Report on *Towards Automated Circuit Discovery for Mechanistic Interpretability*

Juan Belieni
juanbelieni@gmail.com
Rio de Janeiro, Brazil
FGV EMAp

Ana Carolina Erthal
acarolerthal@gmail.com
Rio de Janeiro, Brazil
FGV EMAp

Contents

1 Introduction

This report provides information on the final step on our upskilling process, tracking our progress in replicating results from a paper on Mechanistic Interpretability (MI), *Towards Automated Circuit Discovery for Mechanistic Interpretability* by N. Elhage *et al.* [1].

The main purpose was to be able to reproduce some of the results, allowing us to put in practice the skills acquired during the previous weeks of upskilling, both in deeply understanding the paper's proposed algorithm and actually coding a scaled-down version of it. All implementations are available on GitHub.

2 Brief Paper Revision

The paper introduces a formalization for the circuit discovery process, establishing an algorithm that systematizes the reverse engineering on Transformer models, building on previous work. In addition to this framework, some algorithms for circuit discovery are proposed, with a highlight on the Automatic Circuit DisCovery (ACDC) algorithm. Let's briefly discuss these advances.

2.1 Process Systematizing

The formalization of the reverse engineering process is a significant contribution by N. Elhage *et al.* [1] in MI field. It identifies the general workflow followed by scientists for circuit discovery, establishing a more structured approach to not only save time, but also enhance the consistency of the discoveries, improving results reproducibility in the community. The workflow comprehends:

That is, we aim to identify a specific behavior exhibited by the model when performing a particular task (the circuit we are looking for) and curate a dataset to capture that behavior. Finally, we must select a metric to evaluate the model's performance on that task.

1. **Selecting the task, data and metric:** that is, we must select a behavior that models presents when performing an specific task (the circuit we are looking for), and curate a dataset to obtain that behavior. Last of all, a metric to evaluate the model's performance on that task must be chosen.
2. **Dividing the network into a graph of smaller units:** the model must be represented as a Directed Acyclic Graph (DAG) at some abstraction level, with nodes relations representing the model's computations.
3. **Patching the activations to isolate the relevant subgraph**: the edges on the DAG are recursively tested through activation patching to identify their importance through overwriting activations with corrupted versions and evaluating the impact on the output values using the chosen metric.

The recursive process described on step 3 is automated by the paper's proposed algorithm, ACDC.

2.2 The ACDC Algorithm

The goal of the algorithm is to start from a DAG representing the models computations and reduce it significantly by pruning all nodes that are not relevant for a chosen task, automatically identifying a circuit. The algorithm can be used to aid scientists in the process of discovering new circuits, as automates a time-consuming step of the reverse engineering process.

The idea is that one can specify a threshold for the metric calculation, defining how important an edge must be not to be removed. ACDC iterates over the nodes on the DAG, starting from the output, testing the impact of replacing activations with values from corrupted runs and evaluating the impact on the output. Whenever the metric value is too small, the connection is defined as unimportant to the task and is removed. At last, a subgraph of the original DAG is returned, which is expected to represent a circuit on that task.

Here's a simplification of the ACDC algorithm:

Algoritmo 1.

```
1. $H \leftarrow G$ (Initialize $H$ as the full original DAG, $G$)
2. $H \leftarrow H$.reverse_topological_sort() (Sort $H$ so output comes first)
3. **for** $v \in H$ **do**
    1. **for** $w$ parent of $v$ **do**
        1. $H_{\text{new}} \leftarrow H \setminus \{w \to v\}$ (Temporarily remove candidate edge)
        2. **if** $\text{DKL}(G \| H_{\text{new}}) - \text{DKL}(G \| H) < \tau$ **then** (Check for metric lower than threshold $\tau$)
            1. $H \leftarrow H_{\text{new}}$ (Edge is unimportant, remove permanently)
        3. **end**
    2. **end**
4. **end**
```

Note that DKL stands for Kullback–Leibler divergence, but this metric could be substituted with other chosen metrics.

3 Paper replication

To replicate the paper, we had to narrow the scope due to the numerous tasks and variations of the algorithm covered. For each step, some decisions had to be made.

In step 1, we chose the Induction task to focus on, as it is mentioned in the paper and we now have a good understanding of it from our past upskilling, being the task we focused on the most. For the metric, we opted for the one suggested in the paper, negative log-probability, and also considered Kullback-Leibler Divergence, which the paper defines as a universal metric applicable to various tasks. Regarding the data, we generated a dataset tailored to the task using random tokens, which can be verified in the accompanying code.

For step 2, we needed to generate the Directed Acyclic Graph (DAG). To facilitate the topological reverse sorting, which is required for the algorithm to start from the output node, we used NetworkX, a Python library specifically designed for graph-related tasks.

Finally, step 3 represents the ACDC algorithm implementation itself. Some decisions also had to be made, such as the type of activation patching to be chosen or the type of ablation. We focused on zero and corrupted activations, (replacing the activations that do not have enough influence on the performance with zeros or their correspondent values from a corrupted input, effectively removing them from the graph), and also zero ablation.

Codes for both steps 2 and 3 are available on the referred GitHub.

3.1 The process

Our process on the paper replication involved iterative cycles of trials and corrections to thoroughly understand the algorithm and apply the knowledge we've been acquiring.

3.1.1 Model 1

After developing an initial, simplified version of the ACDC algorithm, we plugged in the model used for induction in ARENA's material, an attention-only 2-layer Transformer with 12 attention heads from the TransformerLens library (Model 1). We then used ACDC to search for the Induction circuit that we had previously encountered in ARENA's material. The correct induction circuit on Model 1 is defined in Figure 1.

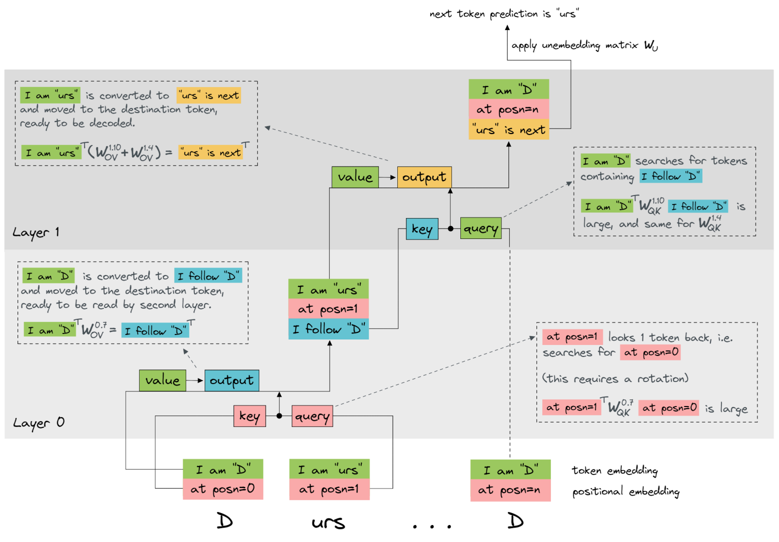

next token prediction is "urs"
apply unembedding matrix W_U
I am "urs" is converted to "urs" is next and moved to the destination token, ready to be decoded.
I am "urs"$^T(W_{OV}^{1.10}+W_{OV}^{1.4})$ = "urs" is next^T
I am "D"
at posn=n
"urs" is next
value
output
I am "D" searches for tokens containing I follow "D"
I am "D"$^T W_{QK}^{1.10}$ I follow "D" is large, and same for $W_{QK}^{1.4}$
key
query
Layer 1
I am "D" is converted to I follow "D" and moved to the destination token, ready to be read by second layer.
I am "D"$^T W_{OV}^{0.7}$ = I follow "D"^{T}
I am "urs"
at posn=1
I follow "D"
at posn=1 looks 1 token back, i.e. searches for at posn=0
(this requires a rotation)
at posn=1$^T W_{QK}^{0.7}$ at posn=0 is large
value
output
key
query
Layer 0
I am "D"
at posn=0
I am "urs"
at posn=1
I am "D"
at posn=n
token embedding
positional embedding
D
urs
. . .
D

Figure 1: Induction circuit in Model 1, ARENA original [2].

Originally, Model 1 had a 40% accuracy in predicting the correct token for the induction task. We ran ACDC using DKL as metric, zero activation patching and zero ablation.

This experiment led us to satisfying results. The circuit we were searching for was largely present in our pruned graph, confirming that we were on the right track. We found a subgraph in which the kept nodes were indeed the ones we expected based on ARENA.

From Figure 1, we anticipated that the circuit would contain Keys (K), Queries (Q) and Values (V) from heads `0.7`, `1.4` and `1.10`. In Figure 2 it can be seen that after pruning, we were mostly left with these heads, along with some additional nodes, getting very close to the actual circuit.

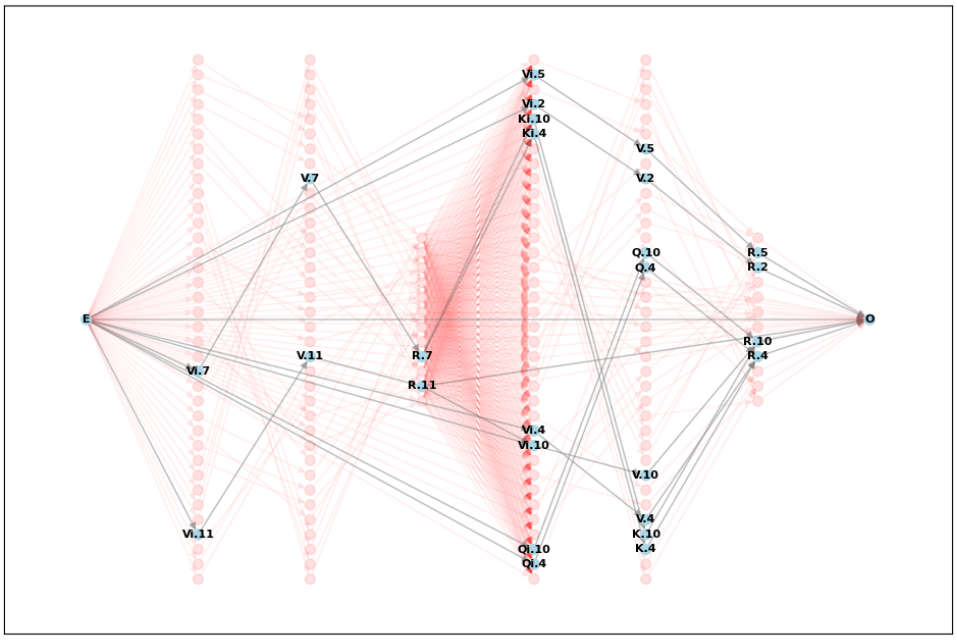


Figure 2: Subgraph that represents the induction circuit for Model 1 found through ACDC, where red nodes and edges were pruned and black ones were kept.

However, there is a drawback. Although the subgraph closely resembles the correct circuit, Q and K nodes from head `0.7` were incorrectly pruned. Losing these nodes ultimately nullified our accuracy, and the model was unable to perform any kind of induction correctly.

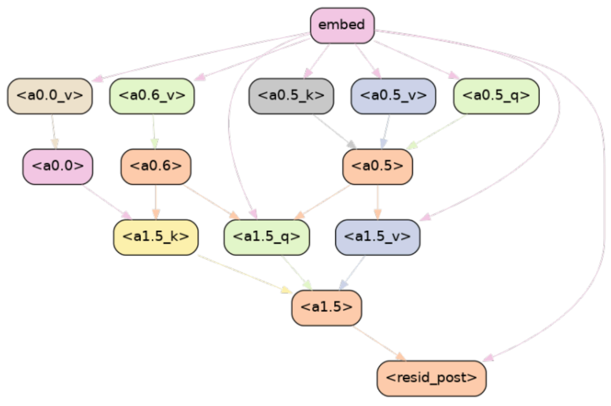
To confirm this result, we manually added these few nodes and edges that were missing, and assessed the models performance gain. We went from 0% to an astonishing 87% accuracy performance.

From this point onwards, we were confident that the full graph we were building was correct, and upon pruning could lead to good circuit discoveries. We were, however, unsure about the metric calculation, as the ACDC was supposed to be keeping important edges to the task, and without the presence of edges to Q and K nodes on `0.7` head, we were left with poor performance.

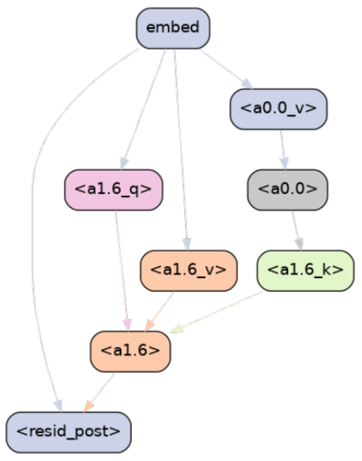
As changing the threshold, type of activation patching (corrupted and zero) and metric (from DKL to negative log-probability) wasn't enough to get the full circuit, we decided to test another model to validate our process.

3.1.2 Model 2

We now used a pretrained model from TransformerLens, `redwood_attn_2l`, a 2-layer 8 heads attention-only Transformer (Model 2), which was used in the paper's GitHub repository for the induction task. The paper contains an image of induction circuits obtained through ACDC (Figure 3). The expectation was to validate our pipeline, and check if we got the same circuit under the same constraints.



(a) Corrupted activations



(b) Zero activations

Figure 3: Subgraphs recovered by ACDC on the induction task with different types of activation by N. Elhage *et al.* [1]

Using zero activations, we found the exact same circuit detailed in Figure 3 (b), as it shows in Figure 4. However, this circuit leads to 0% accuracy for induction.

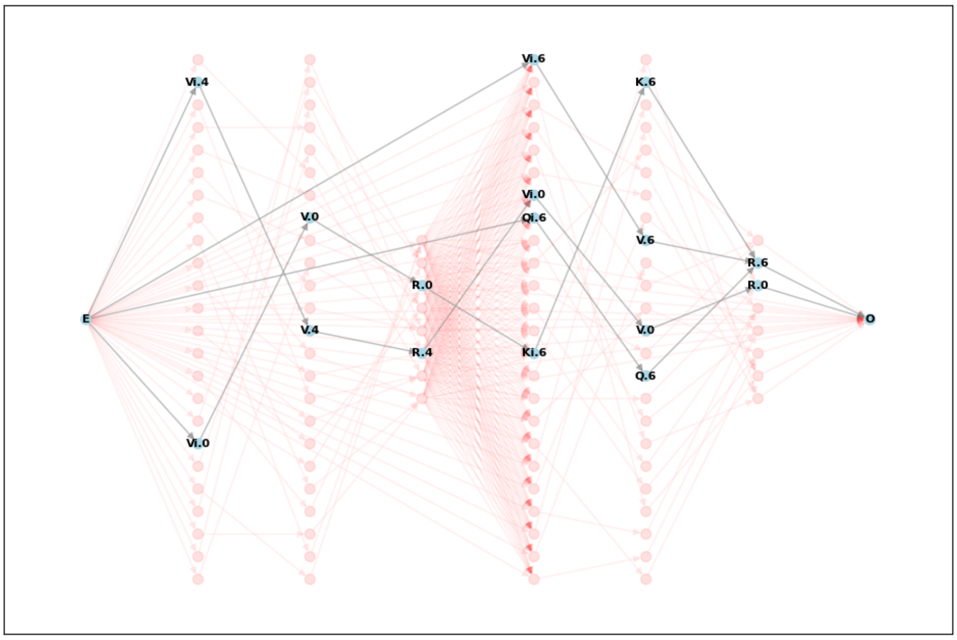


Figure 4: Subgraph by ACDC on induction task for Model 2, similar to the one described in [1]

Moreover, we ran the official ACDC code on the same model and obtained the same circuit, with no differences in nodes and edges. However, the circuit from the official version achieved 10% accuracy on the induction task. Although this performance is lower than the initial 22% for this model, it is significantly different from our version's results.

It was not possible to directly compare the results any further as the induction task, while listed in the paper and repository, is not included in the final results, and no metrics are provided. This left us with the impression that our algorithm might have a misconception regarding activation substitution. Despite the circuit being identical, it failed to perform the targeted task

Although we could not identify the mistake in time for this project delivery, the replication process was very fruitful for our learning and helped us formulate several questions we wish to explore in the future.

4 Future Exploration and Conclusions

After this process, we've developed a good understanding of the circuit discovery framework, the ACDC algorithm, and relevant questions for further investigation. During our exploration, several questions emerged, suggesting areas for future research.

Firstly, the results we obtained when exploring induction were not directly comparable, but we were left with questions on our code replication results. Even though the correct circuit was found, our version's performance was not correct, and we intend on continue to work on the code after the upskilling final date to understand the problem and make corrections, making sure we fully understand the paper.

Additionally, other smaller issues have caught our attention which we believe could be further explored. For example, demonstrating the efficacy of using the topological reverse of the original graph to run the algorithm, which is proposed in the paper without an explanation. Experiments could be run testing other ways of traversing the graph, to better understand the reasoning behind the sorting. Another area of interest is finding ways to make ACDC more efficient, which could enable extensions to slightly larger models.

In conclusion, our paper replication process significantly contributed to our upskilling. Putting our developed knowledge through the past months into practice allowed us to deepen our understanding of circuits and exercise our mechanistic interpretability skills. Despite some difficulties we encountered, we believe to have identified potential areas for us to continue on studying and possibly contribute to the field

References

[1] N. Elhage *et al.*, "A Mathematical Framework for Transformer Circuits," *Transformer Circuits Thread*, 2021.

[2] "ARENA Chapter 1: Transformer Interpretability."