Backward Chaining Circuits in a Transformer Trained on a Symbolic Reasoning Task

Jannik Brinkmann[†1] **Abhay Sheshadri**[†2] **Victor Levoso**[†3] **Paul Swoboda**[4]
Christian Bartelt[1]
[1]University of Mannheim [2]Georgia Institute of Technology [3]Independent
[4]Heinrich-Heine University of Düsseldorf

Abstract

Transformers demonstrate impressive performance on a range of reasoning benchmarks. To evaluate the degree to which these abilities are a result of actual reasoning, existing work has focused on developing sophisticated benchmarks for behavioral studies. However, these studies do not provide insights into the internal mechanisms driving the observed capabilities. To improve our understanding of the internal mechanisms of transformers, we present a comprehensive mechanistic analysis of a transformer trained on a synthetic reasoning task. We identify a set of interpretable mechanisms the model uses to solve the task, and validate our findings using correlational and causal evidence. Our results suggest that it implements a depth-bounded recurrent mechanisms that operates in parallel and stores intermediate results in selected token positions. We anticipate that the motifs we identified in our synthetic setting can provide valuable insights into the broader operating principles of transformers and thus provide a basis for understanding more complex models.

1 Introduction

Transformer-based language models (Vaswani et al., 2017) demonstrate impressive performance on reasoning[1] tasks (Kojima et al., 2023), mathematical problem-solving (Cobbe et al., 2021), and planning (Huang et al., 2022). However, despite strong performance on certain reasoning benchmarks, it remains unclear to what extent these abilities are a result of actual reasoning or simple pattern memorization (Huang & Chang, 2023). To understand the reasoning capabilities of language models, existing work has focused on developing sophisticated benchmarks for behavioral studies (Tafjord et al., 2021; Saparov & He, 2023; Valmeekam et al., 2023). However, the conclusions drawn from these studies *do not provide insights into the internal mechanisms* driving the observed capabilities. In contrast, recent work in the field of mechanistic interpretability attempts to understand the algorithms that models implement by reverse-engineering their internal mechanisms, and describing them at a certain level of abstraction (Elhage et al., 2021). For example, Nanda et al. (2023a) reverse-engineered how a small transformer model implements modular addition, providing insights into the specific computations performed by different components of the model. Similarly, Olsson et al. (2022) discovered "induction heads" in transformers, which enable a distinct copying mechanism that is considered to be crucial for the in-context learning abilities of language models.

Contributions This paper studies reasoning in language models by reverse-engineering the internal mechanisms of a transformer trained on a symbolic multi-step reasoning task. Specifically, we focus on path finding in a tree, a variation of the task proposed in Saparov & He (2023). By analyzing the internal representations of the model, we identify several key mechanisms:

1. A specific type of copying operation implemented in attention heads, which we call *deduction heads*. These are similar to induction heads as observed in Olsson et al. (2022). In our context,

[†] Equal contribution. Correspondence to jannik.brinkmann@uni-mannheim.de. The code is available at github.com/abhay-sheshadri/backward-chaining-circuits.

[1] In this paper, we consider a form of deductive reasoning as studied in Saparov & He (2023). For a discussion of different forms of reasoning, refer to Huang & Chang (2023).

deduction heads intuitively serve the purpose of moving one level up the tree. These heads are implemented in multiple consecutive layers which allows the model to climb the tree multiple layers in a single inference step.

2. A *parallelization motif* whereby the early layers of the model choose to solve several subproblems in parallel that may be relevant for solving harder instances of the task.
3. A *heuristic* for tracking the children of the current node and whether these children are leaf nodes of the tree. This mechanism is used when the model is unable to solve the problem using deduction heads in parallel.

We validate our findings using correlational and causal evidence, using techniques such as linear probing (Alain & Bengio, 2018), activation patching (Vig & Belinkov, 2019), and causal scrubbing (Chan et al., 2022).

2 Methods

Linear Probes. To investigate whether information is encoded in intermediate representations of the model, we use linear probes (Alain & Bengio, 2018) implemented as a linear projection from the residual stream. This involves training a logistic regression model on a dataset of activations $\mathbf{x}_i^\ell$ to predict an auxiliary classification problem, where $\mathbf{x}_i^\ell$ are the activations at position i in layer ℓ.

Activation Patching. To evaluate the importance of a model component for a given prediction, we intervene by patching in the activations it would have had on a different input (also called resampling ablations) (Vig et al., 2020; Meng et al., 2022). This involves using a clean input s with an associated target prediction r, and a corrupted input s' with a different target r'. Then, we cache the activations of the component on s, and evaluate the effect of patching in these activations when running the model on s'. To evaluate the effect of this intervention, we compute the difference in logits: $\text{LD}(r, r') = \text{Logit}(r) - \text{Logit}(r')$.

Causal Scrubbing. To evaluate specific hypotheses about internal mechanisms, we use causal scrubbing which evaluates the effect of behavior-preserving resampling ablations (Chan et al., 2022). Specifically, given a hypothesis about which component of a model implements a specific behavior, we replace the activations of that component on some input with activations on another input, where our hypothesis predicts that the activations represent the same thing. Then, we evaluate the impact of this intervention by computing how much of the initial performance is recovered. In contrast to activation patching, which provides insights about whether a specific activation is causally linked to the output, causal scrubbing provides stronger evidence about the role of activations.

3 Experimental Setup

Task Description We focus on path finding in trees as a modified version of the task proposed by Saparov & He (2023). Our adaptation shifts the focus from reasoning in natural language to abstract symbolic reasoning. This allows us to better understand motifs that the models might be using to solve analogous problems in natural language. In our experimental setup, we generate training samples by generating binary trees uniformly at random from the set of all trees with 16 nodes. Then, for each tree, a leaf node is randomly selected as the goal node (see Figure 1). The model is given the edge list $\texttt{[A}_1\texttt{][B}_1\texttt{]}$, $\texttt{[A}_2\texttt{][B}_2\texttt{]}$, ... $\texttt{[A}_n\texttt{][B}_n\texttt{]}$, the selected goal node $\texttt{[G]}$, and the root node $\texttt{[P}_1\texttt{]}$, and should predict the path from the root node to the goal $\texttt{[P}_2\texttt{][P}_3\texttt{]}$ ... $\texttt{[P}_m\texttt{]}$ such that $\texttt{[P}_m\texttt{]} = \texttt{[G]}$, as shown in Figure 2. Thus, to predict the next step in the path, the model must perform multiple reasoning steps in a single forward pass without relying on techniques such as chain-of-thought or scratchpad (Wei et al., 2022), making the task non-trivial.

Model and Training In our experiments, we use a 6-layer, decoder-only transformer with an embedding dimension of 128, a single attention head per layer, and a feed-forward dimension of 512, resulting in a total of 1.2 million parameters. The training dataset consists of 150,000 generated trees. The edge lists of these trees are shuffled to prevent the model from learning simple heuristics and encourage structural understanding of trees. To evaluate the performance of our model, we compute the accuracy based on the exact match of complete sequences using greedy decoding. Our

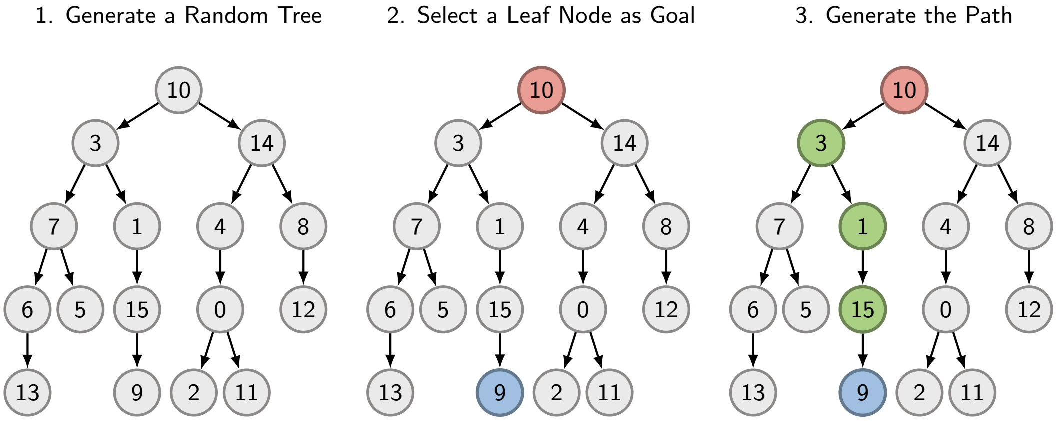

Figure 1: Data Generation. To generate our training set, we (1) generate a binary tree, (2) select a random leaf node as the goal node, and (3) determine the path from the root to the goal node.

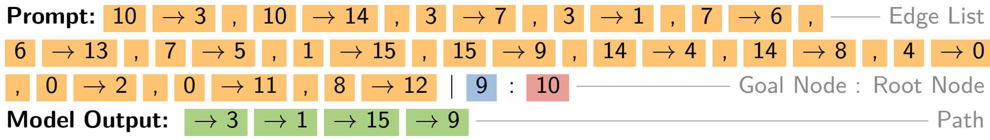

Figure 2: Prompt Format. The model receives input in a structured format, with each box representing a token. The edge list of the tree is denoted as token pairs $[\mathtt{A}_1][\mathtt{B}_1], \ldots, [\mathtt{A}_n][\mathtt{B}_n]$, followed by the task specification, including the goal [G] and the root node $[\mathtt{P}_1]$. The model's objective is to predict the nodes in the path $[\mathtt{P}_2] \ldots [\mathtt{P}_m]$, culminating in the goal node $[\mathtt{P}_m]$ = [G]. For simplification, our tokenization distinguishes tokens representing source and target nodes of each edge, such as [15] and [→15].

model achieves 99.7 % accuracy on a test set of 15,000 unseen trees, despite seeing just a small fraction of all possible trees during training (see Appendix C). This suggests that generalization is required for meaningful performance and that the model is capable of solving the task.

4 Results

In this section, we present a mechanistic interpretation of the internal mechanisms of the model and provide correlation and causal evidence for these mechanisms. Our findings suggest that the model uses an interpretable and meaningful backward chaining algorithm to perform pathfinding in a tree. To help guide the reader, we present an intuitive explanation before breaking down the individual algorithmic steps.

The Backward Chaining Algorithm First, the model aggregates the source and target nodes of each edge in the edge list into the target node position (see Appendix D.1). Then, the model starts at the goal node and moves up the tree one level with each layer of the model. This mechanism is implemented using specific attention heads, which we term "deduction heads" (see Section 4.1). By the composition of multiple deduction heads in consecutive layers, the model can traverse the tree upwards for up to $L - 1$ edges, where L is the number of layers in the model. We refer to this mechanism as backward chaining, inspired by the use of the term in the symbolic artificial intelligence literature (Russell & Norvig, 2009). In more complex scenarios, where the required path exceeds the depth of the model, it relies on backward chaining from multiple nodes in the tree in parallel (see Section 4.2). This creates multiple subpaths that can be combined to find the correct next step. In addition, the model uses a simple heuristic as a fallback mechanism, where it identifies child nodes of the current position and evaluates whether these are leaf nodes of the tree. This enables the model to make informed guesses when backward chaining alone is insufficient to solve the problem (see Appendix D.2).

4.1 Backward Chaining using Deduction Heads

The most important mechanism the model uses to predict the correct next step is an iterative algorithm, which we refer to as backward chaining. The algorithm starts at the goal node and climbs the tree one step at a time. To this end, the model copies the target node `[G]` into the final token position `[P_i]` and then applies what we call "deduction heads" in each subsequent layer.

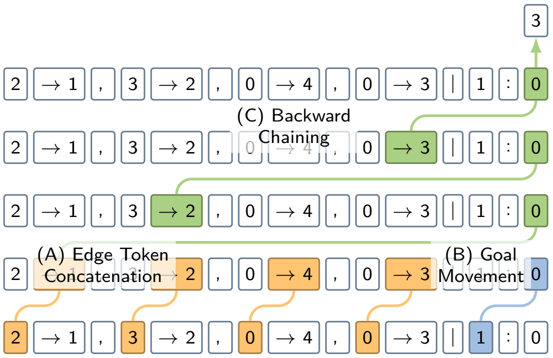

Figure 3: Backward Chaining. Given an input prompt, the model concatenates edge tokens in a single token position (A), and copies the goal node into the final token position (B). The next step is then identified by applying an iterative algorithm that climbs the tree one level per layer (C).

Mechanism: Deduction Heads The function of deduction heads is to search for the edge in the context in which the current position is the target node `[B]`, find the corresponding source token `[A]`, and then copy the source token over to the current position. Thus, deduction heads complete the pattern by mapping `[A]` `[B]` ... `[B]` → `[A]`. In other words, this mechanism enables the model to go one step up the tree and append `[A]` after having seen the last `[B]` in the sequence. This mechanism depends on edge-token concatenation, which copies information about the source token `[A]` onto the target token `[B]` for each edge in the context (see Appendix D.1).

By composition of multiple deduction heads in consecutive layers, the model is able to climb multiple steps up the tree. This creates a subpath at the final token position whose lengths is equivalent to the number of layers involved. In our model, we observe that the attention heads after the first layer can act as deduction heads, resulting in a backward chaining depth of at most $L - 1$ steps.

Experiment: Causal Scrubbing To confirm that the model uses backward chaining to predict the next step for paths up to a depth of $L - 1$, we use causal scrubbing (see Section 2). Specifically, we hypothesize that the attention head of layer ℓ is responsible for writing the node that is $\ell - 1$ edges above the goal to the final token position in the residual stream. This implies that the output of the attention head in layer ℓ should be consistent across trees that share the same node $\ell - 1$ edges above the goal. To test this, we generate a clean and corrupted graph that share the same node $\ell - 1$ edges above the goal node. Then, we substitute the output of the head on the clean graph with the output of the head on the corrupted graph, and measure the difference in the logits.

Results Figure 4 illustrates the effect of this intervention. We find that we can recover most of the performance (almost 100 %) of the model for paths up to a length of $L - 1$, providing strong evidence in favour of our backward chaining hypothesis. We also find that this hypothesis explains the behaviour of the attentional heads in the first four layers of the model for paths with more than $L - 1$ steps; only the attentional heads in the last two layers behave significantly differently, such that the model ends up making incorrect predictions after we apply causal scrubbing.

4.2 Path Merging

In cases where the goal is more than $L - 1$ steps away from the current position, backward chaining from the goal node is insufficient. To address this, we find that the model performs backward chaining in parallel from multiple different positions in the tree and combines the resulting subpaths to facilitate more complex scenarios.

Mechanism: Path Merging To compute paths for which backward chaining on the final token position is not sufficient, the model performs backward chaining in parallel from multiple positions in the tree. To this end, it selects intermediate goals nodes in the tree and performs backward chaining from them. To store the resulting subpaths, the model identifies tokens that do not contain any useful

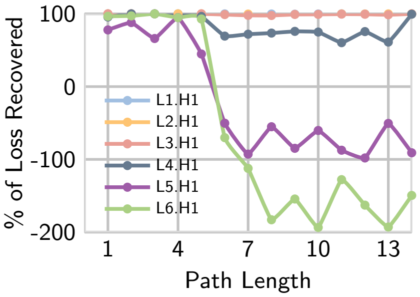

Figure 4: To test whether the model uses backward chaining, we perform causal scrubbing. We find that we can recover close to 100 % of the performance for paths up to length $L-1$, providing strong evidence for our backward chaining hypothesis.

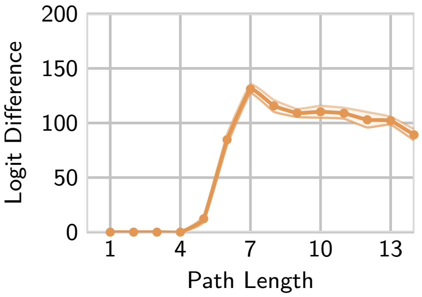

Figure 5: To test whether the model relies on subpaths stored in register tokens, we perform resampling ablations on the register token positions. Our findings demonstrate that these subpaths are causally relevant for predictions on paths of length greater than $L-1$.

information for the actual task and uses them to store intermediate results (see Appendix D.5). We refer to these tokens as *register tokens*. This results in a multiple subpaths being stored at different positions in the sequence. Then, the model can combine these by finding overlapping subpaths. To illustrate, assuming that a subpath `[B]`→`[C]`→`[G]` has been stored in the final token position and a different subpath `[A]`→`[E]`→`[B]` has been stored in some register token, the model combines these subpaths on the final token position, enabling it to move the tree up multiple steps at a time.

Experiment: Register Token Patching To evaluate whether the subpaths stored in the register tokens have a causal effect on the prediction of the model, we perform resampling ablations on the register token positions in (i) trees which contain sufficiently long paths such that simple backward chaining is insufficient, and (ii) positions where nodes have multiple child nodes, ensuring that the model has to decide between multiple options. The corrupted activations are extracted from another tree in the same class. Then, we compute the effect of this intervention using the logit difference.

Results Figure 5 illustrates the impact of patching register tokens on the model predictions at different path lengths. Our results show that the intervention has no effect on performance up to a path depth of four and minimal effect at depth five, which is consistent with our backward chaining hypothesis. Beyond this depth, this intervention has a significant effect on the performance. This suggests that the encoded subpaths are causally relevant for predicting next steps on paths that are more than $L-1$ steps away from the goal. However, our findings also indicate that the predictions are not solely dependent on these subpaths derived, but other factors besides the subpaths contribute to the prediction. This includes the influence of a one-step lookahead mechanism, which identifies child nodes of the current position and increases the probabilities of the children that are not leaf nodes of the tree (see Appendix D.2). This enables the model to make informed guesses in cases where backward chaining alone is not sufficient to solve the problem.

5 Conclusion and Discussion

Our findings in this synthetic setting demonstrate the ability of a transformer to perform deductive reasoning up to a certain reasoning depth, after which it resorts to simple heuristics. By using parallelized computations to store intermediate results in register tokens and then combining these results on the final token position, the model demonstrates a form of multi-step deductive reasoning that, while effective within a given setting, is constrained by architectural inductive biases. This observation suggests that transformers may exhibit a inductive bias towards adopting highly parallelized strategies for tasks involving search, planning, or reasoning.

Acknowledgement

Jannik Brinkmann is supported by the German Federal Ministry for Digital and Transport (BMDV) and the German Federal Ministry for Economic Affairs and Climate Action (BMWK). Abhay Sheshadri and Victor Levoso have been supported by Lightspeed Grants.

References

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2018.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens, 2023.

Emily M. Bender and Alexander Koller. Climbing towards NLU: On meaning, form, and understanding in the age of data. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5185–5198, Online, July 2020. Association for Computational Linguistics.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, pp. 610–623, New York, NY, USA, 2021. Association for Computing Machinery.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7096–7116. Association for Computational Linguistics, November 2020.

Eugène Charles Catalan. Note sur une équation aux différences finies. *Journal de Mathématiques Pures et Appliquées*, 3:508–516, 1838.

Lawrence Chan, Adrià Garriga-Alonso, Nicholas Goldowsky-Dill, Ryan Greenblatt, Jenny, Ansh Radhakrishnan, Buck, and Nate Thomas. Causal Scrubbing: a method for rigorously testing interpretability hypotheses. 2022.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. Neural networks learn representation theory: Reverse engineering how networks perform group operations. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

Arthur Conmy, Augustine N. Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability, 2023.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers, 2023.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural networks and the chomsky hierarchy, 2023.

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.

Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4):681–694, 2020.

Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. Dissecting recall of factual associations in auto-regressive language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Olga Golovneva, Moya Chen, Spencer Poff, Martin Corredor, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. Roscoe: A suite of metrics for scoring step-by-step reasoning, 2023.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens, 2023.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gäel Varoquaux, Travis Vaught, and Jarrod Millman (eds.), *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pp. 11–15, Pasadena, CA USA, Aug 2008.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, David Peng, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Shafiq Joty, Alexander R. Fabbri, Wojciech Kryscinski, Xi Victoria Lin, Caiming Xiong, and Dragomir Radev. Folio: Natural language reasoning with first-order logic, 2022.

Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 1049–1065, Toronto, Canada, July 2023. Association for Computational Linguistics.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9118–9147. PMLR, 17–23 Jul 2022.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023.

Kevin Meng, David Bau, Alex J Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022.

William Merrill, Yoav Goldberg, Roy Schwartz, and Noah A. Smith. Provable limitations of acquiring meaning from ungrounded form: What will future language models understand? *Transactions of the Association for Computational Linguistics*, 9:1047–1060, 2021.

William Merrill, Ashish Sabharwal, and Noah A. Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856, 2022.

Neel Nanda and Joseph Bloom. Transformerlens. `https://github.com/neelnanda-io/TransformerLens`, 2022.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023a.

Neel Nanda, Senthooran Rajamanoharan, János Kramár, and Rohin Shah. Fact finding: Attempting to reverse-engineer factual recall on the neuron level, Dec 2023b. URL https://www.alignmentforum.org/posts/iGuwZTHWb6DFY3sKB/fact-finding-attempting-to-reverse-engineer-factual-recall.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads, 2022.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021.

Stuart Russell and Peter Norvig. *Artificial intelligence*. Pearson, Upper Saddle River, New Jersey, 3 edition, December 2009.

Tilman Räuker, Anson Ho, Stephen Casper, and Dylan Hadfield-Menell. Toward transparent ai: A survey on interpreting the inner structures of deep neural networks, 2023.

Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*, 2023.

Alessandro Stolfo, Yonatan Belinkov, and Mrinmaya Sachan. A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7035–7052, Singapore, December 2023. Association for Computational Linguistics.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. Transformers as recognizers of formal languages: A survey on expressivity, 2023.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 3621–3634. Association for Computational Linguistics, August 2021.

Curt Tigges, Oskar John Hollinsworth, Atticus Geiger, and Neel Nanda. Linear representations of sentiment in large language models, 2023.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model. In Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes (eds.), *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 63–76, Florence, Italy, August 2019. Association for Computational Linguistics.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 12388–12401. Curran Associates, Inc., 2020.

Boshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun. Towards understanding chain-of-thought prompting: An empirical study of what matters. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2717–2739, Toronto, Canada, July 2023a. Association for Computational Linguistics.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023b.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020.

Dylan Zhang, Curt Tigges, Zory Zhang, Stella Biderman, Maxim Raginsky, and Talia Ringer. Transformer-based models are not yet perfect at learning to emulate structural recursion, 2024.

A Extended Discussion of Related Work

Expressiveness of Transformers To understand the capabilities of transformers, one line of work characterizes their theoretical properties (Bhattamishra et al., 2020; Merrill et al., 2021; Pérez et al., 2021; Merrill et al., 2022; Liu et al., 2023). These studies answer questions about the expressiveness of the transformer architecture by treating them as approximators of different classes of functions (Yun et al., 2020), or by characterizing the class of formal languages that a transformer can recognize (Strobl et al., 2023).

Mechanistic Interpretability In contrast to these theoretical investigations, a number of studies have adopted an empirical approach in order to understand what transformers learn in practice Elhage et al. (2021); Delétang et al. (2023); Zhang et al. (2024). Our analysis is inspired by existing work in the field of mechanistic interpretability, attempting to discover and understand the algorithms implemented in a model by reverse-engineering its internal mechanisms (Räuker et al., 2023). To explore these internal mechanisms, the field has adopted techniques such as activation patching Wang et al. (2023b), causal scrubbing (Chan et al., 2022), and circuit discovery (Conmy et al., 2023). In addition, considerable focus has been placed on the study of small models trained on specialized tasks, such as modular addition (Nanda et al., 2023a), or group operations (Chughtai et al., 2023), providing a more manageable framework for understanding complex computational processes.

Evaluating Reasoning Capabilities Existing approaches to evaluate the reasoning capabilities of language models focus on their performance on a range of downstream tasks (Huang & Chang, 2023). To enable a more formal analysis of reasoning, a number of studies have developed sophisticated metrics and benchmarks (Han et al., 2022; Golovneva et al., 2023; Wang et al., 2023a). For example, Saparov & He (2023) use a synthetic question-answering dataset based on a world model expressed in first-order logic to parse the generated reasoning processes into symbolic proofs for formal analysis. Their results suggest that language models are capable of making correct individual deduction steps. However, these approaches stop short of exploring the internal mechanisms that enable these capabilities (Huang & Chang, 2023). The approach that comes closest to our work is Stolfo et al. (2023), presenting a mechanistic interpretation of arithmetic reasoning by investigating the information flow in the model given simple mathematical questions.

B Transformer Notation

Transformers (Vaswani et al., 2017) represent input text as a sequence $t_1, t_2, \dots, t_N$ of tokens, such that $t_i \in V$ where V is a vocabulary. Each token t_i is embedded as a vector $\mathbf{x}_i^0 \in \mathbb{R}^d$ using an embedding matrix $W_E \in \mathbb{R}^{|V| \times d}$, where d is the dimension of the hidden state. These embeddings initialize the residual stream, which is then transformed through a sequence of L transformer blocks, each consisting of a multi-head attention sublayer and an MLP sublayer. The representation of token t_i at layer ℓ is obtained by:

$$\mathbf{x}_i^\ell = \mathbf{x}_i^{\ell-1} + \mathbf{a}_i^\ell + \mathbf{m}_i^\ell \tag{1}$$

where $\mathbf{a}_i^\ell$ and $\mathbf{m}_i^\ell$ are the outputs of the attention and MLP sublayers. To predict the next token in the sequence, it applies an unembedding matrix $W_U \in \mathbb{R}^{|V|\times d}$ to the residual stream $\mathbf{x}_i^L$, translating it into a probability distribution over the vocabulary.

C Experimental Setup

C.1 Implementation and Computing

All experiments were carried out on a single NVIDIA RTX A6000 GPU. The total computation time for training the transformer model was less than 24 hours. To generate the trees, we used `networkx` (Hagberg et al., 2008). For training and execution of all experiments, we used `TransformerLens` (Nanda & Bloom, 2022). For details on the model and training configuration, see Tables 1 and 2.

C.2 Size of Training Set

Our dataset consists of 150,000 randomly generated examples, each including a labeled binary tree with 16 nodes. The number of possible *unlabeled* binary trees with $n + 1$ nodes is given by the n-th Catalan number (Catalan, 1838):

$$C(n) = \frac{(2n)!}{(n+1)! \cdot n!}$$

When considering *labeled* binary trees, this number grows to $(n + 1)! \cdot C(n)$ unique trees. This suggests that memorization is infeasible, and generalization is required for meaningful performance.

C.3 Training Configuration

Table 1: Training Configuration

Parameter	**Value**
Learning Rate	1e-3
Optimizer	`AdamW`
Batch Size	64
Betas	(0.9, 0.99)
Weight Decay	0.01

C.4 Model Configuration

Table 2: Model Configuration

Parameter	**Value**
Number of Layers	6
Number of Heads	1
Residual Stream Dim.	128
Attention Head Dim.	128
Feed-Forward Dim.	512
Activation Function	`gelu`
Vocabulary Size	35
Context Size	63

D Extended Discussion of Results

D.1 Edge Token Concatenation

The attention head in the first layer of the model creates edge embeddings by moving the information about the source token onto the target token for each edge in the context. Thus, for each edge `[A] [B]` it copies the information from `[A]` into the residual stream at position `[B]`. This mechanism has some similarities with "Previous Token Heads", as observed in pre-trained language models (Olsson et al., 2022; Wang et al., 2023a).

Experiment: Linear Probes To validate that the model creates edge embeddings, we train a linear probe to predict the associated edge given the activations $\mathbf{x}^1$ at the positions of target nodes. The probe is trained using 8,000 examples and evaluated on a test dataset of the same size. For comparison, we also report the performance of a linear probe given the activations $\mathbf{x}^0$ at the positions of the target nodes and probes given the activations at the positions of the source node.

Table 3: Performance of linear probes trained to predict the edge `[A][B]` given the residual stream activations at position `[A]` or `[B]`.

	$\mathbf{x}_i^0$	$\mathbf{x}_i^1$
Linear $\{\texttt{[A]} \rightarrow \texttt{[A][B]}\}$	0.13	0.19
Linear $\{\texttt{[B]} \rightarrow \texttt{[A][B]}\}$	0.11	**1.00**

Results Table 3 reports the performance of the linear probe measured using the F1 score. We find that we can successfully extract the source and target token (`[A] [B]`) from the residual stream activations $\mathbf{x}^1$ at the position of the target tokens after the first layer, providing strong evidence for the edge token concatenation hypothesis as described above. Moreover, it does not encode the complete edge in the position of the source token, attributed to the causal masking in the attention mechanism.

D.2 One-Step Lookahead

We find that the model uses an additional mechanism, which identifies child nodes of the current position and increases the prediction probabilities of the children that are not leaf nodes of the tree. This enables the model to make informed guesses in cases where backward chaining is not sufficient. This mechanism is particularly effective on long paths as these have a lower branching factor in our experimental setup. Thus, it is a pragmatic strategy to minimize the training error.

Experiment: Linear Probes To validate that the model represents the child nodes of the current position, including whether these are leaf nodes, we use linear probes. These probes are trained to predict this information given the activations on the final token position. Table 4 reports the performance of the linear probes measured using the F1 score. Our analysis shows that the model starts to collect information about the child and leaf nodes from the fourth layer and represents both aspects in the fifth layer.

	$\mathbf{x}_i^4$	$\mathbf{x}_i^5$	$\mathbf{x}_i^6$
Linear $\{\texttt{[P}_\texttt{i}\texttt{]} \rightarrow \texttt{[Childs}_\texttt{i}\texttt{]}\}$	0.00	47.88	**98.20**
Linear $\{\texttt{[P}_\texttt{i}\texttt{]} \rightarrow \texttt{[Leafs}_\texttt{i}\texttt{]}\}$	0.00	49.71	**95.76**

Table 4: F1 score of linear probes trained to predict the children of the current position and whether these are leafs of the tree given the residual stream activations at position $\texttt{[P}_\texttt{i}\texttt{]}$.

Experiment: Causal Scrubbing To evaluate the impact of the mechanism, we perform causal scrubbing such that it incorporates the aforementioned mechanisms. We reuse the experimental setup from Section 4.1 but add additional constraints to our resampling scheme. We avoid resampling the contributions of the target node and register token positions through the attention heads.

The results are illustrated in Figure 6. We find that we can recover most of the performance of the model for paths across the full training distribution, providing strong evidence that the observed mechanisms together account for most of the model behavior.

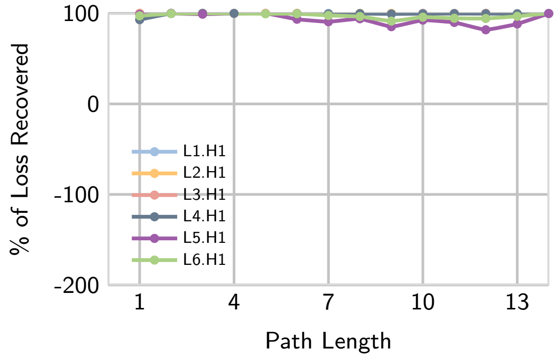

Figure 6: To test the effect of the one-step lookahead, we perform causal scrubbing such that it accounts for this mechanism. We find that we can recover most of the performance of the model for paths across the full training distribution, providing strong evidence that the observed mechanisms together account for most of the model behavior.

Mechanism for Computing Children and Leaf Nodes We provide some insights into how the model determines which nodes are the children of the current token and which of those are not leaf nodes of the tree. We find that attention heads `L5.H1` and `L6.H1` are responsible for this task, as suggested by Table 4. We examine how these attention heads directly compose with the unembedding matrix. We compute the contribution of each target node position to the logits at the path position through the attention heads `L5.H1` and `L6.H1`. The results are shown in Figure 7.

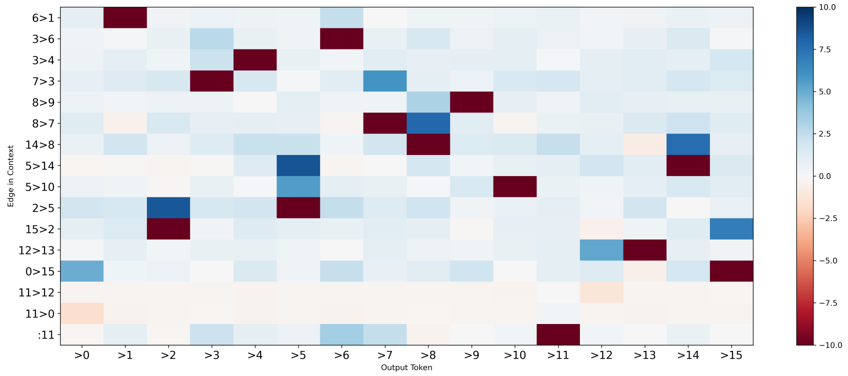

Figure 7: Direct logit attribution from the output of the path position to the target node positions. We sum the contributions of `L5.H1` and `L6.H1` on a specific example.

The QK-circuits of these two heads attend to the target node of every edge, except those for which the source node is the current path position. Since both nodes of the edge will be represented in the target node position (see Section D.1), we will refer to the target node as the edge. We break down the mechanism in the OV-circuits of these heads into three components:

1. Each edge decreases the logit of its target node.
2. Each edge increases the logit of its source node.
3. Each token in the path decreases its logit.

As a result of these mechanisms, the logits of the leaf nodes in the graph will decrease while the logits of the children will increase. For the other nodes, the logit increase from being a parent, and the logit decrease from being a child node roughly cancel out, causing their logit to remain the same.

D.3 Attention Patterns

Here, we visualize attention patterns of our model on a few example inputs to provide intuition for the backward chaining mechanism. In each example, we highlight the attention from the final token position and the register tokens that are causally relevant for the prediction. To determine these, we use attention knockout Geva et al. (2023) on the register token positions. This prevents the final token from attending to register tokens by zero-ing out the attention weights. This allows us to test whether critical information propagates from them. More formally, let $a, b \in [1, N]$ be two positions such that $a <= b$, we block $\mathbf{x}_b^l$ from attending to $\mathbf{x}_a^\ell$ at layer $\ell < L$ by updating the attention weights to that layer:

$$M_{ab}^{l+1} = -\infty \forall j \in [1, H] \tag{2}$$

Thus, this restricts the source position from obtaining information from the target position, at that particular layer. To avoid information leakage across positions, we block attention edges in multiple layers rather than a single one. Specifically, we block attention to the register tokens in the final two layers of the model.

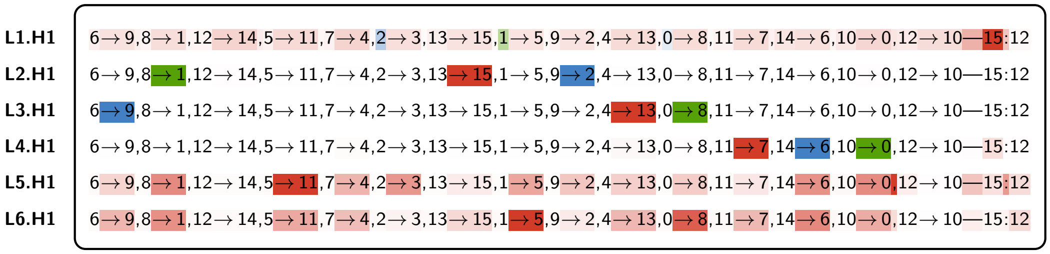
L1.H1 6→ 9,8→ 1,12→ 14,5→ 11,7→ 4,2→ 3,13→ 15,1→ 5,9→ 2,4→ 13,0→ 8,11→ 7,14→ 6,10→ 0,12→ 10—15:12
L2.H1 6→ 9,8→ 1,12→ 14,5→ 11,7→ 4,2→ 3,13→ 15,1→ 5,9→ 2,4→ 13,0→ 8,11→ 7,14→ 6,10→ 0,12→ 10—15:12
L3.H1 6→ 9,8→ 1,12→ 14,5→ 11,7→ 4,2→ 3,13→ 15,1→ 5,9→ 2,4→ 13,0→ 8,11→ 7,14→ 6,10→ 0,12→ 10—15:12
L4.H1 6→ 9,8→ 1,12→ 14,5→ 11,7→ 4,2→ 3,13→ 15,1→ 5,9→ 2,4→ 13,0→ 8,11→ 7,14→ 6,10→ 0,12→ 10—15:12
L5.H1 6→ 9,8→ 1,12→ 14,5→ 11,7→ 4,2→ 3,13→ 15,1→ 5,9→ 2,4→ 13,0→ 8,11→ 7,14→ 6,10→ 0,12→ 10—15:12
L6.H1 6→ 9,8→ 1,12→ 14,5→ 11,7→ 4,2→ 3,13→ 15,1→ 5,9→ 2,4→ 13,0→ 8,11→ 7,14→ 6,10→ 0,12→ 10—15:12

Figure 8: Visualization of multi-layer attention patterns on an example input. We show the attention from three selected positions: the path position, register token at position 39, and register token at position 44. We show that the path node starts backward chaining from the specified goal, while the two register tokens start backward chaining from different subgoals. Each token is highlighted by the color of the token that most strongly attends to it. The intensity of the color is based on the magnitude of the attention score.

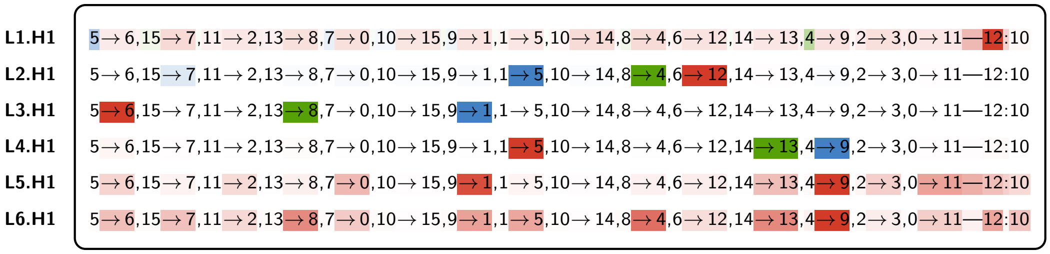
L1.H1 5→ 6,15→ 7,11→ 2,13→ 8,7→ 0,10→ 15,9→ 1,1→ 5,10→ 14,8→ 4,6→ 12,14→ 13,4→ 9,2→ 3,0→ 11—12:10
L2.H1 5→ 6,15→ 7,11→ 2,13→ 8,7→ 0,10→ 15,9→ 1,1→ 5,10→ 14,8→ 4,6→ 12,14→ 13,4→ 9,2→ 3,0→ 11—12:10
L3.H1 5→ 6,15→ 7,11→ 2,13→ 8,7→ 0,10→ 15,9→ 1,1→ 5,10→ 14,8→ 4,6→ 12,14→ 13,4→ 9,2→ 3,0→ 11—12:10
L4.H1 5→ 6,15→ 7,11→ 2,13→ 8,7→ 0,10→ 15,9→ 1,1→ 5,10→ 14,8→ 4,6→ 12,14→ 13,4→ 9,2→ 3,0→ 11—12:10
L5.H1 5→ 6,15→ 7,11→ 2,13→ 8,7→ 0,10→ 15,9→ 1,1→ 5,10→ 14,8→ 4,6→ 12,14→ 13,4→ 9,2→ 3,0→ 11—12:10
L6.H1 5→ 6,15→ 7,11→ 2,13→ 8,7→ 0,10→ 15,9→ 1,1→ 5,10→ 14,8→ 4,6→ 12,14→ 13,4→ 9,2→ 3,0→ 11—12:10

Figure 9: Visualization of multi-layer attention patterns on an example input, similar to Figure 8. We show the attention from three different positions: path position, register token at position 39, and register token at position 45.

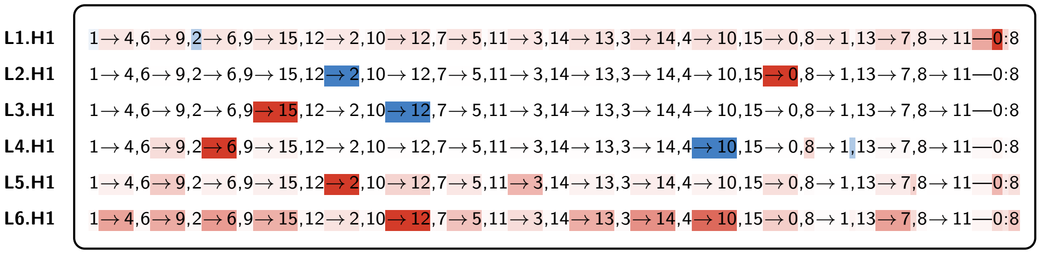

Figure 10: Visualization of multi-layer attention patterns on an example input, similar to Figure 8. We show the attention from three different positions: path position and register token at position 41

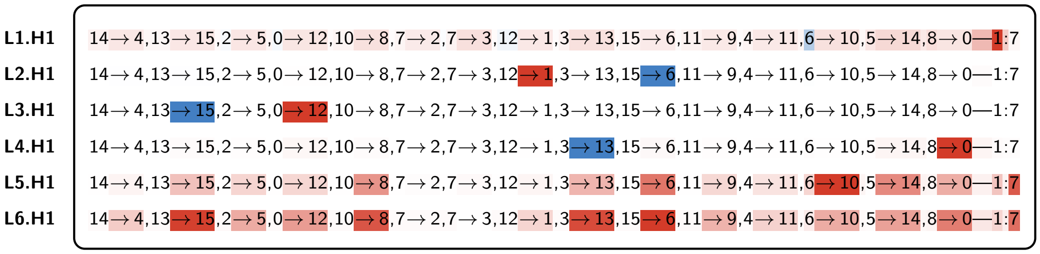

Figure 11: Visualization of multi-layer attention patterns on an example input, similar to Figure 8. We show the attention from three different positions: path position and register token at position 36

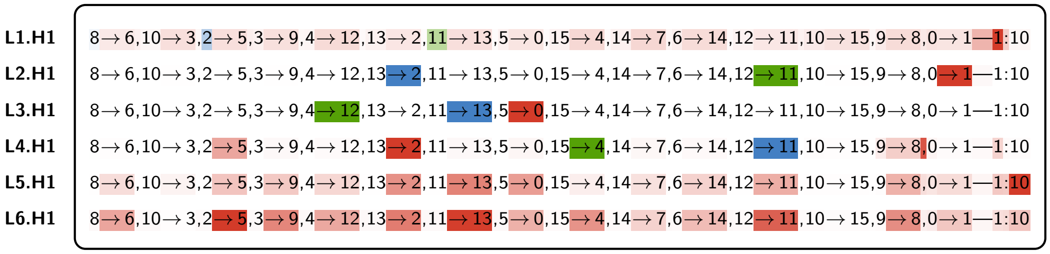

Figure 12: Visualization of multi-layer attention patterns on an example input, similar to Figure 8. We show the attention from three different positions: path position, register token at position 41, and register token at position 42.

D.4 Attention Head Composition

In this section, we perform additional experiments to verify that `L1.H1` performs edge token concatenation, and `L2.H1` is a deduction head. Elhage et al. (2021) show that transformers can learn induction heads in two different ways involving different compositions: the K-composition, where the W_K of the head reads from the output of the previous head, and V-composition, where the W_V of the head reads from the output of the previous head. Our findings suggest that the deduction heads we found in our model are a result of K-composition. In the following subsections, we adopt the conventions of Elhage et al. (2021).

Layer 1 - Edge Token Concatenation Head `L1.H1` serves as a variation of a previous token head studied in Elhage et al. (2021), effectively transferring source node information to the corresponding target node in each edge. This is captured in the QK-circuit:

$$M^0_{QK} = W_P \, W^0_{QK} \, W_P^T$$

M^0_{QK} (see Figure 13) shows that the attention values are maximized when the query vector corresponds to the position embedding of an incoming node, and the key vector corresponds to the position embedding of the immediately preceding outgoing node. Then, following the goal node at position 45, the head persistently attends to the goal position.

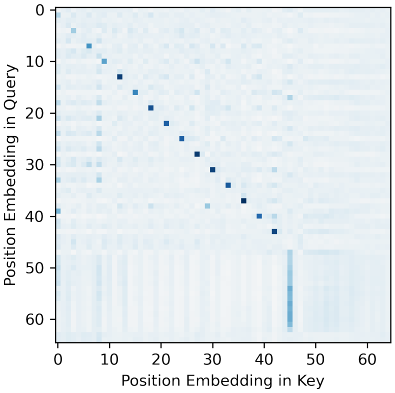

Figure 13: Visualization of M^0_{QK}

Layer 2 - Deduction Head `L2.H1` is a deduction head, which attends to the target node that matches the goal at positions in the path. It then moves information about the source node of that edge into the last position in the window. It can be viewed as a reverse induction head (Olsson et al., 2022) that uses a K-composition (Elhage et al., 2021) to map a sequence `[A]` `[B]` ... `[A]` → `[B]`, where `[B]` represents target nodes and `[A]` represents source nodes. This can again be verified by looking at the QK-circuit:

$$M^1_{QK} = (\text{MLP}^0(W^0_{OV} W_E) + W^0_{OV} W_E) W^1_{QK} W_E^T$$

This matrix shows the interactions of the embedding of the source and target tokens at layer 2 (see Figure 14). Our analysis is complicated by the fact that our model is not attention-only, as attention heads can compose with each other through the MLP, which makes similar analyses in later layers of the model intractable. However, our causal scrubbing results provide evidence that the attention heads in the subsequent layers implement a similar mechanism to `L2.H1`, but use the output of the previous layer's attention head to backward chain further up the tree.

D.5 Register Tokens and Subgoals

In this section, we provide more details on the role of the register tokens in our model. From each register token position, the model attends to a random node in the context and starts backward-chaining from that node. The initial selection of a node by the register token can be viewed as identifying a *subgoal*, from which the model can perform backward chaining. This precomputation

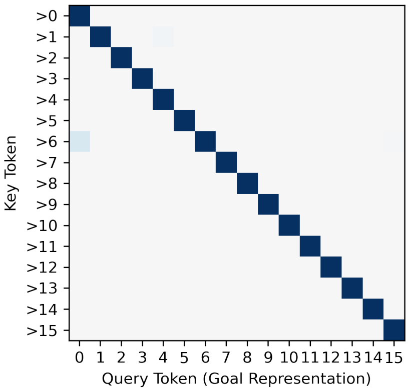

Figure 14: Visualization of a subset of M_{QK}^1 showing interactions between source and target node tokens.

occurs before the actual goal is specified and occurs fully parallel to the main backward-chaining mechanism. These findings hint that transformers may exhibit an inductive bias towards learning highly parallelized algorithms when trained to perform search, planning, or reasoning.

To see whether there is any structure in the selection of subgoals, we empirically study which node the register tokens select as subgoals across 1000 samples. The results are illustrated in Figures 15.

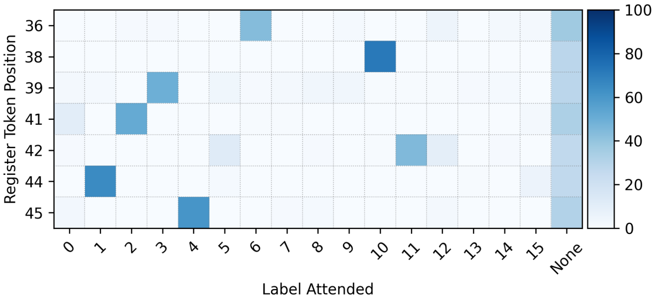

Figure 15: Preferences in Subgoal Selection (1): Ratio of register tokens attending to different source tokens aggregated across 1000 samples. We consider a register token to select a subgoal based on an attention threshold of 0.3.

We observe that the model usually attends to the same tokens, e.g. position 36 attends to token `[6]` most of the time. However, we observe an interesting dynamic in which the register token selects a different subgoal in two cases:

1. If the node doesn't occur before the register token position, it cannot attend to it due to causal masking.
2. If the node is a leaf node of the tree since it doesn't have a corresponding source token to attend to.

To validate this, we again examine the probability of the model selecting subgoals in trees where the most common subgoal occurs before the register token position and is not a leaf node (see Figure 16).

Further exploration reveals that the subgoals selected by each register token position can be somewhat understood through an examination of the embedding matrices. We evaluate a selection of seven register token positions that are used on several different examples and show their preferred subgoals. By composing the embedding and position embedding matrices with the QK-circuit of the first layer's attention head, we define R_P as:

$$R_P = W_P W_{QK}^1 W_E^T$$

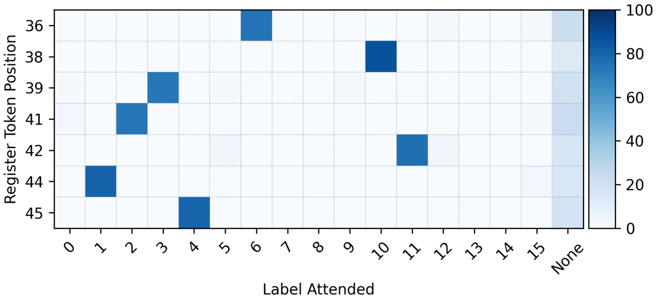

Figure 16: Preferences in Subgoal Selection (2): Ratio of register tokens attending to different subgoals aggregated across 1000 samples on trees where the node is attends to most often is not a leaf node of the tree and occurs before the register token. We again consider a register token to select a subgoal based on an attention threshold of 0.3.

where W_E is the embedding matrix, W_P is the position embedding matrix, and W_E^1, W_Q^1 are the key and query projection matrices of `L1.H1`. This explains how the model selects subgoals, by having the key for each positional embedding of a register token match with some specific source node token.

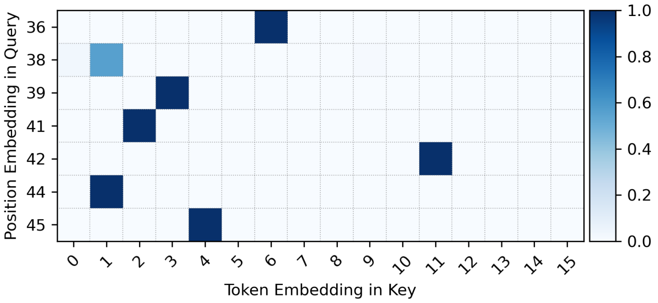

Figure 17: Plot of R_P

E Extended Discussion

Register Tokens Our model uses some token positions as a form of working memory to store intermediate results. This observation aligns with Darcet et al. (2023) which found that image models use some image patches to accumulate global information while discarding spatial information. Similarly, Goyal et al. (2023) show that adding uninformative tokens at the end of each prompt can enhance language model performance on downstream tasks without introducing additional parameters, and Tigges et al. (2023) show that models have "summarization tokens" where information about the sentiment of the context is aggregated on tokens that do not have inherent sentiment. Our findings suggest that these techniques enable the model to store more intermediate results and perform more computations in parallel. This is consistent with theoretical insights from Merrill et al. (2022) which highlights how the effective state of a transformer depends on the number of tokens in the sequence.

Structural Recursion Transformers, which are by definition non-recurrent, struggle with emulating structural recursion and extracting recursive rules from training data (Zhang et al., 2024). This aspect of learning is crucial in domains such as programming and formal mathematics, where understanding complex relationships relies on these abilities. Our analysis provides insights into possible reasons for this limitation. In our setting, training the model using standard objectives for next-token prediction forces the model to unroll the entire recursive structure in a single forward pass. This restricts their abilities to process recursion, leading them to resort to shortcut solutions (Liu et al., 2023).

Reasoning in Transformers There is an ongoing debate about the reasoning capabilities of transformers Huang & Chang (2023). Some argue that these models might just be capable of memorizing patterns without gaining causal understanding, which could lead to diminishing performance on out-of-distribution data Bender & Koller (2020); Floridi & Chiriatti (2020); Bender et al. (2021); Merrill et al. (2021). However, there are several observations that suggest that transformers might be capable of more than just pattern recognition; e.g. Olsson et al. (2022) found a simple algorithm implemented in attention heads that contributes to the in-context learning abilities of transformers and applies independent of the specific tokens. This algorithm is doing more than memorizing patterns and can in some sense work out-of-distribution. In our synthetic setting, we found that the model learned an interpretable and meaningful backward chaining algorithm, supporting the claim that transformers might be capable of a form of reasoning that goes beyond simple pattern memorization. However, it is important to note that findings from our synthetic settings do not support the boarder claim that transformers possess general reasoning capabilities, highlighting the need for further investigations.

F Limitations

Synthetic Task Our experiments were conducted on a symbolic reasoning task. This allowed us to bypass the complexities associated with natural language, such as multi-token embeddings (Nanda et al., 2023b). In addition, our tokenization distinguishes tokens representing source and target nodes of each edge, such as `[15]` and `[→15]`. Therefore, our findings are specific to our model and it remains unclear whether large language models trained on natural language use similar mechanisms to solve this task. However, we anticipate that the motifs we discovered in our synthetic setting can provide valuable insights into the broader operating principles of transformers and thus provide a basis for understanding more complex models.

Input Format To prevent the model from learning shortcuts based on the order of the edges in the prompt, we trained our model on shuffled edge lists. However, our analysis is limited to sequences in which the edge list is presented in backward order. By backward order we mean a listing of edges that starts with the leaf nodes and ascends level by level to the root node, as opposed to a forward order where the listing starts with the root node and progresses downwards through each level. Our investigation does not extend to a detailed examination of alternative arrangements of the edge list. However, preliminary observations suggest that the model uses comparable mechanisms with minor variations, such as the use of different register tokens.

G Tuned Lens

In this section, we provide an additional piece of evidence in favor of the existence of the backward chaining mechanism. To understand how the predictions of a transformer are built layer-by-layer, Belrose et al. (2023) develop the Tuned Lens, a method that involves training a linear model to translate the activations from an intermediate layer directly to the input of the unembedding layer.

Inspired by this approach, we replace the last n layers of the model with a linear transformation trained to predict the next token from the residual stream activations $\mathbf{x}^{L-n}$. Similar to the Tuned Lens, this method allows us to skip over these layers and see the current best prediction that can be made from the model's residual stream. Intuitively, this allows us to peek at the iterative computations a transformer uses to compute the next token. Here, we present a visualization of some example trees and the results of the iterative computation (see Figures 18 to 21). These figures highlight the current best prediction a linear transformation could make based on the internal activations. We project the logits output by the linear model back onto the tree structure to better visualize the backward-chaining procedure.

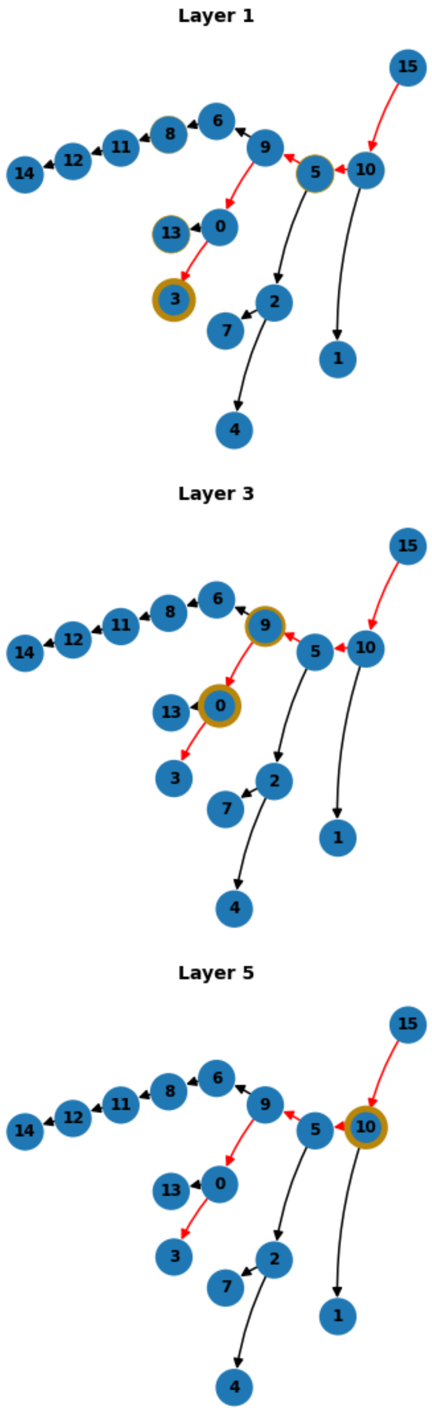

Figure 18: Example 1 (Path Length 5): Results of a linear transformation to predict the next step based on the residual stream activations after each layer, projected onto the tree structure. The yellow border highlights the current best prediction(s) of the linear transformation.

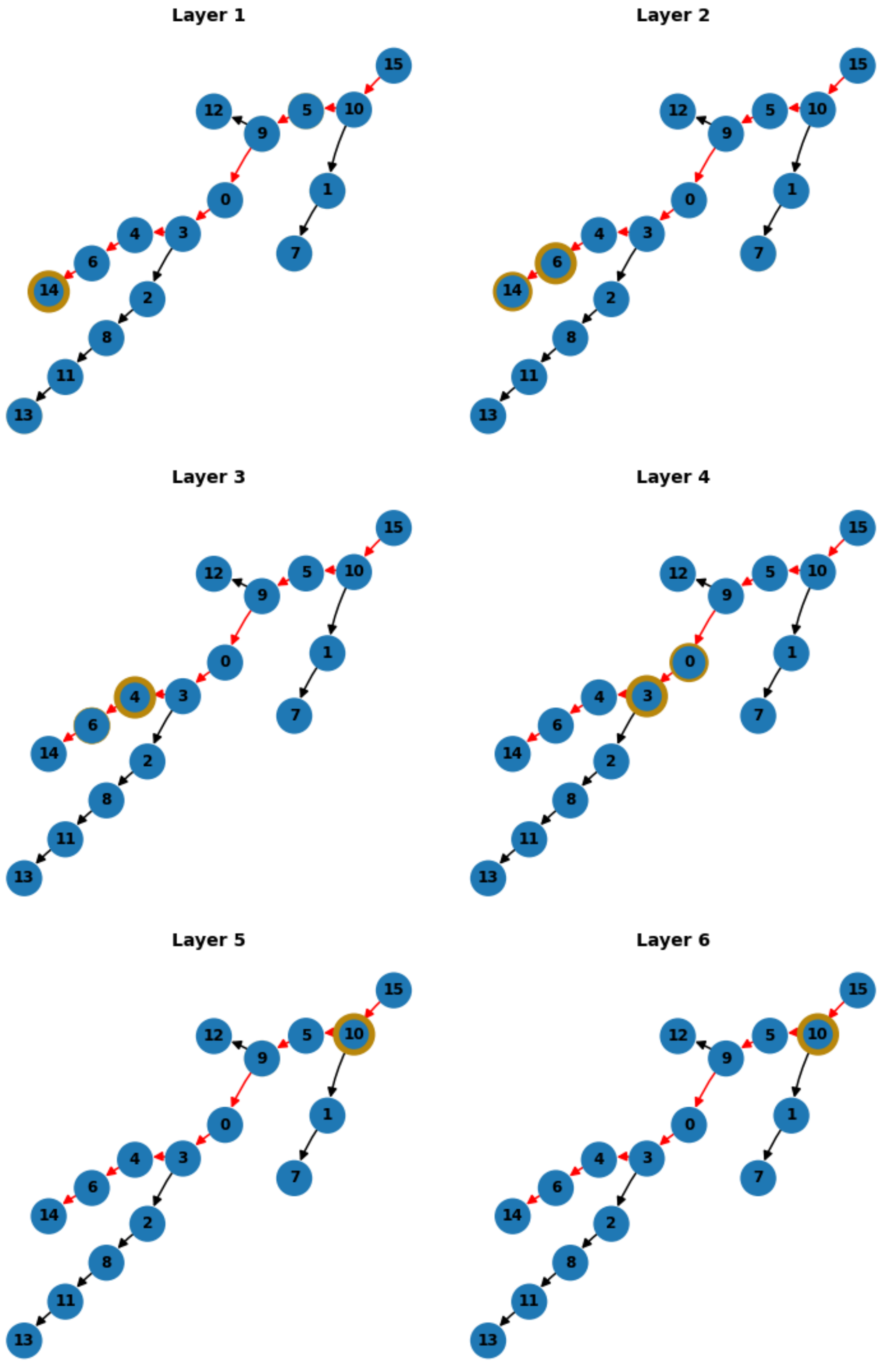

Figure 19: Example 2 (Path Length 8): Results of a linear transformation to predict the next step based on the residual stream activations after each layer, projected onto the tree structure. The yellow border highlights the current best prediction(s) of the linear transformation.

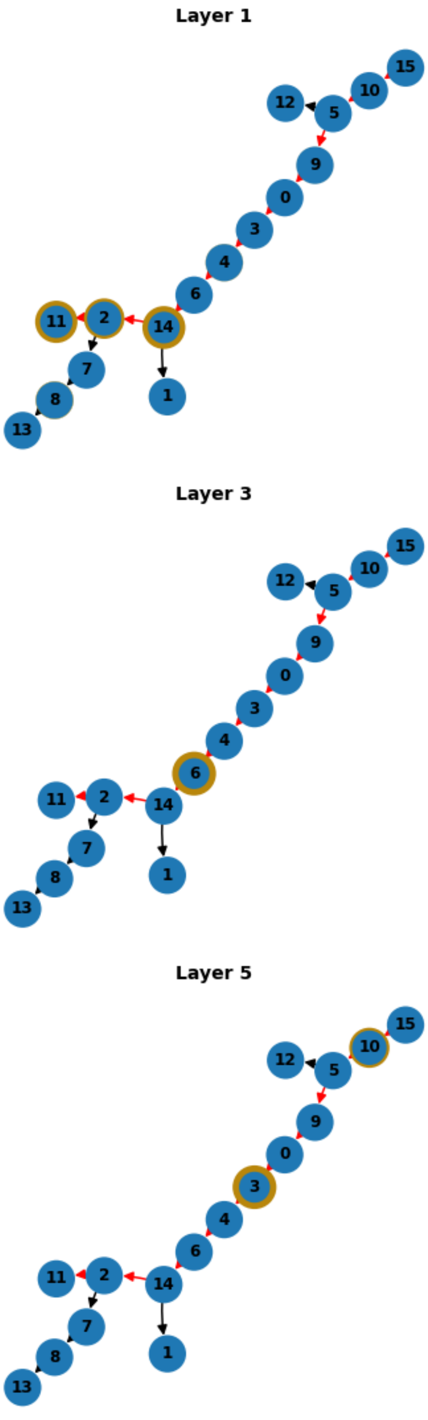

Figure 20: Example 3 (Path Length 10): Results of a linear transformation to predict the next step based on the residual stream activations after each layer, projected onto the tree structure. The yellow border highlights the current best prediction(s) of the linear transformation.

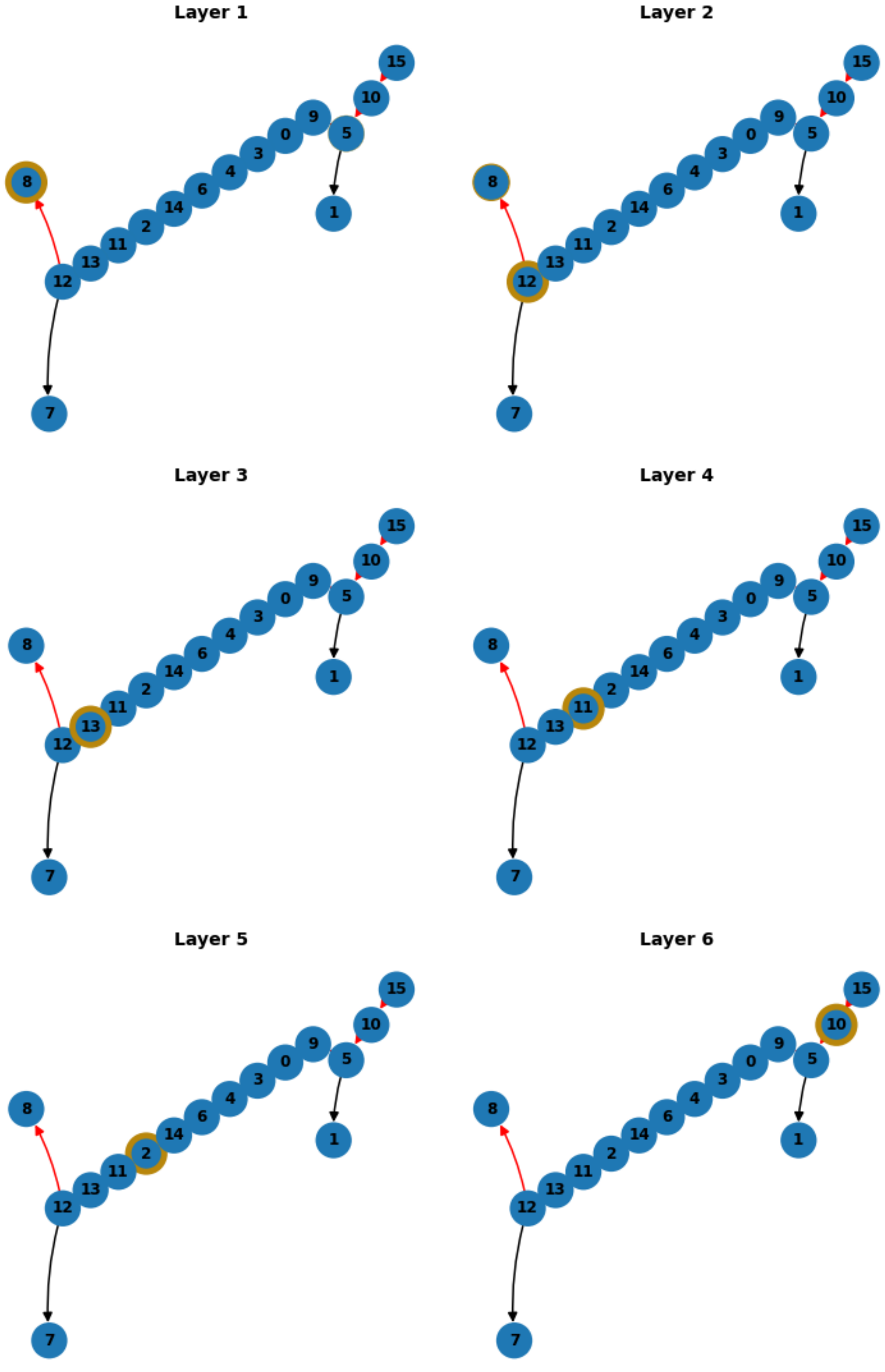

Figure 21: Example 4 (Path Length 13): Results of a linear transformation to predict the next step based on the residual stream activations after each layer, projected onto the tree structure. The yellow border highlights the current best prediction(s) of the linear transformation.