Attribution Patching Outperforms Automated Circuit Discovery

Aaquib Syed
University of Maryland, College Park
asyed04@umd.edu

Can Rager
Independent
canrager@gmail.com

Arthur Conmy
Independent
arthurconmy@gmail.com

Abstract

Automated interpretability research has recently attracted attention as a potential research direction that could scale explanations of neural network behavior to large models. Existing automated circuit discovery work applies activation patching to identify subnetworks responsible for solving specific tasks (circuits). In this work, we show that a simple method based on attribution patching outperforms all existing methods while requiring just two forward passes and a backward pass. We apply a linear approximation to activation patching to estimate the importance of each edge in the computational subgraph. Using this approximation, we prune the least important edges of the network. We survey the performance and limitations of this method, finding that averaged over all tasks our method has greater AUC from circuit recovery than other methods.[1]

1 Introduction

Mechanistic interpretability is a subfield of AI interpretability that focuses on attributing model behaviors to its components, thus reverse engineering the network [1]. This field aims to identify subnetworks (circuits) within the model which are responsible for solving specific tasks [2]. Prior attempts at finding circuits in language models have led to finding networks of attention heads and multi-layer perceptrons (MLPs) that partially or fully explain model behaviors at tasks such as indirect object identification, modular arithmetic, completion of docstrings, and predicting successive dates [3, 4, 5, 6]. However, almost all previous work has been limited to relatively small models since manually applying mechanistic interpretability methods has not currently scaled to end-to-end circuits in larger models [7].

It may be important to scale interpretability to large models as these are the neural networks most widely deployed and used by a wide range of people. Currently, we have little understanding into these models work and failure modes are not always found ahead of deployment. If successful, scaled interpretability could address a wide variety of concerns about the lack of transparency of language models [8], in addition to speculative risks about the alignment of machine learning systems [9].

Automated Circuit Discovery (ACDC; [10]) attempts to automate a large portion of the mechanistic interpretability workflow — the pruning of edges between attention heads and MLPs that do not affect the task being studied. ACDC begins with a computational graph, and recursively calculates the importance of an edge in the graph for a specific task. In our work, we use edges to refer to activations inside models between two components (Section 2 describes this motivation further). ACDC's pruning algorithm applies **activation patching**. (Note that **activation patching** is not **attribution patching**. Both are defined in full in Section 3.3.) At a high level, activation patching edits a specific activation in a model forward pass and measures a model statistic (e.g loss) under this

[1]Our code is available at `https://github.com/Aaquib111/acdcpp`

37th Conference on Neural Information Processing Systems (NeurIPS 2023) ATTRIB Workshop.

intervention. Activation patching is inefficient for circuit discovery because getting each statistic about model activations requires another forward pass. Our work uses **attribution patching** to recover circuits more efficiently (Section 3.3).

Our main contributions are:

1. Introducing a method for using attribution patching on all computational graph edges for automated circuit discovery (Edge Attribution Patching, Section 3.3).
2. Benchmarking Edge Attribution Patching vs existing circuit discovery methods (Section 4).
3. Finding and explaining some limitations with Edge Attribution Patching (Section 5).

2 Related Work

Automated Circuit Discovery refers to finding the important subgraph of models' computational graphs for performance on particular tasks [10]. Existing algorithms include efficient heuristics [11] and gradient-descent based methods [12, 13]. ACDC is related to pruning [14] and other compression techniques [15], but differs in how the compressed networks are reflective of the circuits that model uses to compute outputs to certain tasks and the goal of ACDC is not to speed up forward passes (all techniques studied in this work slow forward passes).

Activation Patching is a technique for analyzing the role of individual components in a model. It involves targeted manipulations of activations during a forward pass (further explained in Section 3.1). Previous works applied this technique under various names, such as Interchange Interventions [16], Causal Mediation Analysis [8] and Causal Tracing [17]. We adapt the terminology used by Conmy et al. [10].

Transformer Circuits. Our work builds upon the framework for understanding transformers for interpretability as introduced by Elhage et al. [18]. The important details include how they formulate forward passes of transformer models. Individual attention heads and MLPs (collectively called nodes) read and write information to a central communication channel, also called the residual stream. In these terms we can examine dependencies of nodes with the output of earlier nodes, i.e we can measure the effect of attention heads in layer 0 on the attention heads in layer 2. In the following, we view these dependencies as edges between nodes, building on existing work using this perspective [5, 6, 3].

3 Edge Attribution Patching

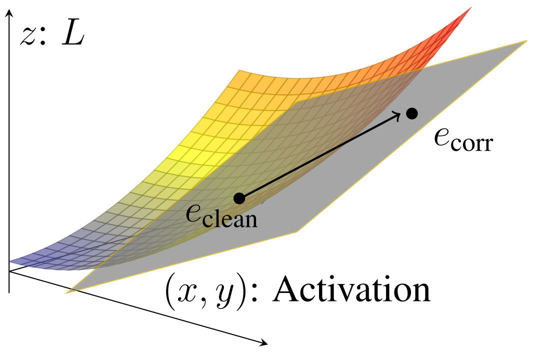
We present **Edge Attribution Patching** (EAP) as a technique to identify relevant model components for solving a specific task. In the following, we view language models as directed, acyclic graphs. In these terms, we aim to find small subgraphs that retain good performance on narrow tasks. We determine the importance of a specific edge through targeted manipulation of activations during a forward pass. We compare two approaches, Attribution Patching and Activation Patching, in order to motivate EAP.

3.1 Activation Patching

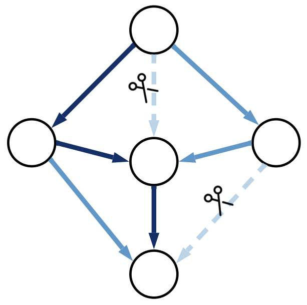
Activation patching refers to replacing the activations from one model forward pass with the activations from a different forward pass. This method is typically applied to measure the counterfactual importance of model components, i.e. to measure a statistic $L(x)$ from model outputs under the activation patching, where x is an input prompt. For example, L often represents loss or logit difference [3].

Following existing work (Section 2), we study the effect of activation patching on specific model edges by setting these equal to activations from different forward passes. Concretely, suppose that an edge E in the computational graph has activation e_{corr} on some corrupted prompt. In this work, we use the change in metric under activation patching

$$|L(x_{\text{clean}}|\,\text{do}(E = e_{\text{corr}})) - L(x_{\text{clean}})| \tag{1}$$



(a) Attribution Patching (Section 3.3) approximates the difference in metric L caused by corrupting edges.



(b) Removing the least important edges.

Figure 1: Edge Attribution Patching (EAP)

to measure the impact of edge E. We use do-notation from causality [19] to emphasise that activation patching is a causal intervention.

3.2 Attribution Patching

Activation patching slows ACDC since each measurement (like Equation (1)) requires another forward pass. ***Attribution patching*** [20] is a technique for estimating Equation (1) for many different edges E using only two forward passes and one backward pass.[2] It linearly approximates the metric difference after corrupting a single edge in the computational graph (Figure 1) by expanding L as a function of the edge activation as a Taylor series with terms up to the first order:

$$L(x_{\text{clean}} | \operatorname{do}(E = e_{\text{corr}})) \approx L(x_{\text{clean}}) + \underbrace{(e_{\text{corr}} - e_{\text{clean}})^{\top} \frac{\partial}{\partial e_{\text{clean}}} L(x_{\text{clean}} | \operatorname{do}(E = e_{\text{clean}}))}_{\text{Call this } \Delta_e L\text{, the } \textbf{attribution score}.} . \tag{2}$$

A simple rearrangement implies that Equation (1) is approximately equal to $|\Delta_e L|$ (3) which we call the **absolute attribution score** for the rest of this paper. In this work we always compute this score across a set of $(x_{\text{clean}}, x_{\text{corr}})$ pairs and take the mean.

In practice, all gradients needed to calculate the attribution scores come from intermediate terms computed in one ordinary backwards pass[3] in PyTorch [21], hence attribution patching is extremely efficient.

3.3 Edge Attribution Patching

We can use the insights from Section 3.2 to build an automated circuit discovery algorithm. This takes two steps: i) use Equation (2) to obtain absolute attribution scores for the importance of all edges in the computational graph and then ii) sort these scores and keep the top k edges in a circuit. We use **Edge Attribution Patching** (EAP) to refer to this algorithm. In the rest of the work we report results for all k values when we evaluate EAP (similar to HISP in [10]).

Note that one limitation of attribution patching is that it will not work when the gradient of the metric is the zero vector. Conmy et al. [10] recommended the use of KL divergence as a metric, which is i) equal to 0 when we run the model without ablations and ii) a non-negative metric. Therefore the zero point is a global minimum and hence all gradients are the zero vector at this point. In this work we use the 'task-specific metrics' (not KL divergence) from [10] so avoid this issue.

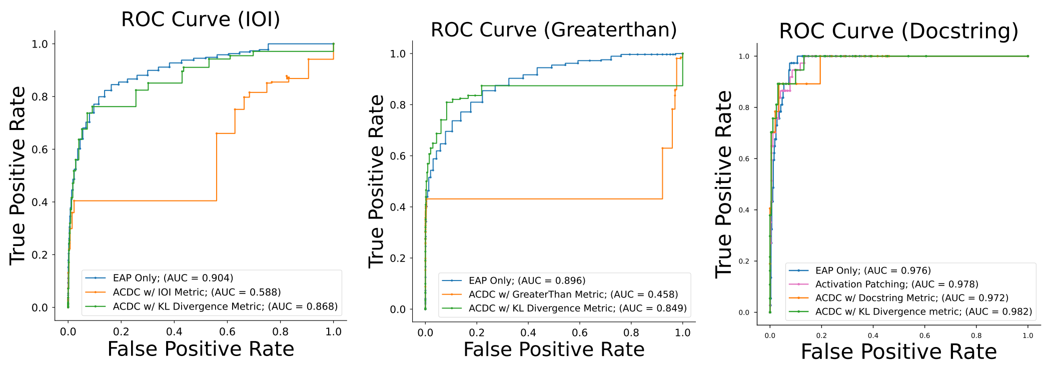
[2]Attribution patching (like activation patching) also applies to nodes and other model internal components that aren't edges, but we only use edges in this work.

[3]In Appendix F we show how only one backwards pass is required.

4 Results

4.1 Edge Attribution Patching vs Activation Patching vs ACDC

We compare Edge Attribution Patching (EAP) and ACDC on the Indirect Object Identification (IOI), Docstring, and Greater-Than tasks. For each of these tasks, previous studies identified a subgraph (circuit) relevant for solving the task. We use their results as a ground truth for benchmarking both methods. We also compare using ACDC with the task-specific metrics (e.g logit difference) and KL Divergence (which was originally recommended). For the docstring task, we also include repeated activation patching as another point of reference for performance comparisons. We applied repeated activation patching by running the same circuit discovery method described in Section 3.3 but using Equation (1) rather than absolute attribution scores. Activation patching was not included in the other tasks as it was too computationally expensive to run on the GPT-2 small models used by IOI and Greater-Than. Subnetworks found using EAP for all three tasks are shown in Appendix A.



(a) IOI task (b) Greater-Than task (c) Docstring task

Figure 2: ROC Curves comparing EAP, ACDC with task metric, and ACDC with KL Divergence. The Docstring plot also compares to Activation Patching.

The ROC curves in Figure 2 suggest the performance of EAP is better than ACDC overall: it has the maximal AUC in Figure 2a-2b, while ACDC used with the KL Divergence metric outperforms EAP in Figure 2c. ACDC outperformed the existing methods HISP and Subnetwork Probing methods [10]. We conclude EAP outperforms all previous methods for circuit discovery, since it is competitive with ACDC on recovering circuits while significantly reducing the computational demand: EAP only takes a constant number of forward and backwards passes while the number of forward passes required by ACDC is scaling exponentially with the number of nodes.

4.2 Validating EAP Attribution Scores

In this section, we look at the approximate metric change (attribution score) EAP assigns to each edge in the model. We aim to understand the relation between the attribution score and the function of the edge in the task being studied. First, we look at the distribution of scores for edges in the circuit compared to edges not in the circuit for each of the three tasks.

Figure 3 shows the distribution of attribution scores for the IOI task. The distributions for the remaining tasks can be found in Appendix B. Qualitatively, attribution scores for edges in the circuit tend to be spread further from zero. Furthermore, there are only 6 edges outside of the interval $[-0.25, 0.25]$ that aren't part of the IOI circuit. We further explore the attribution scores for the IOI circuit's classes of heads in Appendix E.

5 Limitations

We introduced edge activation patching as an approximation to activation patching. However, we found that edge activation patching outperformed ACDC, a technique based on activation patching (Section 4). In this section, we investigate whether attribution patching's success is due to extremely

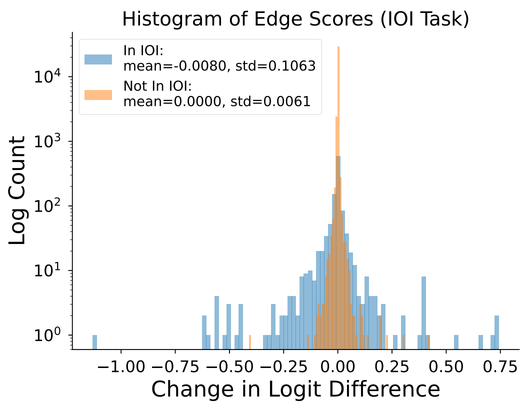


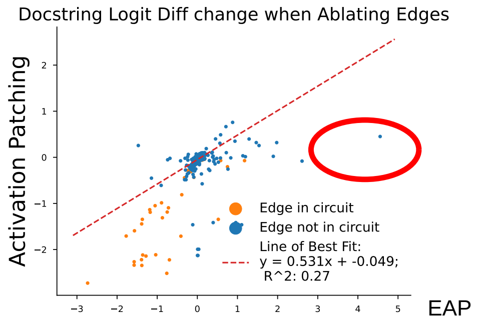
Figure 3: Distribution of Attribution Scores for the IOI Task (Logit Diff)

accurate approximations (in Section 5.1 we find that the answer is no), and whether there is any further use for ACDC (in Section 5.2 we find that the answer is yes). We use the docstring task as a case study due to the small model size used.

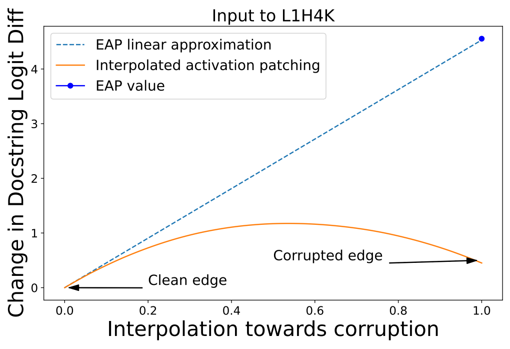
5.1 How faithful are Attribution Patching's approximations?

To study how faithful the approximation Equation (2) is, we plot the attribution patching scores (Equation (2)) against the activation patching scores (Equation (1)) in Figure 4a. Surprisingly, we find a fairly weak correlation between activation and attribution patching scores ($R^2 = 0.27$). Further, the line of best fit has gradient 0.531, suggesting that attribution patching estimates the effect of activation patching as twice as important as it really is.

Moreover, we can gain some sense for the discrepancy between activation and attribution patching by studying the continuous transition between clean (e_{clean}) and corrupted (e_{corr}) activations in Equation (1), i.e studying the values $|L(x_{\text{clean}}|\ \text{do}(E = \lambda e_{\text{corr}} + (1-\lambda)e_{\text{clean}})) - L(x_{\text{clean}})|$ for $0 \leq \lambda \leq 1$. We can compare this to the linear approximations of Attribution Patching $\lambda \Delta_e L$. Figure 4b shows the result for one edge in the docstring circuit where the linear approximation to activation patching is not accurate.



(a) Distribution of attribution scores for edges from activation patching and attribution patching. Circled: outlier EAP point studied in Figure 4b.



(b) Visualizing the rightmost point in Figure 4a. Note that corrupting this edge (surprisingly) slightly increases the logit difference on the Docstring task (higher logit difference is better). However, EAP overestimates how large this increase is.

Figure 4: Visualizing Edge Attribution Patching.

We find that interpolating towards the corrupted input creates a concave curve (Figure 4b) such that the linear approximation at $\lambda = 0$ overestimates the effect of activation patching this edge. In Appendix D we show that this also holds for the other outlier edges in the ellipse in Figure 4a.

5.2 Is there any further use for ACDC?

In Section 5.1 above, we found that EAP overestimates activation patching in cases where the attribution score is concave. This suggests the potential to refine the result by running ACDC on the pruned subgraph returned by EAP. We ran EAP first, then ACDC on the resulting subgraph for the Docstring task, varying pruning thresholds for EAP and ACDC independently. Figure 5 compares the TPR and FPR for the combined methods with the ROC curve of EAP only. The combined methods show increased performance compared to EAP only.

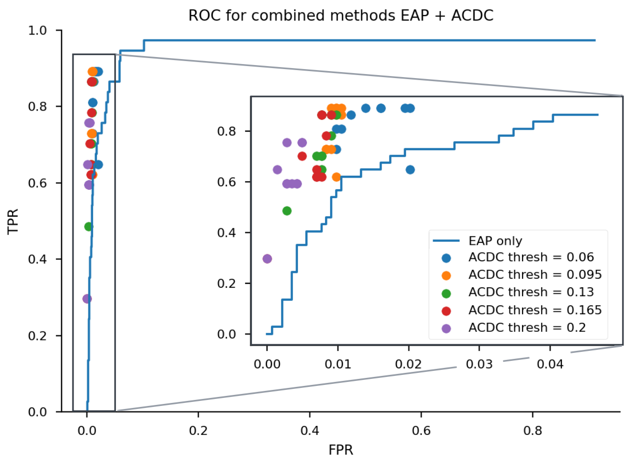


Figure 5: Comparing statistics of the combined EAP + ACDC methods with EAP only. The inset shows a zoom to the significant area of the statistics of the combined method.

Finally, one further limitation of this research is that the metrics used for interpretability do not precisely capture meaningful human understanding. Recovering a subgraph that humans previously recovered is limited because i) we can't evaluate this metric for interpretability tasks that we don't yet understand and ii) human-found circuits are imperfect, increasing the noise in this measurement.

6 Conclusion

We provide evidence that Edge Attribution Patching (EAP) outperforms ACDC in identifying circuits while being substantially faster to compute. This result is surprising, as EAP is an approximation for activation patching, the method applied by ACDC. However, running ACDC on the prepruned subnetwork found by EAP can improve the identification of relevant edges. Therefore, we suggest future circuit discovery experiments to run EAP first and then apply ACDC.

7 Author Contributions

Aaquib Syed and Can Rager proposed combining ACDC with attribution patching methods and implemented initial prototypes. Arthur Conmy advised working on attributing edges rather than nodes and Aaquib made the first findings that this outperformed Automatic Circuit Discovery. All authors worked on the paper's figures, experiments and code.

8 Acknowledgements

We would like to thank Callum McDougall for organising ARENA and providing a great introduction to interpretability, and Rusheb Shah and Lucy Farnik for collaboration on the ARENA hackathon prototype which this work is based on. We would like to thank Neel Nanda for a helpful discussion and János Kramár, Stephen Casper and Euan Ong for suggestions based on a earlier version of this work.

References

[1] Chris Olah. *Mechanistic Interpretability, Variables, and the Importance of Interpretable Bases*. 2022. URL: https://www.transformer-circuits.pub/2022/mech-interp-essay.

[2] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. "Zoom In: An Introduction to Circuits". In: *Distill* (2020). DOI: 10.23915/distill.00024.001.

[3] 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*. 2023. URL: https://openreview.net/forum?id=NpsVSN6o4ul.

[4] 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*. 2023. URL: https://openreview.net/forum?id=9XFSbDPmdW.

[5] Stefan Heimersheim and Jett Janiak. *A circuit for Python docstrings in a 4-layer attention-only transformer*. 2023. URL: https://www.alignmentforum.org/posts/u6KXXmKFbXfWzoAXn/a-circuit-for-python-docstrings-in-a-4-layer-attention-only.

[6] Michael Hanna, Ollie Liu, and Alexandre Variengien. *How does GPT-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model*. 2023. arXiv: 2305.00586 [cs.CL].

[7] Tom Lieberum, Matthew Rahtz, János Kramár, Neel Nanda, Geoffrey Irving, Rohin Shah, and Vladimir Mikulik. *Does Circuit Analysis Interpretability Scale? Evidence from Multiple Choice Capabilities in Chinchilla*. 2023. arXiv: 2307.09458 [cs.LG].

[8] Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Simas Sakenis, Jason Huang, Yaron Singer, and Stuart Shieber. *Causal Mediation Analysis for Interpreting Neural NLP: The Case of Gender Bias*. 2020. arXiv: 2004.12265 [cs.CL].

[9] Evan Hubinger. *An overview of 11 proposals for building safe advanced AI*. 2020. arXiv: 2012.07532 [cs.LG].

[10] Arthur Conmy, Augustine N. Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. "Towards Automated Circuit Discovery for Mechanistic Interpretability". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. arXiv: 2304.14997 [cs.LG].

[11] Paul Michel, Omer Levy, and Graham Neubig. "Are Sixteen Heads Really Better than One?" In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett. 2019, pp. 14014–14024. URL: https://proceedings.neurips.cc/paper/2019/hash/2c601ad9d2ff9bc8b282670cdd54f69f-Abstract.html.

[12] Christos Louizos, Max Welling, and Diederik P. Kingma. "Learning Sparse Neural Networks through L_0 Regularization". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=H1Y8hhg0b.

[13] Steven Cao, Victor Sanh, and Alexander Rush. "Low-Complexity Probing via Finding Subnetworks". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2021, pp. 960–966. DOI: 10.18653/v1/2021.naacl-main.74.

[14] Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. "What is the State of Neural Network Pruning?" In: *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. Ed. by Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze. mlsys.org, 2020. URL: https://proceedings.mlsys.org/book/296.pdf.

[15] Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. *A Survey on Model Compression for Large Language Models*. 2023. arXiv: 2308.07633 [cs.CL].

[16] Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. *Causal Abstractions of Neural Networks*. 2021. URL: `https://arxiv.org/abs/2106.02997`.

[17] Kevin Meng, David Bau, Alex J Andonian, and Yonatan Belinkov. "Locating and editing factual associations in GPT". In: *Advances in Neural Information Processing Systems*. 2022.

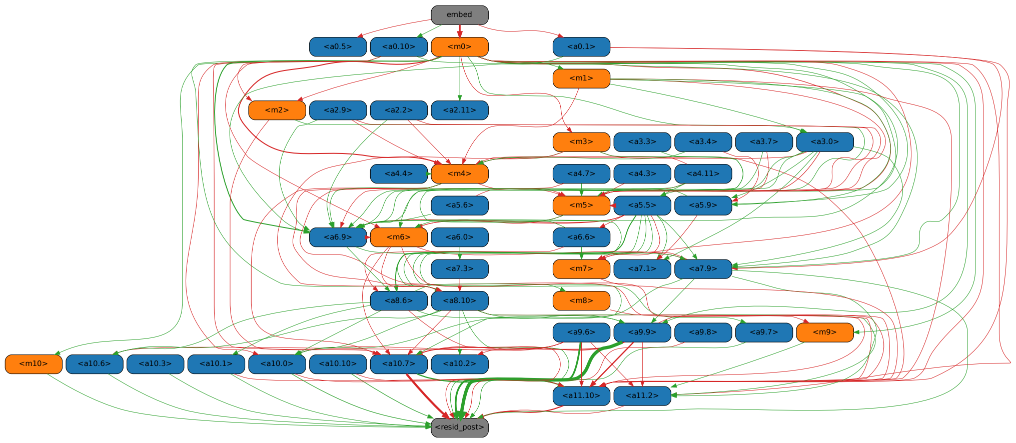
[18] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown,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

[19] Judea Pearl. "Causal diagrams for empirical research". In: *Biometrika* 82.4 (1995), pp. 669–688.

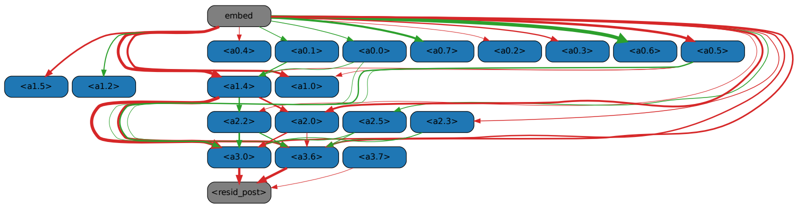
[20] Neel Nanda. *Attribution Patching: Activation Patching At Industrial Scale*. 2023. URL: `https://www.neelnanda.io/mechanistic-interpretability/attribution-patching`.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

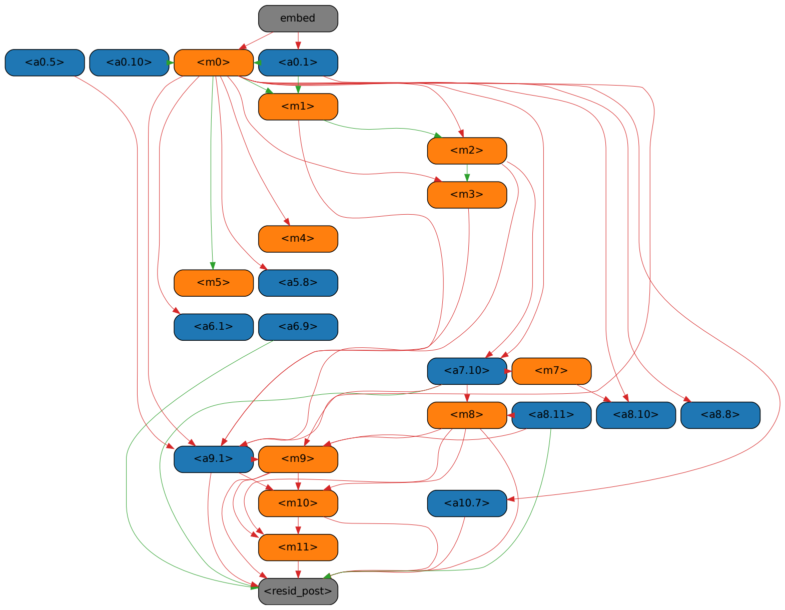
A EAP Subnetworks



(a) IOI Subnetwork, Threshold=0.077



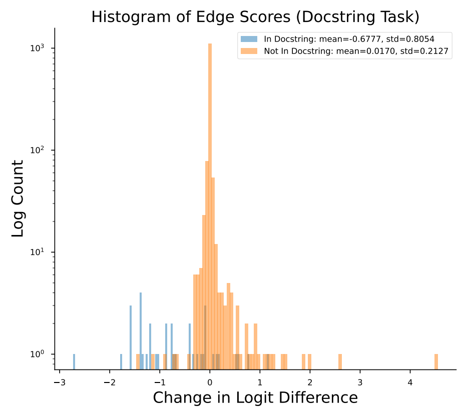
(b) Docstring Subnetwork, Threshold=0.244



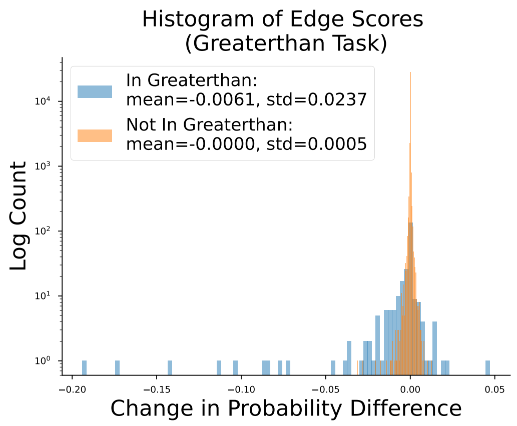
(c) Greaterthan Subnetwork, Threshold=0.009

Figure 6: Resulting subnetworks after EAP at the given thresholds.

B Distribution of EAP Attribution Scores



(a) Distribution of Attribution Scores for the Docstring Task



(b) Distribution of Attribution Scores for the Greater-Than Task

Figure 7: Distribution of Attribution Scores for the Docstring and Greater-Than tasks

C Further investigation into combining EAP with ACDC

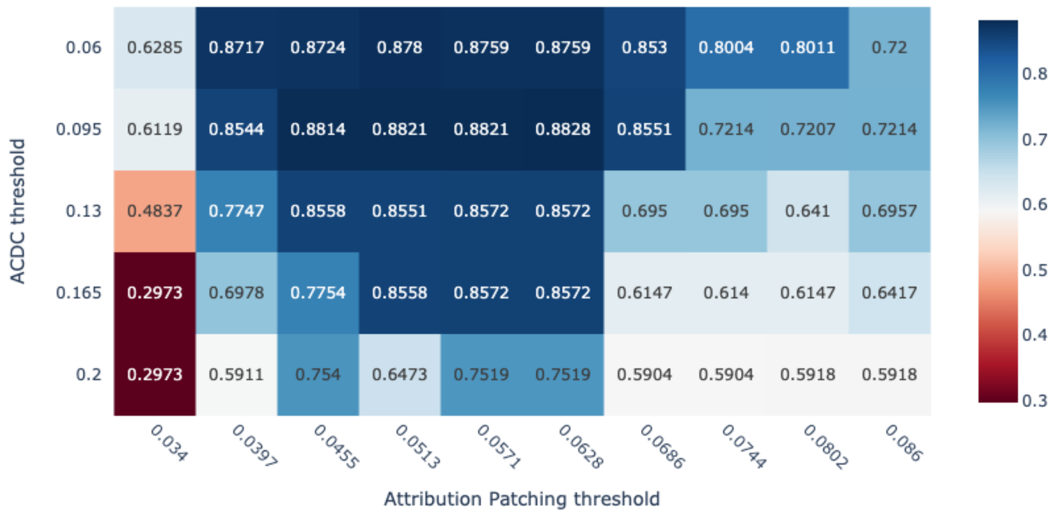


Figure 8: Youdens-J statistic (maximum TPR minus FPR value) for combining EAP and ACDC methods on the docstring task. We applied ACDC to the pruned subgraph returned by EAP.

D Further failures of attribution patching approximation

In Figure 9 we show further cases where in the docstring task attribution patching can be misleading. These cases all involve an edge that comes from the model's embeddings (positional and tokens). Our interpretation is that weighted averages of embeddings are anomalous inputs to the model and cause the concave change in docstring logit diff which doesn't occur when edges ae between non-embedding model components.

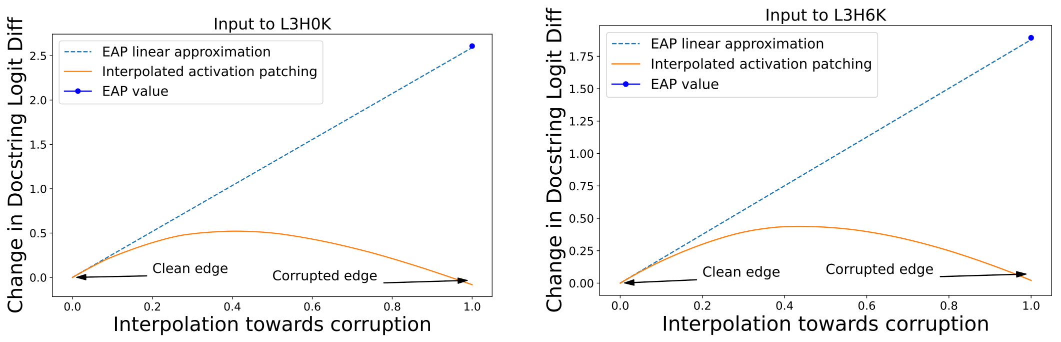


Figure 9: Visualizing Edge Attribution Patching in two further cases where the concave activation patching curve means the linear fit is poor.

E Edges Roles in IOI

We further explore the attribution scores for the IOI circuit. The IOI circuit is comprised of different attention head classes such as Induction heads, S-Inhibition heads, etc. [3]. Figure 10 shows the distributions of scores stratified by the roles of the edges. The edge roles are defined according to the role of their origin node. While edge roles such as Previous Token, Duplicate Token, Induction, and S-Inhibition edges have attribution scores centered around zero, we see a bias in edge scores given to name mover and negative name mover edges. As the name mover edges are directly responsible for the model outputting the indirect object, the attribution scores are largely negative since ablating these edges removes the model's ability to output the indirect object, lowering the logit difference. Similarly, the negative name movers have attribution scores that are largely positive since ablating these edges improves the logit difference. This matches the intuitive function of the edges.

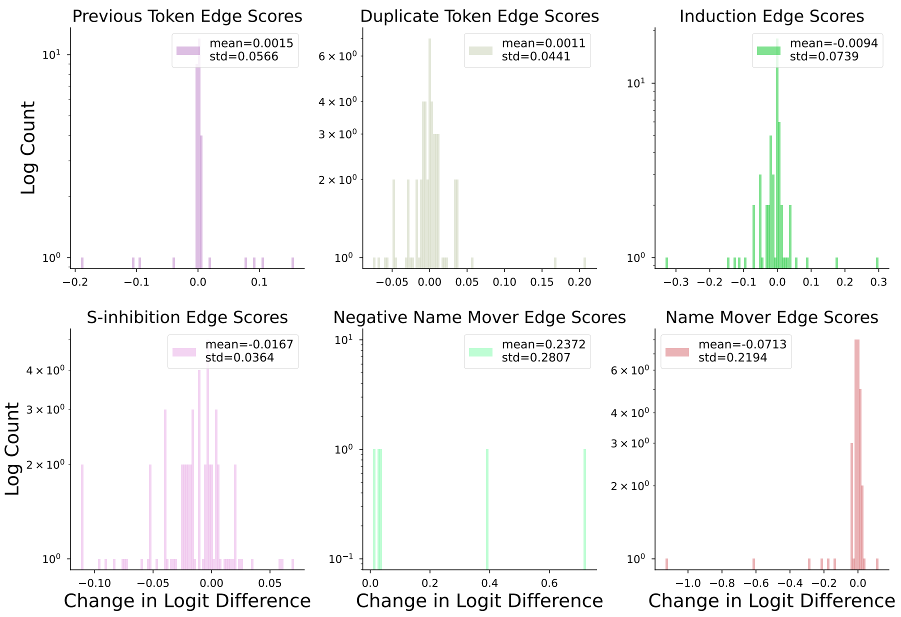


Figure 10: Distribution of Attribution Scores for each Edge Role in the IOI Task.

F Only one backwards pass is required for EAP

Note: it may be easier to understand our implementation `https://github.com/Aaquib111/acdcpp/blob/main/utils/prune_utils.py#L249` rather than read this explanation. Alternatively, this derivation uses essentially the same arguments as Nanda [20][4] though with an updated codebase.

There are only two types of edges iterated over in ACDC: i) residual edges where the result is added at its endpoint, and ii) edges between the residual stream and the query, key and value calculations.[5] Clearly for all edges like ii) we can compute the gradient terms in Equation (2) in one backwards pass.

Interestingly, for all $\Delta_e L$ terms where e is a type i) edge (i.e added at the endpoint), we only need calculate the gradient with respect to the endpoint of the edge! For example, suppose we're calculating the effect of L0H0 on L1H0Q. If we represent the input to L1H0Q as a node V in the computational graph then

$$\frac{\partial}{\partial e_{\text{clean}}} L(x_{\text{clean}} | \operatorname{do}(E = e_{\text{clean}})) = \frac{\partial}{\partial v_{\text{clean}}} L(x_{\text{clean}} | \operatorname{do}(V = v_{\text{clean}})) \tag{3}$$

due to how V is just the sum of all the edges entering V. This allows efficient calculation of all the $\Delta_e L$ values since gradients with respect to nodes in computational graphs are calculated by default in backwards passes.

[4]Specifically, this section: `https://www.neelnanda.io/mechanistic-interpretability/attribution-patching#how-to-think-about-activation-patching=:~:text=axes%20of%20variation.-,Path%20patching,-The%20core%20intuition`

[5]It may be worth looking at some ACDC outputs from [10]. See `https://colab.research.google.com/github/ArthurConmy/Automatic-Circuit-Discovery/blob/main/notebooks/colabs/ACDC_Implementation_Demo.ipynb` for an explanation of this design choice.