Reasoning Fails Small - Scaling Fails ACDC

Davide Cavicchini
University of Trento

davide.cavicchini@studenti.unitn.it

***Abstract*—Understanding how Large Language Models perform logical reasoning remains a significant challenge in mechanistic interpretability. This work investigates the application of ACDC (Automatic Circuit Discovery) to identify computational circuits responsible for logical reasoning, specifically focusing on the Modus Tollens task from LogicBench. Our experiments across multiple model sizes (0.5B to 1.5B parameters) reveal fundamental limitations of circuit discovery methods when applied to challenging reasoning tasks for small models and scaling limitations for more capable ones. We employed complementary analysis techniques, including linear probing, activation patching, and visualization methods, to understand why ACDC fails and characterize the nature of logical reasoning, if even present, in the smaller models. Critical technical limitations, including Grouped Query Attention incompatibility and dataset preparation challenges, constrained our analysis. Our experimentation suggests that current circuit discovery methods may be insufficient for understanding models which, as Antropic says, work on "a bag of heuristics".**

***Index Terms*—circuit discovery, mechanistic interpretability, logical reasoning, transformer models, ACDC**

I. Introduction

Understanding how Large Language Models (LLMs) perform complex reasoning tasks remains one of the central challenges in AI interpretability. By peeking into the internal algorithms they use to solve such tasks, we could massively improve their reasoning capabilities using targeted editing of their weights. Mechanistic interpretability approaches, particularly circuit discovery methods like ACDC [1], promise to identify the minimal computational subgraphs responsible for specific behaviors. However, these techniques have several limitations hindering their application to complex tasks.

In this work, we investigate the limitations of ACDC in discovering circuits for logical reasoning tasks in LLMs. We focus on the task of modus tollens with the LogicBench benchmark [2]. By using complementary techiniques we will then try to understand why ACDC fails to discover meaningful circuits for this task.

II. Related Work

Mechanistic interpretability has emerged as a crucial approach for understanding neural network internals. As already mentioned, it would allow for greater control on what these model learn. The current research views models as a computational graph [3] where information flows from early layers to deeper ones to the output layers.

Early works manually checked the internal representations of the models and tried to make sense of what the model was thinking and what each neuron was modelling. Later work tried to automate this process, mainly by using activation and path **patching**. These techniques usually use a dataset composed of pairs of *clean* and *corrupt* prompts, where the second changed some the content to steer the answer of the model away form a known correct one, then the model activations are stored for the clean sample and when forwarding using the corrupt prompt see how much of the original performance is recovered when the clean ones are restored. This allowed for works like ROME [4], to discover the exact location of factual associations inside of an LLM model. And, later, for finding the computational subgraphs responsible for some of the capabilities observed in LLM thanks to Circuit discovery methods. Particularly, Automatic Circuit Discovery (ACDC) [1] has shown success in identifying computational circuits for relatively simple tasks such as the "greater-then" task for predicting the correct next token in phrases like "The war lasted from 1517 to 15", where the model should answer with tokens whose numerical value is grater than 17.

Although very simple in principle, this patching work requires an expensive inference to find meaningful circuits and is limited to the chosen task and *dataset format*. For this reason, they are mostly applied to models such as GPT2, which are mostly considered outdated by the community and have limited capabilities. Newer techniques employ the use of sparse AutoEncoders trained in an unsupervised fashion, which allows them to learn both where a concept lies and how computations are shared to deeper layers. An application of this idea is the recent paper from Anthropic "On the Biology of a Large Language Model" [5], where the authors trained a sparse AE for each layer and allowed deeper layers to receive information from all the previous ones. The sparse encoding allowed them to locate where concepts lied inside of the network weights and layers, while the numerous skip connections allowed them to flatten the computational graph, allowing the sparse model to clearly show from which layer the information being used is coming from. Although very expressive, these models are very hard to train and require a large amount of computational power; for this reason, they were not the focus of this investigation.

III. Methodology

Our initial objective was to apply the ACDC circuit discovery method to the Modus Tollens task from LogicBench in order to identify the internal computational mechanisms used by transformer models during logical reasoning. However, during the implementation process, we encountered several unexpected limitations—both technical and methodological—that hindered the discovery of meaningful circuits. This led us to expand our methodology to include complementary analysis techniques aimed at understanding why ACDC was unsuccessful in this context.

This section outlines our experimental pipeline, including dataset preparation for patching-based interpretability, model selection, circuit discovery via ACDC, and supporting methods such as linear probing and activation patching for deeper analysis.

A. Dataset Preparation

Modus Tollens is a fundamental logical inference rule: given "If P then Q" and "not Q", one can conclude "not P". The original dataset generated several scenarios describing the first logical implication, then paired with them a series of "if then" questions which has a yes/no answer to them. The first step was to gather the clean and corrupt pairs of prompts, which only required filtering one question with a positive answer and one with a negative for each scenario.

However, another fundamental requirement for applying patching to the model is to have both prompts with the same exact number of tokens. Our initial solution used a manual padding of the shorter one by recursively adding new line characters until the two matched in number of tokens. This worked for the sake of being able to apply the technique, but inserted a strong bias in the prompt structure, which we will explore later in section V-B, that might give a spurious signal to the ACDC technique. For this reason, we also experimented with the use of a local LLM to rephrase the two and remove the negations so that the two prompts have the same number of words. This worked for some of the pairs, but most still slipped past this pre-processing. Using stronger models from the OpenAI API yielded better results, but was deemed too expensive to apply to the whole dataset.

Another requirement for applying these techniques is the performance of these models on the task we are trying to solve. This is fairly obvious; if the models cannot solve the task, then we won't be able to pick up any meaningful signal. Since we had to work with a small model (up to 1.5B parameters) most of the models we tested turned out to have essentially random accuracy, and only the more modern models allowed for accuracy around 0.66. We feared this result to be too small for the ACDC technique to pick up any meaningful information, so for this reason, we also apply a filtering on the datasets where only the correctly classified clean and corrupt sample pairs are used for the analysis. This can be seen as a sensible choice, as maybe the model is only being strayed by a strange name in the prompt, however we later learned that only using the accuracy in isolation is probably not enough to asses the model performance and filter the examples, as we will try to argue in section **??**.

B. Automatic Circuit Discovery

As already mentioned multiple times, ACDC works using path patching, where we substitute the value from an edge in the clean run by using the one obtained in the corrupt run, checks what the effect is on the final result using a metric, and decide whether to keep it or not using a hyperparameter as threshold.

In other words: given a computational graph G of the model, ACDC constructs a subgraph $H \subseteq G$ by iteratively traversing from outputs to inputs, pruning edges while preserving model behavior. At each step, the algorithm evaluates whether removing a set of incoming edges to a node degrades performance on a chosen metric (typically KL divergence), in particular we use $H(x_i, x'_i)$, that denotes the result of the model when x_i is the input to the network, but all the missing edges are substituted with the activation on x'_i. If the degradation is below a threshold τ, the edges are discarded; otherwise, they are retained. The result, when successful, is a sparse subgraph H that approximates the original computation relevant to a specific input-output behavior.

The framework chosen for the experiments allows repeating this process multiple times for different τ values and creating *attribution scores* for each edge retained during the procedure. We can use this to visualize the final graph and try to understand what is the subgraph that the model is using to solve the task at hand.

C. Complementary Analysis Techniques

As we will see later, the results from ACDC hardly explained what the LLMs are doing, for this reason we decided to employ several other techniques to try to peek at the inner world models learned by them. These include:

1) **Linear probing**: Train a linear probe to predict if the model is looking at a clean or corrupt sample just by the last token representations. Using this technique, we want to check whether the models are storing somewhere in their layers the information of being in a "true" world state where the statements in the prompt are logically sound, and a "false" state of the world where we cannot infer for sure the truthfulness of the statements.
2) **Activation patching**: Apply activation patching to the model using a custom metric to showcase the effect of each layer, attention head, and token.
3) **Visualizations**: We developed multiple visualization tools to understand the model behaviour and try to unify the understanding acquired from the different approaches tested.

IV. Experiments

For this investigation, we needed to use models available in the transformer lens library, which allows for easier patching of models, and is small enough to allow these techniques to be run in reasonable timeframes. The options we tested out were:

- EleutherAI/pythia-14m,
- EleutherAI/pythia-410m,

- gpt2-small,
- gpt2,
- meta-llama/Llama-3.2-1B-Instruct,
- Qwen/Qwen2.5-0.5B-Instruct,
- Qwen/Qwen2.5-1.5B-Instruct,

Out of these models, the one we experimented with the most is Qwen/Qwen2.5-1.5B-Instruct, as it showed a good level of performance (around 0.66) and retained 64 pairs out of the 170 in the LogicBench's modus tollens dataset.

A. *Automatic Critcuit Discovery*

For ACDC, we decided to use as tau bases $1, 5, 7, 9$ and exp $-2, -3, -5$, the algorithm is then repeated for all possible combinations of the two. One big limitations later discovered of the library used comes from the fact that it only uses the first batch to compute the patching scores. This hinders massively the experimentations possible as it requires to have a full batch of prompts and *cached activations* to be able to perform the computations. This, in turn, limited our ability to test the robustness of the recovered circuits with multiple samples, although as we will cover in section V-C, this would probably not have changed our final results. Because of these limitations, we only tested with 8 and 1 batch size. The resulting graphs for the execution with a single sample are shown in images 1, 2, and 3.

B. *Linear probing*

For probing, we computed the hidden activations for different datasets and compared the performance obtained by the linear probes. In particular, we tested both with the price game task, as a baseline of what to expect, and the actual modus tollens task with and without the padding procedure required to apply patching. The results are shown in Table I.

Additionally, the data acquired during this step was used to cross-reference the output of ACDC and activation patching, as it allows to manual peeking inside the model representations by mapping them back to the tokens using the detokenization matrix. The app is showcased in Figure 4. We can see this as a ”cheaper” version of a sparse AE, as by using the detokenization matrix, we are computing the similarity to the concepts the model has associated with the tokens.

C. *Activation Patching*

Finally, we directly used the Transformer Lens library to apply activation patching to each attention head and each MLP layer, while also saving a metric that measures how much of the original performance is recovered from this action.

In particular, we first compute the output logits for the clean run and the probability associated with the positive and negative answers by computing the softmax only using the relevant logits we are interested in, these probabilities are then used to compute our baseline as $clean_score = pos\_tokens\_prob - neg\_tokens\_prob$. When patching the corrupt run, we then compute the same value and then divide it by the clean one previously computed

$$score = \frac{patched\_score}{clean\_score}$$

This way, we can interpret the resulting score in the following way:

- $score > 1$ Patching the corrupt run yielded a more ”yes” leaning result than the clean run;
- $score = 1$ Patching brought back the same model confidence on the ”yes” answer;
- $0 < score < 1$ Patching made the model recover partially, as now the answer favours the ”yes”;
- $score = 0$ the model is now completely unsure of the answer;
- $score < 0$ patching the model still made it answer with ”no”;

We can then filter out the less relevant nodes and see which ones contribute the most to the model decision.

In this case, we wanted to explore more how these scores changed for different samples. So we implemented one algorithm to compute the token-level scores for a single sample, and one to compute the average score over multiple runs at the layer level (for the MLP layer, we took the max score when patching one token, while for attention heads an implementation to fully substitute their output is already implemented).

V. Results and Analysis

This section presents the empirical findings from applying ACDC, linear probing, and activation patching to logical reasoning tasks in small-scale language models. While our original aim was to uncover sparse circuits supporting logical inference via ACDC, the experiments ultimately revealed deeper limitations in both the models and the methodology. We progressively broadened our analysis to understand why circuit discovery failed, and whether complementary techniques could offer a clearer view of how these models internally represent logical structure.

A. *ACDC*

Our initial application of ACDC focused on recovering sparse computational subgraphs from the Qwen2.5-1.5B-Instruct model using a single clean/corrupt sample. As seen in Figures 2 and 3, the results proved highly sensitive to the threshold values used. At low thresholds, the resulting graphs retained nearly every node, indicating no clear separation of relevant components. At higher thresholds, ACDC failed to identify any useful circuit at all, the graph did not even reach the unembedding layer.

Moreover, when applying ACDC over larger batches, the attribution scores tended to flatten out across the graph. This resulted in indistinct, noisy outputs where the relative contribution of each edge became difficult to interpret. These results mirror the pattern we later observe in the activation patching analysis V-C), and underscore a limitation of ACDC on hard problems where models either fail often or require more structured prompts, as we will try to argue in our conclusions.

Dataset	Layer	accuracy	F1
Price Game	MLP_L5	0.5	0.0
	MLP_L14	0.76	0.70
Modus Tollens padded	MLP_L5	0.99	0.99
	MLP_L14	0.99	0.99
Modus Tollens base	MLP_L5	0.84	0.84
	MLP_L14	0.97	0.97

TABLE I
PROBING RESULTS ON DIFFERENT DATASETS AND LAYERS FOR THE QWEN/QWEN2.5-1.5B-INSTRUCT MODEL

B. Linear Probing Results

From Table I, we can see how the model is creating a linearly separable representation of the true and false states of the world for the price game task, as we can correctly classify the sample using deeper layers of the model! Instead, in the Modus Tollens task, we get a much different result. We can notice how the performance of the model is suspiciously high, even in early layers of the model. And it improves marginally in deeper ones.

This is even more noticeable when using the ”aligned” prompts for ACDC with the new line padding. This result can either be interpreted as the model having an almost perfect linear representation of the state of the world, which is not the case, given its poor performance, or that the prompts we are using add a significant bias to the representations. This, in turn, partially explains why ACDC struggles to find relevant edges, as substituting the activations from the clean or corrupt run shifts the result for the wrong reasons!

C. Activation Patching Analysis

This analysis revealed significant brittleness in the model's internal reasoning. Patching even a single token often leads to scores ≥ 1, suggesting an inherently fragile and superficial internal representation of logical reasoning.

To understand the reason behind these mostly positive results, we decided to investigate the uncertainty of the model on the correctly classified pairs using an entropy measure on the distribution of positive and negative responses. Surprisingly, the result is that in the positive clean example, the entropy of the model is 0.5395 ± 0.1735. Given that only two possible outcomes exist for the task at hand, this means that the model has no idea of what the correct answer should be. In turn, for the negative corrupt samples the model is strongly committed to its answer, achieving an entropy of 0.1586 ± 0.1745.

This behaviour can in part explain our results, as switching from the clean output distribution and corrupt distribution is not as drastic as if the model were certain of its prediction in both cases. Trying to average the layer-level results on multiple samples yields an even less meaningful result, as the computed scores tend all to 1.

To complement the quantitative analysis, we developed visualizations of token-level patching graphs under different thresholds. These allow us to inspect which layers and tokens contribute positively or negatively to the prediction confidence. While useful for debugging and interpretation, they did not significantly alter our conclusions and mostly served as qualitative confirmation of the observed brittleness. Examples of such graphs at varying thresholds are shown in Figures 6, 5.

VI. PROBLEMS ENCOUNTERED

Our investigation into applying ACDC for discovering logical reasoning circuits was impacted by several critical challenges, spanning computational resources, specific tooling limitations, and dataset preparation for patching-based interpretability.

- **Memory and Computational Constraints:** The primary impediment was the substantial computational overhead of ACDC. The search space exploration inherent in ACDC, even for the Qwen2.5-1.5B model, resulted in long execution times, with single ACDC runs potentially requiring up to a week on an A100 GPU. GPU memory constraints further limited our effective batch size to 8 samples as already explained earlier. This not only decelerated the experimental cycle but also rendered extensive ACDC hyperparameter sweeps computationally prohibitive.
- **Tooling Incompatibility with Grouped Query Attention (GQA):** A significant technical constraint emerged from the ‘transformer_lens‘ library. At the time of our experiments, its support for models employing Grouped Query Attention (GQA) is incomplete. Preventing the utilization of faster algorithms within the ‘automatic-circuit‘ library that necessitate KVQ granularity.
- **Dataset Preparation Challenges for Patching:**
  - *Padding-Induced Bias:* A fundamental prerequisite for patching methodologies like ACDC is the identical token length of clean and corrupt prompts. Our initial strategy of padding shorter prompts with newline characters, while enabling the application of the technique, introduced a potent structural bias. As evidenced by our linear probing experiments (Section V-B), this padding rendered clean and corrupt samples highly distinguishable from their activations even in early network layers, likely giving spurious signals to ACDC.
  - *Ineffectiveness and Cost of Rephrasing:* Efforts to obviate padding by employing local LLMs to rephrase prompt pairs resulted in limited and inconsistent success. While more powerful proprietary models demonstrated superior capabilities, their application across the entire dataset was deemed too expensive for this investigation.
- **Low Baseline Model Performance:** The models accessible for this study (up to 1.5B parameters) demonstrated only modest performance. Although we filtered the dataset to retain only correctly classified sample pairs, this relatively low underlying proficiency implies that the signal for reasoning might be inherently weak or inconsistent, making it hard for ACDC to build a meaningful circuit.

- **ACDC Implementation Nuances:** The specific ACDC implementation utilized had a limitation where patching scores are computed based only on the initial batch of data. Limiting the exploration of the full dataset.

A. Key Experimental Take-Aways

Our investigation yields several insights about the limitations of circuit discovery methods for logical reasoning in smaller language models:

Circuit Discovery Fails on Weak Reasoning Models: ACDC proved ineffective at identifying meaningful computational circuits for Modus Tollens reasoning across all tested model sizes (0.5B-1.5B parameters). The recovered circuits were either trivially dense (retaining most edges) or pathologically sparse (failing to reach output layers), indicating that these models lack the robust, specialized reasoning mechanisms that circuit discovery methods assume.

Superficial Pattern Matching Over Logical Reasoning: Our activation patching analysis revealed that models achieve their modest performance through brittle, superficial pattern matching rather than genuine logical inference. The high uncertainty in model predictions (entropy of 0.54 ± 0.17 for correctly classified positive examples) suggests that apparent ”reasoning” may be largely coincidental.

Dataset Preparation Introduces Spurious Signals: The requirement for identical token lengths in clean/corrupt prompt pairs creates a fundamental challenge. Our padding approach introduced strong structural biases that enabled near-perfect linear separability (99% accuracy) even in early model layers, potentially masking genuine reasoning signals and confounding circuit discovery algorithms.

Methodological Brittleness: Activation patching results showed extreme sensitivity, with single-token substitutions often yielding scores ≥ 1, indicating that the models' internal representations of logical structure are highly fragile.

Scale-Dependent Interpretability Challenges: While smaller models ($\leq$1.5B parameters) were computationally tractable for circuit discovery, they demonstrated insufficient reasoning capabilities. Larger models with better reasoning performance remain computationally prohibitive for circuit analysis for this project.

REFERENCES

[1] A. Conmy, A. N. Mavor-Parker, A. Lynch, S. Heimersheim, and A. Garriga-Alonso, “Towards automated circuit discovery for mechanistic interpretability,” 2023. [Online]. Available: https://arxiv.org/abs/2304.14997

[2] M. Parmar, N. Patel, N. Varshney, M. Nakamura, M. Luo, S. Mashetty, A. Mitra, and C. Baral, “Logicbench: Towards systematic evaluation of logical reasoning ability of large language models,” 2024. [Online]. Available: https://arxiv.org/abs/2404.15522

[3] C. Olah, “Mechanistic interpretability, variables, and the importance of interpretable bases,” https://www.transformer-circuits.pub/2022/mech-interp-essay, 2022.

[4] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, “Locating and editing factual associations in gpt,” 2023. [Online]. Available: https://arxiv.org/abs/2202.05262

[5] J. Lindsey, W. Gurnee, E. Ameisen, B. Chen, A. Pearce, N. L. Turner, C. Citro, and *et al.*, “On the biology of a large language model,” https://transformer-circuits.pub/2025/attribution-graphs/biology.html, 2025.

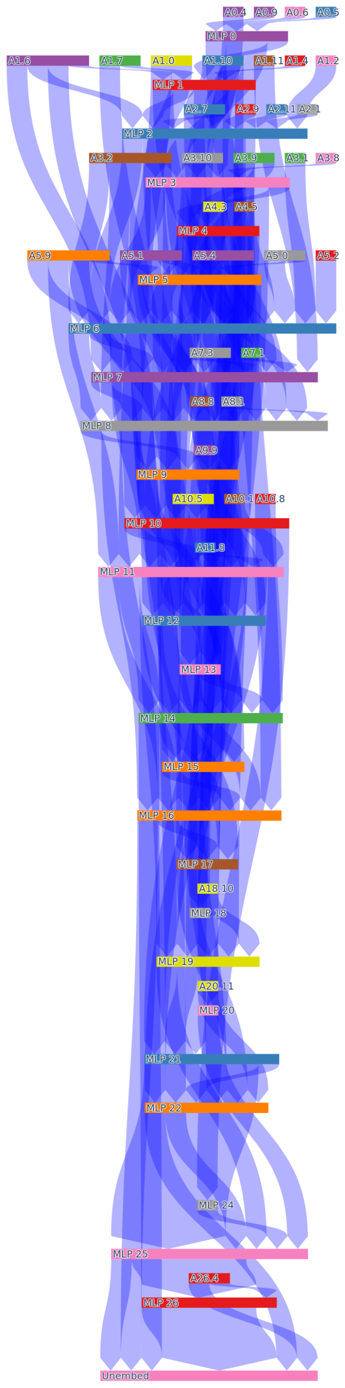


Fig. 1. graph obtained with ACDC with a threshold of $7e-2$

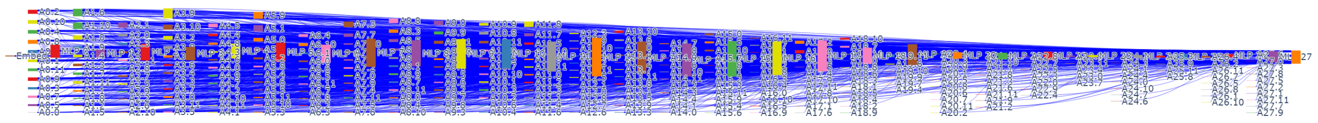


Fig. 2. graph obtained with ACDC with a threshold of $6e-3$

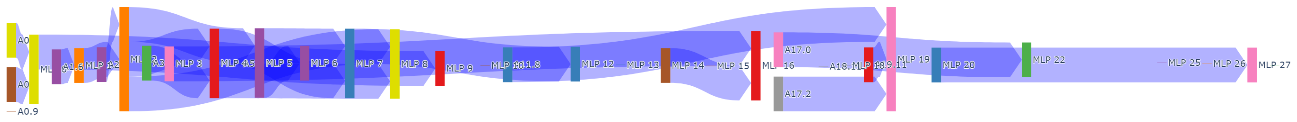
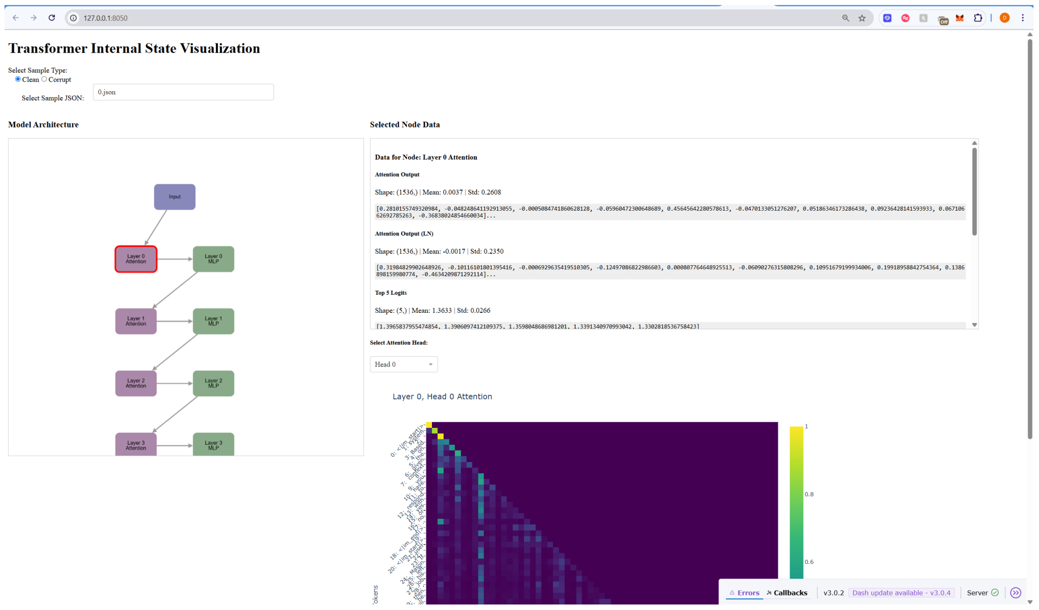


Fig. 3. graph obtained with ACDC with a threshold of $9e-2$



Transformer Internal State Visualization

Select Sample Type:
[✓ Clean] Corrupt

Select Sample JSON: 0.json

Model Architecture

Selected Node Data

Data for Node: Layer 0 Attention

Attention Output

Shape: (1536,) | Mean: 0.0037 | Std: 0.2608

```
[0.2810155749320984, -0.048248641192913055, -0.0005084741860628128, -0.05960472300648689, 0.45645642280578613, -0.0470133051276207, 0.05186346173286438, 0.09236428141593933, 0.067106
62692785263, -0.36838024854660034]...
```

Attention Output (LN)

Shape: (1536,) | Mean: -0.0017 | Std: 0.2350

```
[0.31984829902648926, -0.10116101801395416, -0.0006929635419510305, -0.12497086822986603, 0.000807764648925513, -0.06090276315808296, 0.10951679199934006, 0.19918958842754364, 0.1386
898159980774, -0.4634209871292114]...
```

Top 5 Logits

Shape: (5,) | Mean: 1.3633 | Std: 0.0266

```
[1.3965837955474854, 1.3906097412109375, 1.3598048686981201, 1.3391340970993042, 1.3302818536758423]
```

Select Attention Head:

Head 0

Layer 0, Head 0 Attention

Fig. 4. Visualization app developed to navigate the output of the dataset collected for the probing experiments

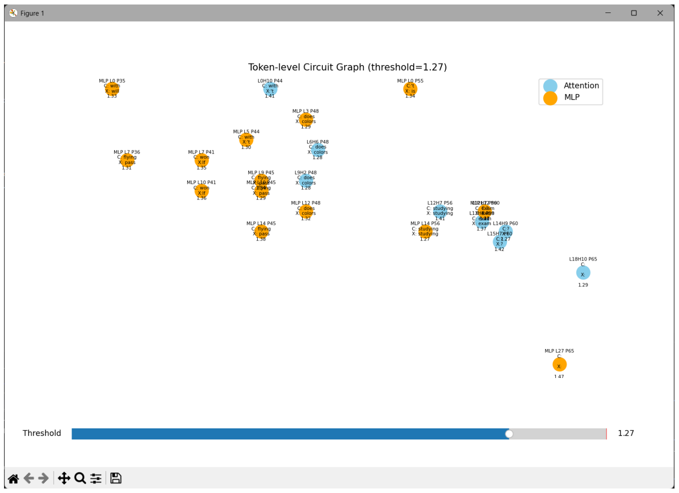


Fig. 5. Token graph extracted using activation parching with a threshold ≥ 1

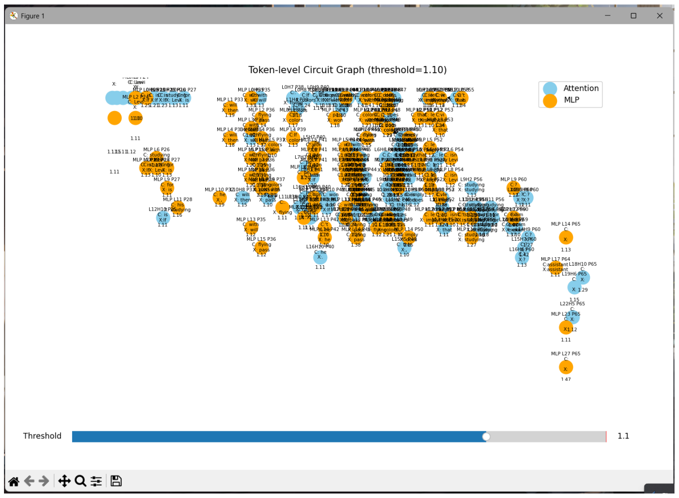


Fig. 6. Token graph extracted using activation parching with a threshold ≈ 1