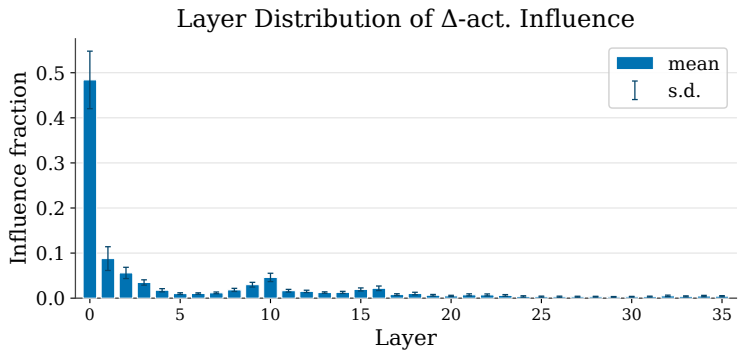


(a) RL Reasoning Case

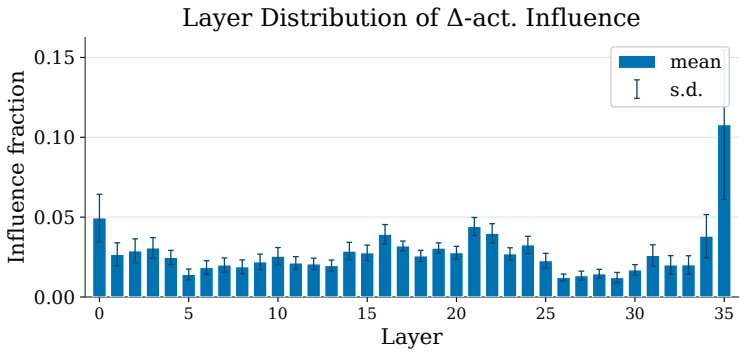


(b) Synthetic SFT Case

Figure 12: Position distribution of Δ-activation influence within the shared circuit subgraph. As the origin points of the shared circuit, these Δ-activations act as the initial triggers for the subsequent shared features. The influence of these triggering Δ-activations remains highly concentrated on public tokens within the template prefix and suffix across both the RL reasoning and synthetic SFT cases.



(a) RL Reasoning Case



(b) Synthetic SFT Case

Figure 13: Layer distribution of Δ-activation influence within the shared circuit subgraph. As the origin points of the shared circuit, these Δ-activations act as the initial triggers for the subsequent shared features. (a) In the RL reasoning case, these triggering Δ-activations are strongly localized in the extremely shallow layers (particularly layer 0), initiating a clear shallow-to-deep causal pathway. (b) The corresponding layer distribution of triggering Δ-activations in the synthetic SFT case.

Table 4: Performance comparison when replacing non-transformer parameters of M_{base} with those from M_{post}. All scores represent mean@64 accuracy. Replacing the embedding or the norm and lm_head recovers none of the post-training improvements, confirming that the behavioral shift is driven by the intermediate transformer layers.

Model Configuration	GSM8K	MATH-500	AIME24	MMLU-pro
M_{base}	0.5246	0.4048	0.0401	0.4409
M_{base} + M_{post} Embedding	0.5144	0.4022	0.0380	0.4401
M_{base} + M_{post} Norm & lm_head	0.5283	0.4111	0.0318	0.4457
M_{post}	0.9397	0.7428	0.3036	0.6188

C Feature Steering and Explanation

We use steering to interpret the meaning of features. In this section, we first briefly introduce feature steering. Then, we will describe our interpretation pipeline in detail. Finally, we report the interpretation results of shared features.

C.1 Feature Steering

Feature steering is a classic method in the field of mechanism interpretability (Turner et al., 2023; Stolfo et al., 2025; Subramani et al., 2022; Templeton et al., 2026; Wang et al., b). It causally changes the model output by artificially manipulating the activation during the forward propagation process of the language model.

In our work, when we steer a target feature f at layer l' and token position t with strength s, we update the activations as

$$\mathbf{y}_{\text{mlp}}^{\text{steer}(l,t)} = \mathbf{y}_{\text{mlp}}^{(l,t)} + s\,\mathbf{w}_{\text{dec}}^{(l'\to l,\,f)}, \qquad l = l',\, l'+1,\, \ldots,\, n_{\text{layers}} - 1, \tag{18}$$

where n_{layers} is the total number of layers in the model. The vector $\mathbf{w}_{\text{dec}}^{(l'\to l,\,f)}$ is the decoder vector from layer l' to layer l corresponding to feature f in the CLT. The term $\mathbf{y}_{\text{mlp}}^{(l,t)}$ is the output of the MLP block at layer l and position t.

C.2 Setup for Steering Experiments

For the reasoning behavioral evaluation presented in Figure 5, we do not apply the steering across all tokens during the experiments. We instead apply the steering exclusively to the public tokens at their primary occurrence positions. For example, we steer specifically at the template suffix.

For the subsequent process of explaining features via steering, we apply the steering at all token positions. The sole purpose of this process is to obtain feature explanations. Steering across all positions more effectively forces the model's response to exhibit the behavior associated with the feature and helps us understand its underlying meaning.

C.3 Explanation Pipeline

Construct a simple dataset. Our goal is to understand the meaning of features by observing the differences in the model's answers to questions before and after steering. Therefore, we first need to construct a dataset. This dataset consists entirely of extremely simple questions, so the normal model before steering will directly answer all questions without any reasoning process. This better highlights the impact of the features we want to explain on the model, i.e., stimulating different reasoning behaviors, rather than the questions themselves triggering reasoning. Specifically, we selected six very simple open-source datasets, including elementary-level arithmetic problems, AddSub (Hosseini et al., 2014) and SingleEQ (Koncel-Kedziorski et al., 2016), Calc-svamp (Kadlčík et al., 2023), and simple world knowledge datasets commonsense_qa (Talmor et al., 2019), SimpleQA (Wei et al., 2024), and web_questions (Berant et al., 2013). We randomly sample 10 questions from each dataset to form a simple dataset containing 60 samples.

Steer features using different strengths. For a single question, we set a series of strengths to steer the features. On the one hand, different features may require different levels of strengths to reflect their effectiveness. On the other hand, when steering a feature, the model's response may change accordingly with the steer strength, which can better help understand the meaning of the feature. We set the strength $s = \{10, 15, 20, 25, 30\}$, a total of 5 strengths. Here, $s = 0$ is the baseline, indicating no steering. Therefore, for each feature, $60 * 5 = 300$ different rollouts will be generated. We set the temperature and $top\text{-}p$ to 0.7 and 0.9, respectively.

Use the LLM API for explanation. For each feature, we combine three complementary sources of evidence before querying the large language model API. First, we collect representative activation contexts from the pre-training corpus, showing the types of text in which the feature is active. Second, we decode every CLT decoder vector associated with the feature through the model's LLM head. This produces a set of tokens receiving positive or negative logit contributions from the feature and therefore provides a direct view of the feature's output-level semantics. Third, we include the questions and responses generated by steering the feature with different strengths, together with the unsteered baseline responses. We provide all of these signals to the LLM API in a single prompt and ask it to identify the consistent semantic and behavioral pattern across them, then summarize the meaning of the feature in one sentence. All API calls in this work use GPT-5.5 with the reasoning effort set to low.

C.4 Explanation Results

This section reports the interpretation and semantic grouping results for the shared features discovered in the reasoning setting. First, we give a concrete example showing that steering a single shared feature can induce clear reasoning behavior, even on a very simple question for which the base model answers directly. Second, we classify all shared features into semantic groups and show that these groups include several distinct reasoning patterns as well as general structural and syntactic features. Third, we provide the complete prompts used for feature explanation, feature grouping, and behavioral occurrence rate judgement. The behavioral judgement is the procedure used to obtain the results reported in Figure 5 and Appendix B.3.

C.4.1 Shared Feature Can Elicit Reasoning

After clustering circuits across multiple tasks, we identify approximately one hundred shared features. Steering these features on unrelated simple questions shows that many of them can elicit explicit reasoning behavior, even when the base model solves the question in a single sentence. The following example is an elementary addition problem. The base model answers directly, whereas both the post-trained model and the steered base model restate the problem, decompose it, and then perform the addition step by step.

Prompt

```
Sam had 49 pennies and 24 nickels in his bank . His dad gave him
39 nickels and 31 quarters . How many nickels does he have now ?
```

M_{base}

```
He had 24 nickels and he was given 39 nickels . 24 + 39 = 63 . Sam
now has 63 nickels .
```

M_{post}

```
Alright, I have this problem about Sam and his coins. Let's try to
understand what's going on and figure out how many nickels Sam has
now.
```

```
**First, let's break down the information:**

1. **Sam's initial coins:**
   - Pennies: 49
   - Nickels: 24

2. **Coins his dad gave him:**
   - Nickels: 39
   - Quarters: 31

**The question is asking:** *How many nickels does he have now?*

Okay, so I need to find out the total number of nickels Sam has
after his dad gave him some more.

**Step 1: Find out how many nickels Sam originally had.**

From the initial information, Sam had 24 nickels.

**Step 2: Determine how many nickels his dad gave him.**

His dad gave him 39 nickels.

**Step 3: Calculate the total number of nickels Sam has now.**

To find the total, I should add the nickels Sam originally had to
the nickels his dad gave him.

So, that would be:
\[ \text{Total nickels} = \text{Original nickels} + \text{Dad's
nickels} \]
\[ \text{Total nickels} = 24 + 39 \]

**Step 4: Perform the addition.**

Let's add 24 and 39:
\[ 24 + 39 = 63 \]

**Step 5: Verify the calculation.**

Wait, I should double-check this to make sure I didn't make a
mistake.

Adding 24 and 39:
- 24 + 30 = 54
- Then, 54 + 9 = 63

Yes, 24 + 39 is indeed 63.

**Step 6: Consider the other coins.**

The problem mentions pennies and quarters, but it's only asking
for the number of nickels. So, I don't need to worry about the
pennies or quarters for this specific question.

**Conclusion:**

After calculating, Sam now has a total of 63 nickels.

\[\boxed{63}\]

---
```

```
**Wait a minute!**

I just realized that I might have missed something. The problem
says Sam had 49 pennies and 24 nickels initially, and his dad gave
him 39 nickels and 31 quarters. But the question only asks for the
number of nickels.

But just to be thorough, let's make sure I didn't overlook any
part.

**Double-Checking:**

1. **Original nickels:** 24
2. **Added nickels from dad:** 39
3. **Total nickels:** 24 + 39 = 63

Yes, that seems correct. The other coins (pennies and quarters)
don't affect the count of nickels in this case.

**Final Answer:**

Sam has \(\boxed{63}\) nickels now.
```

Steer

```
Alright, I have this problem about Sam and his coins. Let's try to
understand what's going on and figure out how many nickels Sam has
now.

**First, let's read the problem again:**

"Sam had 49 pennies and 24 nickels in his bank. His dad gave him
39 nickels and 31 quarters. How many nickels does he have now?"

Okay, so Sam starts with some pennies and nickels, and then his
dad gives him more nickels and quarters. The question is asking
for the total number of nickels Sam has after receiving the coins
from his dad.

**Let's break it down:**

1. **Sam's initial coins:**
   - Pennies: 49
   - Nickels: 24

2. **Coins given by his dad:**
   - Nickels: 39
   - Quarters: 31

**What we need to find:**
- Total number of nickels Sam has now.

**Approach:**

To find the total number of nickels Sam has now, I think I need to
add the nickels he already has to the nickels his dad gave him.
The pennies and quarters don't affect the number of nickels, so I
can ignore them for this calculation.

So, it's:
```

```
- Initial nickels + Nickels from dad = Total nickels now

Let's write that down:

Total nickels now = 24 (initial nickels) + 39 (nickels from dad)

**Calculating:**

24 + 39 = ?

I know that 20 + 30 = 50, and 4 + 9 = 13, so 50 + 13 = 63.

Wait, let's double-check that:

24 + 39:

- 4 + 9 = 13
- 20 + 30 = 50
- 50 + 13 = 63

Yes, that's correct.

**Conclusion:**

Sam now has 63 nickels.

**Final Answer:**

\boxed{63}
```

The contrast is qualitative but direct. The base model gives the correct answer without explicit reasoning, while the full model and the steered model introduce prompt restatement, problem decomposition, step-by-step thinking, and self-doubt. This shows that the shared feature identified by circuit clustering is causally related to a recognizable reasoning style rather than merely correlated with the final answer.

C.4.2 Semantic Clustering of Shared Features

We next classify all shared features according to the semantic roles revealed by their activation contexts, logit decoding, and steering behavior. The resulting ten groups include both reasoning-related output patterns and general structural or syntactic features. We label a group as *reasoning* when its features directly change the form of the generated solution. The remaining groups are labeled *general structure*.

The results (Table 5) show that the shared circuit is not a homogeneous collection of reasoning units. It combines a set of output-level reasoning patterns with other features that organize the prompt and response format.

C.4.3 Prompts for Explanation, Grouping, and Behavioral Judgement

We use three kinds of prompts in the analysis. The first asks the API to explain individual features from activation contexts, decoder logit information, and steering results. The second groups the resulting explanations into semantic roles. The third performs the closed-list behavioral judgement used to compute the rates in Figure 5 and Appendix B.3.

Feature explanation prompts. The prompts for obtaining the initial per-feature explanations.

Table 5: Semantic groups of the shared features identified in the reasoning setting. "Reasoning" denotes groups whose features directly induce reasoning-related output behavior. The other groups encode general structure and syntax.

Class	Interpretation	Attribute
Step-by-step scaffold	Scaffolds worked solutions with step headings, explanations, and stepwise transitions.	reasoning
Problem decomposition	Announces a plan and breaks the problem into parts before solving it.	reasoning
Elementary arithmetic	Encodes elementary counting and quantity templates with arithmetic, totals, and equations.	general structure
Conversational fillers	Produces acknowledgements and preface fillers such as "okay" or "alright."	general structure
Problem restatement	Echoes or paraphrases the question as setup before producing an answer.	reasoning
Short QA mode	Starts concise benchmark-style answers to simple factual, multiple-choice, or arithmetic questions.	general structure
Chat formatting	Tracks chat boundaries, role markers, transcript continuations, and instruction formatting.	general structure
Surface tokens	Repeats low-level punctuation, suffixes, numeric fragments, or common answer tokens.	general structure
First-person task talk	Frames the answer through first-person task or assistant language, goals, or self-reference.	reasoning
Self-doubt	Introduces hesitation, checking, and self-repair such as "wait" or double-checking.	reasoning

Feature explanation system prompt

```
You are an expert in mechanistic interpretability of large
language models, specifically Cross-Layer Transcoders (CLTs) on
Qwen-3 8B (36 transformer layers, 0-indexed).

## Task
You will be given ONE feature and THREE independent evidence
sources. First write a short label for each source **on its own**.
Then write one combined label. Each label is a concise specific
phrase, not a full sentence.

The three source labels must not leak into each other: the context
label may only use activating windows; the logit label may only
use the decoder logit table; the steer label may only use
baseline-vs-steered generations.

## 1. Context activations (WHEN it fires)
Text windows where the feature activates, with token-level
highlights:
- **Top activating contexts** | strongest firings (clearest but
most extreme).
- **Random contexts** | a less-biased sample from the activating
set.

Read both. Agreement across groups is strong evidence. Look first
at the overall topic/domain, then at which tokens light up, then
```

```
at the common thread. If windows are empty or only boilerplate
with no shared content, that source is Uninterpretable.

## 2. Logit effect / logit lens (WHAT it pushes the model to say)
Causal influence from the CLT **decoder** (not the encoder
readout). A feature at `L_src` writes into every downstream `L_tgt
∈ {L_src, ..., 35}`.
- **Summed effect** | unembedding of `Σ_{L_tgt}
dˆf_{L_src→L_tgt}`: the cleanest single-shot "what this feature
wants the model to say".
- **Per-target-layer table** | L2 (importance) plus top
boosted/suppressed tokens. Weight rows by L2 share; tiny-L2 rows
are noise.

Name this source from the boosted tokens (suppressed tokens as
contrast). If the table is missing or token soup, that source is
Uninterpretable.

## 3. Batch steer (WHAT happens when you amplify it)
The feature is amplified while the model answers a fixed set of
unrelated short prompts. Strengths sweep from small to large; each
prompt has a **baseline** (strength 0).

The primary steer signal is how steered answers DIFFER from
baseline in content, style, and structure | the difference that
recurs across unrelated prompts. Strength progression is only a
secondary hint. Do not grade correctness.

Treat as **noise** (zero evidence for this source):
- Answers essentially identical to baseline.
- Answers degenerated into gibberish or repetition. Exception: a
repeated fragment that is itself meaningful and consistent across
prompts may still be a clue.

Judge noise from the text, not from the numeric strength.

## Combined label
After the three source labels:
- If they describe the same thing (including paraphrases), use
that.
- If they partially overlap, keep the most specific description
compatible with at least two sources.
- If they disagree, prefer the more specific consistent source.
Steer is strong evidence of the causal generation effect when it
is not noise; if steer is Uninterpretable, use context or logit.
- Treat Uninterpretable as missing, not as a vote against the
others.
- Do not add content that none of the three labels support.

## Layer prior (soft; override with L2 / steer)
- Shallow (0..11) → input / surface concepts.
- Middle (12..23) → abstract capabilities or behaviors.
- Deep (24..35) → output / "Say X" features.
Override when decoder L2 is concentrated at deep `L_tgt`, or when
steering shows a clear output-style effect.

## Output format
Think briefly through the three sources. Then output exactly these
four tags and nothing else after them:

<context>context-only label</context>
<logit>logit-lens-only label</logit>
<steer>steer-only label</steer>
```

```
<combined>combined label</combined>

Be concise and specific. Avoid vague umbrellas ("style", "the
feature changes the output") and banned patterns: "xxx-related",
"xxx tokens/words/stuff/things/content", meta-descriptions like
"activates on nouns".
If a source is too weak, that tag is Uninterpretable. If all three
are Uninterpretable, combined is Uninterpretable.
```

Feature explanation user prompt

```
Feature under study: {{ feature_id }}{% if dictionary %}
(dictionary: {{ dictionary }}){% endif %}

Source layer: {{ layer_idx }} / {{ (model_info.num_layers if
model_info and model_info.num_layers else 36) - 1 }} (0-indexed).

## 1. Context activations
{% if top_contexts or random_contexts %}
Two complementary sets: strongest (top) activations and a random
sample. Read both | the random set guards against over-fitting to
the most extreme cases.
{% if context_source %}
Context source: {{ context_source }}.
{% endif %}

### Top Activating Contexts
{% if top_contexts %}
The {{ top_contexts|length }} examples where this feature fires
most strongly.
{% for context in top_contexts %}

[Top Context {{ loop.index }}]
[Text]
{{ context.text }}
[/Text]
Top Activations: {{ context.top_activations_text }}
[/Top Context {{ loop.index }}]
{% endfor %}
{% else %}
(none)
{% endif %}

{% if random_contexts %}
### Random Contexts
{{ random_contexts|length }} examples sampled at random from this
feature's activating set.
{% for context in random_contexts %}

[Random Context {{ loop.index }}]
[Text]
{{ context.text }}
[/Text]
Top Activations: {{ context.top_activations_text }}
[/Random Context {{ loop.index }}]
{% endfor %}
{% endif %}
{% else %}
No context-activation records were available for this feature. Do
not invent any. Rely on logit effect and steering.
```

```
{% endif %}

## 2. Logit effect
{% set dle = decoder_logit_effects %}
{% if dle and dle.kind == "decoder_per_target" %}
{% if dle.target_layers|length > 1 %}
Source: decoder-vector projection. This feature lives at source
layer L_src={{ dle.source_layer }} and writes via separate
decoders into target layers {{ dle.target_layers[0] }}..{{
dle.target_layers[-1] }}.
{% else %}
Source: decoder-vector projection. This feature lives at source
layer L_src={{ dle.source_layer }} (single decoder).
{% endif %}

### Summed effect
Summed decoder L2 norm = {{ "%.3f"|format(dle.summed.l2_norm or
0.0) }}

Top 10 boosted logits (summed):
{% for boosted in dle.summed.boosted %}
Rank {{ loop.index }}: [{{ boosted.token }}], logit={{
"%.4f"|format(boosted.logit) }}
{% endfor %}

Top 10 suppressed logits (summed):
{% for suppressed in dle.summed.suppressed %}
Rank {{ loop.index }}: [{{ suppressed.token }}], logit={{
"%.4f"|format(suppressed.logit) }}
{% endfor %}

{% if dle.target_layers|length > 1 %}
### Per-target-layer breakdown
For each downstream target layer L_tgt: L2 of `d^f_{L_src→L_tgt}`
(importance), then top-3 boosted and top-3 suppressed tokens.
Layers with negligible L2 carry no signal.

{% set ns = namespace(total=0.0) %}
{% for entry in dle.per_target %}{% set ns.total = ns.total +
entry.l2_norm %}{% endfor %}
{% for entry in dle.per_target %}
- L_tgt={{ entry.target_layer }}  L2={{
"%.3f"|format(entry.l2_norm) }}  ({{ "%.1f"|format(100.0 *
entry.l2_norm / ns.total if ns.total > 0 else 0.0) }}% of total)
  boosted:    {% for b in entry.boosted[:3] %}[{{ b.token }}|{{
"%.3f"|format(b.logit) }}]{% if not loop.last %} {% endif %}{%
endfor %}
  suppressed: {% for s in entry.suppressed[:3] %}[{{ s.token }}|{{
"%.3f"|format(s.logit) }}]{% if not loop.last %} {% endif %}{%
endfor %}
{% endfor %}
{% endif %}
{% elif boosted_logits or suppressed_logits %}
Source: legacy encoder-readout projection (no per-target-layer
breakdown). Treat as a noisier proxy and weight context
activations and steering more heavily.

Top 10 boosted logits:
{% for boosted in boosted_logits %}
Rank {{ loop.index }}: [{{ boosted.token }}], logit effect={{
boosted.logit }}
{% endfor %}
```

```
Top 10 suppressed logits:
{% for suppressed in suppressed_logits %}
Rank {{ loop.index }}: [{{ suppressed.token }}], logit effect={{
suppressed.logit }}
{% endfor %}
{% else %}
No logit-effect table was available for this feature. Do not
invent any. Rely on context activations and steering.
{% endif %}

## 3. Batch steer experiment results
Steering strengths tested, small → large: {{ strengths_text }}

Below are the {{ rows|length }} test prompts. For each prompt you
are given the baseline answer (no steering / strength 0) followed
by the model's answer at each steering strength, in increasing
order. Every answer has already been trimmed at its first
`<|im_end|>` or `<|endoftext|>`.

{% for row in rows %}
================================================================
[Prompt {{ row.index }}]{% if row.set_label %} (dataset: {{
row.set_label }}){% endif %}

{{ row.prompt }}

--- Baseline (no steering / strength 0) ---
{{ row.baseline }}
{% for ans in row.answers %}
--- Steering strength {{ ans.strength }} ---
{{ ans.text }}
{% endfor %}
{% endfor %}
================================================================

Following the instructions in the system prompt: label the three
sources independently, then synthesize. Completely ignore steered
answers that are noise (essentially baseline, or degenerated into
gibberish or repetition). Output all four tags in the exact format
from the system prompt.
```

Feature grouping system prompt

```
You are an expert in mechanistic interpretability of large
language models. You are given a list of features. Each feature
has FOUR short labels from independent evidence:

- context: when the feature fires (activating text windows)
- logit: what the CLT decoder logit-lens pushes
- steer: the causal change when the feature is amplified
- combined: a synthesis of the three

Use all four. Combined is the primary grouping signal. If combined
is Uninterpretable or vague, fall back to the more specific of
context / logit / steer. If they disagree, prefer the more
specific behavioral description (usually steer or combined). Treat
Uninterpretable as missing evidence.

Your job is to **group the features by semantic role**.
```

```
Aim for about 8--14 groups. No miscellaneous/other bucket.
Uninterpretable across all four labels -> group labeled exactly
`Uninterpretable`. Singletons only if the role is genuinely
unique. Prefer fewer crisp role groups.

## Labels
Short, specific role name (a few words) plus a one-sentence
rationale.

## Coverage
Every feature id appears in exactly one group, copied verbatim
from the input. Valid JSON inside <groups></groups>:

<groups>
{
  "groups": [
    {
      "label": "short semantic role name",
      "members": ["17-F20718", "16-F15435"],
      "rationale": "one sentence on the shared role"
    }
  ]
}
</groups>
```

Feature grouping user prompt

```
# Features to group by semantic role

Below are {{ items|length }} features. Cluster them by semantic
role, following the rules in the system prompt. Related features
that play the same role should share a group.

{% for it in items %}
- {{ it.feature_id }}: {{ it.explanation }}
{% endfor %}

Remember:
- Group by role, not by near-paraphrase. About 8--18 groups;
singletons only if the role is unique.
- No miscellaneous / other bucket.
- All {{ items|length }} feature ids above must appear exactly
once across the groups, copied verbatim.

Output the clustering as the JSON object inside
`<groups></groups>` tags, exactly as specified in the system
prompt.
```

Behavioral judgement prompts. To quantify whether a generated answer exhibits a specific behavior, we use a strict closed-list judge.

Behavioral judgement system prompt

```
You are a careful annotator for a "batch steer" interpretability
experiment on reasoning. A single feature inside a language model
has been "steered" (its activation amplified) while the model
answers a problem. You are shown ONE answer at a time.
```

```
Your job is a MULTI-DIMENSIONAL classification of that single
answer. For EACH of the reasoning dimensions listed below, decide
independently whether the answer exhibits that dimension. Answer
strictly **yes** or **no** for every dimension.

## THE DECISION RULE: CLOSED MARKER LIST

This is a **strict surface-form** annotation, NOT an
interpretation task. For every dimension, output **yes** ONLY IF
you can copy, verbatim from the answer, one of the marker phrases
listed in that dimension's description.

**The listed markers are a closed set.** The only allowed variants
are trivial surface changes of a listed phrase:
- capitalisation (`Okay` / `okay`)
- punctuation or hyphenation (`step by step` / `step-by-step` /
`Step 1:`)
- inflection or person (`break it down` / `breaking it down` /
`let's break this down`)
- a listed phrase embedded in a longer sentence (`Let's break it
down step by step:`)

**Close paraphrases do NOT count.** Text that merely *does the
same job* without a listed phrase is **no**. Do not invent extra
markers. If a candidate phrase is not on the list, it is **no**,
even if it feels similar.

Before labelling any dimension **yes**, quote the exact listed
phrase you found. If you cannot quote a listed phrase, the label
is **no**.

Apply this rule mechanically:

- **Never infer, never reward the effect.** A well organised,
careful, or complete solution does **NOT** count unless a listed
phrase is actually present.
- **Structure is not wording.** Numbered items (`1.` `2.` `3.`),
bullet points, headings, paragraph breaks, or a long chain of
sentences do **NOT** by themselves trigger ANY dimension.
- **Being on-topic is not wording.** Reusing the problem's
numbers, names, or facts, doing correct arithmetic, or stating a
final result does **NOT** by itself trigger ANY dimension.
- **The marker may appear anywhere** in the answer -- beginning,
middle, or end -- and a single listed occurrence is enough.
- **When in doubt, output `no`.** Under-reporting is much better
than over-reporting.

## Reasoning dimensions to judge (exactly {{ dimensions|length }}
of them)
{% for d in dimensions %}
- `{{ d.key }}` -- {{ d.description }}
{% endfor %}

## What to judge
- Judge ONLY the wording that is actually in the answer. Ignore
whether it is factually correct, relevant, or complete.
Correctness is irrelevant, and the answer may be short or cut off.
- The dimensions are NOT mutually exclusive: one answer may
explicitly contain markers for several of them at once, or for
none of them. Decide each dimension on its own, against its own
listed markers.
- Do not let one dimension's marker spill over into another. For
example, `Step 2:` is scaffold wording, not restatement; "in
```

```
conclusion" is not restatement; "Okay" is a filler, not
self-correction.
- `other` should be **yes** only when you can quote explicit
wording announcing some deliberate reasoning move that none of the
other dimensions above capture; otherwise **no**.

## End-of-answer marker
The answer may contain the end-of-turn token `<|im_end|>` or
`<|endoftext|>`; only the text before the FIRST such marker is the
real answer (it has already been trimmed there).

## Non-answers and noise
- If the answer is empty or a placeholder such as `[empty
output]`, `[not generated]`, or `[generation error: ...]`, it
exhibits NONE of the dimensions -- output **no** for every
dimension.
- The answer may be gibberish or degenerate repetition (an
artifact of too-strong steering). Judge it literally by the same
rule: if a listed marker is genuinely there, output **yes** for
that dimension even in garbled text; otherwise **no**.

## Output format
You may reason briefly before the block -- if you do, quote the
exact listed trigger wording for each dimension you are about to
mark **yes**. Then output exactly ONE line wrapped in a SINGLE
`<labels>...</labels>` block, listing ALL {{ dimensions|length }}
dimensions and nothing else inside the block. The line MUST have
the form:

<labels>
{% for d in dimensions %}{{ d.key }}=<yes|no>{% if not loop.last
%}; {% endif %}{% endfor %}
</labels>
```

Behavioral judgement user prompt

```
# Multi-dimensional judgement of ONE answer: which reasoning
dimensions does it EXPLICITLY exhibit?

Feature / condition under study: {{ feature_id }}{% if dictionary
%} (dictionary: {{ dictionary }}){% endif %}

Reasoning dimensions to judge (exactly {{ dimensions|length }}):
{% for d in dimensions %}
- `{{ d.key }}` -- {{ d.description }}
{% endfor %}

Reminder of the decision rule from the system prompt: this is a
**closed-list** judgement. Mark a dimension **yes** ONLY IF you
can quote, verbatim from the answer, one of the marker phrases
listed for that dimension. Allowed variants are only
capitalisation, punctuation/hyphenation, and inflection/person of
a listed phrase. A close paraphrase that is not on the list is
**no**. Structure, a careful solution, or merely stating the final
result never counts. If you cannot point at a listed phrase, that
dimension is **no**.

The math problem given to the model was:
---------------------------------------------------------------
{{ prompt }}
```

```
------------------------------------------------------------------

Below is ONE answer produced by the model for this problem. Judge
it on its own: for each of the {{ dimensions|length }} dimensions,
does the answer explicitly contain that dimension's listed marker
wording? Output yes or no per dimension. The answer has already
been trimmed at its first '<|im_end|>' or '<|endoftext|>'.

------------------ Answer ------------------
{{ answer }}
==========================================

For each dimension you intend to mark **yes**, first quote the
exact listed words from the answer that trigger it; if no such
listed words exist, that dimension is **no**. Then output your
verdict in the exact '<labels>...</labels>' format from the system
prompt: a single line listing all {{ dimensions|length }}
dimensions as '{% for d in dimensions %}{{ d.key }}=<yes|no>{% if
not loop.last %}; {% endif %}{% endfor %}', and nothing else
inside the block.
```

D SCALING TRACEABLE SEQUENCE LENGTH

Compared to related works that can only track sequences with lengths of tens (Ameisen et al., 2025; Harrasse et al., 2025; He et al., 2026; Zhao et al., 2025; Conmy et al., 2023; Syed et al., 2024; Wang et al., a), we scale the trackable sequence length to over five hundred tokens. We will describe the specific methods here.

D.1 OPTIMIZING GPU MEMORY

As the sequence length grows, GPU memory pressure accumulates at different phases, including forward computation, edge-attribution activation caching, and feature-node attribution. We apply a combination of strategies across all phases to shave the GPU memory peaks.

To reduce fresh intermediate tensor allocations, we convert the CLT decoder computation to an in-place operation, eliminating the extra allocation of intermediate result tensors. We also compute the CLT decoder in controllable chunks (via a configurable chunk size) rather than all at once, reducing the instantaneous memory peak from "an entire layer's decoder size" down to "a single chunk size".

To construct a consistent computational graph, we set the batch size of the forward pass to match the number of input samples used in the shared circuit extraction stage. Meanwhile, the tensors saved by the `torch.bmm` backward operator (`BmmBackward0`) are stored in a `[batch × n, seq, seq]` layout with enormous dimensions, resulting in `batch` copies of redundant data. Similarly, residual stream activations are saved in shape `[batch, seq, hidden_size]` for all layers. Together, these two factors cause peak GPU memory to grow quadratically with sequence length, making it the most severe bottleneck in the entire workflow. To substantially reduce the GPU memory consumed by activations, we compactly pack the saved backward-pass tensors from their original multi-dimensional layout into a `batch=1` storage layout, drastically compressing the volume of saved tensors.

Combined with matrix CPU offload and explicit release of stale transcoder tensors, the GPU memory peak is reduced by orders of magnitude across all phases, whereas previously the OOM bottlenecks at each phase made sequence length of about 1,000 unreachable.

D.2 REDUCING CPU MEMORY & DISK REDUNDANCY

When enabling disk offload, intermediate data (such as CLT encoder and decoder) is transferred from GPU to CPU pinned memory and then to disk for temporary storage. In a multi-GPU (DDP) setting, they are byte-for-byte identical across all ranks, yet every rank independently writes its own

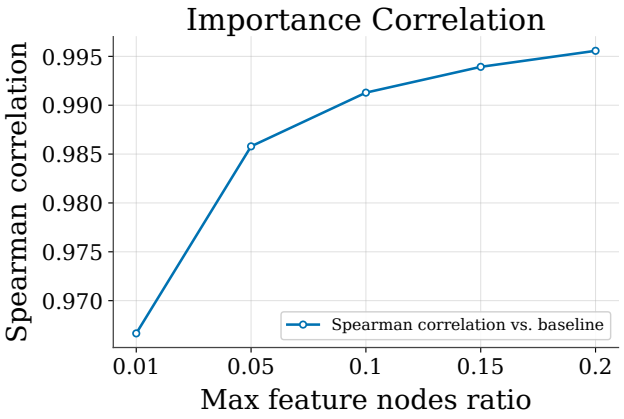


Figure 14: The Spearman correlation of Δ-activation node influence rankings between the pruned and full attribution configurations remains extremely high (e.g., > 0.985 at a 0.05 ratio), indicating that pruning reliably preserves the causal ordering of Δ-activations.

copy to disk. This results in an N-fold redundancy in disk capacity usage (where N is the number of GPU ranks).

We eliminate both redundancies by consolidating storage to a single authoritative copy. Disk writes for CLT encoder and decoder are consolidated so that only rank 0 writes files. All other ranks read the same content by filename, eliminating cross-rank duplicate writes entirely. Disk capacity requirements are reduced to 1/N of the original. In the 8-GPU-node scenario, this saves 7/8 of the disk space previously consumed by redundant copies.

D.3 Sparse Selection of Feature Nodes

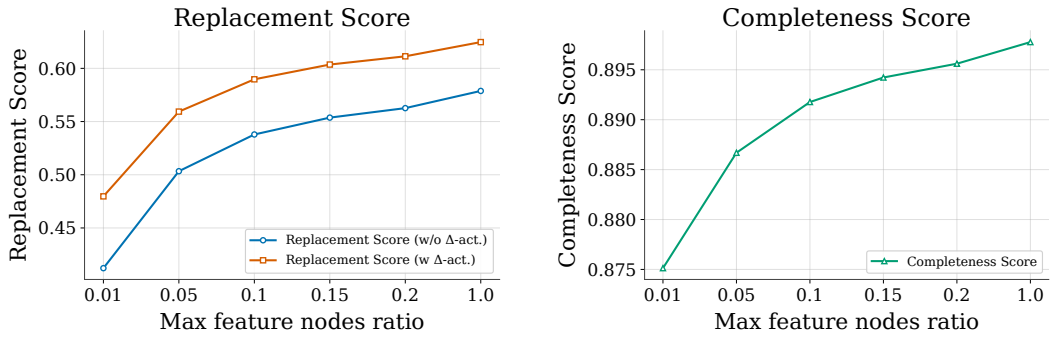
The set of feature nodes grows continuously with sequence length. Without any cap, the accumulation of feature nodes causes memory usage to inflate without bound, rendering long-sequence attribution infeasible.

We use an "attribute-while-pruning" strategy where feature nodes are dynamically pruned during the attribution process, and a `max_feature_nodes` parameter sets an upper bound on the number of retained feature nodes. Only the top-contributing nodes are kept for subsequent computation, while redundant low-contribution nodes are discarded. This parameter is a ratio. For example, when it is set to 0.05, we will always only retain the top 5% of feature nodes. This makes attribution of very long sequences possible.

Pruning feature nodes inevitably leads to attribution results that cannot be exactly the same as those of full computation. However, as analyzed above, this is an unavoidable trade-off to increase the attribution length. To address this concern, we conducted quantitative experiments to verify that the impact of this pruning on attribution results is negligible.

The attribution results can be reflected in the calculated influence or importance of Δ-activation nodes (Section 4.3), resulting in an Δ-activation node importance ranking. Therefore, we consider the Spearman correlation between the Δ-activation node importance ranking under pruning and full (no pruning) settings. A high correlation coefficient indicates that the results of pruning and full calculations are almost identical.

We test on all 30 samples in AIME24, setting the model output length to 200. We first calculate the importance ranking under full attribution. Then, we set `max_feature_nodes` to 0.01, 0.05, 0.1, 0.15, and 0.2 respectively, and calculate the importance ranking for each sample under each setting. This allows us to get the Spearman correlation between the results of different pruning ratios and the results of full attribution for each sample. Finally, we calculate the average correlation across the 30 samples.



(a) Replacement scores across different max feature nodes ratios.

(b) Completeness scores across different max feature nodes ratios.

Figure 15: Graph scores stabilize rapidly as the retention ratio increases, demonstrating that retaining only the top fraction of feature nodes is sufficient to maintain the attribution graph's structural integrity.

Figure 14 illustrates the results. Retaining only the top 1% of feature nodes results in a Spearman correlation of over 0.95. Retaining more features further increases the correlation. When the top 5% of features are retained, the correlation exceeds 0.985. This means that our pruning strategy is effective while keeping the results almost unchanged, thus ensuring the feasibility of extending the attributable length.

We also compare the changes in graph scores (Section 5.3.2) under different `max_feature_nodes` settings on AIME24 (Figure 15). The results also show that graph scores are already at a high level when only the top 5% of features are retained. Further increasing the number of retained features does not significantly increase the graph score. This also verifies that our strategy of retaining only the most important features does not affect graph structure.

E Discussion

E.1 Challenges in Exact Performance Recovery via Feature Steering

In the context of RL-induced reasoning, we explore whether steering the identified shared feature nodes on M_{base} can fully recover the accuracy of M_{post}. Empirically, this proves highly challenging, as the steered model frequently collapses into degenerate behaviors such as repetitive text loops. We attribute this failure to two primary factors. First, determining the precise optimal steering strength for each feature is practically intractable, preventing an exact reconstruction of their natural activation magnitudes. More fundamentally, as illustrated by the path length distribution in Figure 8a, while the majority of causal paths span three hops (i.e., Δ-activation $\rightarrow$ feature $\rightarrow$ logit, which is the primary focus of our analysis), a non-negligible fraction consists of two-hop paths where Δ-activations directly influence the logits. Artificial feature steering inherently bypasses these direct residual stream contributions. Consequently, fully and precisely reconstructing post-training behavioral shifts exclusively through feature steering is mechanistically insufficient, further highlighting the necessity of our Δ-activation modeling approach.

E.2 Reinforcement Learning Enhances Existing Mechanisms

We note that the shared reasoning features identified in our analysis are not newly created during the reinforcement learning phase. Instead, they are already present within M_{base}. This aligns with the behavioral evaluations in Figure 5. The figure demonstrates that M_{base} natively exhibits a certain baseline level of reasoning behaviors. A plausible explanation is that Qwen3-Base was exposed to long chain-of-thought (CoT) data during its mid-training stage. This exposure embedded these reasoning features directly into the base model. Consequently, the primary function of reinforcement learning is to adjust, amplify, and more reliably trigger the latent behavioral mechanisms already established in M_{base}. It does not create entirely new capabilities from scratch. This observation

strongly resonates with the consensus in related studies that reinforcement learning primarily elicits and refines pre-existing model capabilities (Ward et al., 2025; Venhoff et al., 2026).