

Internal states before *wait* modulate reasoning patterns

Dmitrii Troitskii ^{*1,2}, Koyena Pal ^{*2}, Chris Wendler², Callum Stuart McDougall³,

¹Independent, ²Northeastern University, ³Google Deepmind

Correspondence: troitskii.d@northeastern.edu, pal.k@northeastern.edu

Abstract

Prior work has shown that a significant driver of performance in reasoning models is their ability to reason and self-correct. A distinctive marker in these reasoning traces is the token *wait*, which often signals reasoning behavior such as backtracking. Despite being such a complex behavior, little is understood of exactly why models do or do not decide to reason in this particular manner, which limits our understanding of what makes a reasoning model so effective. In this work, we address the question whether model’s latents preceding *wait* tokens contain relevant information for modulating the subsequent reasoning process. We train crosscoders at multiple layers of DeepSeek-R1-Distill-Llama-8B and its base version, and introduce a latent attribution technique in the crosscoder setting. We locate a small set of features relevant for promoting/suppressing *wait* tokens’ probabilities. Finally, through a targeted series of experiments analyzing max-activating examples and causal interventions, we show that many of our identified features indeed are relevant for the reasoning process and give rise to different types of reasoning patterns such as *restarting from the beginning*, *recalling prior knowledge*, *expressing uncertainty*, and *double-checking*.

1 Introduction

A growing class of language models known as reasoning models, for instance, DeepSeek-R1 (DeepSeek-AI et al., 2025), OpenAI o1 series (OpenAI et al., 2024), and others (Team, 2024; Kavukcuoglu, 2025; Anthropic, 2025), produce detailed internal reasoning chains before generating responses. While they showcase sophisticated reasoning capabilities, our understanding of their internal reasoning mechanisms remains limited. A recent work by Venhoff et al. (2025) categorized DeepSeek R1’s reasoning process into behavioral patterns such as example testing, uncertainty estimation, and backtracking to show how models

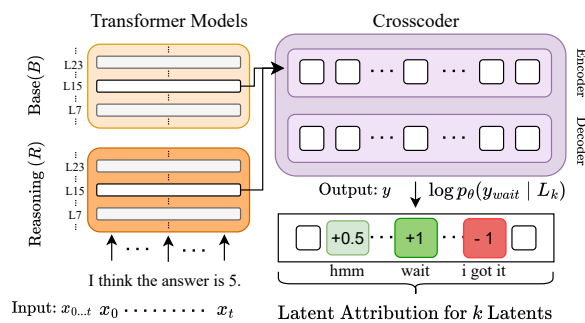


Figure 1: **We determine reasoning related crosscoder features** via latent attribution with respect to *wait* tokens’ logits.

solve reasoning tasks. This leads us to ask the following question: when and how do these reasoning patterns form?

This is a broad question that encompasses both the training dynamics, such as when and how reasoning circuits form, and the emergence of reasoning patterns and behaviors during inference. In this work, we focus on the latter, specifically within the distilled R1 model DeepSeek-R1-Distill-Llama-8B.¹ From observation, in this model the token *wait* is strongly associated with the self-reflection of the model (Baek and Tegmark, 2025), which could relate to reasoning patterns such as backtracking, deduction, and uncertainty estimation (Venhoff et al., 2025). Hence, we hypothesize that the features that modulate this token prediction can be useful in understanding the shifts in the reasoning patterns of the reasoning model.

Contributions. Motivated by this observation, we train Sparse Crosscoders (Lindsey et al., 2024) a recent mechanistic interpretability technique that allows to learn features within a paired base and finetuned model in an unsupervised way and can be used to perform model diffing.

¹Although we work with the distilled model, we will refer to it as R1 for succinctness.

We leverage our crosscoders to discover latent features that modulate reasoning behaviors. To reduce the search space, our case study focuses on features promoting and suppressing *wait* tokens that we hypothesize to play a key role in R1’s reasoning. To this end, we extend attribution patching (Kramár et al., 2024) to the crosscoder setting. Our crosscoder latent attribution method allows us to efficiently score all of our 32,768 features regarding their contribution to the logits of *wait* tokens. As a result, we obtain two short lists of relevant features, the 50 top features with the largest contributions to *wait* logits and the 50 bottom ones, that we can effectively investigate.

By utilizing the crosscoders classification of the features in to base, shared, and reasoning-finetuned we found that surprisingly many reasoning related features fall inside of the bottom bucket, suggesting that both amplifying as well as suppressing *wait* are of key importance to reasoning. By studying max-activating examples we observe a similar pattern. While top features mostly occur in the frequently reported backtracking and self-verification behaviors, we again notice that bottom features stand out in terms of the diversity of the reasoning behaviors within which they occur (e.g., starting again from an earlier step, expressing uncertainty, wrapping up, etc.). Finally, we perform a causal analysis via activation steering, in which we observe effects matching our interpretations based on the max-activating examples for many features, in particular the ones mentioned in parenthesis before. Our work showcases how to reduce the combinatorial complexities of crosscoder feature-based interpretability with the help of our novel crosscoder latent feature attribution technique. We thereby open the door for future researchers to explore such findings in other chat and reasoning models.

2 Methods

To discover features, we create reasoning data instances (Section 2.2) and train Sparse Crosscoders (Section 2.1). We then apply Latent Attribution patching (Section 2.3) to filter features that strongly modulate *wait* tokens. Finally, we steer these features (Section 2.4) to evaluate their impact on the model’s reasoning behavior, helping us determine how actionable they are.

2.1 Sparse Crosscoder

Sparse Autoencoder (SAE) decomposes model activations into a sparse set of linear feature directions. Sparse Crosscoder extends this by decomposing activations from different layers, models, or context positions. To explore feature correspondence between two models, we train L1 Sparse Crosscoders (Lindsey et al., 2024) between DeepSeek-R1-Distill-Llama-8B and its base version. We train three Crosscoders at 25%, 50%, and 75% layers depth, which correspond to layers 7, 15, and 23 respectively, using an expansion factor of 8 to learn 32768 features. For training details, refer to Appendix A.

2.2 Data: Reasoning Instances with *wait*

Venhoff et al. (2025) released rollouts on reasoning problems from DeepSeek-R1-Distill-Llama-8B that have reasoning traces across ten categories that include topics like mathematical logic, creative problem solving, and scientific reasoning. 193 out of 500 samples have either of the following *wait* tokens — “Wait”, “ Wait”, “ wait”, “wait”. For every *wait* in every sample that has a preceding reasoning sequence, we create a subsequence from the beginning of the sentence to the token right before *wait*. This filters for *wait* tokens related to reasoning behavior. This generates 350 subsequences.

2.3 Latent Attribution

As shown in Figure 1, latent attribution observes and computes how much each latent component in the crosscoder contributes to the change in the downstream metric, M_{patch} , which we define as the log probability of the sum of the four tokens of *wait* as follows:

$$M_{patch} = \log\left(\sum_{y \in Y_{wait}} p_{\theta}(y|\mathbf{L})\right) \quad (1)$$

where $\mathbf{L} \in \mathbb{R}^{batch \times seq \times dim}$ are the latents obtained from the crosscoder, and Y_{wait} denotes the token sequence corresponding to *wait*, which includes “Wait”, “ Wait”, “ wait”, “wait”. To investigate the causal contribution of these representations, we intervene on the model’s activations. Extracting an intermediate result from a forward pass and inserting it into another is typically performed across minimally different runs to causally probe the role of specific subcomponents. In our case, we approximate this by replacing the activations with zero activations (i.e., performing ablations), which

is similar to attention attribution, a non-patching variant of attribution patching (Nanda, 2023). See more info on the setup at Appendix B.

2.4 Steering

After training, the columns of the decoder matrix $W^{(R)} \in \mathbb{R}^{d \times n_f}$ can be thought of as n_f linear feature directions/features $W^{(R)} = (\mathbf{f}_1, \dots, \mathbf{f}_{n_f})$, where R denotes to the reasoning model, d is the dimension of the activations being decomposed, and n_f is the number of features extracted by our cross-coder. These features can then be used to modulate the model’s rollouts by adding them to the model’s activations ($a(R)$) during token generation:

$$a_{\text{steered}}^{(R)}(x)_i = \begin{cases} a^{(R)}(x)_i + \alpha \frac{\|a^{(R)}(x)\|}{\|\mathbf{f}_k\|} \mathbf{f}_k, & \text{if } i \geq t \\ a^{(R)}(x)_i, & \text{otherwise,} \end{cases} \quad (2)$$

in which we “steered” with strength $\alpha \in \mathbb{R}$ and the k th feature at the last token of the input $x_1, \dots, x_t$ and each newly generated token position $x_{t+1}, \dots$. We steer at layer 15.

3 Results

Intermediate decoding. As a first check we investigate our selected features through the patchscope-lens (Ghandeharioun et al., 2024). It is an intermediate decoding technique that decodes (some) of the information contained in a latent by inserting it at its corresponding layer in a parallel forward pass that is processing a patchscope-lens prompt. Since we are using a reasoning model, we slightly adapt the patchscope-lens prompt to the reasoning model’s format and thought prefilling (see Appendix C). For each of the top/bottom features, we compute the patchscope’s next token distribution and take the average within the top/bottom groups. As a result, we obtain Figure 4. As can be seen top features indeed promote “Wait” and related tokens, whereas, bottom ones share “ ” as top token.

Model diffing. Crosscoders by design allow to classify their learned features into base, shared, and finetuned. As Minder et al. (2025) have recently shown this classification is not perfect, we ask the reader to take Figure 5 with a grain of salt. Figure 5 shows the fraction of top and bottom features classified into the three categories. Both top and bottom features contain a significant number of shared features, which is expected, since most of the features learned by the crosscoder are in this category. The bottom features also contain some base-only features, - we hypothesize that some of the features that only the base model uses

would decrease the likelihood of the *wait* token in reasoning sequences, since the base model was not trained to predict the *wait* token in such context.

As expected, none of the top features are base-only. Most interestingly, the bottom features contain a larger number for finetuned-only, i.e., reasoning features. This suggests reasoning tuning allocates a substantial number of features to both suppressing as well as promoting *wait* token’s probabilities.

Max activating examples. Next, we examine the features’ max activating examples, which we obtain by computing feature coefficients over a dataset of 20 million tokens. Max-activating examples are input samples of various lengths from the crosscoder training data that elicit the highest activation of a given feature. They are identified by running the trained crosscoder on 200,000 validation samples, recording each feature’s activation, and selecting the top 100. For each sample, we highlight the most activating tokens, aiding interpretation. Top features most positively influence the logprob of the *wait* token, while Bottom features have the strongest negative effect. We manually expect a set of 100 max activating examples and generate an automated annotation for each of our 100 features.

We observe that the Top features largely contribute to backtracking (max activating examples activate on *wait* and “But” tokens). Bottom features correspond to behavior which can be interpreted as the model restarting its thinking process or concluding the reasoning trace. For example, feature #1565 activates on full stops at the end of the reasoning sequence and feature #32252 activates on a final answer at the end of samples with mathematical reasoning.

Feature steering. Finally, we investigate features’ causal impact on the rollouts in the reasoning model by performing steering. This is done by activating a feature, represented as a column vector of our learned decoder matrix, and multiplying it with a steering coefficient that sets its strength before adding it to the residual stream. We steer on a single input example, see Figure 3. To match the setting in which we extracted the features we start steering the rollout from the token before the first *wait* token and from there generate up to 200 tokens while steering. Since steering is sensitive to the steering strength hyperparameter, we steer with multiple strengths -1.5, -1.25, -1.0, -0.75, -0.5, 0.5, 0.75, 1, 1.25, 1.5 for each feature. This results

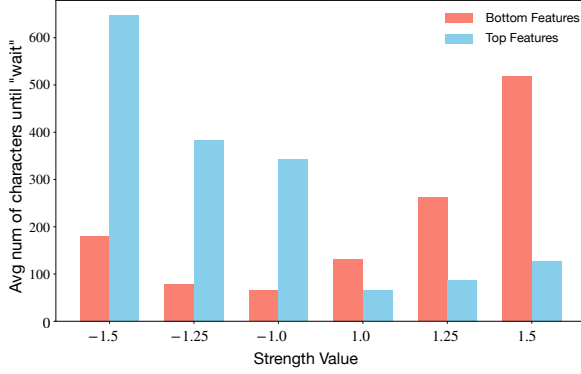


Figure 2: We compute how many characters occur before the first *wait* token in the continued rollout, while steering with all of our features and different intervention strengths. Steering with positive coefficient for the top features slightly increases the distance to the first *wait* which is due to oversteering, for bottom features as one would expect the distance to the first *wait* increases significantly. Negative steering has the opposite effect.

in 10 different continuations for each of the 100 features.

In Figure 2, we verified that the causal role of our features is broadly aligned with our expectations about them based on our feature selection criterion. In particular, we measure the number of characters occurring before the first *wait*. Since we selected our features based on latent attribution with respect to *wait*, for top features the number of characters should be small when steering positively and for bottom features it should be large. Similarly, for features that have meaningful effects in both directions—which is not guaranteed, since feature coefficients are always positive—steering with negative strength should increase the number of characters before ‘wait’ for top features and decrease it for bottom features. Figure 2 matches these expectations.

Steered generations. We show example continuations under our interventions in Figure 3. As can be seen, in particular, the bottom features lead to interesting reasoning patterns that we have not seen before in the literature. Among the top features we observed many instances of steering positively quickly resulting in degenerate sequences like “WaitWaitWait...” or “wait wait wait...”. Steering these into the negative direction leads to *wait* disappearing from the outputs. Our best guess is that those features literally contribute to the *wait* tokens and it is unclear whether they can also trigger any specific reasoning patterns that would be comparable to the ones reported in Fig-

Table 1: **Math500** ($n=100$) **downstream evaluation with feature steering** at layer 15 ($\alpha=1.5$) over first 100 tokens; Acc = exact-match accuracy; Δlen = median token increase; Adh. is LLM-judge score

Feature	Type	Acc	$\Delta\text{Length (med, tok) [\%]}$	Adh.
188	Bottom	84%	+429 (+28%)	83
744	Bottom	81%	+473 (+39%)	58
25929	Bottom	79%	+310 (+21%)	31 [†]
31748	Top	61%	+526 (+45%)	36

ure 3. Additionally, we found several features that when steered positively/negatively lead towards the model wrapping up and providing the final response and when steered into the opposite direction lead to extended reasoning.

Downstream evaluation on MATH500. We compute rollouts for each sample with temperature=0.6, top_p=0.95, and max_tokens=7500, resulting in 86% accuracy. Next, we filter for completions containing the *wait* token. From these, we sample 100 problems with 81 correct / 19 incorrect rollouts. We steer with one feature at a time (IDs 188, 744, 25929, 31748) at layer 15 with strength $\alpha = 1.5$, applied to the first 100 tokens (full-sequence steering was found to degrade continuation quality).

We evaluate using exact-match accuracy, median change in completion length, and LLM-judge adherence score (GPT-5-nano (OpenAI, 2025)). Accuracy remained close to 81% baseline for three features but decreased to 61% for 31748. Detailed results are provided in Table 1, and judge implementation details are included in Appendix D. Manual inspection suggests the judge systematically underestimates adherence, and thus the reported scores should be interpreted as lower bounds.

4 Related Work

To understand reasoning models, Venhoff et al. (2025) looks into the chain-of-thought texts and categorizes the trace into various reasoning behavior patterns such as deduction, adding knowledge, and example testing. They further find steering vectors to increase or decrease the appearance of a particular reasoning behavior. Baek and Tegmark (2025) investigates reasoning features, learned by cross-coders, via qualitative analysis of max activating examples. An additional perspective on token-level and feature-level analysis of reasoning features is a concurrent work by Lee et al. (2025) that performs a case study on the self-verification process of a task-specific reasoning model.

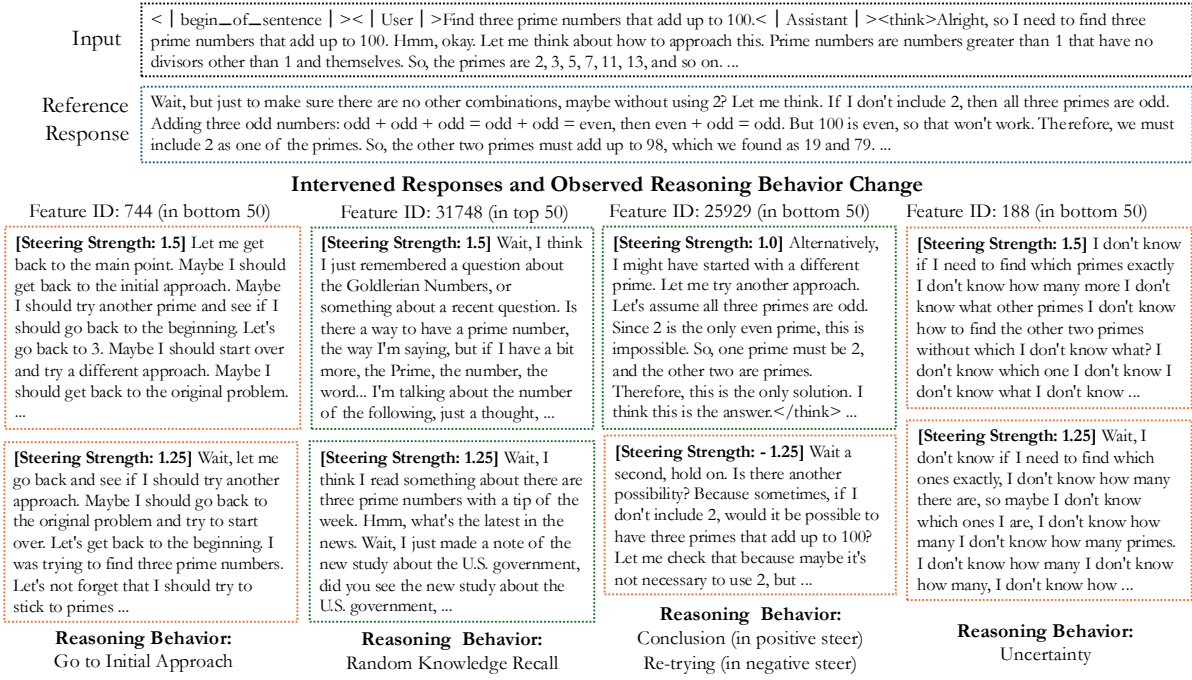


Figure 3: Change in reasoning behavior observed when features are steered in the positive and/or negative directions.

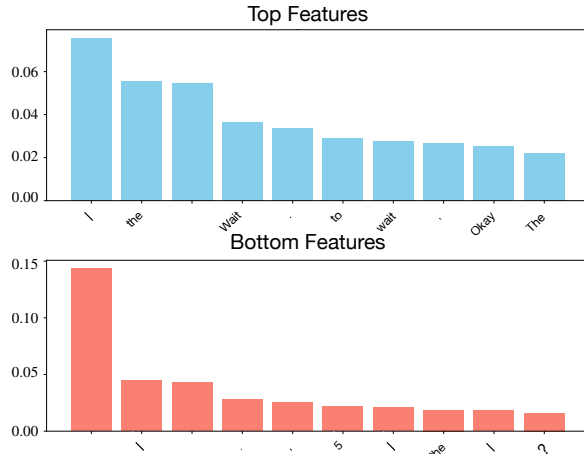


Figure 4: Top token probabilities from patchscope: top features promote 'Wait' and related tokens, while bottom features show no clear pattern.

Similarly, Zhang et al. (2025) show that reasoning models “know” when they are right by training linear probes that can predict the correctness of intermediate solutions from the models’ internals with high accuracy. In contrast to (Venhoff et al., 2025; Lee et al., 2025; Zhang et al., 2025) that focus on specific features and reasoning behaviors on labeled datasets, we are using crosscoders to discover features relevant to distilled reasoning model’s “thought” process with minimal supervision.

Compared Baek and Tegmark (2025) we focus

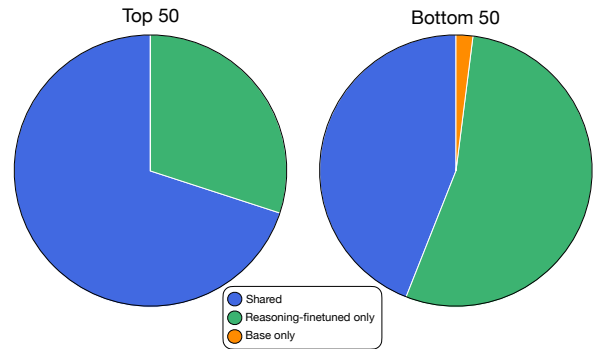


Figure 5: Crosscoder classification of the features into base, shared, reasoning-finetuned categories, with distribution attributed using the Crosscoder Relative Norm Difference

on a more narrow set of the features that we obtain by our latent attribution technique that selects features most relevant for modulating the logits of different “wait” tokens.

5 Conclusion

Our study sheds light on how the model’s latents preceding *wait* tokens signal behaviors like backtracking. We introduce latent attribution patching for identifying and testing which internal features influence these tokens in crosscoder models. We show that these features not only predict *wait* tokens but also shape how the model reasons.

Limitations

While our method identifies and manipulates features influencing *wait* tokens, it is currently limited to a specific model family and may not generalize across architectures without adaptation. Additionally, our analysis focuses on a narrow slice of reasoning behavior, potentially missing out other important markers or mechanisms. Finally, while we demonstrate causal influence on reasoning behavior, fully constructing a circuit that also shows latent-to-latent dependency across layers for these reasoning behaviors remains an open area for future work.

Data and Code Availability

Crosscoder weights: <https://huggingface.co/mitroitskii/Crosscoder-Llama-3.1-8B-vs-Llama-R1-Distill-8B>.

Crosscoder training: https://github.com/science-of-finetuning/crosscoder_learning.

Crosscoders analysis and max activating examples: <https://github.com/science-of-finetuning/sparsity-artifacts-crosscoders>.

Attribution experiments: https://github.com/mitroitskii/interp-experiments/tree/main/reasoning_circuits.

Steering experiments: <https://github.com/wendlerc/r1helpers>.

Acknowledgments

This project began as part of the training phase of Neel Nanda’s MATS 8.0 stream. We are grateful to Neel and Arthur Conmy for their valuable feedback and suggestions during that time. We would also like to thank Clément Dumas and Julian Minder for their helpful guidance on crosscoder setup, Caeden Juang for advice on the attribution setup, Constantin Venhoff for providing the dataset, as well as the Bau Lab for providing compute resources.

References

Anthropic. 2025. [Claude 3.7 sonnet system card](#).

David D. Baek and Max Tegmark. 2025. [Towards understanding distilled reasoning models: A representational approach](#). In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,

Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. [Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning](#). *Preprint*, arXiv:2501.12948.

Asma Ghandeharioun, Avi Caciularu, Adam Pearce, Lucas Dixon, and Mor Geva. 2024. [Patchscopes: A unifying framework for inspecting hidden representations of language models](#). *arXiv preprint arXiv:2401.06102*.

Koray Kavukcuoglu. 2025. [Gemini 2.5: Our most intelligent ai model](#).

János Kramár, Tom Lieberum, Rohin Shah, and Neel Nanda. 2024. [Atp*: An efficient and scalable method for localizing llm behaviour to components](#). *Preprint*, arXiv:2403.00745.

Andrew Lee, Lihao Sun, Chris Wendler, Fernanda Viégas, and Martin Wattenberg. 2025. [The geometry of self-verification in a task-specific reasoning model](#). *Preprint*, arXiv:2504.14379.

Jack Lindsey, Adly Templeton, Jonathan Marcus, Thomas Conerly, Joshua Batson, and Christopher Olah. 2024. [Sparse crosscoders for cross-layer features and model diffing](#).

Samuel Marks, Can Rager, Eric J Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. 2025. [Sparse feature circuits: Discovering and editing interpretable causal graphs in language models](#). In *The Thirteenth International Conference on Learning Representations*.

Julian Minder, Clément Dumas, Bilal Chughtai, and Neel Nanda. 2025. [Latent scaling robustly identifies chat-specific latents in crosscoders](#). In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*.

Neel Nanda. 2023. [Attribution patching: Activation patching at industrial scale](#). *Mechanistic Interpretability* blog post, accessed August 16, 2025.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. [Openai o1 system card](#). *Preprint*, arXiv:2412.16720.

OpenAI. 2025. [Gpt-5-nano — models & capabilities](#). <https://platform.openai.com/docs/models/gpt-5-nano>. Documentation page.

Qwen Team. 2024. [Qwq: Reflect deeply on the boundaries of the unknown](#).

Constantin Venhoff, Iván Arcuschin, Philip Torr, Arthur Conmy, and Neel Nanda. 2025. [Understanding reasoning in thinking language models via steering vectors](#). In *Workshop on Reasoning and Planning for Large Language Models*.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. 2025. Reasoning models know when they’re right: Probing hidden states for self-verification. *arXiv preprint arXiv:2504.05419*.

A Crosscoder Training Details

To train a crosscoder using activations from two models, base B and reasoning-finetuned R , the crosscoder feature activation can be computed as follows:

$$f(x_t) = \text{ReLU}\left(\sum_{i=B,R} W_{enc}^{(i)} a^{(i)}(x_t) + b_{enc}^{(i)}\right) \quad (3)$$

$$a'^{(i)}(x_t) = W_{dec}^{(i)} f(x_t) + b_{dec}^{(i)} \quad (4)$$

where $W_{enc}^{(i)}$ is the model i 's encoder matrix, $W_{dec}^{(i)}$ is model i 's decoder matrix, $a^{(i)}$ is the model i 's activation at token x_t and $a'^{(i)}(x)$ is the reconstructed activation.

The crosscoder training minimizes the loss that consists of the sum of reconstruction MSE and the sum of the per-feature decoder vector's L2 norm.

$$Loss = \sum_{i=B,R} \|a'^{(i)} - a^{(i)}\|^2 + \sum_k f_k(x_t) \sum_{i=B,R} \|W_{dec,k}^{(i)}\| \quad (5)$$

We trained three such crosscoders for 18 hours using 50-100% of x8 A100 GPUs.

B Additional information on Latent Attribution

Let M_{patch} be a scalar metric computed via a computational graph that operationalizes the concept of promoting or suppressing *wait*. Following the attribution setup in Marks et al. (2025), we quantify the importance of each crosscoder latent on this metric via its *indirect effect* on M_{patch} :

$$\text{IE}(M_{patch}; \mathbf{L}, W_{dec}^R) = M_{patch}(\mathbf{L} | \text{do}(\mathbf{a} = \mathbf{a}_{active})) - M_{patch}(\mathbf{L} | \text{do}(\mathbf{a} = \mathbf{0})) \quad (6)$$

Here, $\mathbf{a}_{active}$ represents the value that latent activations during the computation of $M_{patch}(\mathbf{L})$, and $M_{patch}(\mathbf{L} | \text{do}(\mathbf{a} = \mathbf{0}))$ denotes the value of M_{patch} when computing $M_{patch}(\mathbf{L})$ but intervening in the computation by manually setting latent activations to zero.

We cannot compute IEs efficiently for a very large number of latent components $\mathbf{L} \in \mathbb{R}^d$ using Equation (6). We thus employ linear approximations to Equation (6) that can be computed for many latents in parallel. The simplest such approximation that attribution patching (Nanda, 2023) employs is a first-order Taylor expansion:

$$\widehat{\text{IE}}_{\text{atp}}(M_{patch}; \mathbf{L}, W_{dec}^R) = \nabla_{\mathbf{L}} M_{patch} \big|_{\mathbf{L}=\mathbf{L}_{\text{clean}}} \cdot (\mathbf{a}_{active} - \mathbf{0}) \quad (7)$$

which estimates Equation (6) for every latent in parallel using only two forward and one backward pass.

The gradient with respect to latents is computed via the chain rule is

$$\frac{\partial M_{patch}}{\partial \mathbf{L}} = \frac{\partial M_{patch}}{\partial \mathbf{H}} \cdot W_{dec}^R \quad (8)$$

where the matrix multiplication projects the hidden-state gradient onto all decoder directions simultaneously using batch operations (implemented via `einsum('bth, hk→btk', gradient, W_dec)` for batch size b , sequence length t , hidden dimension h , and k latents).

Per-token attribution For token t and latent k , the location contribution is:

$$\text{attr}_{t,k} = \frac{\partial M_{patch}}{\partial H_t} \cdot v_k \times L_{t,k} \quad (9)$$

where v_k is the k -th column of W_{dec}^R . Positive value of $\text{attr}_{t,k}$ implies promoting local effect on the metric, while negative implies a suppressing local effect.

Aggregation to per-feature scores We summarize each latent’s influence by summing over tokens in the analysis window and averaging over examples in the batch:

$$\text{score}(k) = \text{mean}_{\text{batch}} \left[\sum_{t \in \mathcal{W}} \text{attr}_{t,k} \right] \quad (10)$$

Sorting $\text{score}(k)$ produces the top (positive, promoters) and bottom (negative, suppressors) feature lists used in Section 3.

C Patchscope-lens prompt

```
<begin of sentence><User>Continue the
following pattern: cat cat
1135 1135
hello hello
<Assistant><think>
Okay I need to complete: cat cat
1135 1135
hello hello
?
```

D LLM-Judge Prompt

We use an automated judge to score whether steered generations exhibit the intended behavior. The judge returns a 1–100 adherence score and a brief rationale.

Model GPT-5-nano.

Prompt template (verbatim).

"You are an expert evaluator of language model behavior and steering interventions.

Please analyze the following example and assess how much the steered response corresponds to the steering direction "{steering_direction}" on a scale from 1 to 100, where:

- 1-20: Very little to no correspondence with the steering direction
- 21-40: Some correspondence but not strong
- 41-60: Moderate correspondence
- 61-80: Strong correspondence
- 81-100: Very strong/excellent correspondence with the steering direction

****Original Context:****
{text_before}

****Original Response (before steering):****
{original_text_after}

****Steered Response (after steering):****
{steered_text_after}

- **Assessment Criteria:****
- Consider how well the steered response reflects the intended steering direction "{steering_direction}"
 - Look for behavioral changes, linguistic patterns, and content shifts

- Compare the steered response to the original response
- Evaluate the consistency and strength of the steering effect

Please provide your assessment as a JSON object with the following format:

```
{  
  "score": <number between 1-100>,  
  "reasoning": "<brief explanation of your assessment>"  
}
```

Assessment:"