Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small

Anonymous Author(s)
Affiliation
Address
email

Abstract

Research in mechanistic interpretability seeks to explain behaviors of ML models in terms of their internal components. However, most previous work either focuses on simple behaviors in small models, or describes complicated behaviors in larger models with broad strokes. In this work, we bridge this gap by presenting an explanation for how GPT-2 small performs a natural language task that requires logical reasoning: indirect object identification (IOI). Our explanation encompasses 28 attention heads grouped into 7 main classes, which we discovered using a combination of interpretability approaches including causal interventions and projections. To our knowledge, this investigation is the largest end-to-end attempt at reverse-engineering a natural behavior "in the wild" in a language model. We evaluate the reliability of our explanation using three quantitative criteria–*faithfulness, completeness* and *minimality*. Though these criteria support our explanation, they also point to remaining gaps in our understanding. Our work provides evidence that a mechanistic understanding of large ML models is feasible, opening opportunities to scale our understanding to both larger models and more complex tasks.

1 Introduction

Transformer-based language models (Vaswani et al., 2017; Brown et al., 2020) have demonstrated an impressive suite of capabilities, but largely remain black boxes. Understanding these models is difficult because they employ complex non-linear interactions in densely-connected layers and operate in a high-dimensional space. Despite this, they are already deployed in high-impact settings, underscoring the urgency of understanding and anticipating possible model behaviors. Some researchers have even argued that interpretability is necessary for the safe deployment of advanced machine learning systems (Hendrycks & Mazeika, 2022).

Work in mechanistic interpretability aims to discover, understand and verify the algorithms that model weights implement by reverse engineering model computation into human-understandable components (Olah, 2022; Meng et al., 2022; Geiger et al., 2021; Geva et al., 2020). By understanding underlying mechanisms, we can better predict out-of-distribution behavior (Mu & Andreas, 2020), identify and fix model errors (Hernandez et al., 2021; Vig et al., 2020), and understand emergent behavior (Nanda & Lieberum, 2022; Barak et al., 2022; Wei et al., 2022).

In this work, we aim to understand how GPT-2 small (Radford et al., 2019) implements a natural language task. To do so, we locate components of the network that produce specific behaviors, and study how they compose to complete the task. Specifically, we discover *circuits*: induced subgraphs of a model's computational graph that are human-understandable and responsible for a behavior. We employed a number of techniques, most notably activation patching, knockouts, and projections, which we believe are useful, general techniques for circuit discovery.

Submitted to the ML Safety Workshop at the 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute

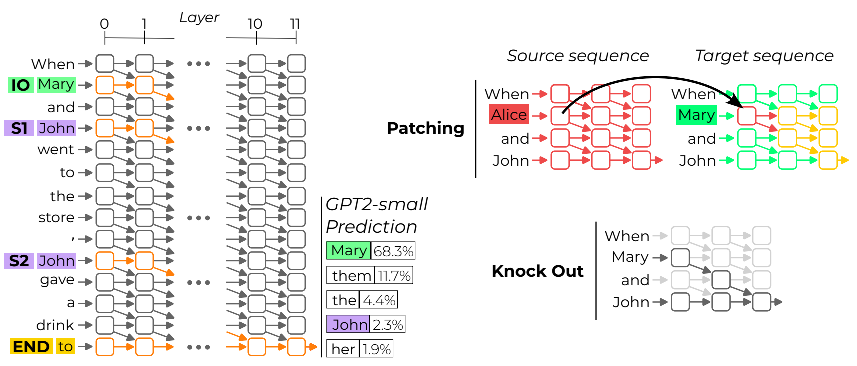

Figure 1: Left: We isolated a *circuit* (in orange) responsible for the flow of information connecting the indirect object 'Mary' to the next token prediction. The nodes are attention blocks and the edges represent the interactions between attention heads. Right: We discovered and validated this circuit using activation experiments, including both patches and knockouts of attention heads.

We focus on understanding a non-trivial, algorithmic natural language task that we call Indirect Object Identification (IOI). In IOI, sentences such as 'When Mary and John went to the store, John gave a drink to' should be completed with 'Mary'. We chose this task because it is linguistically meaningful and admits a complex but interpretable algorithm: of the two names in the sentence, predict the name that isn't the subject of the last clause.

We discover a circuit of 28 attention heads–1.5% of the total number of (head, token position) pairs–that completes this task. The circuit uses 7 different categories of heads (see Figure 2) to implement the algorithm. Together, these heads route information between different name tokens, to the end position, and finally to the output. Our work provides, to the best of our knowledge, the most detailed attempt at reverse-engineering a natural end-to-end behavior in a transformer-based language model.

Explanations for model behavior can easily be misleading or non-rigorous (Jain & Wallace, 2019; Bolukbasi et al., 2021). To remedy this problem, we formulate three criteria to help validate our circuit explanations. These criteria are **faithfulness** (the circuit can perform the task as well as the whole model), **completeness** (the circuit contains all the nodes used to perform the task), and **minimality** (the circuit doesn't contain nodes irrelevant to the task). Our circuit shows significant improvements compared to a naïve (but faithful) circuit, but fails to pass the most challenging tests.

In summary, our main contributions are: (1) We identify a large circuit in GPT-2 small that performs indirect-object identification (Figure 2 and Section 3); (2) Through example, we identify useful techniques for understanding models, as well as surprising pitfalls; (3) We present criteria that ensure structural correspondence between the circuit and the model, and check experimentally whether our circuit meets this standard (Appendix B).

2 Background

In this section, we introduce the IOI task, review the transformer architecture, define *circuits* more formally and describe a technique for "knocking out" nodes in a model.

Task description. In indirect object identification (IOI), two names (the indirect object (IO) and the first occurrence of the subject (S1)) are introduced in an initial dependent clause (see Figure 1). A main clause then introduces the second occurrence of the subject (S2), who is usually exchanging an item. The task is to complete the main clause, which always ends with the token 'to', with the non-repeated name (IO). We create many dataset samples for IOI (p_{IOI}) using 15 templates (see Appendix C) with random single-token names, places and items.

To quantify GPT-2 small's performance on the IOI task, we use two different metrics: logit difference and IO probability. *Logit difference* measures the difference in logit value between the two names, where a positive score means the correct name (IO) has higher probability. *IO probability* measures the absolute probability of the IO token under the model's predictions. Both metrics are averaged over $\mathrm{p_{IOI}}$. GPT-2 small has mean logit difference of 3.55, averaged across over 100,000 dataset examples, and mean IO probability of 49%.

2.1 Circuits

In mechanistic interpretability, we want to reverse-engineer models into interpretable algorithms. A useful abstraction for this goal are *circuits*. If we think of a model as a computational graph M where nodes are terms in its forward pass (neurons, attention heads, embeddings, etc.) and edges are the interactions between those terms (residual connections, attention, projections, etc.), a circuit C is a subgraph of M responsible for some behavior (such as completing the IOI task). This definition of a circuit is slightly different from that in Olah et al. (2020), where nodes are features (meaningful directions in the latent space of a model) instead of model components.

2.2 Knockouts

Just as the entire model M defines a function $M(x)$ from inputs to logits, we also associate each circuit with a function $C(x)$, via *knockouts*. A knockout removes a set of nodes K in a computational graph M with the goal of "turning off" nodes in K but capturing all other computations in M. Thus, $C(x)$ is defined by knocking out all nodes in $M \backslash C$ and taking the resulting logit outputs in the modified computational graph.

A first naïve knockout approach consists of simply deleting each node in K from M. The net effect of this removal is to *zero ablate* K, meaning that we turn its output to 0. This naïve approach has an important limitation: 0 is an arbitrary value, and subsequent nodes might rely on the average activation value as an implicit bias term. Because of this, we find zero ablation to lead to noisy results in practice.

To address this, we instead knockout nodes through *mean ablation*: replacing them with their average activation value across some reference distribution (similar to the bias correction method used in Nanda & Lieberum (2022)). Mean-ablations will remove the influence of components sensitive to the *variation* in the reference distribution (i.e. attention heads that move names in $\mathrm{p_{IOI}}$), but will not influence components using information *constant* in the distribution (i.e. attention patterns that are constant in $\mathrm{p_{IOI}}$). Through mean-ablations, we are interested in finding the components that move information about names, which is the core of the IOI task and also varies with the distribution.

In this work, all knockouts are performed in a modified $\mathrm{p_{IOI}}$ distribution with three random names, so the sentences no longer have a single plausible IO. We mean-ablate on this distribution, which we call the 'ABC' distribution, because mean-ablating on the $\mathrm{p_{IOI}}$ distribution would not remove enough information, like information constant in $\mathrm{p_{IOI}}$ that is helpful for the task. To knockout a single node, a (head, token position) pair in our circuit, we compute the mean of that node across samples of the same template. Computing means across the entire distribution instead of templates would average activations at different tokens, like names, verbs and conjunctions, mixing information destructively.

3 Circuit Overview

We seek to explain how GPT-2 small implements the IOI task (Section 2). Recall the example sentence "When Mary and John went to the store, John gave a drink to". The following human-interpretable algorithm suffices to perform this task:

1. Identify all previous names in the sentence (Mary, John, John).
2. Remove all names that are duplicates (in the example above: John).
3. Output the remaining name.

Our circuit contains three major classes of heads, corresponding to these three components:

- *Duplicate Token Heads* identify tokens that have already appeared in the sentence. They are active at the S2 token, attend primarily to the S1 token and write a 'signal' into the residual stream that token duplication has occurred.
- *S-Inhibition Heads* perform step 2 of the human-interpretable algorithm. They are active at the END token, attend to the S2 token and write to bias the query of the Name Mover Heads against both S1 and S2 tokens.

- *Name Mover Heads*, by default, attend to previous names in the sentence, but due to the S-Inhibition Heads attend less to the S1 and S2 tokens. Their OV matrix is a name copying matrix, so in $\mathrm{p}_{\mathrm{IOI}}$, they increase the logit of the IO token.

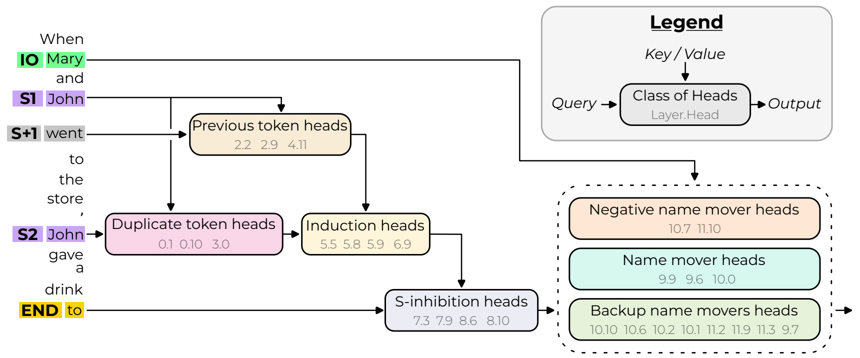

Figure 2: We discover a circuit in GPT-2 small that implements IOI. The input tokens on the left are passed into the residual stream. Attention heads move information between residual streams: the query and output arrows show which residual streams they write to, and the key/value arrows show which residual streams they read from.

A fourth major family of heads writes in the opposite direction of the Name Mover Heads, thus decreasing the confidence of the predictions. We speculate that these *Negative Name Mover Heads* might help the model "hedge" so as to avoid high cross-entropy loss when making mistakes.

There are also three minor classes of heads that perform related functions to the components above:

- *Previous Token Heads* copy the embedding of S to position S+1.
- *Induction Heads* perform the same role as the Duplicate Token Heads through an induction mechanism. They are active at position S2, attend to token S+1 (mediated by the Previous Token Heads), and output a signal that the S token previously appeared in the context.
- Finally, *Backup Name Mover Heads* do not normally move the IO token to the output, but take on this role if the regular Name Mover Heads are knocked out.

In Appendix A, we show step-by-step how we discovered each component, providing evidence that they behave as described above.

4 Discussion

A major motivation for this work was to gain evidence that mechanistic explanations for large language models is possible. Does this approach scale? In initial analyses with GPT-2 medium, we find that GPT-2 medium also has a sparse set of heads writing in the $W_U[IO] - W_U[S]$ direction. However, not all of these heads attend to IO and S, suggesting more complex behavior than the Name Movers Heads in GPT-2 small. Furthering this investigation is an exciting line of future work.

After completing this work, we learned several lessons useful for future interpretability efforts. We found that specifying a behavior and representative distribution for this behavior is a fundamental difficulty. For this reason, we think algorithmic tasks, as opposed to heuristics, are easier to interpret because they impose a clearer structure on model internals. In the circuit discovery process, we found activation patching useful for the discovery of important nodes. Activation patching is particularly useful with algorithmic tasks that have input schemas (Appendix C) because they suggest relevant source distributions for patching.

In this work, we discover, understand and check a circuit in GPT-2 small that identifies indirect objects. However, there are still several components we still do not understand, including the attention patterns of the S-Inhibition Heads, and the effect of MLPs and layer norms. We hope that our work spurs further efforts in mechanistic explanations of larger language models computing different natural language tasks, with the eventual goal of understanding full language model capabilities.

References

Boaz Barak, Benjamin L Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: Sgd learns parities near the computational limit. *arXiv preprint arXiv:2207.08799*, 2022.

Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda B. Viégas, and Martin Wattenberg. An interpretability illusion for BERT. *CoRR*, abs/2104.07143, 2021. URL `https://arxiv.org/abs/2104.07143`.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

Guy Dar, Mor Geva, Ankit Gupta, and Jonathan Berant. Analyzing transformers in embedding space. *arXiv preprint arXiv:2209.02535*, 2022.

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.

Matthew Finlayson, Aaron Mueller, Sebastian Gehrmann, Stuart Shieber, Tal Linzen, and Yonatan Belinkov. Causal analysis of syntactic agreement mechanisms in neural language models, 2021. URL `https://arxiv.org/abs/2106.06087`.

Atticus Geiger, Hanson Lu, Thomas F Icard, and Christopher Potts. Causal abstractions of neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=RmuXDtjDhG`.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*, 2020.

Dan Hendrycks and Mantas Mazeika. X-risk analysis for ai research. *arXiv*, abs/2206.05862, 2022.

Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob Andreas. Natural language descriptions of deep visual features. In *International Conference on Learning Representations*, 2021.

Sarthak Jain and Byron C. Wallace. Attention is not Explanation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3543–3556, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1357. URL `https://aclanthology.org/N19-1357`.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *arXiv preprint arXiv:2202.05262*, 2022.

Jesse Mu and Jacob Andreas. Compositional explanations of neurons. *Advances in Neural Information Processing Systems*, 33:17153–17163, 2020.

Neel Nanda and Tom Lieberum. A mechanistic interpretability analysis of grokking, 2022. URL `https://www.alignmentforum.org/posts/N6WM6hs7RQMKDhYjB/a-mechanistic-interpretability-analysis-of-grokking`.

nostalgebraist. interpreting gpt: the logit len, 2020. URL `https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens`.

Chris Olah. Mechanistic interpretability, variables, and the importance of interpretable bases. `https://www.transformer-circuits.pub/2022/mech-interp-essay`, 2022. Accessed: 2022-15-09.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

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. *Advances in Neural Information Processing Systems*, 33:12388–12401, 2020.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *ArXiv*, abs/2206.07682, 2022.

5 Appendix

A Discovering the Circuit

A.1 Which heads directly write to the output? (Name Mover Heads)

We begin by identifying which attention heads directly affect the model's output: in other words, the heads writing in the residual stream at the END position, in a direction that has high dot product with the logit difference. Formally, let W_U denote the unembedding matrix, $\overline{\text{LN}}$ a layer norm operation (see Appendix H) and $W_U[IO]$, $W_U[S]$ the corresponding unembedding vectors for the IO and S tokens. We searched for heads (i, j) such that

$$\lambda_{i,j} \stackrel{\text{def}}{=} \mathbb{E}_{X \sim \text{p}_{\text{IOI}}}[\langle \overline{\text{LN}} \circ h_{i,j}(X), W_U[IO] - W_U[S] \rangle] \tag{1}$$

had large magnitude. Recall that $h_{i,j}(X)$ is the value that head (i, j) writes into the residual stream on input X. Therefore, heads with $\lambda_{i,j} > 0$ correctly promote the IO token over the S token (on average). The unembedding projection in (1) is called the *logit lens* and has been used in previous work to interpret intermediate activations (nostalgebraist, 2020) and parameters (Dar et al., 2022). We display the values of $\lambda_{i,j}$ in Figure 3 A. We see that only a few heads in the final layers have large logit projection $\lambda_{i,j}$. Specifically, 9.6, 9.9, and 10.0 have a large positive score, while 10.7 and 11.10 have a large negative score.

Name Mover Heads. To understand the positive heads, we first study their attention patterns. We find that they attend strongly to the IO token: the average attention probability of all heads over p_{IOI} is 0.59. Since attention patterns can be misleading (Jain & Wallace, 2019), we check whether attention is correlated with the heads' functionality. We do so by scatter plotting the attention probability against the logit score $\langle h_i(X), W_U[IO] \rangle$. The results are shown in Figure 3 B: higher attention probability on the IO token is linearly correlated with higher output in the IO direction (correlation $\rho > 0.81$, $N = 500$). Based on this result, we hypothesize that these heads (i) attend to names and (ii) copy whatever they attend to. We therefore call these heads *Name Mover Heads*.

To check that the Name Mover Heads copy names generally, we studied what values are written via the heads' OV circuits. We transform the output of the first layer at a name token through the OV matrix of a Name Mover Head and then project to the logits. The copy score is the proportion of samples that contain the input name token in the top 5 logits ($N = 1000$). We find that all three Name Mover Heads have a copy score above 95% (compared to less than 20% for an average head).

Negative Name Mover Heads. In Figure 3, we also observed two heads strongly writing opposite the $W_U[IO] - W_U[S]$ direction. We called these heads *Negative Name Mover Heads*. Their copy score is calculated with the negative of their OV matrix. As described in Figure 3, they share all the properties of Name Mover Heads, except they write in the opposite of names they attend to.

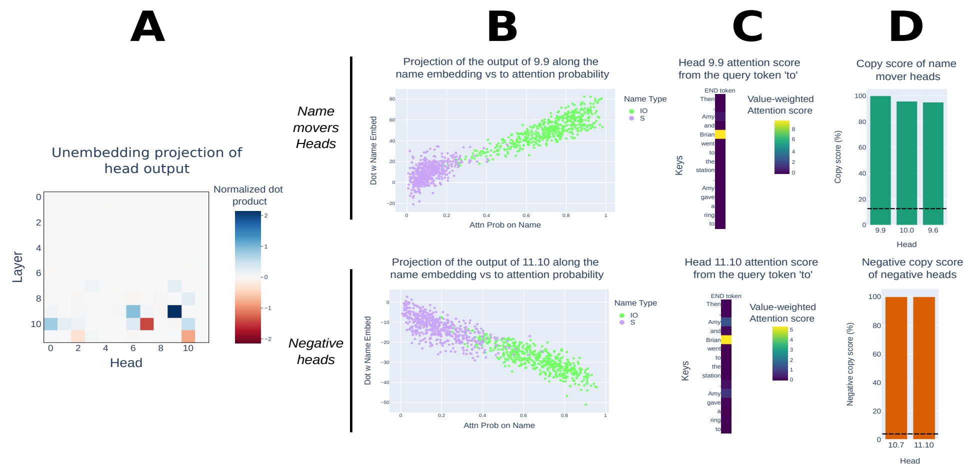

Figure 3: **A:** Name Movers and Negative Name Movers Heads are the heads that most strongly write in the $W_U[IO] - W_U[S]$ direction. **B**: Attention probability vs projection of the head output along $W_U[IO]$ or $W_U[S]$ respectively. Note that for S tokens, we sum the attention probability on both $S1$ and S2. **C**: Value-weighted attention score with the query at the end token. **D, top**: Positive copying score for the Name Mover Heads. **D, bottom**: Negative copying score for the Negative Name Mover Heads. Dashed lines are the average scores for all heads.

A.2 Which heads affect the Name Mover Heads' attention? (S-Inhibition Heads)

Given that the Name Mover Heads are primarily responsible for constructing the output, we ask why these Name Mover Heads pay preferential attention to the IO token. First, there are two ways to affect the Name Mover Heads's attention: through the query vector at the END token or the key vector at the IO token. Since the key vector appears early in the context, it likely does not contain much task-specific information, so we focus on the END query vector.

Then, by investigating Name Mover Heads on the ABC distribution (where the three names are distinct; see Section 2.2), we observed that their attention is not selective: they pay equal attention to the first two names. We thus ask: what has changed from the ABC distribution to the $\mathrm{p}_{\mathrm{IOI}}$ distribution to cause the Name Mover Heads to attend to the IO token preferentially?

To empirically answer this question, we perform a *patching* experiment. As illustrated in Figure 1 this technique consists of two steps. First we save all activations of the network run on a source sequence. Then we run the network on a *target* sequence, replacing some activations with the activations from the source sequence. We can then measure the behavior of the patched model. Doing this for each node individually locates the nodes that explain why model behavior is different in the source and target sequences.

In our case, we run activation patching with source sentences from the ABC distribution and target sentences from $\mathrm{p}_{\mathrm{IOI}}$. We then compute the change in attention probability from END to IO, averaged over the three Name Mover Heads. Since the Name Mover Heads attention on the IO is high in the $\mathrm{p}_{\mathrm{IOI}}$ distribution and low in ABC, patching at important heads from ABC to $\mathrm{p}_{\mathrm{IOI}}$ should decrease Name Mover Heads attention on IO. The results from patching every head at the END token position are shown in Figure 4, right. We observe that patching heads 7.3, 7.9, 8.6, 8.10 causes a decrease in the attention probability on IO, indicating that they are counterfactually important for the Name Mover Heads's attention probability on the IO token. We call these heads **S-Inhibition Heads**, because in $\mathrm{p}_{\mathrm{IOI}}$ they primarily cause the Name Mover Head attention to drop on the S tokens (thus increasing the attention on the IO token).

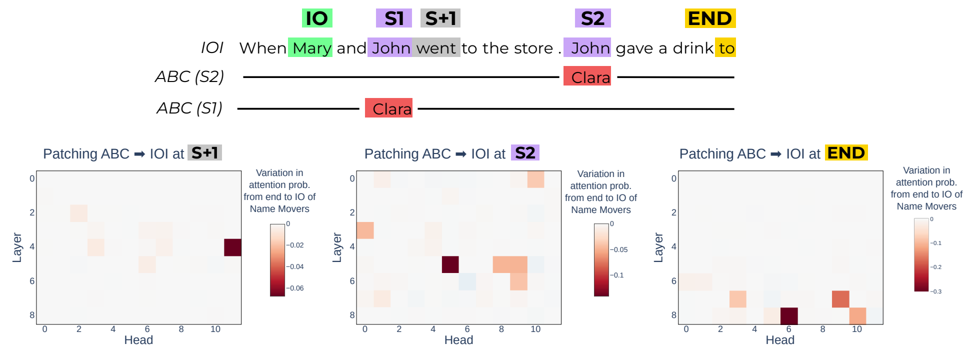

Figure 4: The attention probability to IO averaged over three Name Mover Heads is decreased most by the Previous Token Heads (left), Induction Heads (center) and S-Inhibition Heads (right) when we patch these attention heads from a sentence with a different S2 name (center and right), or a different S1 name (left).

A.3 What information do the S-Inhibition Heads move?

How do the S-Inhibition Heads differentiate between IO and S, so they inhibit one but not the other? We measured their attention pattern and found that they preferentially attend to the S2 token. We therefore studied what information these heads move from the S2 token position to the END position.

Towards this end, we ran a patching experiment at S2 from the ABC distribution to the IOI distribution and measured the variation in Name Mover Heads attention. The results (Figure 4, center) reveal a large set of heads influencing Name Mover Heads' attention that did not appear at the END position. Logically, S-Inhibition Heads mediate this effect, as they are the only heads influencing Name Mover Heads at the END position. This reasoning suggests that the outputs of this set of head is moved by S-Inhibition Heads from S2 to the END token. When we analyze the attention patterns of these heads, we see two distinct groups emerge.

Duplicate Token Heads. One group attends from S2 to S1. We call these Duplicate Token Heads on the hypothesis that they detect duplicate tokens. To validate this, we analyze their attention pattern on sequences of random tokens (with no semantic meaning), we found that 2 of the 3 Duplicate Token Heads pay strong attention to a previous occurrence of the current token if it exists (see Appendix F for more details). How do the duplicate token heads affect the S2 attention patterns? There is strong evidence that Duplicate Token Heads write a 'copying signal' into the residual stream that S2 Inhibition heads are able to attend to, that doesn't encode information about the tokens that are copied. Appendix G explores the correlational and causal case for this behavior.

Induction Heads and Previous Token Heads. The other group of heads attends from S2 to S1+1 (the token after the S1 token): the classic attention pattern of an induction head. Previously described in Elhage et al. (2021), induction heads recognize the general pattern `[A] [B] ... [A]` and contribute to predicting `[B]` as the next token. For this, they act in pair with a Previous Token Head. The Previous Token Head should write information about `[A]` into the residual stream at `[B]`, so that the Induction Head can match the next occurrence of `[A]` to that position (and subsequently copy `[B]` to the output).

We therefore seek to identify Previous Token Heads used by our purported Induction Heads. To this end, we patched activations from a sentence where S1 is replaced by a random name, at the S+1 token index. As shown in figure 4, some heads (and particularly 4.11) appear to influence Name Mover Heads. Then, by looking at the attention pattern of the heads with the most important influence in this patching experiment, we identified 3 Previous Token Heads. After analyzing attention patterns on random token sequences, we found that 2 of the 3 Previous Token Heads and 2 of the 4 Induction Heads demonstrated the expected behavior in this out-of-distribution case (Appendix F).

In our task, the Induction Heads writing into the S2 residual stream is an additional way for S-Inhibition Heads to detect that S occurs earlier in the context, on top of the Duplicate Token Heads' role. These Induction Heads, like Duplicate Token Heads, appear to be writing a copying signal into the residual stream at S2 (Appendix G), making them somewhat unlike traditional induction heads that simply copy the token `[B]`.

A.4 Did we miss anything? The Story of the Backup Name Movers Heads

Each type of head in our circuit has many copies, suggesting that the model implements redundant behavior. To make sure that we didn't miss any copies, we knocked out *all* of the Name Mover Heads at once. To our surprise, the circuit still worked (only 10% drop in logit difference). In addition, many heads write along $W_U[IO] - W_U[S]$ after the knockout, which did not do so previously.

We kept the height heads with the strongest $\lambda_{i,j}$, and call them *Backup Name Mover Heads*. See appendix D for further details on these heads. Among the height heads identified, we investigated their behavior before the knockout. We observe diverse behavior: 3 heads show close resemblance to Name Mover Heads; 3 heads equally attend to IO and S and copy them; 1 head pays more attention to S1 and copies it; 1 head seems to track and copy subjects of clauses, copying S2 in this case.

B Experimental validation

In this section, we check that our circuit provides a good account of GPT-2's true behavior. In general, our criteria depend on a measure F of the performance of a circuit on a task. In our case, suppose $X \sim \mathrm{p_{IOI}}$, and $f(C(X); X)$ is the logit difference between the IO and S tokens when the circuit C is run on the input X. The average logit difference $F(C) \overset{\text{def}}{=} \mathbb{E}_{X\sim \mathrm{p_{IOI}}}[f(C(X); X)]$ is a measure of how much a circuit predicts IO rather than S, i.e performs the IOI task.

Firstly, we check that C is **faithful** to M, i.e. that it computes similar outputs. We do so by measuring $|F(M) - F(C)|$, and find that it is small: 0.2, or only 6% of $F(M) = 3.55$.

In Section B.1 we define a running toy example of a model M for which faithfulness is not sufficient to prescribe which circuits explain a behavior defined by a measure F well. This motivates the criteria of completeness and minimality that we then check on our circuit.

B.1 Completeness

As a running example, suppose a model M uses two similar and disjoint serial circuits C_1 and C_2. The two sub-circuits are run in parallel before applying an OR operation to their results. Identifying only one of the circuits is enough to achieve faithfulness, but we want explanations that include both C_1 and C_2, since these are both used in the model's computation.

To solve this problem, we introduce the **completeness** criterion: for every subset $K \subseteq C$, $|F(C \setminus K) - F(M \setminus K)|$ should be small. In other words, C and M should not just be similar, but remain similar under knockouts.

In our running example, we can show that C_1 is not complete by setting $K = C_1$. Then $C_1 \setminus K$ is the empty circuit while $M \setminus K$ still contains C_2. The metric $|F(C_1 \setminus K) - F(M \setminus K)|$ will be large because $C_1 \setminus K$ has trivial performance while $M \setminus K$ successfully performs the task. However, this definition doesn't cover all cases of 'incompleteness' we would wish to cover[1].

The criterion of completeness requires a search over exponentially many subsets $K \subseteq C$. This is computationally intractable given the size of our circuit, hence we use three sampling methods to find examples of K that give large completeness score:

- The first sampling method chooses subsets $K \subseteq C$ uniformly at random.
- The second sampling method set K to be an entire class of circuit heads G, e.g the Name Mover Heads. $C \setminus G$ should have low performance since it's missing a key component, whereas $M \setminus G$ might still do well if it has redundant components that fill in for G.
- Thirdly, we greedily optimized K node-by-node to maximize the completeness score (see appendix K for the detail of the optimization procedure).

These first two methods of sampling K suggested to us that our circuit was ε-complete for a small value of ε. However, the third resulted in sets K that had high completeness score: up to 3.09. All such results are found in figure 5, on the left.

[1]If two nodes in $M \setminus K$ perfectly cancel each other, independent from C then the completeness definition could be met without including either of these important nodes in a circuit. We could also check that $\mathrm{F(C)}$ doesn't change after adding nodes: $\forall K \subseteq M, |F(C \setminus K) - F(M \setminus K)| \leq \varepsilon$. However, in practice such cases of compensation are improbable.

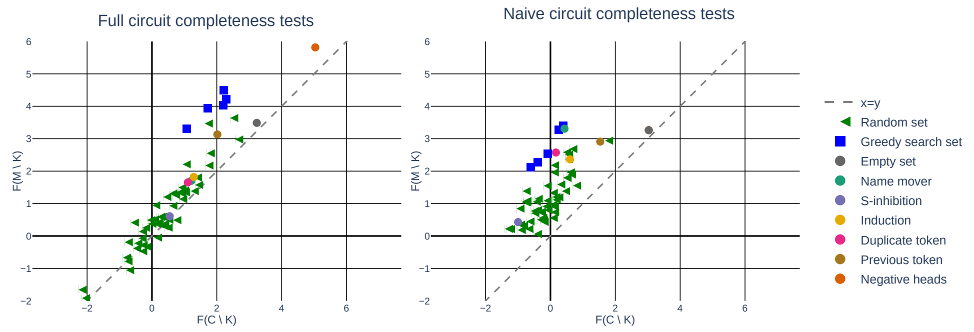

Figure 5: Plot of points $(x_K, y_K) = (\mathrm{F}(M \setminus K), \mathrm{F}(C \setminus K))$ for our circuit (left) and a naive circuit (right). Each point is for a different choice of K: 50 uniformly randomly chosen $K \subseteq C$, $K = \emptyset$, and the five K with the highest completeness score found by greedy optimization. Since the completeness score is $|x_K - y_K|$, we show the line $y = x$ for reference.

B.2 Minimality

A faithful and complete circuit may contain unnecessary components, and so be overly complex. To avoid this, we should check that each of its nodes v is actually necessary. This can be evaluated by showing that v can significantly recover F after knocking out a set of nodes K.

Formally, the **minimality** criterion is whether for every node $v \in C$ there exists a subset $K \subseteq C \setminus \{v\}$ that has minimality score $|F(C \setminus (K \cup \{v\})) - F(C \setminus K)| \geq A$. We call a circuit A-**minimal** if this holds.

In the running example, $C_1 \cup C_2$ is A-minimal for some non-trivial A. We can sketch a proof of this result given an informal defintion of 'non-trivial'. To show this, note that if $v_1 \in C_1$ and $K = C_2$, then the minimality score is equal to $|F(C_1 \setminus \{v_1\}) - F(C_1)|$ which is large since C_1 is a serial circuit and so removing v_1 will destroy the behavior. We then proceed symmetrically for $v_2 \in C_2$.

What happens in practice for our circuit? We need to exhibit for every v a set K such that the minimality score is at least A. For most heads, removing the class of heads G that v is a part of provides a reasonable minimality score. We describe the sets K that are required for them in Appendix J. The importance of individual nodes is highly variable, but they all have a significant impact on the final metric (at least 3% of the original logit difference). These results ensure that we did not interpret irrelevant nodes, but do show that the individual contribution of some single attention heads is small.

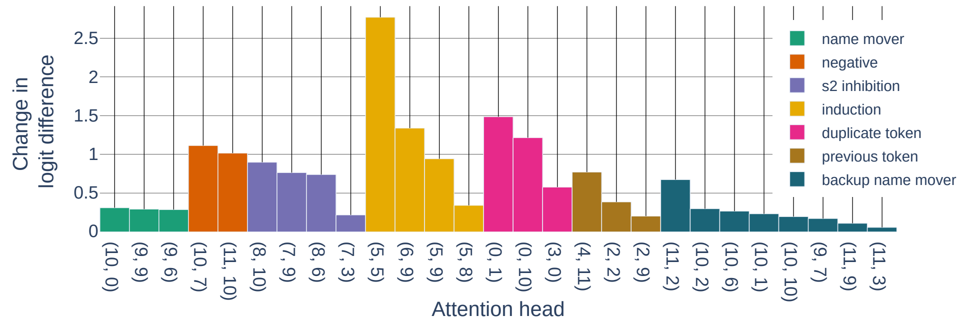

Figure 6: Plot of minimality scores $|F(C \setminus (K \cup \{v\})) - F(C \setminus K)|$ for all components v in our circuit. The sets K used for each component, as well as the initial and final values of the logit difference for each of these v is in Appendix J. Our circuit is 0.06-minimal.

B.3 Comparison with a naive circuit

In the previous sections, we reviewed our circuit on the three quantitative criteria. But without a relative comparison, these numbers are not particularly useful. In order to get a relative sense of the success of our explanation by our criteria, we compare the results on a naïve circuit that consists of the Name Mover Heads (but no Backup Name Mover Heads), S-Inhibition Heads, two Induction

Heads, two Duplicate Token Heads and two Previous Token Heads. This circuit has a faithfulness score 0.1, a score comparable to our circuit's faithfulness score. However, contrary to our circuit, the naive circuit can be easily proven incomplete: by sampling random sets or by knocking-out by classes, we see that $F(M \setminus K)$ is much higher than $F(C \setminus K)$ (Figure 5, left). Nonetheless, when we applied the greedy heuristic to optimize for the completeness score, both circuits have similarly large completeness scores. Thus, we conclude that our worst-case completeness criteria was too high a bar, which future work could use as a high standard to validate circuit discovery.

C IOI Templates

We list all the template we used in Table 7. Each name was drawn from a list of 100 English first names, while the place and the object were chosen among a hand made list of 20 common names. All the word chosen were one token long to ensure proper sequence alignment computation of the mean activations.

Templates in p_{IOI}
Then, [B] and [A] went to the [PLACE]. [B] gave a [OBJECT] to [A]
Then, [B] and [A] had a lot of fun at the [PLACE]. [B] gave a [OBJECT] to [A]
Then, [B] and [A] were working at the [PLACE]. [B] decided to give a [OBJECT] to [A]
Then, [B] and [A] were thinking about going to the [PLACE]. [B] wanted to give a [OBJECT] to [A]
Then, [B] and [A] had a long argument, and afterwards [B] said to [A]
After [B] and [A] went to the [PLACE], [B] gave a [OBJECT] to [A]
When [B] and [A] got a [OBJECT] at the [PLACE], [B] decided to give it to [A]
When [B] and [A] got a [OBJECT] at the [PLACE], [B] decided to give the [OBJECT] to [A]
While [B] and [A] were working at the [PLACE], [B] gave a [OBJECT] to [A]
While [B] and [A] were commuting to the [PLACE], [B] gave a [OBJECT] to [A]
After the lunch, [B] and [A] went to the [PLACE]. [B] gave a [OBJECT] to [A]
Afterwards, [B] and [A] went to the [PLACE]. [B] gave a [OBJECT] to [A]
Then, [B] and [A] had a long argument. Afterwards [B] said to [A]
The [PLACE] [B] and [A] went to had a [OBJECT]. [B] gave it to [A]
Friends [B] and [A] found a [OBJECT] at the [PLACE]. [B] gave it to [A]

Figure 7: Templates used in the IOI dataset. All templates in the table fit the 'BABA' pattern, but we also use templates that fit the 'ABBA' pattern as well (not included for simplicity).

D Backup Name Mover Heads

Here we discuss in more detail the discovery of the Backup Name Mover Heads. As shown in figure 8, knocking-out the three main Name Mover Heads doesn't leave the rest of the heads in a similar state as before. They seem to "compensate" the loss of function from the Name Mover Heads such that the logit difference is only 10% lower. We observe that the Negative Name Mover Heads head write less negatively in the direction of $W_U[IO] - W_U[S]$, 10.7 even write positively in this direction afterwards, while other heads that wrote slightly along $W_U[IO] - W_U[S]$ before the knock-out becomes the main contributor. Both the reason and the mechanism of this compensation effect are still unclear, we think that this could be an interesting phenomenon to investigate in future works. Among those last categories, we identify S-inhibition heads and a set of other head that we called *Backup Name Mover Heads*. We arbitrarily chose to keep the height heads that were not part of any other groups, and wrote in the direction of $W_U[IO] - W_U[S]$ above the threshold of 0.05.

In figure 9 we analyze the behavior of those newly identified heads with similar techniques as Name Mover Heads. Those can be grouped in 4 categories.

- 3 heads (10.1, 10.10 and 10.6) that behave similarly as Name Mover Heads according to their attention pattern, and scatter plots of attention vs dot product of their output with $W_U[IO] - W_U[S]$ (as 10.10).
- 3 heads (10.2, 11.9, 11.3) that pay equal attention to S1 and IO and wrote both of them (as 10.2 in Figure 9).

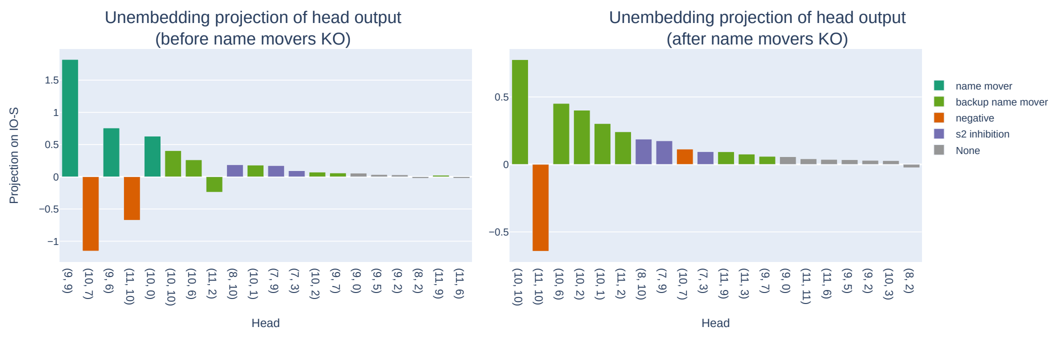

Figure 8: Discovery of the Backup Name Mover Heads. After knock-out of the Name Mover Heads (right) some heads write stronger in the $W_U[IO]$ or $W_U[S]$ direction than before (left). We also observed that negative heads seems inhibited by this operation.

- One head, 11.2, that pays more attention to S1 and write preferentially in the direction of $W_U[S]$
- One head, 9.7, that pays attention to S2 and write negatively.

We did not thoroughly investigate this diversity of behavior, more work can be done to precisely describe these heads. However, these heads are also the ones with the less individual importance for the task (as shown by their minimality score in Figure 6). The exact choice of Backup Name Mover Heads doesn't change significantly the behavior of the circuit.

E GPT-2 small full architecture

GPT-2 small is a decoder-only transformer with 12 layers and 12 attention heads per attention layer. In this work, we mostly focus on understanding the mechanisms of attention heads, which we describe using notation similar to Elhage et al. (2021). We leave a full description of the model to Appendix E.

The input to the transformer is the sum of position and token embeddings, $x_0 \in \mathbb{R}^{N\times d}$, where N is the number of tokens in the input and d is the model dimension. This input embedding is the initial value of the *residual stream*, which all attention layers and MLPs read from and write to. Attention layer i of the network takes as input $x_i \in \mathbb{R}^{N\times d}$, the value of the residual stream before it. The attention layer output can be decomposed into the sum of attention heads $h_{i,j}$. If the output of the attention layer is $y_i = \sum_j h_{i,j}(x_i)$, then the residual stream is updated to $x_i + y_i$.

Focusing on individual heads, each head $h_{i,j}$ is parametrized by four matrices $W_Q^{i,j}$, $W_K^{i,j}$, $W_V^{i,j} \in \mathbb{R}^{d\times \frac{d}{H}}$ and $W_O^{i,j} \in \mathbb{R}^{\frac{d}{H}\times d}$. We rewrite these parameters as low-rank matrices in $\mathbb{R}^{d\times d}$: $W_{OV}^{i,j} = W_O^{i,j}W_V^{i,j}$ and $W_{QK}^{i,j} = (W_Q^{i,j})^T W_K^{i,j}$. The QK matrix is used to compute the attention pattern $A_{i,j} \in \mathbb{R}^{N\times N}$ of head (i, j), while the OV matrix determines what is written into the residual stream. At the end of the forward pass, a layer norm is applied before the unembed matrix W_U projects the residual stream into logits.

Here we define all components of the GPT-2 Architecture, including those we don't use in the main text. GPT-2 small has the following hyperparameters:

- N: number of input tokens.
- V: vocabulary of tokens.
- d: residual stream dimension.
- L: number of layers.
- H: number of heads per layer.
- D: hidden dimension of MLPs

It uses layer norms, the non-linear function

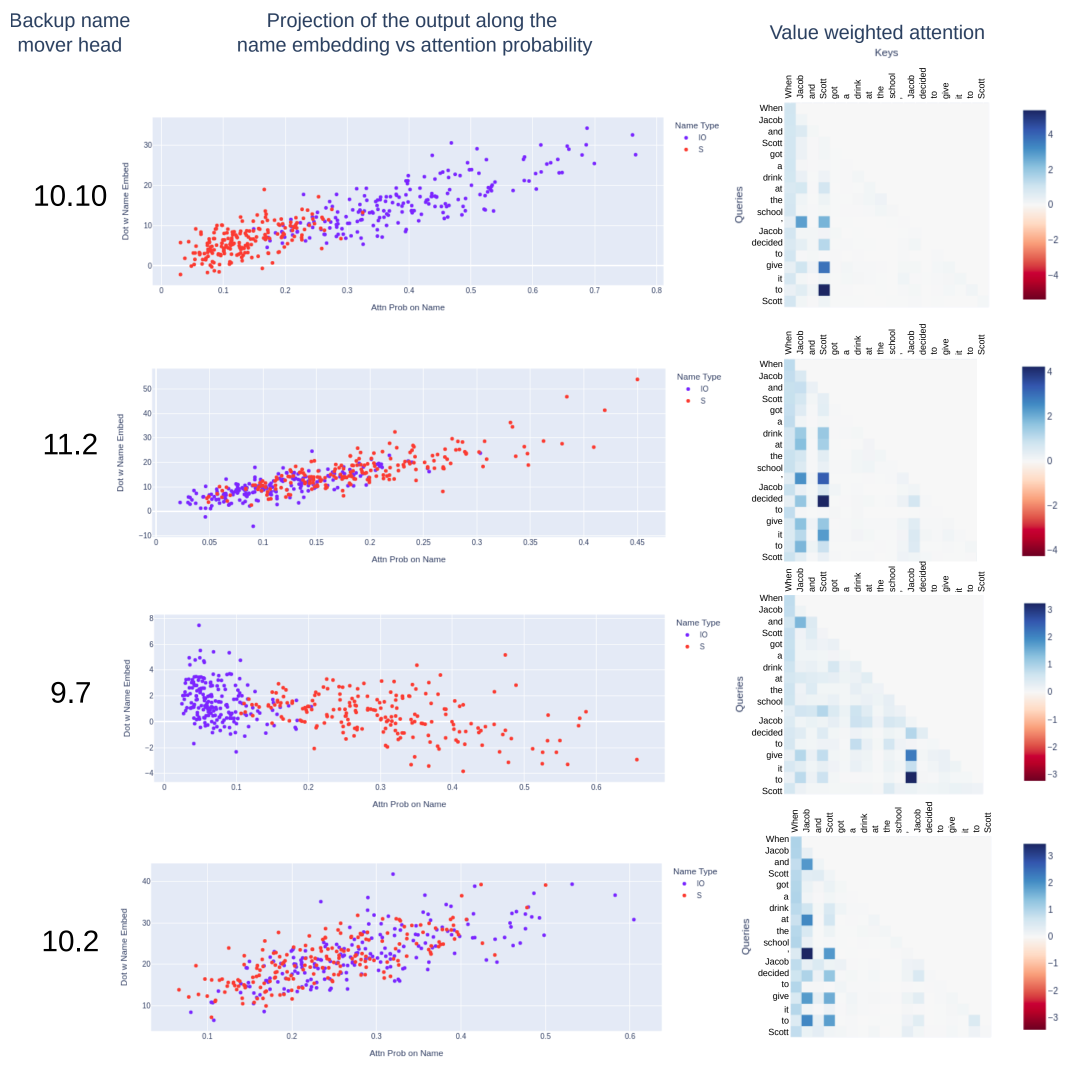

Figure 9: Four examples of Backup Name Mover Heads. Left: attention probability vs projection of the head output along $W_U[IO]$ or $W_U[S]$ respectively. Right: Attention pattern on a sample sequence.

$$\mathrm{LN}(x) \overset{\mathrm{def}}{=} \frac{x - \bar{x}}{\sqrt{\sum_i (x_i - \bar{x}_i)^2}}, \tag{2}$$

where the mean and the difference from the mean sum are over all components of the dimension d vector in each sequence position.

In GPT-2 the MLPs all have one hidden layer of dimension D and use the GeLU non-linearity.

We addressed the parametrisation of each attention head in the main text, and cover the technical details of the W_{QK} and W_{OV} matrix here: the attention pattern is $A_{i,j} = \mathrm{softmax}(x^T W_{QK}^{i,j} x)$ where the softmax is taken for each token position, and is unidirectional. We then have $h_{i,j}(x) \overset{\mathrm{def}}{=} (A_{i,j} \otimes W_{OV}^{i,j}).x$.

Algorithm 1 GPT-2.

```
Require: Input tokens $T$; returns logits for next token.
 1: $w \leftarrow$ One-hot embedding of T
 2: $x_0 \leftarrow W_E w$ (sum of token and position embeddings)
 3: for $i = 0$ to $L$ do
 4:     $y_i \leftarrow 0 \in \mathbb{R}^{N\times d}$
 5:     for $j = 0$ to $H$ do
 6:         $y_i \leftarrow y_i + h_{i,j}(x_i),$ the contribution of attention head $(i, j)$
 7:     end for
 8:     $y'_i \leftarrow m_i(x_i),$ the contribution of MLP at layer $i$
 9:     $x_{i+1} \leftarrow x_i + y_i + y'_i$ (update the residual stream)
10: end for
11: return $W_U \circ M \circ \text{LN} \circ x_L$
```

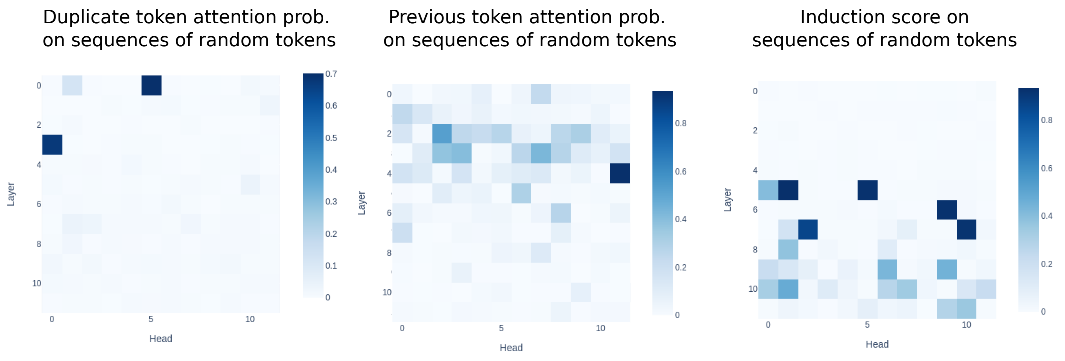

Figure 10: Sum of attention probabilities on position determined by the role. Left: duplicate score, the average attention probability from a token to its previous occurrence. Center: Previous token attention score, it is the average of the off diagonal attention probability. Right: Induction score. Average attention probability from the second occurence of `[A]` to `[B]` on `[A][B]...[A]`.

F Attention pattern analysis on sequences of random tokens

We run GPT-2 small on sequences of 100 tokens sampled uniformly at random from GPT-2's token vocabulary. Each sequence `A` was duplicated to form `AA`, a sequence twice as long where the first and second half are identical. On this dataset, we computed three scores from the attention patterns of the attention heads:

- The duplicate token score: for each token T_i in the second half of a sequence S, we average the attention probability from T_i to its previous occurrence in the first half of S (i.e. T_{i-100}).
- The previous token score: we averaged the attention probability on the off-diagonal. This is the attention from the token at position i to position $i - 1$.
- The induction score: the attention probability from T_i to the token that comes after the first occurrence of T_i (i.e. T_{i-99})

These three score are depicted in Figure 10 for all attention heads. We can identify 3.0 and 0.1 as duplicated token heads that also appear in our circuit, 5.5 and 6.9 have high induction score and were also identified as induction heads in our investigation and 4.11 and 2.2 have a high previous token score. Note that the heads identified are also the ones that have the highest influence in the patching experiment shown in Figure 4.

G Duplicate Token and Induction Copying Signal

We consider an experimental setup similar to the kind that generated Figure 6. We knock out all heads in a circuit class, record the logit difference, and then add back in a vertex of that circuit class and measure change in logit difference. We see extremely small effect sizes for Induction Heads and Duplicate Token Heads, when we knockout on the IOI distribution (Figure 11).

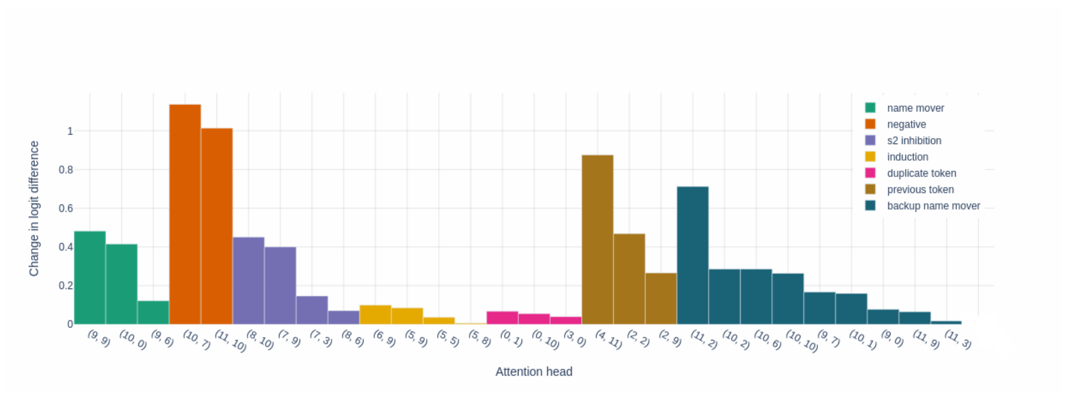

Figure 11: Very small effect change for Duplicate Token Heads and Induction Heads on the IOI distribution.

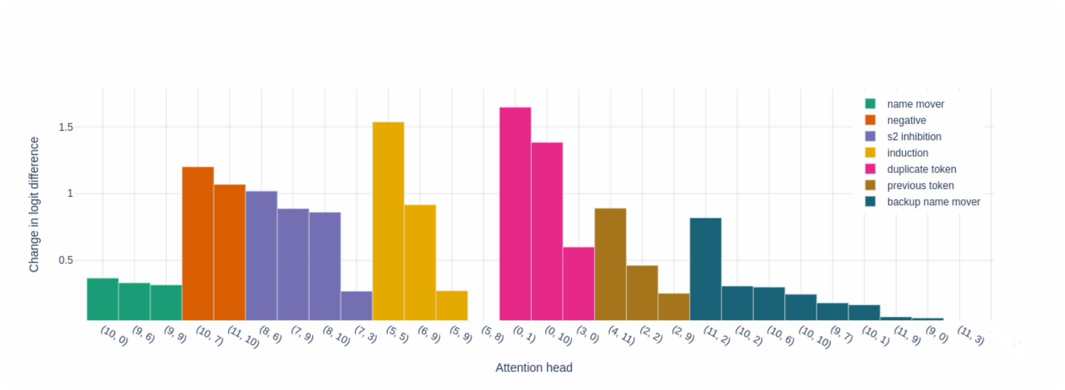

Figure 12: Similar results on the ABC distribution, besides the Duplicate Token Heads and Induction Heads.

On the other hand these two classes exhibit some of the largest effect sizes on the ABC distribution (Figure 12).

How would this occur? One hypothesis is that that these heads are not copying information about the token into the residual stream, but a constant 'signal' that copying is occurring. If these heads were doing this then we would expect that changing slightly where they attend to would not affect this signal being written, as copying can occur over a range of token positions. If there is no copying they would attend to the first token which wouldn't cause the copying signal to be written (Elhage et al. (2021) note induction heads attend to the first token when no induction is detected). We could test this causally by artificially cyclically permuting the attention patterns of these heads (provided we don't permute into the first token), and we would predict that they would still be able to write the copying signal. We permute the attention pattern of all Previous Token and Duplicate Token heads at the S token and the next 4 positions. For control, we also permute the Previous Token Head attention patterns at the same positions. We see almost constant logit difference for all these cyclic permutations, indeed suggesting the constant signal story is correct.

H Layer norm and the residual stream

The attention heads and MLPs in GPT-2 small write into the residual stream. Suppose x is the final state of the residual stream after the 12 layers, at a particular sequence position. This is then converted into logits via $W_U \circ M \circ \text{LN}(x)$, where LN is defined in Appendix E, M is the linear transformation of the layer norm operation and W_U is the unembedding matrix.

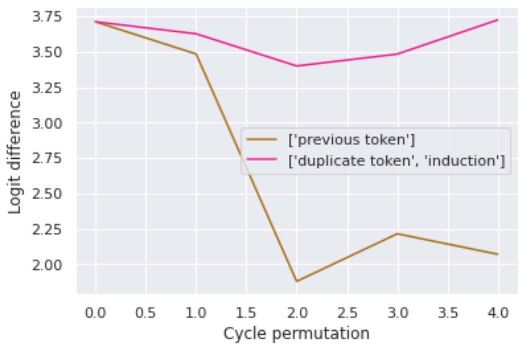

Figure 13: Cyclically permuting attention patterns of duplicate token and induction heads does not affect performance.

In order to attribute the extent to which an attention head $h_{i,j}$ writes in a direction $W_U[T]$ where T is a token (always IO or S in our case), we can't simply compute $\langle M \circ \text{LN} \circ h_{i,j}(X), W_U[T] \rangle$, as the scaling factor that's used is $\sqrt{\sum_i (x_i - \overline{x_i})^2}$. Therefore $\overline{\text{LN}}$ in the main text uses this scaling factor:

$$\overline{\text{LN}}(h) \overset{\text{def}}{=} M \circ \frac{h - \overline{h}}{\sqrt{\sum_i (x_i - \overline{x_i})^2}} \tag{3}$$

where h is the output of a head at the same sequence position as x.

I Role of MLPs in the task

In the main text, we focused our investigation on attention heads. Since they are the only module able of moving information across token position – a crucial component of the IOI task – they were our main subject of interest. However, MLP can still play a significant role in structuring the residual stream at a given position. We explored this possibility by performing knock-out of the MLP layers (Figure 14). We observe that MLP0 have a significant influence on logit difference after knock-out (-100% variation) but the other layers don't seem to play a big role. We hypothesize that MLP0 can be used to perform low level token processing that latter layers relies on.

Moreover, we also investigated the writing of MLP along the $W_U[IO] - W_U[S]$ direction. As shown in Figure 14, bottom, their λ_i score is negligible compared to attention heads (Figure 3, left).

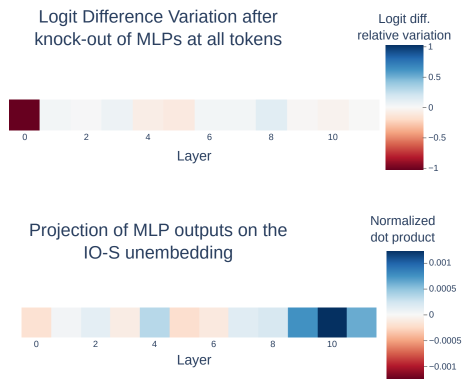

Figure 14: Top: Relative variation in logit difference from knocking out MLP layers. Only MLP0 causes a significative decrease in logit difference after knock-out. Bottom: We measure how much MLPs write along the $W_U[IO] - W_U[S]$ direction by computing the λ_i score for each MLP layer.

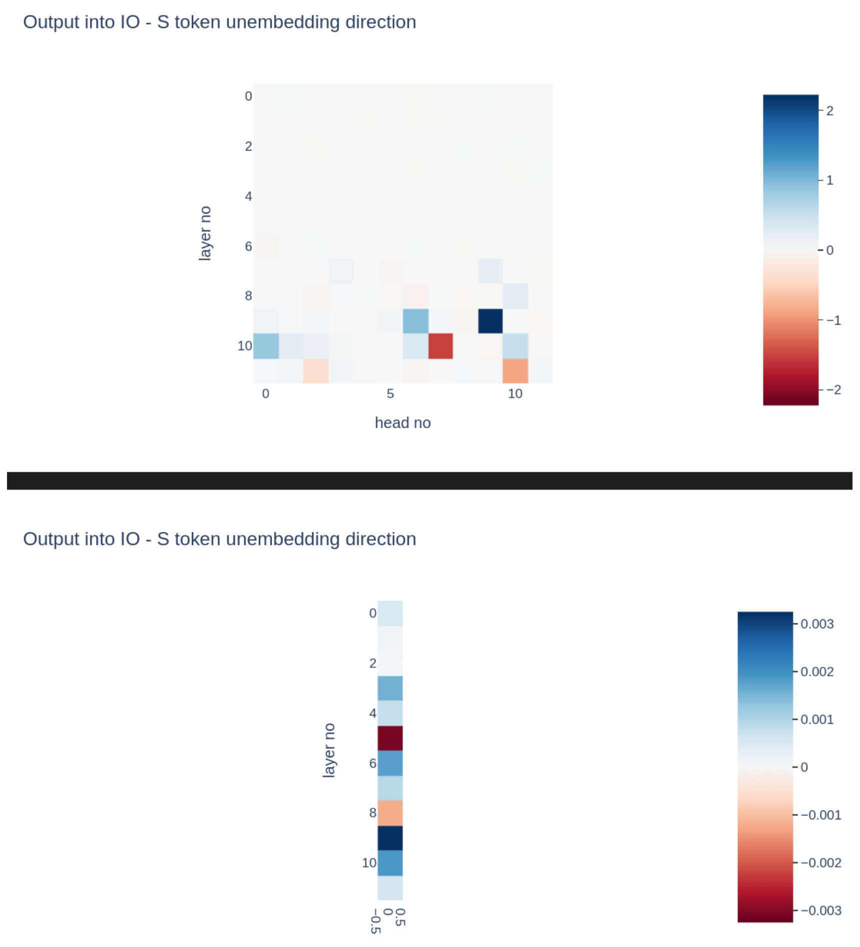

Figure 15: MLPs have a very small effect on writing in the IO - S direction (see scale).

J Minimality sets

The sets that were found for the minimality tests are listed in Table 16.

K Greedy Algorithm

The Algorithm 2 describes the procedure used to sample sets for checking the completeness criteria using greedy optimization. In practice, because the naïve and the full circuit are not of the same size, we chose respectively $k = 5$ and $k = 10$ to ensure a similar amount of stochasticity in the process. We run the procedure 10 times and kept the 5 sets with the maximal important completeness score (including the intermediate K).

Algorithm 2 The greedy sampling procedure for sets to validate the completeness citeria.

```
1: $K \leftarrow \emptyset$
2: for $i$ to $N$ do
3:     Sample a random subset $V \subseteq C$ of $k$ nodes uniformly.
4:     $v_{\text{MAX}} \leftarrow \arg\max_{v \in V} |\mathrm{F}(C \setminus (K \cup \{v\})) - \mathrm{F}(C \setminus K)|$
5:     $K \leftarrow K \cup \{v_{\text{MAX}}\}$
6: end for
7: return $K$
```

As visible in Table 17 the sets found by the greedy search contains a combination of nodes from different class. Nonetheless, the overlap between different K suggest that we are missing components

v	Class	$K \cup \{v\}$	$\mathrm{F}(C \setminus (K \cup \{v\}))$	$\mathrm{F}(C \setminus K)$
(9, 9)	Name Mover	[(9, 9)]	2.78	3.14
(10, 0)	Name Mover	[(9, 9), (10, 0)]	2.43	2.78
(9, 6)	Name Mover	[(9, 9), (10, 0), (9, 6)]	2.77	2.43
(10, 7)	Negative Name Mover	All Negative Name Mover Heads	5.11	3.84
(11, 10)	Negative Name Mover	All Negative Name Mover Heads	5.11	4.06
(7, 3)	S-Inhibition	All S-Inhibition Heads	0.33	1.15
(7, 9)	S-Inhibition	All S-Inhibition Heads	0.33	1.12
(8, 6)	S-Inhibition	All S-Inhibition Heads	0.33	1.10
(8, 10)	S-Inhibition	All S-Inhibition Heads	0.33	0.55
(5, 5)	Induction	Induction Heads and Negative Heads	1.06	3.95
(5, 8)	Induction	All Induction Heads	1.06	2.58
(5, 9)	Induction	All Induction Heads	4.40	5.11
(6, 9)	Induction	Induction Heads and Negative Heads	4.76	5.11
(0, 1)	Duplicate Token	All Duplicate Token Heads	1.14	2.52
(0, 10)	Duplicate Token	All Duplicate Token Heads	1.14	2.29
(3, 0)	Duplicate Token	All Duplicate Token Heads	1.14	1.65
(2, 2)	Previous Token	All Previous Token Heads	2.03	2.80
(2, 9)	Previous Token	All Previous Token Heads	2.03	2.42
(4, 11)	Previous Token	All Previous Token Heads	2.03	2.27
(10, 10)	Backup Name Mover	All NMs and previous Backup NMs	2.40	2.63
(10, 2)	Backup Name Mover	All NMs and previous Backup NMs	0.89	1.09
(11, 2)	Backup Name Mover	All NMs and previous Backup NMs	0.72	0.89
(10, 6)	Backup Name Mover	All NMs and previous Backup NMs	2.63	2.77
(10, 1)	Backup Name Mover	All NMs and previous Backup NMs	1.34	1.47
(9, 7)	Backup Name Mover	All NMs and previous Backup NMs	0.85	1.02
(11, 9)	Backup Name Mover	All NMs and previous Backup NMs	1.02	1.13
(11, 3)	Backup Name Mover	[(9, 9), (10, 0), (9, 6), (10, 10), (11, 3)]	2.53	2.59

Figure 16: K sets for minimality for each v.

K found by greedy optimization
(9, 9), (9, 6), (5, 8), (5, 5), (2, 2), (2, 9)
(9, 9), (11, 10), (10, 7), (8, 6), (5, 8), (4, 11)
(10, 7), (5, 5), (2, 2), (4, 11)
(9, 9), (11, 10), (10, 7), (11, 2), (3, 0), (5, 8), (2, 2)

Figure 17: 4 sets K found by the greedy optimization procedure on our circuit.

from M that can take the place of induction heads or S-inhibition Heads when some Name Mover Heads are knocked-out.

L Techniques Overview

This work involved a variety of techniques that were required to explain model behavior.

- **Knockouts**:
  We used knockouts in two different ways: knocking out singular components of models, and knocking out everything in the model except particular circuits. The former was somewhat useful, and the latter we found powerful.
  - Knockout of single components: as an attribution method, knocking out singular components was not always as powerful as techniques such as projections, since the compensation (or backup) nature of Backup Name Mover Heads in this task allowed components to be knocked out and their true effect size masked.
  - Knockouts of all components except a circuit: on the other hand, knocking out all components except a circuit enabled us to isolate behaviors in this task where behavior was sparse, and check the components of our circuit while ignoring the vast percentage of components of the network, making work manageable.

What was very important for the success of knockout experiments was the choice of reference distribution for knockout. The analysis in Appendix G shows the large difference between the IOI distribution knockouts and the ABC distribution knockouts in our work, for example. This situation more generally exposes how delicate and important the distribution for mean-ablations is. For a more general knockout, the OpenWebText dataset, GPT's training data, can be used. However, we found that this led to noisier results (though our circuit components still passed the tests of significance when these ablations were used).

- **Attention pattern analysis**:
  Using attention patterns to explain behavior is always worrying due to the possibility that information has accumulated on that token primarily from previous tokens, or that the position with large attention paid to isn't actually writing an important value into the residual stream. In our work however, analyzing attention patterns was generally a necessary first step before further experiments could be ran, and in this small model, both of the worrying cases did not generally arise.
- **Patching**:
  Patching was an important method we used to verify causal explanations that were generally formed from correlational evidence. In this way our use case is similar to (Finlayson et al. (2021)). We were surprised however that in general patching gave clear signal on the changes in behavior. This may be because we generally patched from inputs like the ABC distribution (which was successful in knocking out too). Therefore, keeping the context of the sentence templates may be generally useful. This could be either because the other words in the templates allow the model to realise that it should be doing IOI, or that introducing inputs from other distributions introduces noise that the model picks up on and uses, when this is not intended.