(How) Do LLMs Plan in One Forward Pass?

Michael Hanna*
Institute for Logic, Language, and Computation
University of Amsterdam

Emmanuel Ameisen
Anthropic

Abstract

Planning underpins many human linguistic abilities that modern LLMs now possess. However, both the extent to which LLMs plan and what precisely this entails is poorly understood. In this paper, we propose a stringent definition of latent planning, *in one forward pass*: a LLM engages in latent planning only if it has a representation that is causally implicated in its generation of both a planned-for token or concept and a preceding context that licenses it. We next use circuits to show that some LLMs plan in simple scenarios: they possess features that represent a planned-for word like *accountant*, and cause them to output *an* rather than *a*; ablating such features changes their output. On the more complex task of completing rhyming couplets, we find that models often identify a rhyme ahead of time, but even large models seldom plan far ahead. However, we can elicit some planning that increases with scale when steering models towards planned words in prose. In sum, we offer a framework for measuring planning and mechanistic evidence of how models' planning abilities grow with scale.

1 Introduction

Planning is a cornerstone of human language: when humans speak, we plan our utterances, organizing their structure and content [11, 31, 17]. Modern large language models (LLMs) now possess many linguistic abilities too, producing text that is not only grammatical but also relatively meaningful and coherent [5], though gaps with humans remain. Despite this, empirical evidence regarding LLMs' ability to perform latent, unverbalized planning—the same sort that supports humans' linguistic abilities—remains limited, though work on planning in LLMs' chains of thought is more common [16, 30, 32]. Current evidence for latent planning in LLMs is largely observational: studies generally show that future tokens or text attributes can be extracted from model activations [26, 27, 6]. Only recently has causal evidence for planning emerged, in large, closed models [19].

In this paper, we argue that claims of planning must be based on causal, not observational evidence, lest we apply the "planning" label to too broadly. Instead, we should consider an LLM to be planning only if it possesses an abstract feature that represents a planned-for token (or token sequence), and is causally implicated in its generation of the token (sequence) and a preceding context licensing it.

Armed with this definition, we test models on simple tasks that could involve planning, like completing "Someone who handles financial records is → a/**an** (accountant)". We find that across Qwen-3 models, only models of 8B+ parameters succeed. We then use feature circuits [21, 7, 1] to understand how models perform these tasks, and find that there exist planning features that represent future outputs like *accountant* and upweight relevant outputs like "an". Moreover, although smaller models fail, they still possess planning-relevant features that promote the correct answer.

We next have models complete rhyming couplets, where Lindsey et al. observed longer-range planning in Claude Haiku. We find that models employ a circuit that tracks information related to poetry, such as when a line is about to end, or what to rhyme with; however, even large models do not en-

*Completed as part of the Anthropic Fellows Program. Correspondence to m.w.hanna@uva.nl.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

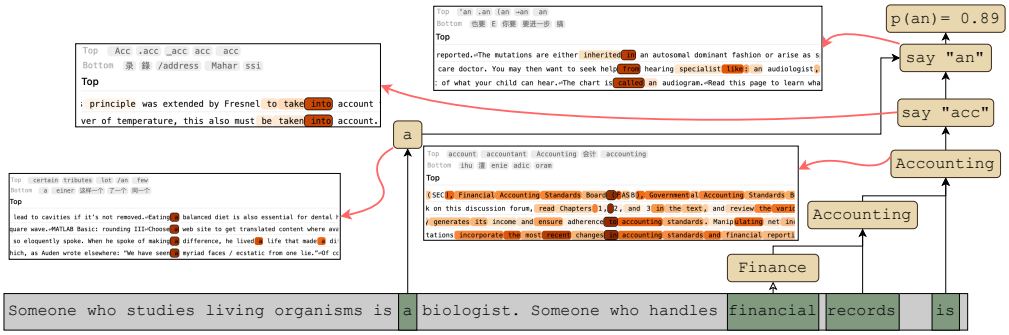

Figure 1: Feature circuit for the input *Someone who studies living organisms is a biologist. Someone who handles financial records is*, explaining Qwen-3 (14B)'s output, *an*. The model first determines the word it plans to say (*accountant*), causing it to output the appropriate article, *an*. Labeled nodes are sets of active transcoder features with a similar role. Edges indicate that the source node increases the target node's activation when active. We demonstrate the role of certain nodes by selecting one of its features and showing its top-activating inputs, and the vocabulary item that it up-/down-weights.

gage in backward planning. Finally, we then test intermediate planning abilities by steering models towards planned words in prose, and observe latent planning, increasing with scale.

2 What is planning in LLMs?

Planning is behavior in which one reasons about which actions must be taken (and in which order) to achieve a goal. However, most past work on latent planning in LLMs searches model internals for evidence of a goal, not goal-oriented reasoning. For example, Dong et al. [6] prompt LLMs to write stories, and probe the LLMs' representations of the input prompt for information about their future outputs. They equate successful probing with latent planning, but see App. E for evidence to the contrary. Pochinkov [27] takes the residual stream of LLMs that are about to start a new paragraph, and attempts to decode the topic thereof using Patchscopes [12]; again, successful decoding is taken to entail planning. Pal et al. [26] also decode models' future tokens with probes and Patchscopes—though they do not call this planning. Lindsey et al. [19] are unique in providing causal evidence: studying LLMs' ability to complete rhyming couplets, they not only observe representations of the rhyming word that the model plans to output, but also causally intervene on them, changing the upcoming word and its preceding context that accommodates it.

We argue that, if LLM planning entails reasoning about the steps needed to output a specific future token, decoding the future token is insufficient to evidentiate planning. Consider a model that always outputs the same token, or one that outputs *0, 2, 4, 6,...*; in both cases, a probe could likely predict many future tokens, but neither task requires planning. More generally, the decodability of a given attribute from model representations does not entail its use in model processing; probes are known to decode unused information from model representations [29]. Instead, if planning is a *mechanism* that models deploy, a definition of planning should make causally verifiable *mechanistic* claims.

We thus propose a stringent definition of planning, based on Lindsey et al.'s [19] findings. A LLM given a length-n input engages in planning if it possess a representation of a token or concept that:

Condition 1: *causes it to output the specific token or concept* t *at some position* $n + k$, $k > 1$. This definition strengthens the decodability criterion from past work: we require that some feature *causes* the LLM to produce t, not just that t is predictable from the LLM's features.

Condition 2: *causes it to output a context that licenses said token or concept* t. This formalizes the idea that planning entails actively working towards outputting t. Consider the input $s =$ *The capital of Texas* $\rightarrow$ *is* $\rightarrow$ *Austin*. Strong LLMs likely have an *Austin* feature at the *Texas* position of s; ablating it stops the model from later outputting *Austin*. However, we would not call this planning unless the *Austin* feature causes the production of the intervening token *is*; this seems unlikely, as knowing *Austin* is not necessary to predict *is*. Note that Condition 2 runs contrary to past planning work studying features that are *not* useful for immediate next-token prediction [33].

Category	Input	Next Token	Planned
a / an	Someone who handles financial records is	an	accountant
is / are	There were 5 dogs but 4 left. Now there	is	1

Table 1: Two simple planning tasks. Each task prompts the model to output a planned token, preceded by a next token with two possible forms; the correct form is determined by the planned token.

3 Transcoders and Transcoder Feature Circuits

To identify planning representations, we decompose model activations into sparse features using transcoders [7]; we then find *feature circuits*, or causally relevant subgraphs of features [21, 1].

Transcoders: Transcoders are auxiliary models that replace the model's MLPs [7]; each transcoder takes in one MLP's inputs and predicts its outputs. Formally, a transcoder takes in a given MLP's input activations $\mathbf{h} \in \mathbb{R}^d$ and computes a sparse representation $\mathbf{z} \in \mathbb{R}^n$ as $\mathbf{z} = f\left(\mathbf{W}_{enc}\mathbf{h} + \mathbf{b}_{enc}\right)$. It then reconstructs the MLP's output activations $\mathbf{h}' \in \mathbb{R}^d$ as $\tilde{\mathbf{h}'} = \mathbf{W}_{dec}\mathbf{z} + \mathbf{b}_{dec}$. f is an activation function, while $\mathbf{W}_{enc}, \mathbf{b}_{enc}, \mathbf{W}_{dec}$, and $\mathbf{b}_{dec}$ are learned parameters.

Transcoders are useful because they are trained to compute representations $\mathbf{z}$ that are *sparse* and *monosemantic*: most dimensions (or *features*) are zero on any given input. We interpret the i^{th} feature of a given transcoder by inspecting the inputs that maximize its activation $\mathbf{z}_i$, as well as the tokens whose unembedding vectors have the highest and lowest dot product with the feature's column in $\mathbf{W}_{dec}$. See Figure 1 for example feature visualizations, used to manually label features. For more background and technical details on transcoders, see Appendix D.1.

Transcoder Feature Circuits: Given a model, transcoders trained on each MLP thereof, and an input, we construct a transcoder feature circuit [1]: a weighted acyclic digraph describing the causal relationships between the model's inputs, transcoder features, and logits. Each edge weight indicates the source node's direct effect on the target, i.e. the amount by which it directly increases the latter's value. Once features are annotated, and similar features grouped together, the circuit serves as a mechanistic explanation for a model's behavior on the input, as seen in Figure 1.

We compute feature circuits using Ameisen et al.'s algorithm. Unlike other feature circuit techniques, it computes *exact* direct effect values—conditional on the model's attention patterns and LayerNorm denominators. We thus know the precise causal relationship between features, ignoring contributions to these quantities. This paradigm helps ensure that any planning features found fulfill our conditions: features are guaranteed to be causally relevant, and we can see what intermediate features represent. We use the `circuit-tracer` library for circuit-finding and interventions [14].

4 Qwen-3 models possess simple planning mechanisms

To test models' planning abilities, we study two simple tasks to which LLMs are likely exposed during pre-training; LLM abilities are often stronger on such tasks [22]. Each task (Table 1) consists of an input that pushes the model to produce a specific content word, preceded by a function word that must agree with it. For example, in the *is / are* task example in Table 1, the model needs to output 1, but must output the correct form of *to be* prior to it. See Appendix A for details on the composition of these datasets. We discuss *a/an* in the main text, and *is/are* in Appendix B.

We start by evaluating Qwen3 (0.6B-32B) [34] on these tasks, testing their ability to predict the correct next token. We find (Figure 2, top left) that all models easily recall the majority class *a* (recall > 0.8). Recall of the minority class *an*, is high (> 0.8) for models with 14B+ parameters, and lower for those below; small models tend to predict *a*. This is not entirely attributable to models' inability to perform the task at hand: in Appendix C, we show that small models can calculate the answer to is/are questions, but fail to predict the correct verb, producing outputs like *... there are 1 dog*. It thus appears that simple planning (and not just fundamental task ability) is the issue.

We next find transcoder feature circuits underlying these tasks, as described in the previous section; we study Qwen3- 0.6B to 14B, as 32B is prohibitively large. We visualize and qualitatively analyze a subset of the feature circuits, annotating each feature in a given circuit and grouping similar features together. We find that in the *a/an* dataset many features represent the planned token: their top-activating texts are instances of that token. These features feed into features that upweight the correct next token; see Figure 1for an example. This suggests that models plan for the target token

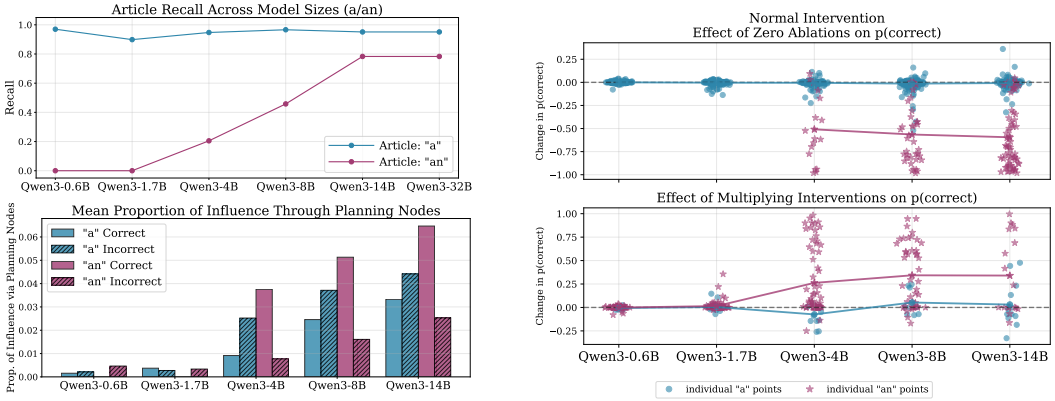

Figure 2: **Top Left**: Qwen-3 family models' recall of correct article on the *a/an* task. All models can recall *a*, but models $\leq$ 14B have lower recall on the less-common *an*. **Bottom Left**: The mean proportion of influence flowing through planning nodes in the *a/an* dataset. When the model correctly predicts the next token, more influence tends to flow through the planning nodes—but only on examples from the minority class *an*. **Right**: Change in $p(\textit{correct article})$ caused by zero and multiplying interventions on planning features. The former generally harm performance, while the latter improve it. Both affect only examples requiring *an*, the minority class.

(e.g. *accountant*), leading them to output the correct next token (e.g. *an*). Even smaller (4B) models often have such planning features, suggesting the existence of nascent planning mechanisms.

We now verify that these planning features truly drive the model's prediction of the correct next token. We start by programmatically finding planning features for each example; a feature is considered planning-relevant if is located at the last position in the sentence, and it either upweights the planned word (e.g., *accountant*), or contains it in its top-activating texts.

With these features, we perform two causal relevance analyses. First, we ask—how important are planning nodes according to our circuits? Each edge in the circuit reflects the direct influence of a source node on a target node, but we can also consider the total flow from a source to a target node, which might travel via multi-node paths. To quantify the importance of the planning nodes, we measure the proportion of the total flow between the circuit's inputs and logits that is mediated by the planning features, comparing the flow in cases where the model is in/correct.

We find that (Figure 2, bottom left) that when models predict *an* correctly, more of their total influence flows through the planning nodes; this greater influence flow may drive model correctness. However, this is not true in the *a* case: the trend is reversed, with more influence flowing through planning nodes in *incorrect* cases. This hints that models may only plan in minority-class *an* cases.

Second, we support our interpretation of these circuits with causal interventions. For each model, we a) take the examples on which the model succeeds and ablate the planning features, setting them to zero, and b) take the examples on which the model fails and highly upweight their planning features, setting their activations to 5$\times$ their usual activations. If these features indeed cause the models to output the planned token, the ablations should harm and improve performance respectively.

We find (Figure 2, right) that this is the case: feature ablation harms performance of models from 4B-14B—those with non-zero performance originally. Boosting planning features improves performance drastically on incorrect examples, though only for those same models. These interventions also primarily affect *an* examples, further suggesting that planning nodes are more important for minority-class examples where models must work against their priors.

5 Qwen-3 models lack complex planning mechanisms

The preceding experiments show that larger Qwen-3 models plan more, but does not address longer-range planning mechanisms. Lindsey et al. [19] observe these, finding that given the first line of a rhyming couplet, Claude Haiku produces a next line using a planning feature feature to control the rhyming word and generate a coherent context. We thus study Qwen-3 models on rhyming couplets.

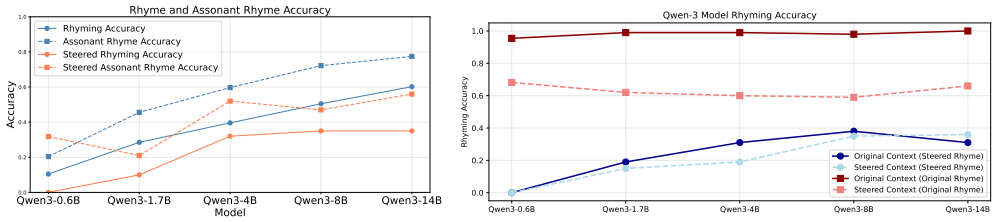

Figure 3: **Left**: Qwen-3 rhyming accuracy. In the base case, models have moderate rhyming accuracy, reaching 0.6 at 14B parameters (solid blue); when we consider assonant (vowel-only) rhyme, Qwen-3 (14B) achieves 0.8 (dashed blue). When steered to predict a new rhyme, model accuracy is only moderate for perfect rhymes (solid orange), but improves with scale, and is better on assonant rhyme (dashed orange). **Right**: Model rhyming accuracy when trying to predict a token satisfying the couplet's the steered rhyme (blue lines) or original rhyme (red lines), given the original (solid) or steered (dashed) context. The model predicts the steered-for rhyme with similar accuracy given the original or steered context. This suggests that the steered context does not better license the rhyme.

We first prompt Qwen-3 (32B) to generate a dataset of 985 first lines of couplets. We then greedily sample a second line from each model, and evaluate its rhyme with the first couplet using CMUDict [4, 2]. We find (Figure 3, left) that larger models rhyme with 50+% accuracy; smaller ones fail more often. However, models often engage in assonant (vowel-only) rhyme, pairing e.g. *craze* with *page*.

For each model, we compute transcoder circuits for 100 inputs where it rhymed correctly. Given a couplet like *Fury burns where calm once stayed,... Hope flickers where the shadows laid*, we find the circuit explaining the model's prediction of *laid*. The resulting circuits have a common structure: midway through the second line, models attend to the end of the couplet's first line, which contains features indicating a rhyme family, e.g. *-ayed*. Similar rhyming features then activate in the second line, causing models to output a rhyming word. If these circuit involve planning, we view the rhyming features at the end of the couplet's first line as the most likely planning features; we thus define rules to identify these programmatically, as in the prior section. See Appendix F for details.

To test if these rhyming features truly control the rhyme, we intervene on each couplet for which we have a circuit. We downweight the rhyme features at the end of its first line, multiplying their activations by -3. We then sample a random couplet with a distinct rhyming sound, and upweight its rhyme features, multiplying their original activations by 7; we find these steering hyperparameters via manual search. We then generate a completion to the first line of the original couplet, while steering on the end of the first line. We measure rhyming accuracy with respect to the new rhyme.

We also check if the context generated by the model facilitates the new rhyme, by measuring which context (the original or steered one) best enables the model to predict the new rhyme, when we steer it towards the new rhyme. If the new context licenses the new rhyme better, the model should more accurately predict a rhyming word given it. We thus feed the model both the original and steered couplet completions, with their last word removed. We record the model's generation given each, when steered towards the new rhyme, and compute rhyming accuracy with respect to the new rhyme.

We find (Figure 3, left) that steering on the rhyme features changes the model's rhyme to the new rhyme in the case of larger models (8B-14B). Though accuracy is only moderate (40%), normal rhyming accuracy was similarly modest at 60%, and assonant rhyme accuracy is higher (up to 60%). However, the intermediate context generated under intervention does not license the new rhyme better (Figure 3, right). When we steer the model, it is equally likely to output the injected rhyme given the steered intermediate context (light blue, dashed line) as when given the original one (dark blue, solid line). Giving the model the intermediate context produced with steering, but not steering it, elicits the original rhyme with relatively high accuracy: near 60% across models (light red, dashed line). Though this is low compared to the accuracy given the original context (near 100%; dark red, solid line), this is confounded by the fact that we only intervened on examples where models rhymed successfully. These results suggest that models do not plan backwards, crafting a context that licenses the rhyming word that they output. However, see App. G for experiments where we are able to elicit planning behavior that scales by steering on planning features taken from couplets.

Overall, these results suggest that while simple planning and rhyming abilities have emerged with model scale, longer-term planning abilities have not yet fully emerged in Qwen-3 models.

Acknowledgments

The authors thank Jim Maar, Denis Paperno, Ana Lučic, Yaniv Nikankin, Yonatan Belinkov, and Sandro Pezzelle for insightful conversations and feedback on this work. The authors also thank the Anthropic Fellows Program for enabling and supporting this project.

References

[1] Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, 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. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/methods.html.

[2] Steven Bird and Edward Loper. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/P04-3031/.

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

[4] Carnegie Mellon University. The carnegie mellon pronouncing dictionary, 2014. URL http://www.speech.cs.cmu.edu/cgi-bin/cmudict. Version 0.7b.

[5] Tyler A. Chang and Benjamin K. Bergen. Language model behavior: A comprehensive survey. *Computational Linguistics*, 50(1):293–350, 03 2024. ISSN 0891-2017. doi: 10.1162/coli_a_00492. URL https://doi.org/10.1162/coli_a_00492.

[6] Zhichen Dong, Zhanhui Zhou, Zhixuan Liu, Chao Yang, and Chaochao Lu. Emergent response planning in LLMs. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Ce79P8ULPY.

[7] Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable LLM feature circuits. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=J6zHcScAo0.

[8] Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english?, 2023. URL https://arxiv.org/abs/2305.07759.

[9] 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.

[10] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/toy_model/index.html.

[11] Fernanda Ferreira and Benjamin Swets. How incremental is language production? evidence from the production of utterances requiring the computation of arithmetic sums. *Journal*

of Memory and Language, 46(1):57–84, 2002. ISSN 0749-596X. doi: https://doi.org/10.1006/jmla.2001.2797. URL https://www.sciencedirect.com/science/article/pii/S0749596X01927974.

[12] Asma Ghandeharioun, Avi Caciularu, Adam Pearce, Lucas Dixon, and Mor Geva. Patchscopes: a unifying framework for inspecting hidden representations of language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[13] Mario Giulianelli, Jack Harding, Florian Mohnert, Dieuwke Hupkes, and Willem Zuidema. Under the hood: Using diagnostic classifiers to investigate and improve how language models track agreement information. In Tal Linzen, Grzegorz Chrupała, and Afra Alishahi, editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 240–248, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5426. URL https://aclanthology.org/W18-5426/.

[14] Michael Hanna, Mateusz Piotrowski, Jack Lindsey, and Emmanuel Ameisen. circuit-tracer. https://github.com/safety-research/circuit-tracer, 2025. The first two authors contributed equally and are listed alphabetically.

[15] Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

[16] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=Th8JPEmH4z.

[17] Eun-Kyung Lee, Sarah Brown-Schmidt, and Duane G. Watson. Ways of looking ahead: Hierarchical planning in language production. *Cognition*, 129(3):544–562, 2013. ISSN 0010-0277. doi: https://doi.org/10.1016/j.cognition.2013.08.007. URL https://www.sciencedirect.com/science/article/pii/S0010027713001649.

[18] 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*, October 2024. URL https://transformer-circuits.pub/2024/crosscoders/index.html.

[19] 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.

[20] Samuel Marks and Max Tegmark. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=aajyHYjjsk.

[21] Samuel Marks, Can Rager, Eric J Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=I4e82CIDxv.

[22] R. Thomas McCoy, Shunyu Yao, Dan Friedman, Mathew D. Hardy, and Thomas L. Griffiths. Embers of autoregression show how large language models are shaped by the problem they are trained to solve. *Proceedings of the National Academy of Sciences*, 121(41):e2322420121, 2024. doi: 10.1073/pnas.2322420121. URL https://www.pnas.org/doi/abs/10.1073/pnas.2322420121.

[23] Neel Nanda and Joseph Bloom. Transformerlens. https://github.com/TransformerLensOrg/TransformerLens, 2022.

[24] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. URL https://distill.pub/2017/feature-visualization.

[25] Bruno A. Olshausen and David J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325, 1997. ISSN 0042-6989. doi: https://doi.org/10.1016/S0042-6989(97)00169-7. URL https://www.sciencedirect.com/science/article/pii/S0042698997001697.

[26] Koyena Pal, Jiuding Sun, Andrew Yuan, Byron Wallace, and David Bau. Future lens: Anticipating subsequent tokens from a single hidden state. In Jing Jiang, David Reitter, and Shumin Deng, editors, *Proceedings of the 27th Conference on Computational Natural Language Learning (CoNLL)*, pages 548–560, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.conll-1.37. URL https://aclanthology.org/2023.conll-1.37/.

[27] Nicky Pochinkov. Parascopes: Do language models plan the upcoming paragraph?, 2025. URL https://www.lesswrong.com/posts/9NqgYesCutErskdmu/parascopes-do-language-models-plan-the-upcoming-paragraph.

[28] Shauli Ravfogel, Grusha Prasad, Tal Linzen, and Yoav Goldberg. Counterfactual interventions reveal the causal effect of relative clause representations on agreement prediction. In Arianna Bisazza and Omri Abend, editors, *Proceedings of the 25th Conference on Computational Natural Language Learning*, pages 194–209, Online, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.conll-1.15. URL https://aclanthology.org/2021.conll-1.15/.

[29] Abhilasha Ravichander, Yonatan Belinkov, and Eduard Hovy. Probing the probing paradigm: Does probing accuracy entail task relevance? In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3363–3377, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.295. URL https://aclanthology.org/2021.eacl-main.295/.

[30] Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. Chain of thoughtlessness? an analysis of cot in planning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=kPBEAZU5Nm.

[31] Valentin Wagner, Jörg D Jescheniak, and Herbert Schriefers. On the flexibility of grammatical advance planning during sentence production: Effects of cognitive load on multiple lexical access. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 36(2):423–440, 2010. doi: 10.1037/a0018619.

[32] Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. PlanGenLLMs: A modern survey of LLM planning capabilities. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19497–19521, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.958. URL https://aclanthology.org/2025.acl-long.958/.

[33] Wilson Wu, John Xavier Morris, and Lionel Levine. Do language models plan ahead for future tokens? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BaOAvPUyBO.

[34] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin

Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

A Simple Planning Dataset Details

We construct two datasets for testing simple planning, the *a/an* and *is/are* datasets. The *a/an* dataset consists of 108 examples of professions (86 requiring *a* and 22 requiring *an*) and descriptions thereof. These were augmented with 350 concrete nouns (267 *a* / 83 *an*) and descriptions thereof. All descriptions were generated by Claude 4 Sonnet, but filtered manually and rewritten if necessary, e.g. because they were too vague. The *is/are* dataset was generated programmatically, and consists of (positive) differences between numbers ranging from 1 and 9; the animals are sampled from a manually curated list of 10 animals. This yields 360 examples.

Note that, in the case of the *a/an* dataset, one randomly-sampled in-context example from our dataset is prepended to each input to the model in order to encourage it to output *a/an*; otherwise, the model does not understand the task structure, and outputs other tokens. The full prompt is thus something like `Someone who provides treatment for physical or mental conditions is a therapist. Someone who heals sick pets is`. This is fed directly to the model as the user input, and the model simply completes the input (rather than generating a separate assistant response). The *el/la* dataset is formatted in the same way.

Similarly, we prepend *is/are* examples with *Repeat and finish the following sentence:*, as we found that this increased performance over simply sampling next tokens without requesting the repetition. The full prompt is thus something like `/no_think Repeat this sentence and complete it. At first there were 2 cats. Then, 1 went away. Now, there`. The `/no_think` prevents models from thinking before answering. During attribution, we prefill the model's assistant response with `<think>\n\n</think>\n At first there were 2 cats. Then, 1 went away. Now, there`. We attribute from the top logits (which are always *is* and/or *are*).

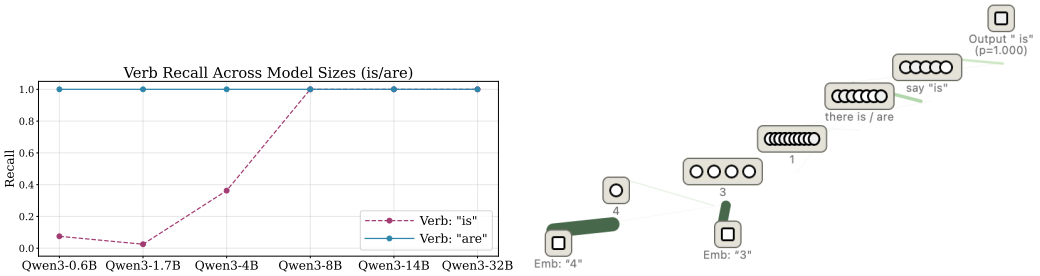

Figure 4: **Left**: Recall of *is* and *are* on the *is/are* dataset, by model. Models below 8B in size mostly fail to predict *is*, while larger models perform perfectly. All models can predict *are*. **Right**: Qwen-3 (8B)'s circuit for the input *At first there were 4 cats. Then, 3 went away. Now, there*. The model possess a set of *1* features, which activate on and upweight *1*, and cause *is* features to activate.

B *Is-Are* Results

Here, we report results for experiments on the *is/are* dataset, which largely mirror those performed on the *a/an* dataset. Figure 4 shows that models behave similarly on the *is/are* to the *is/are* dataset: all models do well on the majority class *are*. Models below 8B in size fail (0.6-1.7B) or perform poorly on the task when the correct answer is *is*; Qwen3-4B scores just below chance. Starting at 8B, models score perfectly on *is* as well, just as with *a/an*.

We perform circuit analysis on *is/are* dataset as well, and find similar, but not identical trends compared to the *a/an* case. Models again have features corresponding to planning features some of the

time. However, *1* features (and *2* and *3* features to a lesser extent) are more common than other numbers' features. Whether this is a real phenomenon (models have special representations for lower numbers due to their frequency) or a transcoder-driven phenomenon (higher numbers also have corresponding features, but transcoders miss these) is unclear. This may also be related to the fact that such features are more important / necessary in the minority class case (*1/is*) than in the majority class case. In the case where such features do exist, there is once more a clear pattern of e.g. *1* features activating features that induce the model to say *is*; see Figure 4 (right) for an example.

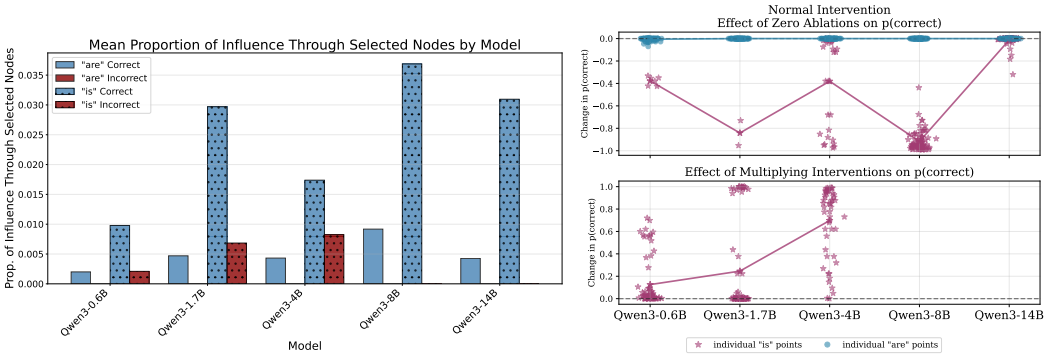

Figure 5: **Left**: The mean proportion of influence flowing through planning nodes in the *is/are* dataset, by model, verb, and correctness; recall that the only incorrect examples are small models failing to predict *is*. The most influence flows through the planning nodes in the *is* examples, where more planning nodes are present. Still, more influence flows through these nodes in correct than incorrect *is* examples. **Right**: Change in $p(\textit{correct verb})$ caused by zero and multiplying interventions on planning features. The former generally harm performance, while the latter improve it. Both affect only *is* examples, which have the most planning nodes, and also are the only examples models answer incorrectly.

Finally, we perform the flow and intervention experiments done on the *a/an* dataset. These are complicated somewhat by the fact that there are more planning nodes in the *is* case than in any of the *are* cases, and that models do not fail on *are* cases. Still, in Figure 5 (left), we can see that in the *is* case, more influence flows through the planning nodes in correct than incorrect examples, as in the *a/an* dataset. Moreover, Figure 5 (right) shows that both zeroing and multiplicative interventions are effective *on is examples*. This is likely because these have the most planning nodes; however, it may also be related to the fact that *is* is the minority class, and "needs" these features more.

C Model Performance on Non-Planning Parts of Simple Planning Tasks

The fact that small models fail to plan on the simple *a/an* and *is/are* planning tasks may raise the question: do small models fail because they cannot perform the tasks at all? To show this is not the case, we generate models' planned tokens, both given the correct next token, and the incorrect next token. We then measure whether the output token in each case matches our expected planned token.

Performance differs by task. On the *a/an* task (Figure 6, left), models have generally high accuracy (> 0.6) when given the correct next token, but lower accuracy when given the incorrect one; the highest scoring models in that scenario achieve an accuracy of 0.3-0.4. Baseline accuracy here is in theory near 0, as we do not constrain the model to output a valid profession. This suggests that although models are not always planning for the precise profession we intend (and indeed, there are cases where we find no nodes corresponding to the planned profession) they often are. And in some cases, they plan so strongly for the intended profession that they output it even when it conflicts with the article.

This trend is much stronger on the *is/are* task. Our results from the analogous experiment (Figure 7) show perfect accuracy for all models when the correct verb form is given. Given the incorrect article, smaller models are (near-)perfectly accurate at predicting the correct number, but larger models (Qwen3-14B and 32B) perform much worse. This provides strong evidence that small models can perform the task (and that a lack of task abilities does not underlie their poor planning performance). However, the root of the behavior of large models is less clear. They appear to be more sensitive

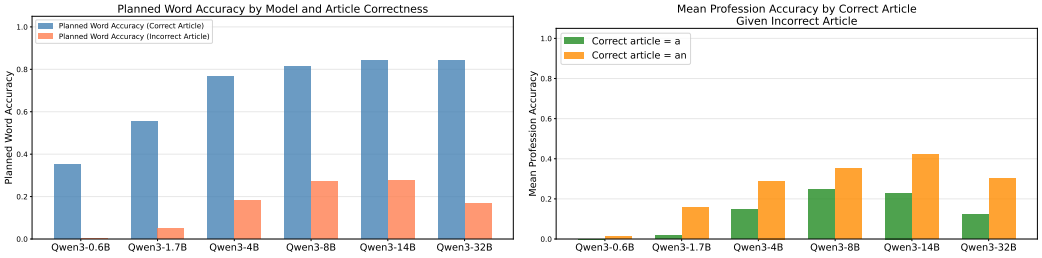

Figure 6: **Left**: Profession accuracy, i.e. whether the model's predicted profession matches the intended profession, when given the correct or incorrect article. Models above 4B in size are highly accurate when given the correct article ($> 80\%$), and even smaller model achieve accuracies above 60%. Given the wrong article, accuracies are lower, but still non-zero, indicating that models may have a strong planning goal that prevails even when the profession is at odds with the article. **Right**: Profession accuracy given the wrong article, by correct article (*a* or *an*). Though accuracy is low, models succeed on both *a* and *an* examples, indicating that successes are not driven by one class.

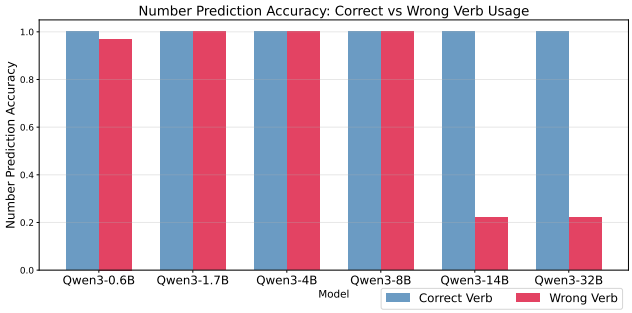

Figure 7: Number accuracy, i.e. whether the model's predicted number of animals matches the intended number, when given the correct or incorrect verb (*is* / *are*). Notably, small models produce the correct number regardless of whether they are given the correct or incorrect verb. In contrast, Qwen3-14B and 32B have starkly reduced accuracy when given the wrong verb form.

to (subject-verb) agreement, and thus produce outputs that agree with the number of the verb; in particular, given *is* as an incorrect next token, they tend to output *1*, rather than a number that agrees with the original animal quantities. In contrast, weak models do produce outputs like . . . *now there are 1 dog remaining.*

D Details of Transcoders and Feature Circuits

D.1 Transcoders

In this section we provide details on transcoders in general and the specific transcoders we use.

Transcoders Past work has attempted to characterize the features encoded in model activations by examining the inputs that most strongly activate each neuron (dimension) of a given activation vector. However, interpreting neurons is difficult, as they are seldom zero and often polysemantic, firing for multiple reasons [24, 10]. Sparse dictionary learning solves this problem by decomposing activations into sparse and (ideally) monosemantic feature vectors [25, 3]. As only a few dimensions, or *features*, of the vector are active on a given input, and each feature fires on only one concept, these are much easier to interpret.

Sparse dictionaries come in many forms. Sparse autoencoders (SAEs; 3, 15) are the most common type, encoding and reconstructing activations from the same location. We use *per-layer* transcoders, which encode MLP inputs and reconstruct MLP outputs [7]; see Figure 8 for a diagram. Lindsey et al. [18] also introduce *cross-layer* transcoders, which take in MLP inputs, and are jointly trained to predict contributions to all downstream MLPs' outputs. These are generally sparser (for a given level of reconstruction error) but also more computationally costly to train and more memory-intensive

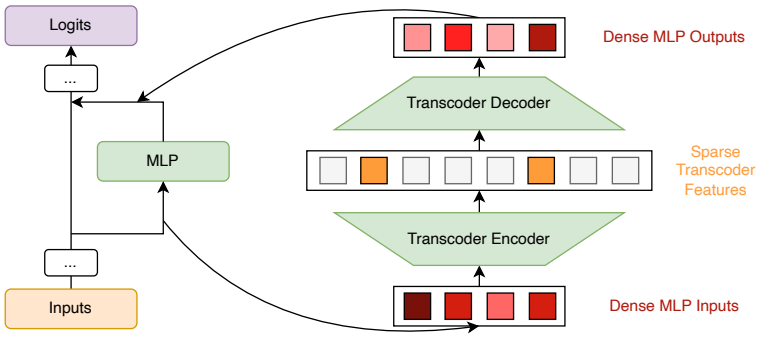

Figure 8: A diagram of a transcoder. The transcoder takes in the dense MLP inputs, computes a sparse representation thereof, and then reconstructs the MLP's dense outputs.

to deploy. Importantly, while Ameisen et al. [1] use cross-layer transcoders for their circuit-finding, per-layer transcoders can also be used.

Formally, a (per-layer) transcoder takes in activations $\mathbf{h} \in \mathbb{R}^d$ from a given MLP's inputs, computes the sparse representation $\mathbf{z} \in \mathbb{R}^n$, and reconstructs the MLP's output activations $\mathbf{h}' \in \mathbb{R}^d$ as follows:

$$\mathbf{z} = f\left(\mathbf{W}_{enc}\mathbf{h} + \mathbf{b}_{enc}\right) \tag{1}$$

$$\tilde{\mathbf{h}'} = \mathbf{W}_{dec}\mathbf{z} + \mathbf{b}_{dec}, \tag{2}$$

Here, f is an activation function (often ReLU, JumpReLU, or Top-k), and $\mathbf{W}_{enc}, \mathbf{b}_{enc}, \mathbf{W}_{dec}$, and $\mathbf{b}_{dec}$ are learned parameters. LLM transcoders are trained to minimize both the MSE between $\mathbf{h}'$ and $\tilde{\mathbf{h}'}$ and the norm of $\mathbf{z}$[2]. The LLM is frozen, and the transcoder trains on up to billions of tokens. The reduction in polysemanticity is achieved by setting the sparse representation size to be much larger than the input size. In doing so, one reduces the pressure on the model to cram many features into a small number of dimensions, as is thought to cause polysemanticity [10].

We interpret the ith feature of a given transcoder by displaying the inputs that maximize its activation $\mathbf{z}_i$. We also display the tokens whose unembedding vectors have the highest and lowest dot product with the feature's column in $\mathbf{W}_{dec}$; these are the vocabulary items that it directly up- and downweights. See Figure 1 for example feature visualizations, used to manually label features.

We often intervene with respect to transcoder features, to verify our interpretation of a given feature. To do so, we take the original feature vector $\mathbf{z}$ and perform desired interventions on it by e.g. zeroing out a feature's activation, yielding $\mathbf{z}'$. We compute $\Delta = \mathbf{W}_{dec}\left(\mathbf{z}' - \mathbf{z}\right)$, and add Δ to the output of the corresponding MLP during the model's forward pass.

Qwen-3 Transcoders For our experiments, we use Hanna et al.'s [14] Qwen-3 transcoders. These circuits are ReLU transcoders, all with a hidden dimension of 163840. They take in MLP inputs post-input-LayerNorm, and predict the MLP's outputs.

D.2 Transcoder Feature Circuits

Formally, feature circuits are weighted acyclic digraphs. The source nodes are input embeddings and nodes corresponding to each transcoder's reconstruction error $\tilde{\mathbf{h}'} - \mathbf{h}'$. These flow through transcoder feature nodes, to nodes that correspond to a given vocabulary item's logit. Each edge's weight is the direct effect of the source node on the target, i.e. the source node's effect on the target's value, unmediated by other nodes.

We compute feature circuits using Ameisen et al.'s [1] algorithm, which works as follows.

Local Replacement Model The first step of attribution is to incorporate the transcoders into the model's computations for a given input. We thus replace the model's MLPs with their corresponding

[2]This is often done by penalizing $\mathbf{z}$'s L_1 norm. However, note that some activation functions, namely Top-k and variants, inherently limit the number of active features, making this unnecessary.

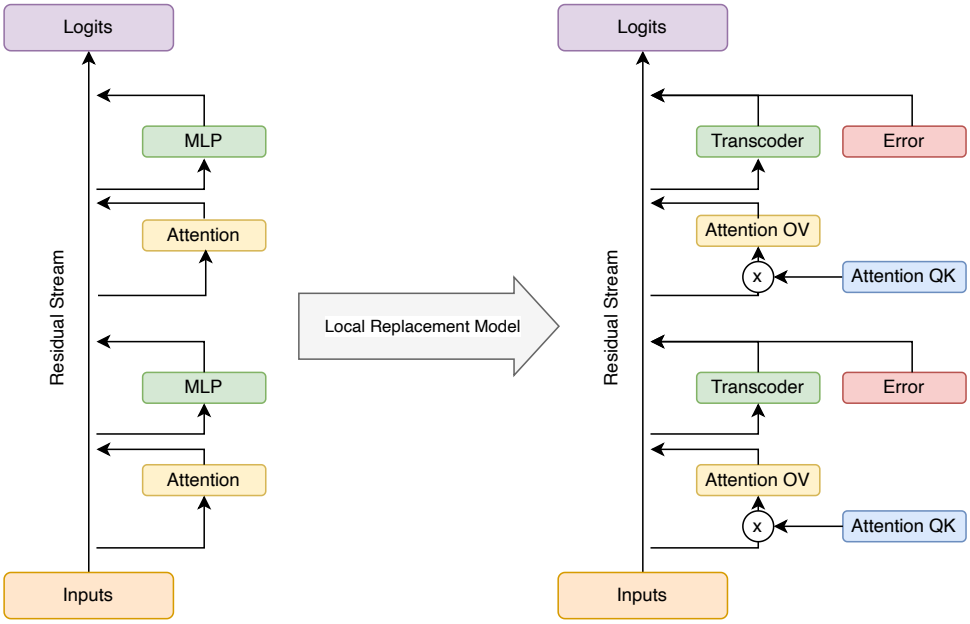

Figure 9: A 2-layer transformer LM, and its corresponding local replacement model. We replace model's MLPs with transcoders, as well as error terms unique to the given input. The attention patterns (from the QK matrix) have been frozen, detaching them from the computation graph. Despite this, the OV-matrix of each attention block is still attached. Thus, when we refer to e.g. the direct effect of a feature of the layer-0 transcoder on a vocabulary logit, this direct effect may pass through the residual stream alone, or additionally through the OV matrix of the attention, a linear transformation. The direct effect of any given feature on any other feature (or any vocabulary logit) is thus linear. See Elhage et al. [9] for more on QK/OV matrices and the residual stream.

transcoders, plus a reconstruction error term equal to the difference between the MLP's output and the transcoder's reconstruction. This yields a *local replacement model*, which behaves identically to the original model, but only on the given input, as reconstruction error terms are input-specific.

We next freeze the model's attention patterns and denominators of any layer normalization terms, treating them as constant values; this entails detaching them from the graph (`.detach()` in Pytorch). See Figure 9 for a depiction of this process. We also detach the transcoder feature activations themselves, so no gradients flow through them.

In so detaching these components, we remove all nonlinearities from our local replacement model: both the attention softmax nonlinearity and the normalization nonlinearities are gone. The activation of any given feature is now linear in the activations of the nodes prior to it. This simplifies the process of computing the direct effect of one node on another, and means that these direct effect values are exact; however, they will not account for features' impact on the model's attention *patterns*.

Attribution We can thus compute the direct effects of a source node on a target node as follows. We define an input vector for the target node: if the node is a feature, this is its input vector (from $\mathbf{W}_{enc}$), and if the node is a logit, this is the corresponding unembedding vector, minus the mean unembedding vector. We inject this gradient at the node's input location—either the MLP input for transcoder features, or the final residual stream for logit nodes; this injection can also be operationalized as a dot product with the residual stream, followed by a `.backward()` call. Then, for each upstream node, its direct effect is the gradient at its output location, multiplied by its output vector: the input embedding or error vector for input and error nodes respectively, or the source feature's activation multiplied by its decoder vector, for feature nodes. With each call of `.backward()`, we find weights for all edges into the target node; repeating this for all nodes attributes the whole attribution graph.

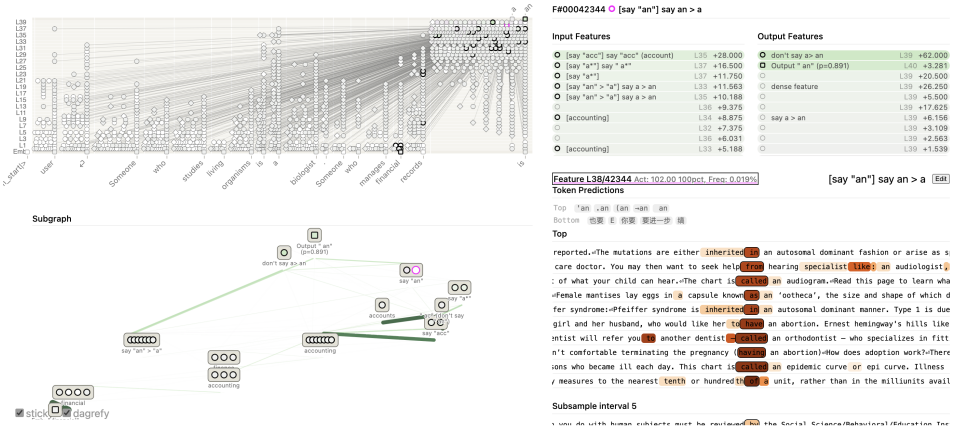

Figure 10: The interface used for circuit visualization / annotation, from `circuit-tracer`.

Methods We limit attribution to the top 7500 most influential feature nodes, as remaining nodes are unlikely to be important, and attributing from many nodes leads to large graphs that fit poorly in memory. We determine which nodes are most influential by intermittently computing each node's influence using the procedure described in Ameisen et al. [1]. For logit nodes, we choose to attribute from the minimum required to capture 0.95 of the model's next-token probability, or the top 10 logit nodes, whichever is smaller (generally the former). Ultimately, the attribution process is quick, from seconds for Qwen-3 (0.6B) to a minute or two for Qwen-3 (14B).

For visualization purposes, it is often useful to prune graphs, removing low-influence nodes and edges. As done by Ameisen et al. [1], we do so by computing the total influence of each node and edge in the circuit. We then set a threshold for each, and take the minimum number of top nodes / edges that sum to that influence; we choose nodes whose influence sums to 80% of the total, and edges whose influence sums to 98%. Our circuit-finding interface, provided by `circuit-tracer` [14], is shown in Figure 10.

E Animal Probing and Intervention Experiments

As done by Dong et al. [6] in their, we set up probing experiments as follows. We take 1000 stories from the validation set of Tinystories, and extract the first sentence. We then feed each first sentence to the model in the following prompt. *Here's the first sentence of a story:* $\{$*sentence1*$\}$. *Continue this story with one sentence that introduces a new animal character.* We then generate (greedy sampling) a next sentence, and recorded the animal contained therein.

We then filter this data down to only the datapoints containing the top-4 most common animals; typically, this leaves 600 or more examples. We then split the data 60/20/20 into train, validation, and test, and collected (transformer layer output) activations from the last token of each prompt. We then train a single-layer MLP probe to predict the animal that the model would predict, from these activations. We use a hidden dimension of 64 for our MLPs, as Dong et al. report that performance plateaus at $d = 64$. We run this analysis on all Qwen3 models, as well as on Llama-3-8B-Instruct, used by Dong et al., and report results across hidden layers in. Figure 11 shows that our results on Llama-3 (8B) are similar to the original findings, with high F1 scores (0.6-0.7) across all layers but the first. Probing results for other models are varied; Qwen3-8B and 32B perform well (F1 near 0.6), while other models exhibit middling performance (F1 < 0.5).

We then verify the causal relevance of the features found by these probes. If the probe has found a causally relevant feature at the end of the prompt that determines the animal that is output, altering that feature should alter the animal that is output. There are a variety of interventions that could be used to verify the features found by the probe: Ravfogel et al. [28] intervening by reflecting representations across probe decision boundaries, while Giulianelli et al. [13] compute the gradient of the probe's prediction (error) with respect to the model representations, and update the representations based on this. We could also use less probe-specific interventions like difference in means [20].

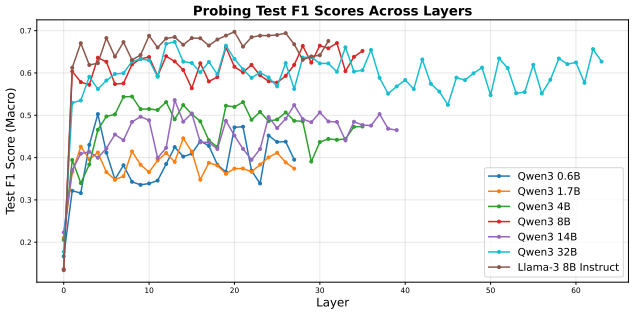

Figure 11: Macro F1 scores when probing models' last-token representations for the animal that the model will output in the following sentence, by model layer. F1 scores are high for certain models—Llama-3 (8B) and Qwen-3 (8B/32B)—but notably lower for others.

We opt for a simpler intervention: we pair each prompt in our dataset with a random prompt that led to the production of a distinct animal. We then generate a continuation to the first prompt, but patch the last-token activations of the second prompt onto the last token of the first prompt. We do so at all layers, effectively replacing all model activations at this position. This means that the next generated token is guaranteed to be the next token of the second prompt; furthermore, attention back to the patched position will receive the patched values. Since we have patched all possible layers in which the relevant features could reside, this intervention should cause the model to produce the animal from the second prompt, if the probed features are indeed relevant. We perform this intervention across the same set of models as the previous experiment, with the exception of Qwen3-32B, as it is missing from TransformerLens [23], the interpretability framework used for this study.

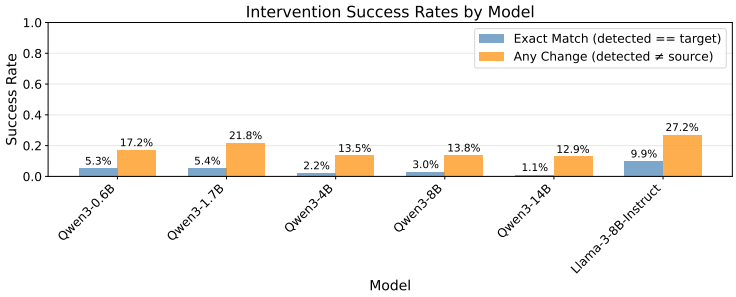

Figure 12: Success rates of our patching intervention, where we patch one prompt (p_2)'s last token activations onto another (p_1)'s last token during generation. We report both exact match (True if the output animal is p_2's animal) and any change (True if the output animal differs from p_1's original animal). In general, exact match is low, below 10%, while any change is higher, but under 30%.

Our results (Figure 12) suggest that the features found are not highly causally relevant. In relatively few cases ($< 10\%$ for all models) do we observe the output animal change that of the second prompt. In fact, in the majority of cases, the output animal does not change at all. This seems to be a violation of our Condition 1, that the found feature must have a causal impact on the model's planned token. We note, however, that Llama-3 (8B), the only model from Dong et al. that we test, does have higher intervention efficacy. Moreover, if there are multiple features relevant for planning the animal to be produced, it would be necessary to find and intervene on all of these to produce a strong effect.

Despite this, we find it unlikely that planning takes place in this scenario. This is because the continuations corresponding to each animal output are generic: they do not hint to animal that will be produced. Consider, for example, the prompt and continuation *Mia and Dad were busy polishing their car…As they worked, a small, curious* ***fox*** *darted into the garage, tail wagging playfully.* The left context of *fox* imposes few constraints on the animal that is to follow; many animals can be *small* and *curious*. This hints that our Condition 2 may not be fulfilled here either: the model does not actually have to plan / prepare a context that licenses the animal eventually output.

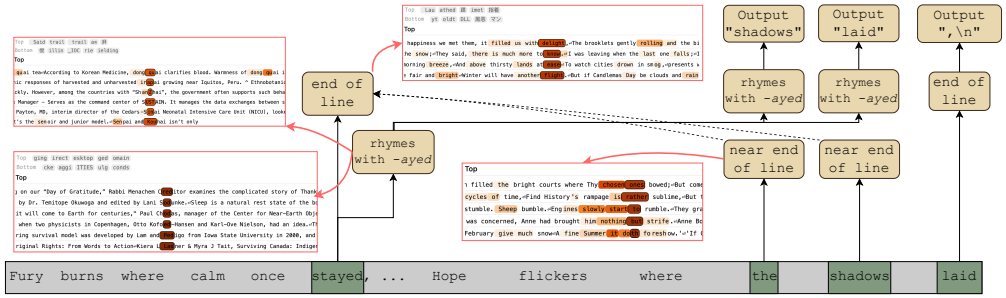

Figure 13: A feature circuit for the couplet *Fury burns where calm once stayed,\n Hope flickers where the shadows laid*, explaining Qwen-3 (14B)'s decision to output *shadows laid,\n*. Halfway through outputting the couplet's second line, the model's "near end of a line of poetry" features activate. These cause it to attend back to the end of the first line, where "end of a line of poetry" features are active, and to move "rhymes with *-ayed*" features into the second line. These influence the model's outputs, eventually leading to *laid*. *End of line* features then cause it to output .\n.

F Couplet Completion Experiments

As discussed in the main test, to test whether models plan when completing couplets, we use transcoder circuits. For each model, we filter the examples from our dataset to those where the model completes the couplet's second line with a rhyming word. We then attribute from this rhyming word's logit, given the input leading up to the rhyming word; that is, given an input like *Fury burns where calm once stayed,... Hope flickers where the shadows laid*, we find the circuit explaining the model's prediction of *laid*. We limit this to 100 examples per model.

We qualitatively analyze the circuits, and find that in larger models, an interpretable circuit emerges. Given the first line of the couplet, the model begins to generate the second with little planning. Near the end of the second line, the model recognizes that it is near the end of a line of rhyming poetry, activating *near end of line* features. These cause it to attend to the end of the first line, drawn by the *end of line* features active there. Rhyming features (e.g. *rhymes with "-ayed"*) at the end of the first line thereby activate similar features in the second line. There, these features remain active until they eventually cause the model to output a rhyming token. Once the model completes the rhyme, it activates *end of line* features and stops generation. Figure 13 depicts this process.

Definition of rhyming features We define rules to automatically find rhyming features. This is challenging, as Qwen-3 models represent e.g. an *-ayed* feature via separate *-ai-* and *-d* sound features, which specify the vowel and final consonant of the rhyme. The top-activating tokens for such features tend to be subwords, and may employ multiple, potentially nonstandard spellings for a given sound; see Figure 13 for example features. As a heuristic, we identify features whose top-10 max-activating tokens are short (under 5 characters), and do not represent a single word (they activate on the same word at most 5 times). We also require that at least 7 of these 10 tokens start with the same vowel, or end with the same consonant, to ensure that the feature's top-tokens all represent one sound. This definition captures rhyming-relevant features with relatively high precision but only moderate recall.

G Larger Models May Use Local Planning Features

Though the couplet planning results are mostly negative, results for larger models (8B-14B) trend in the right direction: they more accurately predict the steered rhyme given the steered context, and less accurately predict the original rhyme; in App. F, we see that their steered generations overlap less with original generations. Moreover, manually inspecting Qwen-3 (14B) couplet-completion circuits showed that while most couplet circuits involve second-line rhyming features that upweight rhyming words, some instead involve *say X* features that upweight a specific upcoming word. These often coincide with rhymes that require some setup, such as a *say "night"* feature occurring before the model outputs *in the night*. These are prime candidates for *local* planning features, that plan for

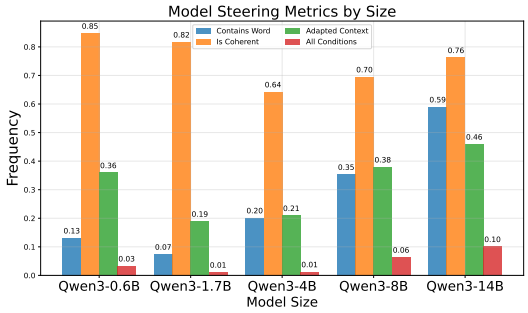

Figure 14: Adaptation metrics by model. As models grow, so does the proportion of (1) outputs containing *X* (blue), and (3) coherent and *X*-containing outputs that also adapt the context to license *X* (green). Few examples do all three.

short phrases, but not whole lines; we thus test whether they elicit backward planning in models, as in Condition 2.

We first identify potential planning features, searching our couplet circuits for *say X* features that upweight the output rhyming word, but are active prior to when *X* is output: such features might adapt the preceding context to license that word. We then steer models using these features on 100 inputs from the TinyStories dataset [8], which we use as a source of neutral input text. For each steered output, we check if it (1) contains the steered word, (2) is coherent, and (3) adapted the context to fit the steered word. We evaluate (1) programmatically, use Claude 4 Sonnet to evaluate (2), and manually verify (2) and evaluate (3) on a subset of outputs that satisfy (1) and (2).

We find (Figure 14) that steering on these *say X* features often induces models to output *X* (blue bars). Moreover, for outputs that are coherent and contain *X*, models—especially larger ones—do adapt their outputs to produce whole phrases like *in the night* or *had a recurring dream* (green bars). The scaling trend likely occurs because larger models have more such local planning features in their couplet circuits. However, this phenomenon is sensitive to steering strength, and these features occur only in a small minority of couplets. We hypothesize that such features are part of an emerging planning mechanism in larger models, much as *a/an* and *is/are* planning can be seen to emerge in Qwen-3 (4B); at larger scale, models may more reliably engage in local planning. Still, more study is needed to confirm the role these features play.