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

Kevin Wang[1], Alexandre Variengien[1], Arthur Conmy[1], Buck Shlegeris[1] & Jacob Steinhardt[1,2]
[1]Redwood Research
[2]UC Berkeley
`kevin@rdwrs.com, alexandre@rdwrs.com,`
`arthur@rdwrs.com, buck@rdwrs.com, jsteinhardt@berkeley.edu`

Abstract

Research in mechanistic interpretability seeks to explain behaviors of machine learning (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 called indirect object identification (IOI). Our explanation encompasses 26 attention heads grouped into 7 main classes, which we discovered using a combination of interpretability approaches relying on causal interventions. 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, pointing toward opportunities to scale our understanding to both larger models and more complex tasks. Code for all experiments is available at `https://github.com/redwoodresearch/Easy-Transformer`.

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 (Zhang et al., 2022; Caldarini et al., 2022), underscoring the urgency of understanding and anticipating possible model behaviors. Some researchers have argued that interpretability is critical 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 mechanistically understand how GPT-2 small (Radford et al., 2019) implements a simple natural language task. We do so by using *circuit analysis* (Räuker et al., 2022), identifying an induced subgraph of the model's computational graph that is human-understandable and responsible for completing the task.

To discover the circuit, we introduce a systematic approach that iteratively traces important components back from the logits, using a causal intervention that we call "path patching". We supplement this mainline approach with projections in the embedding space, attention pattern analysis, and activation patching to understand the behavior of each component.
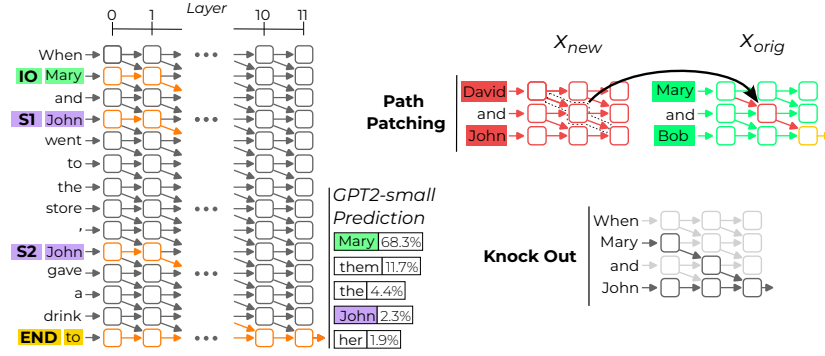
Figure 1: Left: We isolated a *circuit* (in orange) responsible for the flow of information connecting the indirect object "Mary" to the next token prediction. The nodes are attention layers and the edges represent the interactions between these layers. Right: We discovered and validated this circuit using causal interventions, including both path patching and knockouts of attention heads.

We focus on understanding a specific 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 an 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 26 attention heads–1.1% of the total number of (head, token position) pairs–that completes the bulk of this task. The circuit uses 7 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 an end-to-end behavior in a transformer-based language model.

By zooming in on a crisp task in a particular model, we obtained several insights about the challenges of mechanistic interpretability. In particular:

- We identified several instances of heads implementing redundant behavior (Michel et al., 2019). The most surprising were "Backup Name-Mover Heads", which copy names to the correct position in the output, but only when regular Name-Mover Heads are ablated. This complicates the search for complete mechanisms, as different model structure is found when some components are ablated.
- We found known structures (specifically induction heads (Elhage et al., 2021)) that were used in unexpected ways. Thus mainline functionality of a component does not always give a full picture.
- Finally, we identified heads reliably writing in the *opposite* direction of the correct answer.

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 (Section 4).

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. A sentence containing indirect object identification (IOI) has an initial dependent clause, e.g "When Mary and John went to the store", and a main clause, e.g "John gave a bottle of milk to Mary". The initial clause introduces the indirect object (IO) "Mary" and the subject (S) "John". The main clause refers to the subject a second time, and in all our examples of IOI, the subject gives an object to the IO. The IOI task is to predict the final token in the sentence to be the IO. We use 'S1' and 'S2' to refer to the first and second occurrences of the subject, when we want to specify position. We create dataset samples for IOI using 15 templates (see Appendix E) with random single-token names, places and items. We use the notation p_{IOI} for the distribution over sentences generated by this procedure.

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 p_{IOI}. Over 100,000 dataset examples, GPT-2 small has mean logit difference of 3.56 (IO predicted over S 99.3% of the time), and mean IO probability of 49%.

Transformer 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 behavior of attention heads, which we describe below using notation similar to Elhage et al. (2021). We leave a full description of the model to Appendix G.

The input x_0 to the transformer is a sum of position and token embeddings and lies 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 at layer i. 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 $W_Q^{i,j}, W_K^{i,j}, W_O^{i,j} \in \mathbb{R}^{d \times \frac{d}{H}}$ and $W_V^{i,j} \in \mathbb{R}^{\frac{d}{H} \times d}$. 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 $W_{QK}^{i,j} = W_Q^{i,j} (W_K^{i,j})^T$. 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.

2.1 Circuits and Knockouts

In mechanistic interpretability, we want to understand the correspondence between the components of a model and human-understandable concepts. A useful abstraction for this goal is *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 more coarse-grained than the one presented in Olah et al. (2020), where nodes are features (meaningful directions in the latent space of a model) instead of model components.

Just as the entire model M defines a function $M(x)$ from inputs to logits, we also associate each circuit C 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 set 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. Perhaps 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 remove the information that *varies* in the reference distribution (e.g. the value of the name outputted by a head) but will preserve *constant* information (e.g. the fact that a head is outputting a name).

In this work, all knockouts are performed in a modified p_{IOI} distribution called p_{ABC}. It relies on the same generation procedure, but instead of using two names (IO and S) it used three unrelated random names (A, B and C). In p_{ABC}, sentences no longer have a single plausible IO, but the grammatical structures from the p_{IOI} templates are preserved.

We chose this distribution for mean-ablating because using p_{IOI} would not remove enough information helpful for the task. For instance, some information constant in p_{IOI} (e.g. the fact that a name is duplicated) is removed when computing the mean on p_{ABC}.

When knocking out a single node, a (head, token position) pair in our circuit, we want to preserve the grammatical information unrelated to IOI contained in its activations. However, the grammatical role (subject, verb, conjunction etc.) of a particular token position varies across templates. To ensure that grammatical information is constant when averaging, we compute the mean of a node across samples of the same template.

3 DISCOVERING THE CIRCUIT

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 duplicated (in the example above: John).
3. Output the remaining name.

Below we present a circuit that we claim implements this functionality. Our circuit contains three major classes of heads, corresponding to the three steps of the algorithm above:

- *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 signal that token duplication has occurred by writing the position of the duplicate token.
- *S-Inhibition Heads* remove duplicate tokens from Name Mover Heads' attention. They are active at the END token, attend to the S2 token, and write in the query of the Name Mover Heads, inhibiting their attention to S1 and S2 tokens.
- *Name Mover Heads* output the remaining name. They are active at END, attend to previous names in the sentence, and copy the names they attend to. Due to the S-Inhibition Heads, they attend to the IO token over the S1 and S2 tokens.
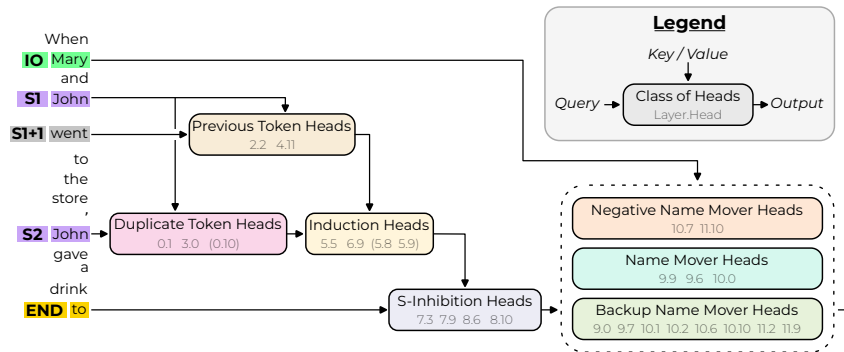

Figure 2: We discover a circuit in GPT-2 small that implements IOI. The input tokens on the left are passed into the residual stream. Attention heads move information between residual streams: the query and output arrows show which residual streams they write to, and the key/value arrows show which residual 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 information about the token S to token S1+1, the token after S1.
- *Induction Heads* perform the same role as the Duplicate Token Heads through an induction mechanism. They are active at position S2, attend to token S1+1 (mediated by the Previous Token Heads), and their output is used both as a pointer to S1 and as a signal that S is duplicated.
- 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 all of our experiments, we do not intervene on MLPs, layer norms, or embedding matrices, focusing our investigation on understanding the attention heads. In initial investigations, we found that knocking out each MLP individually led to good task performance, except for the first layer. However, knocking out all MLPs after the first layer makes the model unable to perform the task (Appendix J). A more precise investigation of the role of MLPs is left for future work.

Below we show how we discovered each of the seven components above, providing evidence that they behave as claimed. We found that it was easiest to uncover the circuit starting at the logits and working back step-by-step. Each step is divided into two parts. First, we trace the information flow backward from previously discovered components by finding attention heads that directly influence them; we do this using a technique we introduce called *path patching*, which generalizes the application of causal mediation analysis to language models introduced in Vig et al. (2020). Second, we characterize the newly identified heads by examining their attention patterns and performing experiments tailored to their hypothesized function.

3.1 WHICH HEADS DIRECTLY AFFECT THE OUTPUT? (NAME MOVER HEADS)

Tracing back the information flow. We begin by searching for attention heads h directly affecting the model's logits. To differentiate indirect effect (where the influence of a component is mediated by another head) from direct effect, we designed a technique called **path patching** (Figure 1).

Path patching replaces part of a model's forward pass with activations from a different input. Given inputs x_{orig} and x_{new}, and a set of paths $\mathcal{P}$ emanating from a node h, path patching runs a forward pass on x_{orig}, but for the paths in $\mathcal{P}$ it replaces the activations for h with those from x_{new}. In our case, h will be a fixed attention head and $\mathcal{P}$ consists of all direct paths from h to a set of components R, i.e. paths through residual connections and MLPs (but not through other attention heads); this measures the counterfactual effect of h on the members of R. The layers after the element from R are recomputed as in a normal forward pass. See Appendix B for full details of the method.

We will always take x_{orig} to be a sample from p_{IOI}, and x_{new} the corresponding sample from p_{ABC} (i.e. where the names in the sentence are replaced by three random names). We run path patching on many random samples from p_{IOI} and measure how this affects the average logit difference. Pathways $h \rightarrow R$ that are critical to the model's computation should induce a large drop in logit difference when patched, since the p_{ABC} distribution removes the information needed to complete the IOI task.

To trace information back from the logits, we run path patching for the pathway $h \rightarrow$ Logits for each head h at the END position and display the effect on logit difference in Figure 3a. We see that only a few heads in the final layers cause a large effect on logit difference. Specifically, patching 9.6, 9.9, and 10.0 causes a large drop (they thus contribute positively to the logit difference), while 10.7 and 11.10 cause a large increase (they contribute negatively to the logit difference).

Name Mover Heads. To understand the heads positively influencing the logit difference, 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. We hypothesize that these heads (i) attend to the correct name and (ii) copy whatever they attend to. We therefore call these heads *Name Mover Heads*.

To verify our hypothesis, we design experiments to test the heads' functionality. Let W_U denote the unembedding matrix, and $W_U[IO]$, $W_U[S]$ the corresponding unembedding vectors for the IO and S tokens. We scatter plot the attention probability against the logit score $\langle h_i(X), W_U[N] \rangle$, measuring how much head h_i on input X is writing in the direction of the logit of the name N
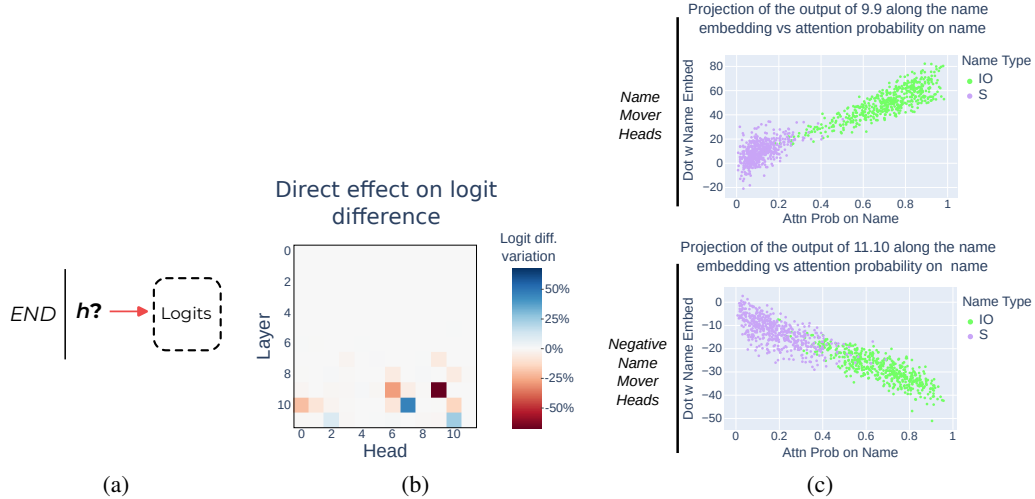
Figure 3: (a) We are searching for heads h directly affecting the logits using path patching. (b) Results of the path patching experiments. Name Movers and Negative Name Movers Heads are the heads that have the strongest direct effect on the logit difference. (c) Attention probability vs projection of the head output along $W_U[IO]$ or $W_U[S]$ respectively. For S tokens, we sum the attention probability on both S1 and S2.

(IO or S). The results are shown in Figure 3c: higher attention probability on the IO or S token is correlated with higher output in the direction of the name (correlation $\rho > 0.81$, $N = 500$)[1].

To check that the Name Mover Heads copy names generally, we studied what values are written via the heads' OV matrix. Specifically, we first obtained the state of the residual stream at the position of each name token after the first MLP layer. Then, we multiplied this by the OV matrix of a Name Mover Head (simulating what would happen if the head attended perfectly to that token), multiplied by the unembedding matrix, and applied the final layer norm to obtain logit probabilities. We compute the proportion of samples that contain the input name token in the top 5 logits ($N = 1000$) and call this the copy score. 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 3b, we also observed two heads causing a large increase, and thus negatively influencing the logit difference. We called these heads *Negative Name Mover Heads*. These share all the same properties as Name Mover Heads except they (1) write in the opposite direction of names they attend to and (2) have a large *negative* copy score–the copy score calculated with the negative of the OV matrix (98% compared to 12% for an average head).

3.2 WHICH HEADS AFFECT THE NAME MOVER HEADS' ATTENTION? (S-INHIBITION HEADS)

Tracing back the information flow. Given that the Name Mover Heads are primarily responsible for constructing the output, we next ask what heads they depend on. As illustrated in Figure 4a, there are three ways to influence attention heads: by their values, keys, or queries. In the case of Name Mover Heads, their value matrix appears to copy the input tokens (see previous section), so we do not study it further. Moreover, as the IO token appears early in the context, the Name Mover Heads' key vectors for this position likely do not contain much task-specific information. Therefore, we focus on the query vector, which is located at the END position.

We again use path patching, identifying the heads that affect each Name Mover Head's query vector. Specifically, for a head h we patch the path $h \rightarrow$ Name Mover Heads' queries (Figure 4a) and report

---

[1]In GPT-2 small, there is a layer norm before the final unembedding matrix. This means that these dot products computed are not appropriately scaled. However, empirically we found that approaches to approximating the composition of attention head outputs with this layer norm were complicated, and resulted in similar correlation and scatter plots.
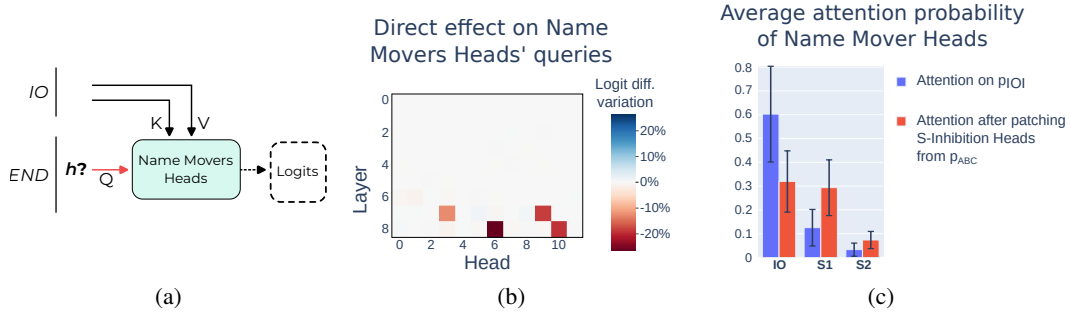
Figure 4: (a) Diagram of the direct effect experiments on Name Mover Heads' queries. We patch in the red path from the p_{ABC} distribution. Then, we measure the indirect effect of h on the logits (dotted line). All attention heads are recomputed on this path. (b) Result of the path patching experiments. The four heads causing a decrease in logit difference are S-Inhibition Heads. (c) Attention of Name Mover Heads on p_{IOI} before and after path patching of S-Inhibition Heads $\rightarrow$ Name Mover Heads' queries for all four S-Inhibition Heads at once (black bars show the standard deviation). S-Inhibition Heads are responsible for Name Mover Heads' selective attention on IO.
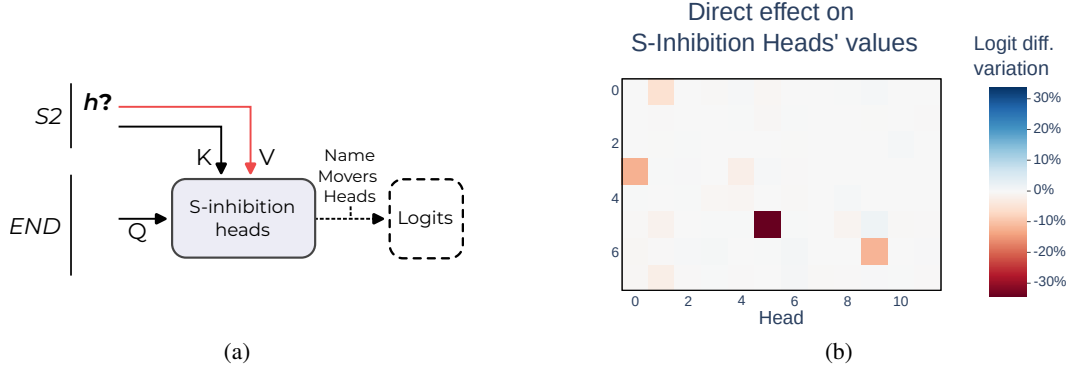

Figure 5: (a) Diagram of the direct effect experiment on S-Inhibition Heads' values. On the indirect path from h to the logits (dotted line), all elements are recomputed, Name Movers Heads are mediating the effect on logits. (b) Result of the path patching experiments for the S-Inhibition Heads' values.

the effect on logit difference in Figure 4b. Heads 7.3, 7.9, 8.6, 8.10 directly influence the Name Mover Heads' queries, causing a significant drop in logit difference through this path.

S-Inhibition Heads. To better understand the influence of these heads on the Name Movers Heads, we visualized Name Movers Heads' attention before and after applying path patching of all four previously identified heads at once (Figure 4c). Patching these heads leads to a greater probability on S1 (and thus less on IO). We therefore call these heads *S-Inhibition Heads*, since their effect is to decrease attention to S1.

What are S-Inhibition heads writing? We observed that S-Inhibition heads preferentially attend to the S2 token (the average attention from END to S2 was 0.51 over the four heads). We conjectured that they move information about S that causes Name Movers to attend less to S1. We discovered two signals through which S-Inhibition Heads do this. The first, the *token signal*, contains the value of the token S and causes Name Mover Heads to avoid occurrences of that token. The second, the *position signal*, contains information about the position of the S1 token, causing Name Mover Heads to avoid the S1 position no matter the value of the token at this position.

To disentangle the two effects, we designed a series of counterfactual experiments where only some signals are present, and the others are inverted with respect to the original dataset (see Appendix A). Both effects matter, but position signals have a greater effect than token signals. These two signals suffice to account for the full average difference in Name Mover Heads' attention to IO over S1.

S-Inhibition heads output information about the position of S1 but don't attend to it directly. They thus must rely on the output of other heads through value composition, as we will see next.

3.3 WHICH HEADS AFFECT S-INHIBITION VALUES?

Tracing back the information flow. We next trace the information flow back through the queries, keys, and values of the S-Inhibition Heads, using path patching as before. We found the values were the most important and so focus on them here, with keys discussed in Appendix C (queries had no significant direct effect on the S-Inhibition Heads).

In more detail, for each head h at the S2 position we patched in the path $h \rightarrow$ S-Inhibition Heads's values (Figure 5a) and reported results in Figure 5b.

We identified 4 heads with a significant negative effect on logit difference. By looking at their attention pattern, we separated them into two groups. The heads from the first group attend from S2 to S1, while those from the second group attend from S2 to the token after S1 (S1+1 in short).

Duplicate Token Heads. Two of the heads attend from S2 to S1. We hypothesize that these heads attend to the previous occurrence of a duplicate token and copy the position of this previous occurrence to the current position. We therefore call them *Duplicate Token Heads*.

To partially validate this hypothesis, we investigated these heads' attention patterns on sequences of random tokens (with no semantic meaning). We found that the 2 Duplicate Token Heads pay strong attention to a previous occurrence of the current token when it exists (see Appendix I).

Induction Heads and Previous Token Heads. The other group of two heads attends from S2 to S1+1. This matches the attention pattern of *Induction Heads* (Elhage et al., 2021), which recognize the general pattern [A] [B] ... [A] and contribute to predicting [B] as the next token. For this, they act in pairs with Previous Token Heads. The Previous Token Heads write information about [A] into the residual stream at [B]. Induction Heads can then match the next occurrence of [A] to that position (and subsequently copy [B] to the output).

To test the hypothesis that this group of heads consists of Induction Heads, we sought out the Previous Token Heads that they rely on through key composition. To this end, we apply path patching to find heads that influence logit difference through the S1+1 keys of the Induction Heads (see Appendix D). We identified two heads with significant effect sizes. To test the induction mechanism, we followed the method introduced in Olsson et al. (2022) by checking for prefix-matching (attending to [B] on pattern like [A] [B] ... [A]) and copying (increasing the logit of the [B] token) of Induction Heads on repeated random sequences of tokens. We also analyzed the attention patterns of Previous Token Heads on the same dataset. As reported in Appendix H, Previous Token Heads attend primarily to the previous token and 2 Induction Heads demonstrate both prefix-matching and copying.

Locating the position signal. Despite their differences, the attention patterns of both groups of heads are determined by the position of S1, suggesting that their output could also contain this information. We thus hypothesized that Duplicate Token Heads and Induction Heads are the origin of the position signal described in Section 3.2, which is then copied by S-Inhibition Heads.

To test this hypothesis, we applied the same technique from the previous section to isolate token and position signals. We patched the output of the Induction and Duplicate Token Heads from sentences with altered position signals. We observed a drop in logit difference of almost the same size as patching the S-Inhibition Heads (at least 88%). However, patching their output from a dataset with a different S1 and S2 token but at the same position doesn't change the logit difference (less than 8% of the drop induced by patching S-Inhibition Heads).

Missing validations and caveats. To fully validate the claimed function of the Duplicate Token and Induction Heads, we would want to perform additional checks that we omitted due to time constraints. First, for Duplicate Token Heads, we would want to study the interaction of the QK matrix with the token embeddings to test if it acts as a collision detector. Moreover, we would investigate the OV matrix to ensure it is copying information from positional embeddings. For Induction Heads, we would perform the more thorough parameter analysis in Elhage et al. (2021) to validate the copying and key composition properties at the parameter level.

Moreover, in the context we study, Induction Heads perform a different function compared to what was described in their original discovery. In our circuit, Induction Heads' outputs are used as positional signals for S-Inhibition values. We did not investigate how this mechanism was implemented. More work is needed to precisely characterize these newly discovered heads and understand how they differ from the original description of Induction Heads.

3.4 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[2]. To make sure that we didn't miss any copies, we knocked out all the Name Mover Heads at once. To our surprise, the circuit still worked (only 5% drop in logit difference). After the knockout, we found the new heads directly affecting the logits using the same path patching experiment from Section 3.1. These new heads compensate for the loss of Name Movers Heads by replacing their role.

We selected the eight heads with the highest direct influence on logit difference after knockout and call them *Backup Name Mover Heads*. We investigated their behavior before the knockout. We observe diverse behavior: 4 heads show a close resemblance to Name Mover Heads, 2 heads equally attend to IO and S and copy them, 1 head pays more attention to S1 and copies it, and 1 head seems to track and copy subjects of clauses, copying S2 in this case. See Appendix F for further details on these heads.

We hypothesize that this compensation phenomenon is caused by the use of dropout during training. Thus, the model was optimized for robustness to dysfunctional parts. More work is needed to determine the origin of this phenomenon and if such compensation effects can be observed beyond our special case.

4 EXPERIMENTAL VALIDATION

In the previous section, we provided an end-to-end explanation of how the model produces a high logit difference between IO and S on the IOI task. Yet Section 3.4 shows that our experimental methods do not fully account for all name-moving in the model: some new heads take on this role after the main Name Mover Heads are knocked out. We therefore seek more systematic validations to check that our account in Section 2.1 includes all components used by the model to perform IOI.

To formalize this, we first introduce notation to measure the performance of a circuit C on the IOI task. Suppose $X \sim \mathsf{p}_{\text{IOI}}$, 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 (recall that $C(X)$ is defined by knocking out all nodes outside C, as described in Section 2.1). We define the average logit difference $F(C) \stackrel{\text{def}}{=} \mathbb{E}_{X \sim \mathsf{p}_{\text{IOI}}}[f(C(X); X)]$ as a measure of how well C performs the IOI task.

Using F, we introduce three criteria for validating a circuit C. The first one, **faithfulness**, asks that C has similar performance to the full model M, as measured by $|F(M) - F(C)|$. For the circuit C from Figure 2, we find that $|F(M) - F(C)| = 0.46$, or only 13% of $F(M) = 3.56$. In other words, C achieves 87% of the performance of M.

However, as the Backup Name Mover Heads illustrate, faithfulness alone is not enough to ensure a complete account. In Section 4.1 we introduce a toy example that illustrates this issue in more detail and use this to define two other criteria—completeness and minimality—for checking our circuit.

4.1 COMPLETENESS

As a running example, suppose a model M uses two identical 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.

---

[2]It is unlikely that SGD would select for perfectly redundant modules, but redundancy is more likely when analyzing a sub-distribution of all of a model's training data or a subtask of its behavior.
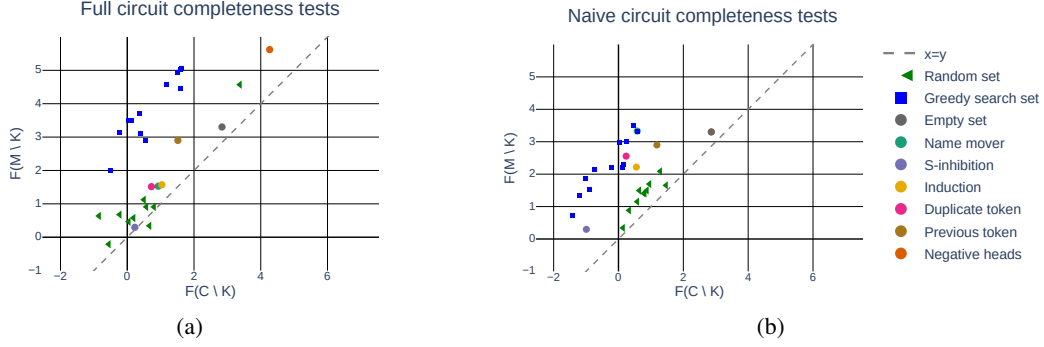
Figure 6: Plot of points $(x_K, y_K) = (F(C \setminus K), F(M \setminus K))$ for (a) our circuit and (b) a naive circuit. Each point is for a different choice of K. We show sets K obtained from different sampling strategies: 10 uniformly randomly chosen $K \subseteq C$, $K = \emptyset$, K a class of heads from the circuit, and 10 K found by greedy optimization. Since the incompleteness score is $|x_K - y_K|$, we show the line $y = x$ for reference.

To solve this problem, we introduce the **completeness** criterion: for every subset $K \subseteq C$, the *incompleteness score* $|F(C \setminus K) - F(M \setminus K)|$ 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.

Testing 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 seek sets K that give large incompleteness scores:

- 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 incompleteness score (see Appendix M for details).

All results are shown in Figure 6a. The first two methods of sampling K suggested to us that our circuit was complete, as every incompleteness score computed with those methods was small. However, the third resulted in sets K that had high incompleteness score: up to 3.09 (87% of the original logit difference). These greedily-found sets were usually not semantically interpretable (containing heads from multiple categories), and investigating them would be an interesting direction of future work.

4.2   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 high *minimality score* $|F(C \setminus (K \cup \{v\})) - F(C \setminus K)|$.

In the running example, we can show that $C_1 \cup C_2$ is minimal. The proof has two symmetrical cases: $v \in C_1$ and $v \in C_2$. Without loss of generality, we chose $v \in C_1$, and then define $K = C_2$. The minimality score for this choice of node v and set K is equal to $|F(C_1 \setminus \{v\}) - F(C_1)|$, which is large since C_1 is a serial circuit and so removing v will destroy the behavior.

What happens in practice for our circuit? We need to exhibit for every v a set K such that the minimality score is high. For most heads, removing the class of heads G that v is a part of provides
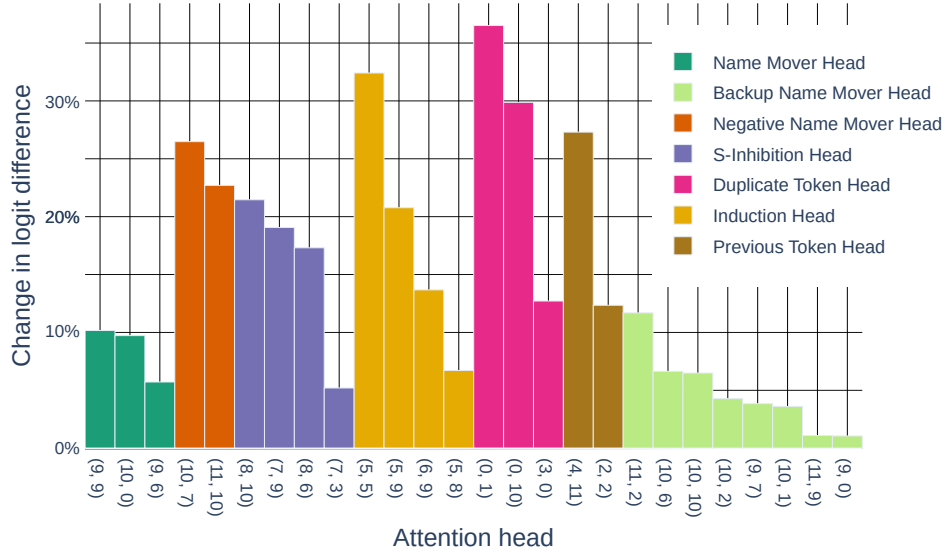
Figure 7: Plot of minimality scores $|F(C \setminus (K \cup \{v\})) - F(C \setminus K)|$ (as percentages of $F(M)$) for all components v in our circuit. The sets K used for each component, as well as the initial and final values of the logit difference for each of these v are in Appendix K.

a reasonable minimality score, but in some instances a more careful choice is needed; we provide full details in Appendix K and display final results in Figure 7. The importance of different nodes varies, but all have a nontrivial impact (at least 1% of the original logit difference). These results ensure that we did not interpret irrelevant nodes, but do show that the individual contribution of some attention heads is small.

4.3 COMPARISON WITH A BASELINE CIRCUIT

In the previous sections, we evaluated our circuit on three quantitative criteria. To establish a baseline for these criteria, we compare to a naïve circuit that consists of the Name Mover Heads (but no Backup nor Negative Name Mover Heads), S-Inhibition Heads, two Induction Heads, two Duplicate Token Heads, and the Previous Token Heads. This circuit has a faithfulness score of 0.1, comparable to the full circuit C. However, the naïve circuit can be more 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 6a). Nonetheless, when we applied the greedy heuristic to optimize for the incompleteness score, both circuits have similarly large incompleteness scores.

4.4 DESIGNING ADVERSARIAL EXAMPLES

As argued in Räuker et al. (2022), one way to evaluate the knowledge gained by interpretability work is to use it for downstream applications. In this section, we do this by using knowledge of the circuit to construct simple adversarial examples for the model.

As presented in Section 3, the model relies on duplicate detection to differentiate between S and IO. Motivated by this, we constructed passages where both the S and IO tokens are duplicated. An example is "John and Mary went to the store. Mary had a good day. John gave a bottle of milk to"; see Appendix L for full details. We find that this significantly reduces the logit difference and causes the model to predict S over IO 23% of the time (Figure 8).

To ensure that the observed effect is not an artifact of the additional sentences, we included a control dataset using the same templates, but where the middle sentence contains S instead of IO. In these sentences, S appears three times in total and IO only appears once. On this distribution, the model has an even higher logit difference than on p_{IOI}, and predicts S over IO only 0.4% of the time.

Limitations of the attack. Despite being inspired by our understanding of our circuit, those examples are simple enough that they could have been found without our circuit with enough effort.

11

Distribution	Logit difference	IO probability	Proportion of S logit greater than IO
p_{IOI}	3.55	0.49	0.7%
Additional occurrence of S (natural sentence)	3.64	0.59	0.4%
Additional occurrence of IO (natural sentence)	1.23	0.36	23.4%

Figure 8: Summary of GPT-2 performance metrics on the IOI task on different datasets. In the line order: for p_{IOI}, for the dataset where we added an occurrence of S (thus S appears three times in the sentence) and for the adversarial dataset with duplicated IO in natural sentences.

Moreover, we do not have a full understanding of the mechanisms at play in these adversarial examples. For instance, the S-Inhibition Heads attend not only to S2, but also to the second occurrence of IO. As this pattern is not present in p_{IOI} nor in p_{ABC}, it is beyond the analysis presented in Section 3. The study of the behavior of the circuit on these adversarial examples could be a promising area for future work.

5 DISCUSSION

In this work, we isolated, understood and validated a set of attention heads in GPT-2 small composed in a circuit that identifies indirect objects. Along the way, we discovered interesting structures emerging from the model internals. For instance:

- We identified an example of attention heads communicating by using pointers (sharing the location of a piece of information instead of copying it).
- We identified heads compensating for the loss of function of other heads, and heads contributing negatively to the next-token prediction. Early results suggest that the latter phenomenon occurs for other tasks beyond IOI (see Appendix H).

However, our work also has several limitations. First, despite the detailed analysis presented here, we do not understand several components. Those include the attention patterns of the S-Inhibition Heads, and the effect of MLPs and layer norms. Second, the number of parameters in GPT-2 small is orders of magnitude away from state-of-the-art transformer language models. A future challenge is to scale this approach to these larger models.

As a starting point, we performed a preliminary analysis of GPT-2 medium. We found that this model also has a sparse set of heads directly influencing the logits. However, not all of these heads attend to IO and S, suggesting more complex behavior than the Name Movers Heads in GPT-2 small. Continuing this investigation is an exciting line of future work.

More generally, we think that zooming in on a clearly defined task in a single model enables a rich description of phenomena that are likely to be found in broader circumstances. In the same way model organisms in biology enable discoveries valuable beyond a specific species, we think these highly detailed (though narrow) examples are crucial to developing a comprehensive understanding of machine learning models.

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.

Guendalina Caldarini, Sardar Jaf, and Kenneth McGarry. A literature survey of recent advances in chatbots. *Information*, 13(1):41, 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.

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.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

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.

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.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

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

Tilman Räuker, Anson Ho, Stephen Casper, and Dylan Hadfield-Menell. Toward transparent ai: A survey on interpreting the inner structures of deep neural networks, 2022. URL https://arxiv.org/abs/2207.13243.

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.

Angela Zhang, Lei Xing, James Zou, and Joseph C Wu. Shifting machine learning for healthcare from development to deployment and from models to data. *Nature Biomedical Engineering*, pp. 1–16, 2022.

6   APPENDIX

A   DISENTANGLING TOKEN AND POSITIONAL SIGNAL IN THE OUTPUT OF S-INHIBITION HEADS

In Section 3.2, we discovered that S-Inhibition Heads are responsible for the Name Mover Heads' specific attention on the IO token, and in particular we discovered that their output is a signal for Name Movers Heads to avoid the S1 token.

More precisely, we discover that they were outputting *token signals* (information about the value of the token S) and *positional signals* (related to the value of the position S1). We describe here the details of this discovery.

To disentangle the two effects, we design a series of counterfactual datasets where only some signals are present, and some are inverted with respect to the original dataset. In this section, we conducted activation patching (or simply patching). Instead of patching isolated paths, we patch the output of a set of heads and recompute the forward pass of the model after this modification. As shown in Section 3.4, the output of later (Backup) Name Mover Heads and Negative Heads depend on earlier heads from these classes, such that it is not possible to ignore the interactions between heads inside a class (as it is the case in path patching in Appendix B).

By patching the output of S-Inhibition heads from these datasets, we can quantify in isolation the impact of each signal on the final logit difference.

We constructed six datasets by combining three transformations of the original p_{IOI} distribution.

- **Random name flip**: we replace the names from a given sentence with random names, but we keep the same position for all names. Moreover, each occurrence of a name in the original sentence is replaced by the same random name. When we patch outputs of S-Inhibition heads from this

	Original positional signal	Inverted position signal
Original S token signal	3.55 (baseline)	-0.99
Random S token signal	2.45	-1.96
S↔IO inverted token signal	1.77	-3.16

Figure 9: Logit difference after patching S-Inhibition heads from signal-specific datasets. The effect on logit difference can be decomposed as a sum of the effects of position and token signal.
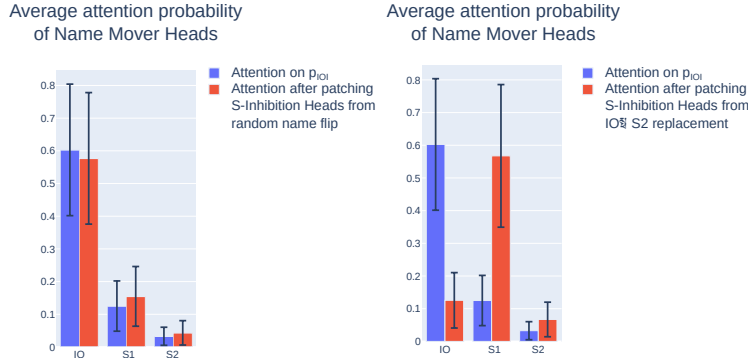

Figure 10: Name Mover Heads' attention probability before and after patching S-Inhibition Heads from signal-specific datasets. Left: patching from the dataset generated by random flip of name (same position signal, random token signal). Right: patching from the dataset generated by IO←S2 replacement (inverted position signal, inverted token signal). Black bars represent the standard deviation.

sentence, only positional signals are present, the token signals are unrelated to the names of the original sequence.

- **IO↔S1 flip**: we swap the position of IO and S1. The output of S-inhibition heads will contain correct token signals (the subject of the second clause is the same) but inverted positional signals (because the position of IO and S1 are swapped)
- **IO←S2 replacement**: we make IO become the subject of the sentence and S the indirect object. In this dataset, both token signals and positional signals are inverted.

We can also compose these transformations. For instance, we can create a dataset with no token signals and inverted positional signals by applying IO↔S1 flip on the dataset with random names. In total, we can create all six combinations of original, inverted, or uncorrelated token signal with the original and inverted positional signal.

From each of those six datasets, we patched the output of S-Inhibition heads and measured the logit difference. The results are presented in Figure 9.

These results can be summarized as the sum of the two effects. Suppose we define the variable S_{tok} to be 1 if the token signal is the original, 0 when uncorrelated and -1 when inverted. And similarly S_{pos} to be 1 if the position signal is the original and -1 if inverted. Then the Figure 9 suggests that the logit difference can be well approximated by $2.31 S_{pos} + 0.99 S_{tok}$, with a mean error of 7% relative to the baseline logit difference.

For instance, when both the positional and token signals are inverted, the logit difference is the opposite of the baseline. This means that the S token is predicted stronger than the IO token, as strong as IO before patching. In this situation, due to the contradictory information contained in the output of S-Inhibition heads, the Name Movers attend and copy the S1 token instead of the IO token (see Figure 10, right). In the intermediate cases where only one of the signals is modified, we observe a partial effect compared to the fully inverted case (e.g. Figure 10, left). The effect size depends on the altered signals: positional signals are more important than token signals.

Can we be more specific as to what the token and positional signals are? Unfortunately, we do not have a complete answer, but see this as one of the most interesting further directions of our work. We expect that the majority of the positional information is about the *relative* positional embedding
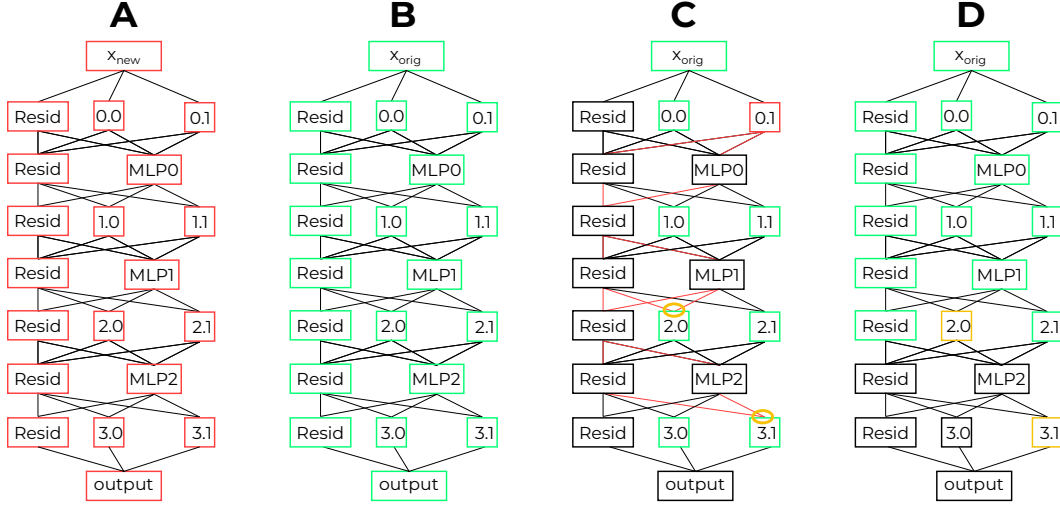
Figure 11: Illustration of the four forward passes in the path patching method. $h = 0.1$ and $R = \{2.0, 3.1\}$. The layer norms are not shown for clarity. In the forward pass C, we show in red all the paths through which h can influence the value of heads in R. Nodes in black are recomputed, nodes in color are patched or frozen to their corresponding values.

between S1 and S2 (such pointer arithmetic behavior has already been observed in Olsson et al. (2022)). When patching in S2 Inhibition outputs from a distribution where prefixes to sentences are longer (but the distance between S1 and S2 is constant), the logit difference doesn't change (3.56 before patching vs 3.57 after). This suggests that the positional signal doesn't depend on the absolute position of the tokens, as long as the relative position of S1 and S2 stays the same.

B PATH PATCHING

In this section we describe the technical details of the path patching technique we used in many experiments. This technique depends on

- x_{orig} the original data point,

- x_{new} the new data point,

- h the *sender* attention head, and

- $R \subseteq M$ the set of *receiver* nodes in the model's computational graph M. In our case, R is either the input (key, query or values) of a set of attention heads or the end state of the residual stream[3].

It is then implemented as Algorithm 1, which can be summarized in five steps.

1. Gather activations on x_{orig} and x_{new}. [Lines 1-2, Figure 11A-B]

2. Freeze[4] all the heads to their activation on x_{orig} except h that is patched to its activation on x_{new}. [Lines 4-5, Figure 11C]

3. Run a forward pass of the model on x_{orig} with the frozen and patched nodes (MLPs and layer norm are recomputed). [Line 7, Figure 11C]

4. In this forward pass, save the activation of the model components $r \in R$ as if they were recomputed (but this value is overwritten in the forward pass because they are frozen). [Line 6-9, Figure 11C]

---

[3]The model uses the end state of the residual stream to compute the logits (after a layer norm), so we choose R equal to this end state when we wish to measure the direct effect of h on the logits.

[4]We use 'freezing' of an activation a to refer to the special case of patching when we begin a forward pass on an input x, edit some earlier activations, and then want to patch a to its value on a forward pass of x (which did not have the edits of earlier activations).

16

5. Run a last forward pass on x_{orig} patching the receiver nodes in R to the saved values. [Lines 12-20, Figure 11D]

Each of the A variables in Algorithm 1 (A_{orig}, A_{new} and A_{patch}) stores activations for all nodes in M. A_{orig} and A_{new} contain output activations while A_{patch} contains the activation of the nodes in R (keys, queries or values of attention heads, or the end state of the residual stream).

In forward pass C of the algorithm all attention heads except h have their output frozen to their value on x_{orig}. This means that we remove the influence of every path including one (or more) intermediate attention heads p of the form $h \to p \to r$, where $r \in R$. In all cases in the paper, we measure the logit difference on the forward pass D compared to the logit difference of the model, as a measure of the importance of the path $h \to r$. We additionally compute this logit difference as an average over $N > 200$ pairs $(x_{\text{orig}}, x_{\text{new}})$.

Algorithm 1 Path patching

1: Compute all activations A_{new} on x_{new} (forward pass A)
2: Compute all activations A_{orig} on x_{orig} (forward pass B)
3: **for** $r \in R$ **do**
4:      $A^r_{\text{patch}} \leftarrow A_{\text{orig}}$
5:      $A^r_{\text{patch}}[h] \leftarrow A^r_{\text{new}}[h]$
6:      **for** each MLP m between h and r **do**
7:          Recompute $A^r_{\text{patch}}[m]$ from the activations in A^r_{patch} (forward pass C)
8:      **end for**
9:      Recompute $A^r_{\text{patch}}[r]$ from the activations in A^r_{patch} (forward pass C)
10: **end for**
11:
12: $A_{\text{final}} \leftarrow \emptyset$
13: **for** $c \in M$ nodes of the computational graph, in topologically sorted order, **do** (forward pass D)
14:      **if** $c \in R$ **then**
15:          $A_{\text{final}}[c] \leftarrow A^c_{\text{patch}}[c]$
16:      **else**
17:          Compute $A_{\text{final}}[c]$ from activations in A_{final}
18:      **end if**
19: **end for**
20: **return** $A_{\text{final}}[\text{Logits}]$

C DIRECT EFFECT ON S-INHIBITION HEADS' KEYS

In this section, we present the direct effect analysis of the S-Inhibition Heads' keys. The experiment is similar to the investigation of the S-Inhibition Heads' values presented in Section 3.2.

The results presented in Figure 12b show that some heads significantly influence the logit difference through S-Inhibition Heads' keys. We observe that Duplicate Token Heads (3.0 and 0.1) appear to also influence S-Inhibition Heads' values, but their effect is reversed.

Moreover, we identify 3 new heads influencing positively the logit difference: 5.9, 5.8 and 0.10.

Fuzzy Duplicate Token Head By looking at its attention pattern, we identified that 0.10 was paying attention to S1 from S2. However, the attention pattern was fuzzy, as intermediate tokens also have non-negligible attention probability. We thus call it a fuzzy Duplicate Token Head. On Open Web Text (OWT), the fuzzy Duplicate Token Head attend to duplicates in a short range (see Appendix I).

Fuzzy Induction Heads The heads 5.9, 5.8 are paying attention to S1+1. For 5.8, S1+1 is the token with the highest attention probability after the start token. But the absolute value is small (less than 0.1). The head 5.9 is paying attention to S1+1 but also to tokens before S. Because of these less interpretable attention patterns, we called them fuzzy Induction Heads. (see Appendix H for more details about their behavior outside IOI).

By influencing the keys of the S-Inhibition Heads, we hypothesize that those heads are amplifying the positional signal written by the other Induction and Duplicate Token Heads.
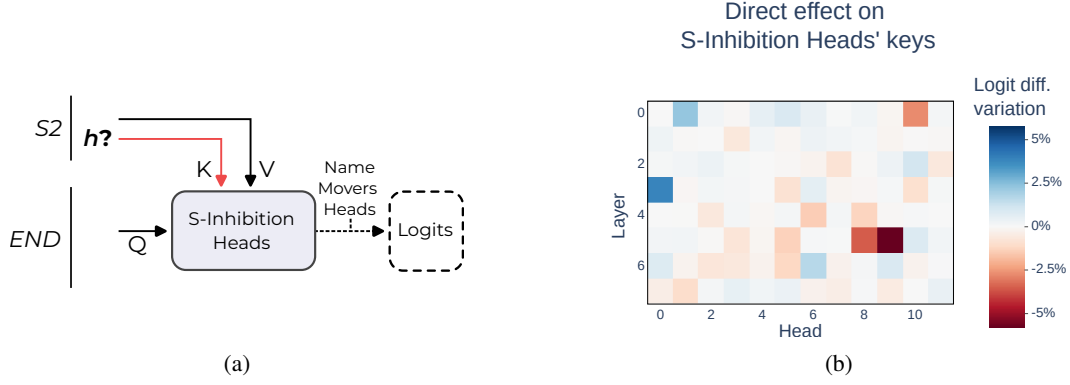
(a)

(b)

Figure 12: (a) Diagram of the direct effect experiment on S-Inhibition Heads' keys at S2. (b) Result of the path patching experiments for direct effect experiment on S-Inhibition Heads' keys.
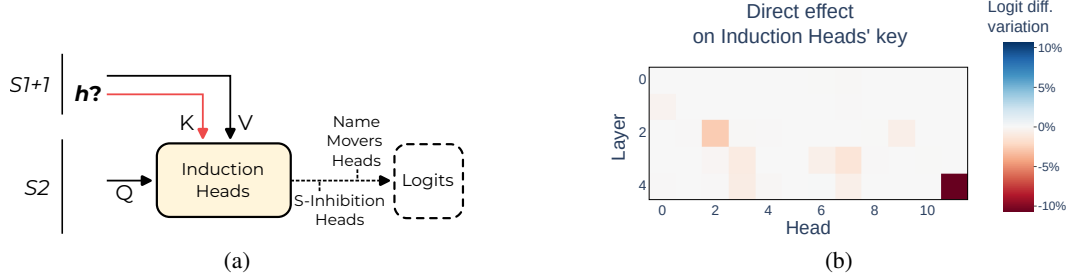

(a)

(b)

Figure 13: (a) Diagram of the direct effect experiment on Induction Heads' keys at S1+1. The effects on logits are mediated by S-Inhibition Heads and Name Mover Heads (b) Result of the path patching experiments on Induction Heads' keys at S1+1.

D   IDENTIFICATION OF PREVIOUS TOKEN HEADS

Induction Heads rely on key composition with Previous Token Heads to recognize patterns of the form [A] [B] ... [A]. In the context of IOI, the repeated token [A] is S2. We thus searched for heads directly affecting Induction Heads keys at the S1+1 position (the [B] token in the general pattern). For this, we used path patching. The results of the experiment are visible in Figure 13. We identify two main heads causing a decrease in logit difference (and thus contributing positively to the logit difference): 4.11 and 2.2. These heads pay primary attention to the previous token. This is coherent with the Previous Token Heads we were expecting. We thoroughly investigate their attention patterns outside of p_{IOI} in Appendix H.

Olsson et al. (2022) also describes an induction mechanism relying on query composition in GPT-2. We performed path patching to investigate heads influencing the Induction Heads queries at the S2 position, but did not find any significant effect.

E   IOI TEMPLATES

We list all the templates we used in Figure 14. 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 words. All the words chosen were one token long to ensure proper sequence alignment computation of the mean activations.

Templates in p$_{\text{IOI}}$
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 14: Templates used in the IOI dataset. All templates in the table fit the 'BABA' pattern, but we use templates that fit the 'ABBA' pattern as well (i.e, by swapping the first instances of [B] and [A] in all of the above).

F   BACKUP NAME MOVER HEADS

Here we discuss in more detail the discovery of the Backup Name Mover Heads. As shown in Figure 15, knocking-out the three main Name Mover Heads surprisingly changes the behavior of the other heads that write in the IO-S direction (both positively and negatively). These heads compensate for the loss of function from the Name Mover Heads such that the logit difference is only 5% lower. We observe that the Negative Name Mover Heads have a less negative effect on logit difference, and 10.7 even has a positive effect on the logit difference after the knockout. The other heads that affected slightly positively the logit difference before the knock-out become the main contributors. 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 the heads influencing positively the logit difference after knockout, we identified S-Inhibition Heads and a set of other heads that we called *Backup Name Mover Heads*. We arbitrarily chose to keep the eight heads that were not part of any other groups, and affected the logit difference with an effect size above the threshold of 2%.

In Figure 16 we analyze the behavior of those newly identified heads with similar techniques as Name Mover Heads. Those can be grouped in 4 categories.

- Four heads (9.0, 10.1, 10.10 and 10.6) that behave similarly to Name Mover Heads in their attention patterns, and the scatter plots of attention vs. dot product of their output with $W_U[IO] - W_U[S]$ (e.g 10.10 in Figure 16).

- Two heads (10.2, 11.9) that pay equal attention to S1 and IO and write both of them (e.g 10.2 in Figure 16).

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

- One head, 9.7, pays attention to S2 and writes 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 7). The exact choice of Backup Name Mover Heads doesn't change significantly the behavior of the circuit.
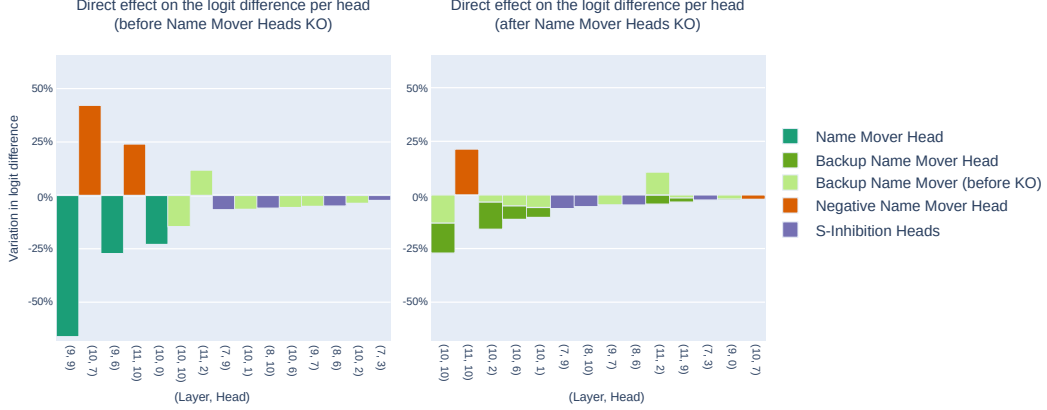
Figure 15: Discovery of the Backup Name Mover Heads. Left: results from the path patching experiment in Figure 3b. Right: the same path patching experiment results except computed after knocking out the Name Mover Heads. In both plots, the heads are ordered by decreasing order of the absolute value of their effect.

G   GPT-2 SMALL FULL ARCHITECTURE

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 parameters

- N: number of input tokens.
- V: length of 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 norm, the non-linear function

$$\text{LN}(x) \stackrel{\text{def}}{=} \frac{x - \bar{x}}{\sqrt{\sum_i (x_i - \bar{x}_i)^2}}, \tag{1}$$

where the mean and the difference from the mean sum are over all components of the dimension d vector in each sequence position. This is then followed by a learned linear transformation M (different for each layer norm).

In GPT-2 the MLPs all have one hidden layer of dimension D and use the GeLU non-linearity. Their input is the layer normed state of the residual stream.

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 $A_{i,j} = \text{softmax}(x^T W_{QK}^{i,j} x)$ where the softmax is taken for each token position, and is unidirectional. We then have $h_{i,j}(x) \stackrel{\text{def}}{=} M \circ \text{LN}((A_{i,j} \otimes W_{OV}^{i,j}).x)$.

Algorithm 2 describe how these elements are combined in the forward pass of GPT-2 small.

H   VALIDATION OF THE INDUCTION MECHANISM 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 two scores from the attention patterns of the attention heads:
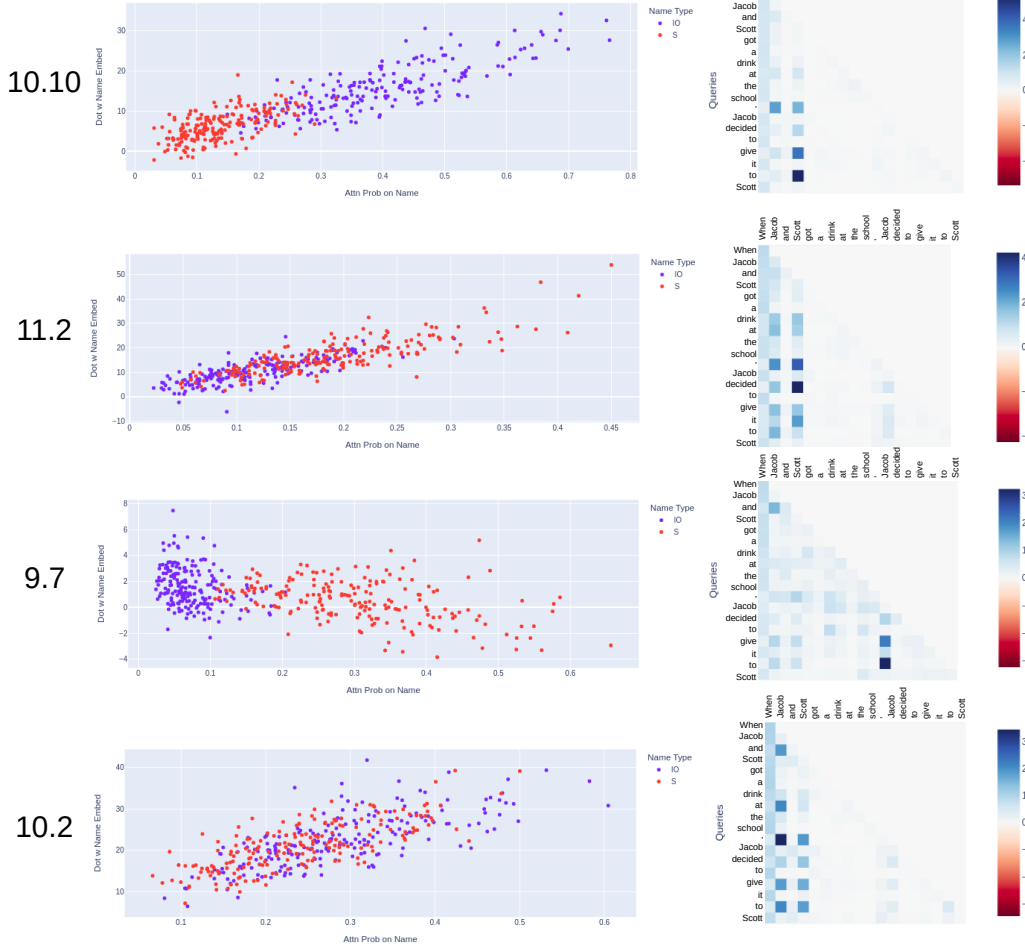
Figure 16: Four examples of Backup Name Mover Heads. Left: attention probability vs projection of the head output along $W_U[IO]$ or $W_U[S]$ respectively. Right: Attention pattern on a sample sequence.

Algorithm 2 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:      $y_i' \leftarrow m_i(y_i)$, 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**
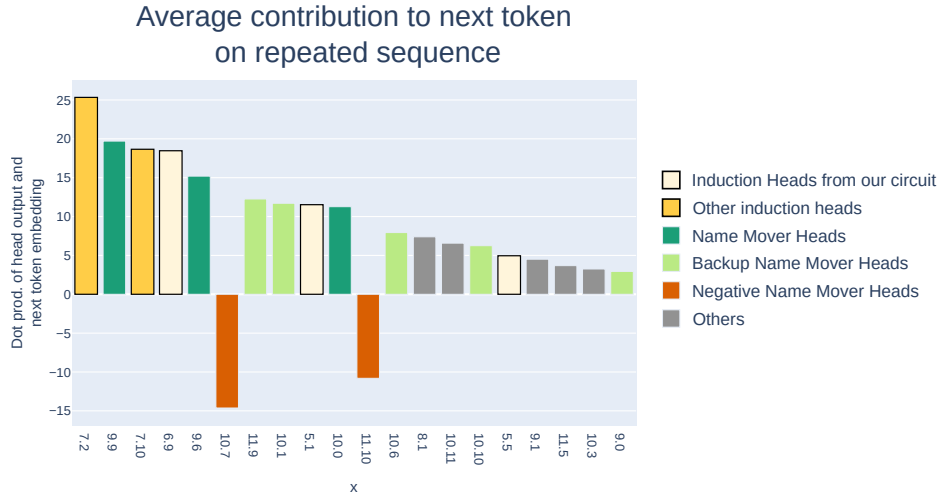11: **return** $W_U \circ M \circ \text{LN} \circ x_L$

Figure 17: Contribution to the next token prediction per head on repeated sequences of tokens. The heads are ordered by decreasing absolute values of contribution. Black contour: heads with attention patterns demonstrating prefix matching property.

- The *previous token score*: we averaged the attention probability on the off-diagonal. This is the average attention from the token at position i to position $i - 1$.

- The *induction score*: the average attention probability from T_i to the token that comes after the first occurrence of T_i (i.e. T_{i-99})

These two scores are depicted in Figure 18 (center and right) for all attention heads.

Previous Token Heads. 4.11 and 2.2 are the two heads with the highest previous token score on sequences of random tokens. This is a strong validation of their role outside p_{IOI}.

Induction Heads. Olsson et al. (2022) define an Induction Head according to its behavior on repeated sequences of random tokens. The attention head must demonstrate two properties. i) Prefix-matching property. The head attends to `[B]` from the last `[A]` on pattern like `[A] [B] ... [A]` ii) Copy property. The head contribute positively to the logit of `[B]` on the pattern `[A][B]...[A]`.

5.5 and 6.9 are among the 5 heads with the highest induction score. This validates their prefix-matching property introduced in Olsson et al. (2022).

To check their copy property, we computed the dot product $\langle h_i(X), W_U[B] \rangle$ between the output of the head h_i on sequence X and the embedding of the token `[B]` on repeated sequences of random tokens. The results are shown in Figure 17. The two Induction Heads (5.5 and 6.9) appear in the 20 heads contributing the most to the next token prediction. Thus validating their copying property.

We also noticed that the majority of the Negative, Backup and regular Name Mover Heads appear to write in the next token direction on repeated sequences of random tokens, and Negative Name Movers Heads contribute negatively. This suggests that these heads are involved beyond the IOI task to produce next-token prediction relying on contextual information.

I   VALIDATION OF DUPLICATE TOKEN HEADS

On the repeated sequences of random tokens (Appendix H) we also computed the *duplicate score*. For each token T_i in the second half of a sequence, we average the attention probability from T_i to its previous occurrence in the first half of the sequence (i.e. T_{i-100}).

The duplicate token scores for all attention heads are depicted in Figure 18. 3.0 and 0.1 are among the three heads with the highest duplicate token score (Figure 18). This is evidence of their role of Duplicate Token Heads outside the circuit for the IOI task.
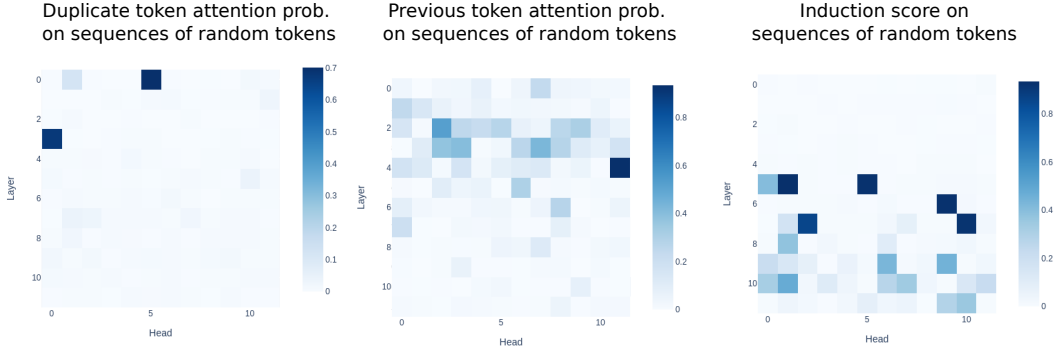
Figure 18: Attention scores on sequences of repeated random tokens. Left: Duplicate score, the average attention probability from a token to its previous occurrence. Center: Previous token attention score, it is the average of the off-diagonal attention probability. Right: Induction score. Average attention probability from the second occurrence of `[A]` to `[B]` on patterns `[A][B]...[A]`.

However, the fuzzy Duplicate Head 0.10 doesn't appear on this test. By qualitatively investigating its attention patterns on Open Web Text, we found that this head attends strongly to the current token. Moreover, when the current token is a name, and it is duplicated, the head attends to its previous occurrence.

J   ROLE OF MLPs IN THE TASK

In the main text, all of the circuit components are attention heads. Attention heads are the only modules in transformers that move information across token positions – a crucial component of the IOI task – so they were our main subject of interest. However, MLPs can still play a significant role in transforming the information in each residual stream. We explored this possibility by measuring the direct and indirect effects of each of the MLPs in Figure 19. In these experiments, for each MLP in turn, we did a path patching experiment (Section 3.2) to measure the direct effect and a knock-out experiment for the indirect effect.

We observe that MLP0 has a significant influence on logit difference after knock-out (it reverses the sign of the logit difference) but the other layers don't seem to play a big role when individually knocked out. When all MLP layers other than the first layer are knocked out, however, the logit difference becomes -1.1 (a similar effect to the knockout of MLP0 alone).
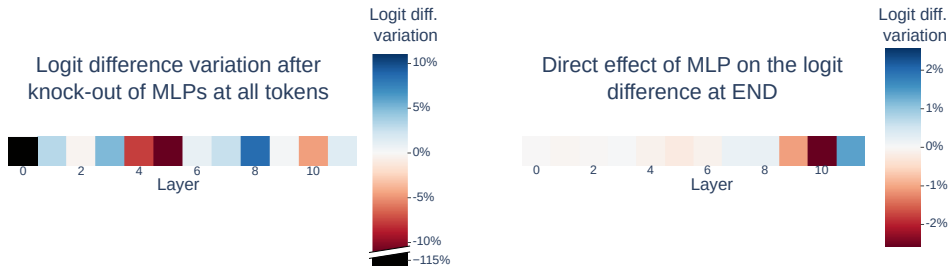

Figure 19: Left: change in logit difference from knocking out each MLP layer. Right: change in logit difference after a path patching experiment investigating the direct effect of MLPs on logits.

K   MINIMALITY SETS

The sets that were found for the minimality tests are listed in Figure 20.

v	Class	$K \cup \{v\}$	$F(C \setminus (K \cup \{v\}))$	$F(C \setminus K)$
(9, 9)	Name Mover	[(9, 9)]	2.26	2.62
(10, 0)	Name Mover	[(9, 9), (10, 0)]	1.91	2.26
(9, 6)	Name Mover	[(9, 9), (10, 0), (9, 6)]	2.11	1.91
(10, 7)	Negative	[(11, 10), (10, 7)]	4.07	3.13
(11, 10)	Negative	[(11, 10), (10, 7)]	4.07	3.27
(8, 10)	S Inhibition	[(7, 9), (8, 10), (8, 6), (7, 3)]	0.24	1.01
(7, 9)	S Inhibition	[(7, 9), (8, 10), (8, 6), (7, 3)]	0.24	0.92
(8, 6)	S Inhibition	[(7, 9), (8, 10), (8, 6), (7, 3)]	0.24	0.86
(7, 3)	S Inhibition	[(7, 9), (8, 10), (8, 6), (7, 3)]	0.24	0.43
(5, 5)	Induction	[(5, 9), (5, 5), (6, 9), (5, 8)]	0.97	2.12
(5, 9)	Induction	[(11, 10), (10, 7), (5, 9)]	3.33	4.07
(6, 9)	Induction	[(5, 9), (5, 5), (6, 9), (5, 8)]	0.97	1.46
(5, 8)	Induction	[(11, 10), (10, 7), (5, 8)]	3.83	4.07
(0, 1)	Duplicate Token	[(0, 1), (0, 10), (3, 0)]	0.60	1.90
(0, 10)	Duplicate Token	[(0, 1), (0, 10), (3, 0)]	0.60	1.66
(3, 0)	Duplicate Token	[(0, 1), (0, 10), (3, 0)]	0.60	1.05
(4, 11)	Previous Token	[(4, 11), (2, 2)]	1.31	2.28
(2, 2)	Previous Token	[(4, 11), (2, 2)]	1.31	1.75
(11, 2)	Backup Name Mover	All previous NMs and backup NMs	0.95	1.37
(10, 6)	Backup Name Mover	All previous NMs and backup NMs	1.65	1.88
(10, 10)	Backup Name Mover	All previous NMs and backup NMs	1.88	2.11
(10, 2)	Backup Name Mover	All previous NMs and backup NMs	1.49	1.65
(9, 7)	Backup Name Mover	All previous NMs and backup NMs	0.81	0.95
(10, 1)	Backup Name Mover	All previous NMs and backup NMs	1.37	1.49
(11, 9)	Backup Name Mover	All name movers and negative heads	0.41	0.45
(9, 0)	Backup Name Mover	All name movers and negative heads	0.41	0.45

Figure 20: K sets for minimality for each v.

K found by greedy optimization
(9, 9), (9, 6), (5, 8), (5, 5), (2, 2)
(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 22: 4 sets K found by the greedy optimization procedure on our circuit.

L   TEMPLATE FOR ADVERSARIAL EXAMPLES

The design of adversarial examples relies on adding a duplicate IO to the sentences. To this end, we used a modification of the templates described in appendix E. We added an occurrence of [A] in the form of a natural sentence, independent of the context. The list of sentence is visible in Figure 21.

[A] had a good day.
[A] was enjoying the situation.
[A] was tired.
[A] enjoyed being with a friend.
[A] was an enthusiast person.

Figure 21: Templates for the natural sentences used in the generation of adversarial examples. The sentences were chosen to be independent of the context.

M   GREEDY ALGORITHM

The Algorithm 3 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 incompleteness score (including the intermediate K).

Algorithm 3 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 $V \subseteq C \setminus K$ of k nodes uniformly.
4:     $v_{\text{MAX}} \leftarrow \arg\max_{v \in V} |F(C \setminus (K \cup \{v\})) - F(C \setminus K)|$
5:     $K \leftarrow K \cup \{v_{\text{MAX}}\}$
6: **end for**
7: **return** K

As visible in Figure 22 the sets found by the greedy search contain a combination of nodes from different classes. This means that it is difficult to interpret how our circuit is incomplete, as mentioned in the main text.