Adversarial Circuit Evaluation

Niels uit de Bos [1] **Adrià Garriga-Alonso** [2]

Abstract

Circuits are supposed to accurately describe how a neural network performs a specific task, but do they really? We evaluate three circuits found in the literature (IOI, greater-than, and docstring) in an adversarial manner, considering inputs where the circuit's behavior maximally diverges from the full model. Concretely, we measure the KL divergence between the full model's output and the circuit's output, calculated through resample ablation, and we analyze the worst-performing inputs. Our results show that the circuits for the IOI and docstring tasks fail to behave similarly to the full model even on completely benign inputs from the original task, indicating that more robust circuits are needed for safety-critical applications.

1. Introduction

Neural networks' vast size and complexity make them difficult to reverse engineer. To address this issue, circuits have been proposed (Olah et al., 2020) as one paradigm. By isolating the subset of components of a neural network that perform a chosen, narrow task, we hope to obtain a subnetwork that is smaller and disentangled from all other tasks the full network performs, making it easier to understand. We call this subset of components the *circuit*.

For the circuit to be helpful in understanding the original, full model, it is crucial that the circuit's behavior coincides with the full model's behavior on the chosen task. In particular, on task-specific inputs, the circuit should produce the same output distribution as the full model. Previous work has mostly assessed a circuit's performance by testing its ability to output the same distribution as the full model *on average* on the task-specific inputs.

In this paper, we argue that, besides looking at the average performance, it is worthwhile to assess a circuit by analyzing its *worst-case* performance: on which inputs and how large a proportion of inputs does the circuit fail to emulate the full model's behavior? We propose a method to evaluate circuits from this adversarial perspective and apply it to several circuits found in the literature: Indirect Object Identification (IOI) (Wang et al., 2022), greater-than (Hanna et al., 2023), and docstring (Heimersheim & Janiak, 2023).

Adversarial circuit evaluation is important for several reasons. First, we cannot say we to truly understand the full model if the circuit behaves differently on a certain fraction of the inputs. For example, our analysis (Section 3) showsthat the circuit for the IOI task fails to emulate the full model's behavior on a significant fraction of inputs. We speculate that especially when romantic objects are involved, components outside of the circuit play a crucial role. Since romantic objects could be only a small fraction of the inputs tested, the circuit's average performance can be high, even though we may be missing a crucial piece of the puzzle.

In particular, if we ever want to use circuits for guarantees or in safety-critical applications, it is crucial to describe the neural network's behavior on all inputs. As a general principle, evolutionary pressures and adversarial attacks can successfully discover and exploit edge cases. For example, if we edit a reward model to align more with human values, the policy optimizing for it may find a regime where our edits fail. In the absence of specific circuit-based safety interventions, this remains speculation. However, the benign-seeming yet adversarial examples we find in this paper might convince the reader that any safety measure could only be built with more robust circuits.

Moreover, we argue that the adversarial metrics are not only useful for evaluating circuits but also for improving them. Our analysis of the worst-performing inputs for the circuits (Tables 4 to 6) shows failure modes that a researcher could try to inspect and address manually. Alternatively, our adversarial evaluation metrics could be plugged in to automatic circuit discovery techniques, likely leading to more robust circuits.

Our main contributions are the following:

- We provide a method to calculate the proposed adversarial metrics (Section 2). [1]

[1] Independent. Work done at MATS [2] FAR AI, San Diego, CA, United States. Correspondence to: Niels uit de Bos <niels@uit-de-bos.nl>.

ICML 2024 Mechanistic Interpretability Workshop

[1] Our code is available on GitHub at `https://github.com/Nielius/AdversarialCircuitEvaluation`.

- We prove a formula to calculate how many task data points are needed to bound the circuit's worst-case performance with high probability (Section 2.1); in more technical terms, we calculate the sample size required to find high-probability upper bounds for percentiles (e.g., the 99th percentile) of the KL divergence between the circuit's output and the full model's output on a distribution of task data points.

- We identify subtasks of IOI and Docstring where the circuit especially fails to explain (Section 3; in particular, Table 4 and Table 5). For IOI, the circuit fails most strongly on inputs featuring a romantic object ("kiss" or "necklace"); for Docstring, patch inputs that have `file` as one of their parameters disrupt the circuit's performance, causing it to predict `file` as the next parameter, even though it does not appear in the clean input.

- In contrast, we find the Greater-Than circuit is more robust than the other two and does not have exhibit any significant edge case failures (Section 3 and Table 6).

2. Methodology

The core component of our adversarial circuit evaluation method is the calculation of the KL divergence between the circuit's output and the full model's output for a large sample of input points. The adversarial metrics we are interested in extracting are the maximum KL divergence and several high percentile values from the resulting distribution of KL divergences. In this section, we describe the technical details of this calculation.

Resample ablation Let $\mathcal{D}$ denote the set of inputs for the task. To calculate the output for the circuit, we ablate all components not in the circuit. While any kind of ablation could be used, we use *resample ablation* in our experiments. That means that in addition to the dataset $\mathcal{D}$ of inputs for the task, we need a dataset $\widetilde{\mathcal{D}}$ of *corrupted inputs* (sometimes called *patch inputs*). On a forward pass with ablations, we replace the output of an ablated component with its output from the (unablated) forward pass on a corrupted input. Resample ablation, also sometimes known as *patching with different activations* (e.g. (Conmy et al., 2023)), is a kind of *interchange intervention* (Geiger et al., 2023); more background and justification can also be found in (Chan et al., 2022) and (Zhang & Nanda, 2024). The circuits analyzed in this paper were identified through resample ablation.

For the reader's benefit, we explain in more detail how resample ablation works in the context of the circuits analyzed in this paper. The neural networks we are dealing with are transformers, and the circuits are subsets of edges between nodes that represent MLPs, transformer heads (each individually), the embedded input, and the output before the unembedding. The transformer heads have three different inputs: the key, the query, and the value. Because of the residual stream's additivity, each node can causally impact all downstream nodes, so there is an edge from each node to all downstream nodes. If an edge from a node X to a node Y is not part of the circuit C and we want to calculate the output $C(x, \tilde{x})$ of the circuit C on a clean input x with a corrupted input $\tilde{x}$, then X's contribution $X(x)$ to the input of Y in a forward pass on x is replaced with X's contribution $X(\tilde{x})$ to the input of Y from a forward pass on $\tilde{x}$. In our implementation, we achieve this by subtracting $X(x)$ from the input to Y, and adding $X(\tilde{x})$; this works because of the residual stream's additivity. In this way, intuitively speaking, a component can output clean outputs to some of its dependent downstream components, while simultaneously outputting corrupted outputs to other components.

Evaluation metrics For $x \in \mathcal{D}$, we denote by $M(x)$ the output of the full model M on x. For $x \in \mathcal{D}, \tilde{x} \in \widetilde{\mathcal{D}}$, we denote by $C(x, \tilde{x})$ the output of a forward pass of M where we have resample-ablated all components outside of the circuit C using the corrupted input $\tilde{x}$. The outputs $C(x, \tilde{x})$ and $M(x)$ are categorical probability distributions over the model vocabulary. We use the KL divergence $\mathrm{D}_{\mathrm{KL}}\left(M(x) \parallel C(x, \tilde{x})\right)$ to measure how close the circuit's output is to the model's output. To obtain our adversarial evaluation metrics, we sample a sufficiently large number of points from the distribution

$$\begin{aligned} \mathrm{D}_{\mathrm{KL}}\left(M(x) \parallel C(x, \tilde{x})\right) \text{ for } & x \sim \mathrm{Unif}(\mathcal{D}), \\ & \tilde{x} \sim \mathrm{Unif}(\widetilde{\mathcal{D}}). \end{aligned} \tag{1}$$

We then take the maximum or high percentiles of this distribution of KL divergences as our adversarial evaluation metric.

2.1. How many samples are needed?

The method described above samples from a distribution and then takes a percentile from that sample. How close is the sample percentile to the true percentile, and how many points do you need to sample to get a good estimate?

The following result, which we prove in Appendix C, provides an answer. Let X be a real-valued probability distribution and let $0 < p < 1$. Denote by x_p the true p-th percentile of X. Because we are looking at worst-case scenarios, we would like a tight upper bound $\hat{x}_p$ for x_p. Take any $\epsilon > 0$ with $p + \epsilon < 1$ and write $\hat{x}_p$ for the $\lceil (p + \epsilon) \cdot n \rceil$-th order statistic of n i.i.d samples from X, i.e., the $\lceil (p + \epsilon) \cdot n \rceil$-th smallest value of the n i.i.d. samples. We then have the following result.

Proposition 2.1. *The probability $\Pr(\hat{x}_p \geq x_p)$ that $\hat{x}_p$ is an upper bound for the true p-th percentile x_p of X can be* *calculated as*

$$\Pr\left(\hat{x}_p \geq x_p\right) = F_{\text{Binom}}(\lceil (p+\epsilon)\cdot n \rceil - 1; n, p) \qquad (2)$$

where $F_{\text{Binom}}(x; n, p)$ is the cumulative distribution function of the binomial distribution with parameters n and p.

Using this result in combination with either the Chernoff bound or Hoeffding's inequality, we can derive the following two bounds that show the asymptotic behavior:

Corollary 2.2. *We have*

$$\begin{aligned}&\Pr\left(\hat{x}_p \geq x_p\right) \geq \\ &\qquad 1 - \exp\left(-n\, \mathrm{D}_{\mathrm{KL}}\left(\mathrm{Bern}(p+\epsilon) \parallel \mathrm{Bern}(p)\right)\right) \qquad (3)\end{aligned}$$

where $\mathrm{D}_{\mathrm{KL}}\left(\mathrm{Bern}(p+\epsilon) \parallel \mathrm{Bern}(p)\right)$ is the KL divergence between the Bernoulli distribution with parameter $p+\epsilon$ and the Bernoulli distribution with parameter p. A simpler, but less tight lower bound is given by

$$\Pr\left(\hat{x}_p \geq x_p\right) \geq 1 - \exp\left(-2n\epsilon^2\right). \qquad (4)$$

We prove both results in Appendix C.

We want to apply these results in the following situation. We fix $0 < p, \delta < 1$, $\epsilon > 0$ with $p + \epsilon < 1$ and want to know how many points we need to sample such that we get

$$\Pr(\hat{x}_p \geq x_p) \geq \delta. \qquad (5)$$

By setting the right hand sides of the equations above equal to δ and solving for n, we obtain the values of n shown in Table 1.

In the results in Appendix B, we sample a million pairs of input and patch inputs independently. Applying the results from this section, if we take $\epsilon = 5 \times 10^{-4}$, then for the 95th-percentile, $\delta = 0.9891$, whereas for the 99-th, 99.9-th and 99.99-th percentile, δ is indistinguishably close to 1 in the `stats.scipy` package. For our main results (described in Section 3), however, we independently sample a thousand inputs and a thousand patch inputs, for a total of a one million pairs. However, these million pairs are not independent, so the results from this section do not apply directly. We can instead consider them as 1000 i.i.d. samples, even though that underestimates the true results, and then some reasonable numbers to consider would be the following: for $\epsilon = 0.01$ and $p = 0.95$, we find $\delta = 0.9194$; for $\epsilon = 5 \times 10^{-3}$ and $p = 0.99$, we find $\delta = 0.9339$.

3. Results

We applied the method described in Section 2 to three circuits found previously in the literature: the circuits for the IOI task (Wang et al., 2022), for the docstring task (Heimersheim & Janiak, 2023), and for the greater-than task (Hanna et al., 2023). In each of these tasks, the model needs to complete a sentence created from a task-specific template. For example, in the IOI task, the template is of the form "Afterwards, `[name1]` and `[name2]` went to the `[place]`. `[name2]` gave a `[object]` to ....", and the model's task is to complete the sentence with the token for `[name1]`. These task-specific templates are described in Table 2.

Table 1. A table showing how many samples are needed such that $\hat{x}_p$ is an upper bound of the p-th percentile of a real-valued distribution with probability at least δ; i.e., such that we have $\Pr(\hat{x}_p \geq x_p) \geq \delta$ (equation (5)). The column labeled "exact" uses the exact calculation from equation (equation (2)); the column "Chernoff" uses the formula based on the Chernoff bound (equation (3)); and the column "Hoeffding" uses the formula based on Hoeffding's inequality (equation (4)). The "exact" column will always provide a more precise value; the other columns are included to give the reader a sense of the approximation quality.

p	δ	ϵ	exact	Chernoff	Hoeffding
0.95	0.95	0.01	1282	2659	14 979
0.95	0.99	0.01	2437	4088	23 026
0.95	0.95	0.04	59	122	937
0.99	0.95	0.005	1049	1937	59 915
0.99	0.99	0.005	1736	2978	92 104
0.999	0.999	0.0005	31 236	44 987	13 815 511

First, to determine the circuit's explanatory power over a range of inputs, we sampled 1000 clean inputs and 1000 corrupted inputs, for a total of 1 million pairs for each task. We used the same data distributions as in (Conmy et al., 2023); these distributions sample each template value from a pre-determined list with equal probability. A crucial difference, though, is that we mixed all clean inputs with all corrupted inputs, whereas the original datasets paired them up in more restrictive ways. For example, in IOI, we allowed the corrupted input point to involve a different object and place, whereas originally the clean and corrupted inputs coincided on everything but the names. In Appendix B, we argue for this approach over only using the corrupted inputs that were used in the original dataset, but for completeness, that section also contains our evaluation of the circuit using only the corrupted inputs from the original dataset.

We then calculated the KL divergence between the model's output and the circuit's output for all those pairs and plotted the results as histograms in Figures 1 to 3. Summary statistics for these distributions are displayed in Table 3. The high percentiles and the max KL divergence shown in that table can then be considered as the adversarial evaluation metrics.

Secondly, to get a better understanding of the worst-case behavior of these circuits, we took the top 10 worst-performing (input, corrupted input) pairs for each circuit, and performed a forward-pass on the circuit and the model to obtain the top three most likely outputs for both the model and the circuit. These results are displayed in Appendix A, Tables 4 to 6. We discuss some of their implications below.

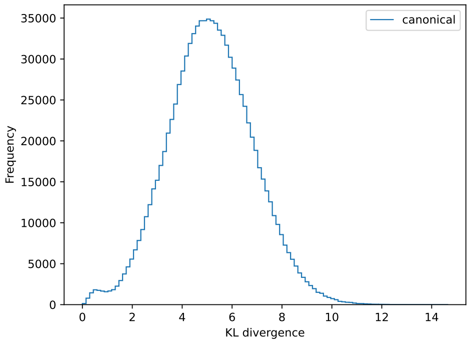

Figure 1. A histogram of the KL divergence for the IOI task. The x-axis shows the KL divergence between the model's output and the circuit's output on an input-corrupted-input pair, and the y-axis shows the number of input-corrupted-input pairs from our random sample of 1 million points that fall into each bin. There are 100 bins of equal size between the values of 0 and the maximum KL divergence achieved. Summary statistics of the plotted distribution are displayed in Table 3.

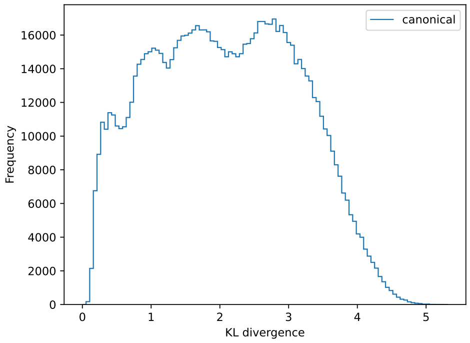

Figure 2. A histogram of the KL divergence for the greater-than task.

Comparing the KL divergence distributions The table of summary statistics (Table 3) for the distribution of KL divergences for the three tasks, computed on our random sample of 1 million input-corrupted-input pairs, shows that each circuit's worst-case performance is quite far from its mean performance. For the IOI and docstring tasks, the standard deviation is quite large, the worst points we found are more than 5 standard deviations away from the mean, and the z-scores indicate that the distributions have slightly thicker tails than the normal distribution. All of this indicates that it is worthwhile to pay attention to the tails of the distribution when evaluating the circuit's performance.

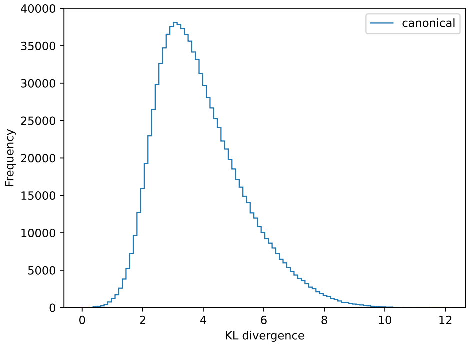

Figure 3. A histogram of the KL divergence for the docstring task.

Docstring A notable feature of the 10 worst-performing input pairs for the docstring task is that 7 out of the 10 have the same corrupted input (`def image(self, key, file, ...)`), which heavily skews the output logits towards the parameters from that corrupted input (notably the `file` parameter). This indicates that there are some components outside of the circuit that play a strong role in this task and perhaps only activate on certain inputs.

Greater-than The greater-than circuit is the best-performing circuit of the three: the worst-performing input pairs have a much lower KL divergence, and their KL divergence does not deviate as much from the mean as the other two tasks (see Table 3). The output analysis in Table 6 shows that even in the 10 worst cases, the circuit's most likely output always coincides with the model's most likely output, and the top three most likely outputs are almost always admissible.

Moreover, unlike in the other two tasks, there is a straightforward explanation that could have been predicted in advance: the worst-performing points are those where the clean input has a very high two-digit number (e.g., 94), so that there are very few allowed completions (only the two-digit numbers > 94), whereas the corrupted input has a very low two-digit number (e.g. 01), allowing almost all two-digit numbers as completion. If we assume that the full model's output distributions are approximately uniform over all allowable two-digits completions, for example, then the KL divergence between the clean input's output and the corrupted input's output is maximal, which plausibly explains why these inputs are the worst-performing inputs. We provide more evidence for this claim in D.

Table 2. A table summarizing the tasks for the circuits we analyze. Note that in each case, the template values in the corrupted input are completely independent from the template values in the clean input.

Task		Input template	Expected output	LLM	Notes
Docstring	clean:	Python function definitions with a docstring that starts describing the function's parameters, but crucially does not list all parameters.	The name of the next undescribed parameter in the docstring.	`attn-only-4l`[7]	
	corrupted:	Similar to clean, but the parameters in the docstring are not necessarily the same as in the function definition.			
Greater-than	clean:	"The `[noun]` lasted from the year `[year` $d_1d_2d_3d_4$`]` to `[`d_1d_2`]`"	any 2-digit number higher than d_3d_4	gpt2-small	
	corrupted:	Similar to clean, but the last two digits of the year are always `01`.			
IOI	clean:	"Afterwards, `[name1]` and `[name2]` went to the `[place]`. `[name2]` gave a `[object]` to "	`[name1]`	gpt2-small	This template can easily be varied, e.g. by switching the order of the names or changing some of the non-templated words, such as replacing "went to" with "decided to go to".
	corrupted:	"Afterwards, `[name1']` and `[name2']` went to the `[place']`. `[name3']` gave a `[object']` to "			

Table 3. Summary statistics from the KL divergence distributions plotted in Figures 1 to 3. The columns labelled "abs" show the absolute values of the KL divergence, whereas the columns labelled "z-score" show the difference between the percentile and the mean expressed as a multiple of the standard deviation.

	docstring		greaterthan		ioi	
	abs	z-score	abs	z-score	abs	z-score
count	1000000.00		1000000.00		1000000.00	
mean	3.91		2.09		5.15	
std	1.45		1.04		1.70	
min	0.10	-2.63	0.08	-1.92	0.03	-3.01
25%	2.85	-0.73	1.23	-0.82	4.01	-0.67
50%	3.66	-0.17	2.07	-0.01	5.12	-0.02
75%	4.75	0.58	2.91	0.79	6.27	0.66
95%	6.66	1.90	3.77	1.61	7.99	1.67
99%	8.03	2.85	4.23	2.05	9.25	2.41
99.9%	9.46	3.84	4.63	2.44	10.81	3.33
99.99%	10.58	4.61	4.91	2.71	12.25	4.17
max	12.07	5.64	5.31	3.09	14.64	5.57

IOI One striking feature of the worst-performing input pairs for the IOI task is that they often seem to involve romantic items. We provide more evidence for this observation in D. This behavior was even more apparent in earlier iterations of our experiment where we fixed the corrupted input. A plausible hypothesis is that parts of the model outside of the circuit are dormant in normal contexts but activate when romantic items are involved.

It is also worth noting that the IOI dataset from (Conmy et al., 2023) that we used only has eight possible values [8] for the object being given. It seems plausible that the circuit could behave very poorly on other objects as well.

4. Discussion

We have found that the IOI and Docstring circuits can produce very different outputs than the full model, even on inputs from the original task. In both cases, the worst-case performance is quite far from the mean performance. This casts doubt on the possibility of using these circuits to understand the full model's behavior. We expect this discrepancy to be even worse on untested input data or under minor distributional shifts: what happens when Mary *has secret plans* to give *an atomic bomb*?

Some of the badly performing inputs seem to follow a pattern, e.g., IOI's failure in romantic contexts and Docstring's tendency to pick up on the `file` parameter in the corrupted input. It seems likely that we could improve the circuits by addressing these specific issues. However, there are also aspects of the circuits' failure that seem more random and inscrutable, and it is unclear if these issues can be fixed, or if there is some fundamental, inherent limitation to the circuits' explanatory power.

We conclude that it is important to find circuits that are more robust, and speculate that we might achieve this by using adversarial evaluation metrics in circuit discovery techniques.

5. Future Work

This paper proposes a method for evaluating circuits adversarially. As we have already alluded to, these evaluation criteria could be integrated into circuit discovery algorithms. In future work, we aim to do this and test its effectiveness. It might improve both the average and worst-case performance.

Additionally, the hope is that this will lead to circuits that are more robust under distributional shifts. The results of this paper show that even under small changes in the input, the circuit can lose its explanatory power. If we want to use circuits in safety-critical applications, they need to be more robust. It would be worthwhile to measure how robust current circuits are to distributional and to try to improve this robustness.

Acknowledgements

We thank MATS, AI Safety Support, and the Long-Term Future Fund (LTFF) for providing funding for this research and providing an amazing research environment. Additional, we are grateful for the compute resources provided by FAR AI, and for ChengCheng Tan for her editorial assistance in improving the readability of this paper. We also thank Iván Arcuschin Moreno for helpful discussions that improved this research, and Thomas Kwa for working on the edge-level subnetwork probing code that we adapted to run forward passes with ablations.

Impact statement

This paper aims to advance the field of mechanistic interpretability. While there are many potential societal consequences of our work, none need to be specifically highlighted here.

References

Arratia, R. and Gordon, L. Tutorial on large deviations for the binomial distribution. *Bulletin of Mathematical Biology*, 51:125–131, 1989. URL `https://api.semanticscholar.org/CorpusID:189884382`.

Chan, L., Garriga-Alonso, A., Goldwosky-Dill, N., Greenblatt, R., Nitishinskaya, J., Radhakrishnan, A., Shlegeris, B., and Thomas, N. Causal scrubbing, a method for rigorously testing interpretability hypotheses. *AI Alignment Forum*, 2022. `https://www.alignmentforum.org/posts/JvZhhzycHu2Yd57RN/causal-scrubbing-a-method-for-rigorously-testing`.

Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., and Garriga-Alonso, A. Towards Automated Circuit Discovery for Mechanistic Interpretability, October 2023.

Geiger, A., Potts, C., and Icard, T. Causal Abstraction for Faithful Model Interpretation, January 2023.

Hanna, M., Liu, O., and Variengien, A. How does GPT-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model, November 2023.

Heimersheim, S. and Janiak, J. A circuit for Python docstrings in a 4-layer attention-only transformer.

[8] The objects are: *ring*, *kiss*, *bone*, *basketball*, *computer*, *necklace*, *drink*, and *snack*.

https://www.alignmentforum.org/posts/u6KXXmKFbXfWzoAXn/a-circuit-for-python-docstrings-in-a-4-layer-attention-only, February 2023.

Hoeffding, W. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi: 10.1080/01621459.1963.10500830. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500830.

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the Wild: A Circuit for Indirect Object Identification in GPT-2 small, November 2022.

Zhang, F. and Nanda, N. Towards best practices of activation patching in language models: Metrics and methods, 2024.

A. Tables of worst-performing input points

See Tables 4 to 6 for the tables of (input, corrupted input) pairs on which the circuits perform the worst, together with the most likely outputs for those inputs. See Section 3 for more details.

Table 4. Top 10 worst-performing input pairs and corresponding 3 most likely outputs for the IOI task. The first two columns show the top 10 worst-performing input pairs, with the worst on top. The third column displays the KL divergence between the full model's output, and the circuit's output when run with resample ablation using the patch input, as explained in detail in Section 2. The last 6 columns show the three most likely output tokens for the model and the circuit, with that output's unnormalized logit score shown in parentheses beneath it.

			model			circuit		
input	patch input	loss	1st	2nd	3rd	1st	2nd	3rd
Then, Tiffany and Sean went to the house. Sean gave a basketball to	Then, Samuel and Adam went to the garden. Daniel gave a drink to	14.64	’ Tiffany’ (19.60)	’ the’ (14.24)	’ Sean’ (13.91)	’ them’ (17.16)	’ the’ (16.74)	’ Daniel’ (15.45)
Then, Crystal and Tyler went to the restaurant. Tyler gave a necklace to	Then, Samuel and Adam went to the garden. Daniel gave a drink to	14.30	’ Crystal’ (18.17)	’ the’ (14.59)	’ a’ (13.30)	’ them’ (16.80)	’ the’ (16.56)	’ Samuel’ (15.68)
Then, Tiffany and Sean went to the house. Sean gave a basketball to	Then, Samuel and Gregory went to the house. William gave a snack to	14.03	’ Tiffany’ (19.60)	’ the’ (14.24)	’ Sean’ (13.91)	’ them’ (16.64)	’ the’ (16.31)	’ Tiffany’ (14.82)
Then, Erica and Justin went to the house. Justin gave a kiss to	Then, Mark and David went to the garden. Paul gave a drink to	14.01	’ Erica’ (19.99)	’ her’ (15.19)	’ the’ (15.07)	’ them’ (17.55)	’ the’ (16.41)	’ David’ (14.69)
Then, Brittany and Brian went to the garden. Brian gave a basketball to	Then, Samuel and Adam went to the garden. Daniel gave a drink to	13.54	’ Brittany’ (18.97)	’ Brian’ (15.04)	’ the’ (14.80)	’ them’ (17.16)	’ the’ (16.63)	’ Samuel’ (14.91)
Then, Tiffany and Jason went to the school. Jason gave a basketball to	Then, Samuel and Adam went to the garden. Daniel gave a drink to	13.53	’ Tiffany’ (18.30)	’ the’ (14.47)	’ her’ (13.88)	’ them’ (17.29)	’ the’ (16.75)	’ his’ (14.82)
Then, Allison and Kevin went to the school. Kevin gave a necklace to	Then, Joseph and Joseph went to the garden. Thomas gave a basketball to	13.50	’ Allison’ (19.08)	’ the’ (14.58)	’ her’ (14.08)	’ Allison’ (17.10)	’ them’ (15.63)	’ the’ (15.48)
Then, Erica and Justin went to the house. Justin gave a kiss to	Then, Timothy and Samuel went to the house. Jesse gave a drink to	13.47	’ Erica’ (19.99)	’ her’ (15.19)	’ the’ (15.07)	’ them’ (17.35)	’ the’ (16.56)	’ Timothy’ (15.14)
Then, Erica and Justin went to the house. Justin gave a kiss to	Then, Samuel and Adam went to the garden. Daniel gave a drink to	13.40	’ Erica’ (19.99)	’ her’ (15.19)	’ the’ (15.07)	’ them’ (17.01)	’ the’ (16.60)	’ his’ (14.97)
Then, Erica and Justin went to the house. Justin gave a kiss to	Then, Benjamin and John went to the house. Charles gave a snack to	13.35	’ Erica’ (19.99)	’ her’ (15.19)	’ the’ (15.07)	’ Erica’ (17.19)	’ the’ (15.93)	’ them’ (15.88)

Table 5. Top 10 worst-performing input pairs and corresponding 3 most likely outputs for the docstring task.

			model			circuit		
input	patch input	loss	1st	2nd	3rd	1st	2nd	3rd
\ndef port(self, order, match, fields, model, old, parent): \n"""agent rule manager \n\n:param fields: set song \n:param model: plane action \n:param	\ndef image(self, key, file, filename, files, line, expected): \n"""package crime framework \n\n:param host: dollar author \n:param command: cup spring \n:param	12.07	’ old’ (19.66)	’ new’ (17.75)	’ fields’ (16.64)	’ file’ (18.87)	’ filename’ (17.41)	’ line’ (16.74)
\ndef default(self, node, user, current, text, port, item): \n"""export manager mission \n\n:param current: song spot \n:param text: delay draft \n:param	\ndef model(self, shape, message, group, file, result, fields): \n"""content host bed \n\n:param new: share stage \n:param page: lift range \n:param	12.07	’ port’ (20.52)	’ current’ (15.69)	’ str’ (15.31)	’ port’ (17.82)	’ filename’ (17.39)	’ message’ (17.22)
\ndef default(self, node, user, current, text, port, item): \n"""export manager mission \n\n:param current: song spot \n:param text: delay draft \n:param	\ndef image(self, key, file, filename, files, line, expected): \n"""package crime framework \n\n:param host: dollar author \n:param command: cup spring \n:param	12.06	’ port’ (20.52)	’ current’ (15.69)	’ str’ (15.31)	’ file’ (19.44)	’ filename’ (17.97)	’ line’ (17.93)
\ndef create(self, token, field, request, content, order, new): \n"""tree cut hell \n\n:param request: king bar \n:param content: income creation \n:param	\ndef image(self, key, file, filename, files, line, expected): \n"""package crime framework \n\n:param host: dollar author \n:param command: cup spring \n:param	11.95	’ order’ (20.48)	’ request’ (18.78)	’ field’ (16.64)	’ file’ (20.19)	’ filename’ (17.68)	’ line’ (17.47)
\ndef values(self, json, module, count, end, model, index): \n"""lead respect dust \n\n:param count: hell step \n:param end: volume pair \n:param	\ndef image(self, key, file, filename, files, line, expected): \n"""package crime framework \n\n:param host: dollar author \n:param command: cup spring \n:param	11.90	’ model’ (21.46)	’ models’ (16.23)	’ id’ (15.41)	’ file’ (20.41)	’ filename’ (18.51)	’ line’ (17.95)
\ndef match(self, results, default, order, check, row, field): \n"""activity path strength \n\n:param order: product plane \n:param check: fan bell \n:param	\ndef image(self, key, file, filename, files, line, expected): \n"""package crime framework \n\n:param host: dollar author \n:param command: cup spring \n:param	11.88	’ row’ (20.65)	’ check’ (16.86)	’ bool’ (16.57)	’ file’ (19.56)	’ check’ (19.51)	’ line’ (18.09)
\ndef command(self, code, instance, create, size, sub, run): \n"""border horse trip \n\n:param create: bishop attack \n:param size: duty horse \n:param	\ndef image(self, key, file, filename, files, line, expected): \n"""package crime framework \n\n:param host: dollar author \n:param command: cup spring \n:param	11.80	’ sub’ (20.32)	’ run’ (15.97)	’ name’ (15.80)	’ file’ (20.00)	’ bool’ (17.52)	’ filename’ (17.31)
\ndef default(self, node, user, current, text, port, item): \n"""export manager mission \n\n:param current: song spot \n:param text: delay draft \n:param	\ndef error(self, order, shape, match, filename, message, results): \n"""star opening risk \n\n:param file: cycle second \n:param content: race staff \n:param	11.53	’ port’ (20.52)	’ current’ (15.69)	’ str’ (15.31)	’ item’ (18.44)	’ text’ (18.06)	’ int’ (17.22)
\ndef item(self, old, code, header, response, node, sub): \n"""game phase birth \n\n:param header: cap session \n:param response: break player \n:param	\ndef image(self, key, file, filename, files, line, expected): \n"""package crime framework \n\n:param host: dollar author \n:param command: cup spring \n:param	11.44	’ node’ (21.31)	’ code’ (17.17)	’ child’ (15.85)	’ file’ (20.06)	’ line’ (18.03)	’ node’ (17.91)
\ndef expected(self, root, results, host, module, names, files): \n"""horse boot sector \n\n:param host: thinking rock \n:param module: rent tie \n:param	\ndef error(self, action, image, source, old, text, content): \n"""charge conduct wife \n\n:param task: meaning shadow \n:param field: warning self \n:param	11.41	’ names’ (21.83)	’ name’ (19.86)	’ files’ (17.22)	’ image’ (17.93)	’ file’ (17.43)	’ name’ (17.00)

Table 6. Top 10 worst-performing input pairs and corresponding 3 most likely outputs for the greater-than task.

			model			circuit		
input	patch input	loss	1st	2nd	3rd	1st	2nd	3rd
The dispute lasted from the year 1694 to 16	The voyage lasted from the year 1601 to 16	5.31	’95’ (27.43)	’97’ (26.22)	’96’ (26.19)	’95’ (27.09)	’99’ (25.66)	’97’ (25.32)
The dispute lasted from the year 1694 to 16	The expedition lasted from the year 1701 to 17	5.22	’95’ (27.43)	’97’ (26.22)	’96’ (26.19)	’95’ (27.07)	’99’ (25.37)	’97’ (24.50)
The dispute lasted from the year 1694 to 16	The pilgrimage lasted from the year 1601 to 16	5.18	’95’ (27.43)	’97’ (26.22)	’96’ (26.19)	’95’ (27.45)	’99’ (26.14)	’97’ (25.98)
The dispute lasted from the year 1694 to 16	The pilgrimage lasted from the year 1601 to 16	5.18	’95’ (27.43)	’97’ (26.22)	’96’ (26.19)	’95’ (27.45)	’99’ (26.14)	’97’ (25.98)
The dispute lasted from the year 1694 to 16	The voyage lasted from the year 1101 to 11	5.17	’95’ (27.43)	’97’ (26.22)	’96’ (26.19)	’95’ (24.12)	’99’ (23.75)	’50’ (23.06)
The dispute lasted from the year 1694 to 16	The voyage lasted from the year 1101 to 11	5.17	’95’ (27.43)	’97’ (26.22)	’96’ (26.19)	’95’ (24.12)	’99’ (23.75)	’50’ (23.06)
The dispute lasted from the year 1694 to 16	The voyage lasted from the year 1101 to 11	5.17	’95’ (27.43)	’97’ (26.22)	’96’ (26.19)	’95’ (24.12)	’99’ (23.75)	’50’ (23.06)
The dispute lasted from the year 1694 to 16	The pilgrimage lasted from the year 1201 to 12	5.15	’95’ (27.43)	’97’ (26.22)	’96’ (26.19)	’95’ (25.83)	’99’ (25.40)	’97’ (25.16)
The raids lasted from the year 1788 to 17	The expedition lasted from the year 1701 to 17	5.13	’89’ (28.32)	’90’ (27.60)	’99’ (27.38)	’89’ (28.30)	’90’ (27.61)	’93’ (27.17)
The raids lasted from the year 1788 to 17	The voyage lasted from the year 1601 to 16	5.12	’89’ (28.32)	’90’ (27.60)	’99’ (27.38)	’89’ (27.21)	’90’ (27.16)	’99’ (26.71)

B. Results with less adversarial patch inputs

For our main results in Section 3, we took a very adversarial approach towards the patch inputs: any clean input could be paired with any patch input for resample ablation. However, the circuits were originally found and tested with resample ablations that were more restrictive:

- In the IOI task, the location and object in the corrupted input were the same as in the clean input.
- In the docstring task, the only difference between the corrupted input and the clean input was the parameter names in the docstring.
- In the greater-than task, the event and the first two digits of the years in the corrupted input were the same as in the clean input.

(We recall that a short description of these tasks is shown in Table 2.)

We believe the our more adversarial approach that allows any corrupted input to be matched with any clean input, is justified for the following reasons:

- The high-level explanation in (Wang et al., 2022) suggests that the model identifies the indirect object through a mechanism that does not depend in any way on the location or object. Some attention heads are name identifiers, others duplicate detectors, others name inhibitors – none of these depend on the location or object.

- The additional information that is ablated is not necessary to complete the task.
- The tasks inputs still follow the same restrictive template.

For completeness, this section presents the adversarial evaluation metrics on random samples of 1 million input-corrupted-input pairs where the patch inputs are matched in the same way as in the original dataset.

Table 7 shows the standard deviations and the means are lower than if we allow any corrupt input, indicating that the circuits indeed perform better on these matched input-corrupted-input pairs. However, the worst points are many standard deviations (9.97 and 15.47 for docstring and IOI, respectively) removed from the mean, so there are still inputs on which the circuits perform very poorly.

The table with the top 10 worst inputs for IOI (Table 8) shows that many of our conclusions still hold: the worst inputs look very benign, but the model correctly predicts the next token, whereas the circuit either takes a name from the patched input or repeats the subject rather than identifying the indirect object, with very high confidence.

The greater-than circuit performs very well, and on the top 10 worst inputs, all the most likely tokens are permissible. Most of the 10 worst inputs for the docstring task usually predict a token that is indeed one of the parameters, but it has already occurred before in the clean input.

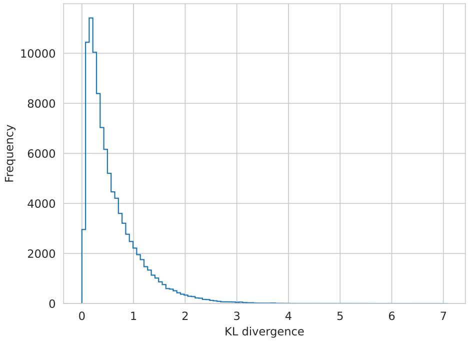

Figure 4. A histogram of the KL divergence for the IOI task, where all input-coruputed-input pairs are matched in the same way as in the original dataset, i.e., with the same location and object in the corrupted input as in the clean input. The x-axis shows the KL divergence between the model's output and the circuit's output on an input-corrupted-input pair, and the y-axis shows the number of input-corrupted-input pairs from our random sample of 1 million points that fall into each bin. There are 100 bins of equal size between the values of 0 and the maximum KL divergence achieved. Summary statistics of the plotted distribution are displayed in Table 3.

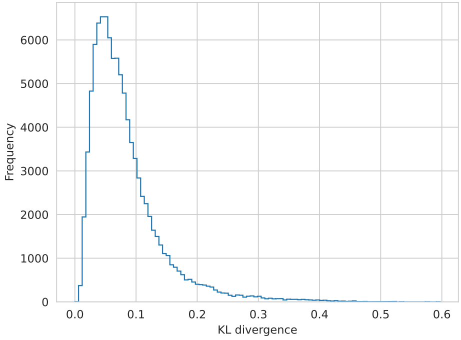

Figure 5. A histogram of the KL divergence for the greater-than task where all input-coruputed-input pairs are matched in the same way as in the original dataset, i.e., with the same event and first two digits in the corrupted input as in the clean input.

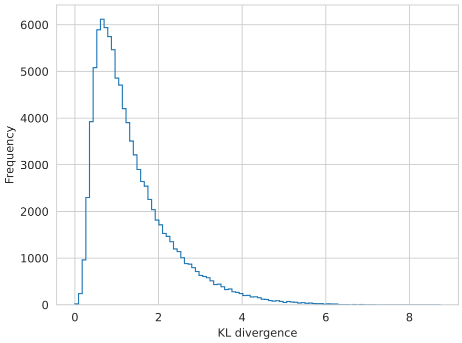

Figure 6. A histogram of the KL divergence for the docstring task where all input-corruputed-input pairs are matched in the same way as in the original dataset, i.e., the only difference between the corrupted input and the clean input is the parameter names in the docstring (but the parameters in the function signature are the same).

Table 7. Summary statistics from the KL divergence distributions plotted in Figures 4 to 6. The columns labelled "abs" show the absolute values of the KL divergence, whereas the columns labelled "z-score" show the difference between the percentile and the mean expressed as a multiple of the standard deviation.

	docstring		greaterthan		ioi	
	abs	z-score	abs	z-score	abs	z-score
count	1000000.00		1000000.00		1000000.00	
mean	1.38		0.08		0.59	
std	0.93		0.06		0.53	
min	0.01	-1.46	0.01	-1.29	0.00	-1.10
25%	0.71	-0.72	0.04	-0.66	0.21	-0.71
50%	1.11	-0.28	0.07	-0.25	0.42	-0.32
75%	1.78	0.43	0.10	0.32	0.80	0.39
95%	3.29	2.04	0.20	1.94	1.65	1.98
99%	4.57	3.41	0.33	3.99	2.49	3.57
99.9%	6.06	5.01	0.46	6.14	3.85	6.12
99.99%	7.33	6.37	0.54	7.45	5.34	8.92
max	9.97	9.19	0.60	8.47	8.83	15.47

Table 8. Top 10 worst-performing input pairs and corresponding 3 most likely outputs for the IOI task where all input-corrupted-input pairs are matched in the same way as in the original dataset. The first two columns show the top 10 worst-performing input pairs, with the worst on top. The third column displays the KL divergence between the full model's output, and the circuit's output when run with resample ablation using the patch input, as explained in detail in Section 2. The last 6 columns show the three most likely output tokens for the model and the circuit, with that output's unnormalized logit score shown in parentheses beneath it.

input	patch input	loss	model 0	model 1	model 2	circuit 0	circuit 1	circuit 2
Then, Stephen and Jacob had a lot of fun at the house. Jacob gave a necklace to	Then, Jacob and Kelly had a lot of fun at the house. Adam gave a necklace to	7.07	’ Stephen’ (17.77)	’ the’ (14.43)	’ his’ (14.32)	’ Jacob’ (20.96)	’ the’ (14.43)	’ his’ (14.27)
Then, Alicia and Steven had a lot of fun at the hospital. Steven gave a kiss to	Then, Jacob and Jose had a lot of fun at the hospital. Amber gave a kiss to	6.62	’ Alicia’ (18.22)	’ her’ (15.33)	’ the’ (15.10)	’ Steven’ (20.09)	’ her’ (14.66)	’ the’ (14.60)
Then, Brandon and Rachel had a lot of fun at the store. Rachel gave a basketball to	Then, Rachel and Jesse had a lot of fun at the store. Paul gave a basketball to	6.37	’ Brandon’ (18.98)	’ Rachel’ (15.06)	’ the’ (14.11)	’ Rachel’ (20.26)	’ the’ (14.15)	’ her’ (13.58)
Then, Brandon and Rachel had a lot of fun at the garden. Rachel gave a kiss to	Then, Rebecca and Gregory had a lot of fun at the garden. Aaron gave a kiss to	6.25	’ Brandon’ (18.37)	’ Rachel’ (16.05)	’ the’ (14.73)	’ Rachel’ (21.39)	’ the’ (15.49)	’ her’ (15.17)
Then, Stephanie and Joseph had a lot of fun at the restaurant. Joseph gave a necklace to	Then, Joseph and Nathan had a lot of fun at the restaurant. Jennifer gave a necklace to	6.19	’ Stephanie’ (19.24)	’ the’ (14.61)	’ her’ (13.82)	’ Joseph’ (19.28)	’ the’ (15.24)	’ them’ (14.32)
Then, Patrick and Rachel had a lot of fun at the restaurant. Rachel gave a basketball to	Then, Samuel and Lauren had a lot of fun at the restaurant. Patrick gave a basketball to	5.86	’ Patrick’ (18.38)	’ Rachel’ (14.38)	’ the’ (14.34)	’ Rachel’ (19.87)	’ the’ (14.48)	’ a’ (13.68)
Then, Joshua and Rachel had a lot of fun at the garden. Rachel gave a kiss to	Then, Christina and Jonathan had a lot of fun at the garden. Melissa gave a kiss to	5.75	’ Joshua’ (18.42)	’ Rachel’ (15.34)	’ the’ (15.12)	’ Rachel’ (20.42)	’ the’ (15.26)	’ her’ (14.71)
Then, Vanessa and Stephen had a lot of fun at the garden. Stephen gave a kiss to	Then, Sara and Travis had a lot of fun at the garden. Rebecca gave a kiss to	5.68	’ Vanessa’ (18.52)	’ the’ (15.26)	’ her’ (14.95)	’ Stephen’ (17.42)	’ the’ (15.46)	’ her’ (14.78)
Then, Richard and Erin had a lot of fun at the store. Erin gave a ring to	Then, Allison and Jose had a lot of fun at the store. Nicholas gave a ring to	5.63	’ Richard’ (17.33)	’ the’ (14.18)	’ Erin’ (13.54)	’ Erin’ (20.11)	’ the’ (14.36)	’ a’ (13.34)
Then, Thomas and Dustin had a lot of fun at the store. Dustin gave a necklace to	Then, Allison and Jose had a lot of fun at the store. Amy gave a necklace to	5.60	’ Thomas’ (16.60)	’ the’ (14.34)	’ his’ (13.40)	’ Dustin’ (19.55)	’ the’ (14.61)	’ Dust’ (13.71)

Table 9. Top 10 worst-performing input pairs and corresponding 3 most likely outputs for the docstring task where all input-corrupted-input pairs are matched in the same way as in the original dataset.

input	patch input	loss	model 0	model 1	model 2	circuit 0	circuit 1	circuit 2
\ndef date(self, options, result, context, user, tag, error): \n"""bench round model \n\n:param context: input sense \n:param user: album second \n:param	\ndef date(self, options, result, port, shape, new, error): \n"""bench round model \n\n:param parent: input sense \n:param order: album second \n:param	8.73	’ tag’ (22.25)	’ tags’ (16.94)	’ context’ (16.52)	’ result’ (18.15)	’ error’ (16.81)	’:’ (16.53)
\ndef client(self, url, image, file, server, values, request): \n"""fuel scale acid \n\n:param file: pub resident \n:param server: cell disk \n:param	\ndef client(self, url, image, token, code, state, request): \n"""fuel scale acid \n\n:param content: pub resident \n:param msg: cell disk \n:param	7.64	’ values’ (23.30)	’ value’ (19.80)	’ data’ (17.54)	’ server’ (19.55)	’ file’ (18.60)	’ request’ (17.34)
\ndef source(self, content, group, project, tag, run, test): \n"""seed post sample \n\n:param project: command distance \n:param tag: bank delay \n:param	\ndef source(self, content, group, results, options, name, test): \n"""seed post sample \n\n:param default: command distance \n:param current: bank delay \n:param	7.54	’ run’ (21.77)	’ test’ (16.69)	’ project’ (16.68)	’ project’ (16.79)	’ target’ (16.23)	’ group’ (16.22)
\ndef check(self, action, last, text, base, run, table): \n"""message duty scope \n\n:param text: bank height \n:param base: post sum \n:param	\ndef check(self, action, last, title, path, url, table): \n"""message duty scope \n\n:param current: bank height \n:param call: post sum \n:param	7.43	’ run’ (20.48)	’ base’ (16.45)	’ line’ (16.41)	’ table’ (17.80)	’ str’ (17.56)	’ base’ (17.25)
\ndef call(self, path, end, option, log, instance, msg): \n"""style drop demand \n\n:param option: colour entry \n:param log: impact cancer \n:param	\ndef call(self, path, end, task, update, new, msg): \n"""style drop demand \n\n:param node: colour entry \n:param header: impact cancer \n:param	7.42	’ instance’ (20.95)	’ str’ (15.92)	’ bool’ (15.80)	’ log’ (17.91)	’ path’ (17.09)	’ str’ (16.92)
\ndef date(self, options, num, page, table, files, default): \n"""root fund boy \n\n:param page: bar finger \n:param table: lane storm \n:param	\ndef date(self, options, num, value, config, order, default): \n"""root fund boy \n\n:param valid: bar finger \n:param group: lane storm \n:param	7.29	’ files’ (23.10)	’ file’ (21.28)	’ filename’ (17.47)	’ table’ (17.95)	’ num’ (16.62)	’ str’ (16.54)
\ndef tag(self, content, port, test, end, model, count): \n"""top release drop \n\n:param test: collection reading \n:param end: protein dream \n:param	\ndef tag(self, content, port, date, target, text, count): \n"""top release drop \n\n:param string: collection reading \n:param index: protein dream \n:param	7.26	’ model’ (19.99)	’ test’ (16.13)	’ models’ (15.54)	’ string’ (16.73)	’ int’ (16.02)	’ bool’ (15.57)
\ndef instance(self, state, size, project, image, fields, run): \n"""father sort horse \n\n:param project: dollar protein \n:param image: duty net \n:param	\ndef instance(self, state, size, server, end, target, run): \n"""father sort horse \n\n:param config: dollar protein \n:param description: duty net \n:param	7.12	’ fields’ (21.71)	’ field’ (18.85)	’ name’ (16.97)	’ value’ (14.27)	’ str’ (14.22)	’ int’ (14.15)
\ndef data(self, parent, new, url, model, found, count): \n"""bone trip user \n\n:param url: user location \n:param model: device object \n:param	\ndef data(self, parent, new, date, order, message, count): \n"""bone trip user \n\n:param field: user location \n:param command: device object \n:param	7.12	’ found’ (19.72)	’ discovered’ (15.77)	’ data’ (15.38)	’ url’ (16.38)	’ description’ (15.97)	’ model’ (15.96)
\ndef user(self, current, server, table, tag, result, group): \n"""cake saving pub \n\n:param table: fashion user \n:param tag: committee tree \n:param	\ndef user(self, current, server, fields, base, match, group): \n"""cake saving pub \n\n:param order: fashion user \n:param old: committee tree \n:param	7.07	’ result’ (22.19)	’ user’ (16.85)	’ current’ (16.37)	’ table’ (16.64)	’ user’ (16.43)	’ server’ (16.14)

Table 10. Top 10 worst-performing input pairs and corresponding 3 most likely outputs for the greater-than task where all input-corrupted-input pairs are matched in the same way as in the original dataset.

input	patch input	loss	model 0	model 1	model 2	circuit 0	circuit 1	circuit 2
The sanctions lasted from the year 1520 to 15	The sanctions lasted from the year 1501 to 15	0.60	’30’ (25.73)	’25’ (25.17)	’40’ (24.95)	’21’ (25.54)	’23’ (25.27)	’22’ (25.15)
The sanctions lasted from the year 1520 to 15	The sanctions lasted from the year 1501 to 15	0.60	’30’ (25.73)	’25’ (25.17)	’40’ (24.95)	’21’ (25.54)	’23’ (25.27)	’22’ (25.15)
The reforms lasted from the year 1520 to 15	The reforms lasted from the year 1501 to 15	0.59	’30’ (25.91)	’25’ (25.26)	’40’ (25.14)	’21’ (25.38)	’23’ (25.07)	’22’ (24.98)
The accord lasted from the year 1520 to 15	The accord lasted from the year 1501 to 15	0.58	’30’ (25.75)	’25’ (25.24)	’40’ (24.78)	’21’ (25.85)	’22’ (25.39)	’23’ (25.30)
The accord lasted from the year 1520 to 15	The accord lasted from the year 1501 to 15	0.58	’30’ (25.75)	’25’ (25.24)	’40’ (24.78)	’21’ (25.85)	’22’ (25.39)	’23’ (25.30)
The accord lasted from the year 1520 to 15	The accord lasted from the year 1501 to 15	0.58	’30’ (25.75)	’25’ (25.24)	’40’ (24.78)	’21’ (25.85)	’22’ (25.39)	’23’ (25.30)
The flights lasted from the year 1580 to 15	The flights lasted from the year 1501 to 15	0.56	’90’ (27.01)	’85’ (24.96)	’80’ (24.83)	’90’ (26.09)	’83’ (25.51)	’85’ (25.41)
The flights lasted from the year 1580 to 15	The flights lasted from the year 1501 to 15	0.56	’90’ (27.01)	’85’ (24.96)	’80’ (24.83)	’90’ (26.09)	’83’ (25.51)	’85’ (25.41)
The flights lasted from the year 1680 to 16	The flights lasted from the year 1601 to 16	0.54	’90’ (28.71)	’80’ (26.84)	’85’ (26.71)	’90’ (27.09)	’83’ (26.79)	’82’ (26.49)
The flights lasted from the year 1680 to 16	The flights lasted from the year 1601 to 16	0.54	’90’ (28.71)	’80’ (26.84)	’85’ (26.71)	’90’ (27.09)	’83’ (26.79)	’82’ (26.49)

C. Proof of Percentile Bounds

In this appendix, we prove Proposition 2.1 and Corollary 2.2.

We remind the reader of the setup. Let X be some randomly distributed variable, let $0 < p < 1$, and let $\epsilon > 0$ with $p + \epsilon < 1$. Suppose we have a sample of n i.i.d. draws from X and we want to use that sample to find an upper bound of the real (but unknown) p-th percentile of X. We denote the real p-th percentile by x_p and we take as our estimate for the upper bound

$$\hat{x}_p := \text{the } \lceil n(p+\epsilon) \rceil\text{-th element in the sample ordered by value from low to high.} \tag{6}$$

Note that ϵ can be considered a kind of safety margin: by making it bigger, we get a less tight estimate of the upper bound, but we increase the probability that it is actually an upper bound.

Proof of Proposition 2.1. The probability $\mathcal{P}(\hat{x}_p \geq x_p)$ is the same as the probability that fewer than $\lceil n(p+\epsilon) \rceil$ elements from our sample come from the lower p percentiles of the distribution — indeed, this is equivalent to saying that the $\lceil n(p+\epsilon) \rceil$-th element comes from the upper $1-p$ percentiles, and is hence at at least as large as x_p.

We can calculate this probability with the binomial distribution $\mathrm{Binom}(n, p)$, because the probability of drawing a sample from the lower p percentiles is precisely p. □

Let $Y \sim \mathrm{Binom}(n, p)$ be a binomially distributed random variable, and let $p < a < 1$. Then the Chernoff bound (Arratia & Gordon, 1989, Theorem 1) says

$$\Pr(Y \geq an) \leq \exp(-n\,\mathrm{D}_{\mathrm{KL}}\left(\mathrm{Bern}(a) \parallel \mathrm{Bern}(p)\right)). \tag{7}$$

Alternatively, Hoeffding's inequality (Hoeffding, 1963, Theorem 1) says

$$\Pr\left(\frac{1}{n}Y - p \geq a - p\right) \leq \exp(-n(a-p)^2) \tag{8}$$

which we can rewrite to

$$\Pr(Y \geq an) \leq \exp(-n(a-p)^2). \tag{9}$$

Proof of Corollary 2.2. Proposition 2.1 tells us

$$\Pr\left(\hat{x}_p \geq x_p\right) = F_{\text{Binom}}(\lceil (p+\epsilon)\cdot n \rceil - 1; n, p) \tag{10}$$

We can rewrite the right hand side:

$$\begin{aligned} F_{\text{Binom}}(\lceil (p+\epsilon)\cdot n \rceil - 1; n, p) &= \Pr\left(Y \leq \lceil (p+\epsilon)\cdot n \rceil - 1\right) \\ &= 1 - \Pr\left(Y \geq \lceil (p+\epsilon)\cdot n \rceil\right) \end{aligned} \tag{11}$$

where $Y \sim \text{Binom}(n, p)$. Applying the Chernoff bound (7), or alternatively applying the Hoeffding inequality (9), gives us the inequalities we're looking for. □

D. Analysis of circuit performance grouped by prompt fields

In 3, we remarked on some patterns in the top 10 worst-performing inputs listed in Appendix A. In this appendix, we provide additional support for those claims, by not just looking at the top 10 worst-performing inputs, but by grouping all inputs based on a template value in their prompt (e.g. in the IOI task, the place, or the object that is being given). The data shows that certain template values lead to higher losses more often.

For IOI, Figures 7 to 9 show that the performance of the IOI circuit in the higher percentiles varies considerably with the object and the location that appear in the clean input prompt. The more romantic objects, such as "kiss" and "necklace", perform especially poorly, but there are also other objects and object-location combinations that perform poorly. In future work we hope to find a mechanistic explanation for the circuit's failure in these cases.

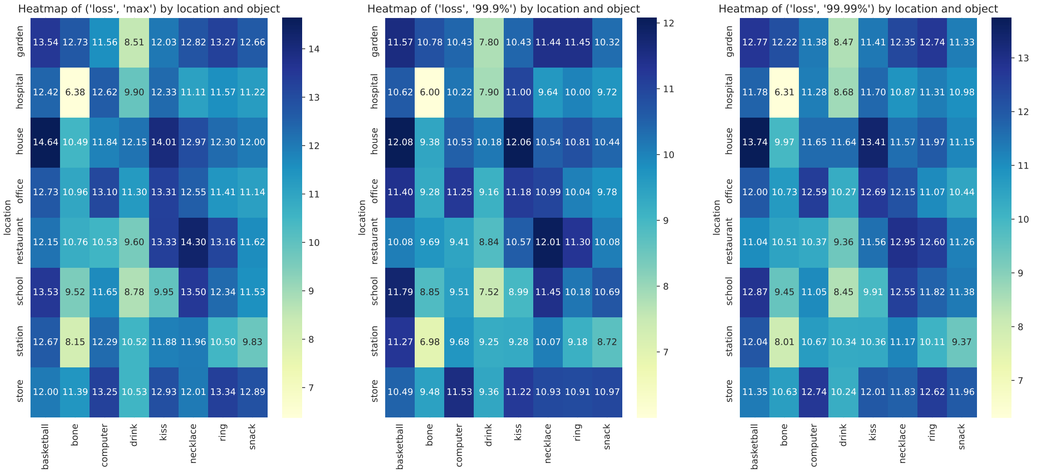

Figure 7. IOI: Three heatmaps, showing different percentiles (the max, the 99.9th percentile, and the 99.99th percentile) of the distribution of KL divergences between the circuit's output and the model's output on a sample of 1 million input-corrupted-input pairs, as in Section 3, plotted against the location and the object in the clean input prompt.

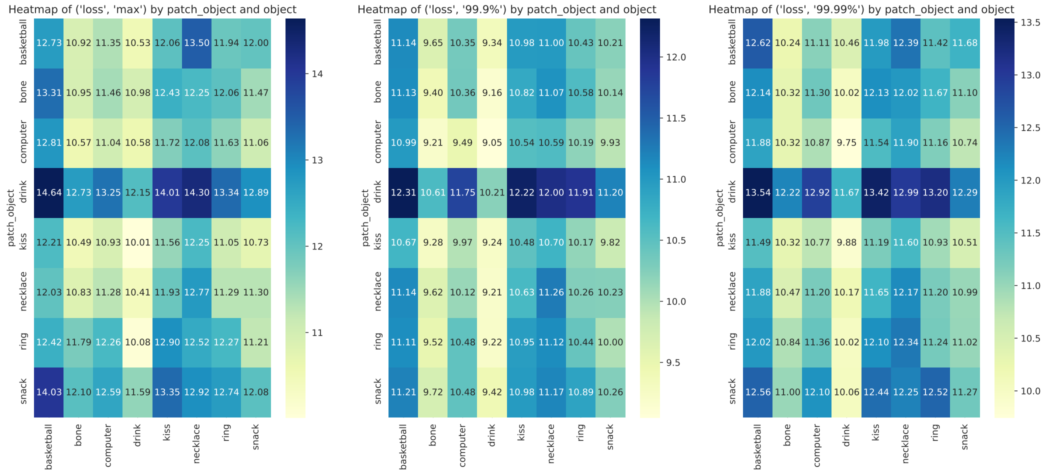

Figure 8. IOI: Three heatmaps, showing different percentiles (the max, the 99.9th percentile, and the 99.99th percentile) of the distribution of KL divergences between the circuit's output and the model's output on a sample of 1 million input-corrupted-input pairs, as in Section 3, plotted against the object in the clean input and the object in the patch input.

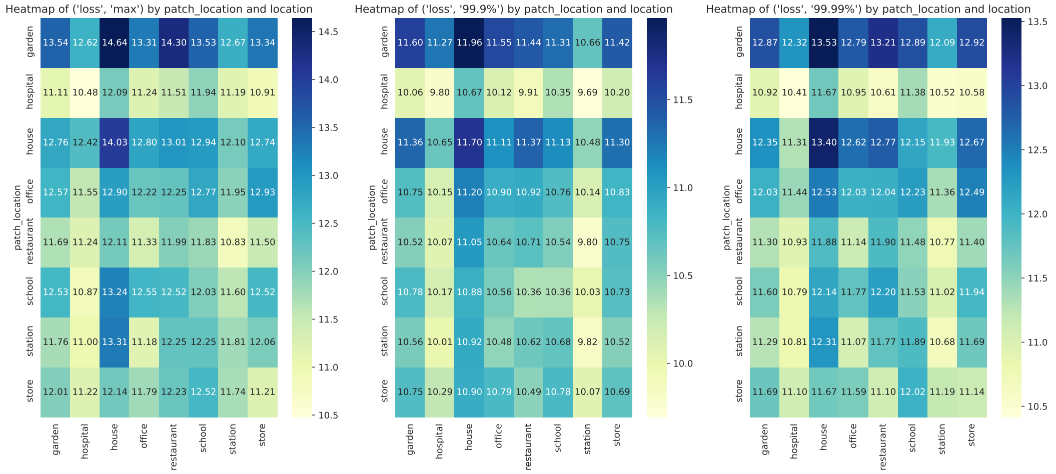

Figure 9. IOI: Three heatmaps, showing different percentiles (the max, the 99.9th percentile, and the 99.99th percentile) of the distribution of KL divergences between the circuit's output and the model's output on a sample of 1 million input-corrupted-input pairs, as in Section 3, plotted against the location in the clean input and the location in the patch input.

For the docstring task, we could not identify and then statistically confirm a clear hypothesis for why some inputs fared much worse than others.

For the greater-than task, Figure 10 confirms that the circuit performs especially well when the last two digits of the year in the clean input are very low (e.g. 1705), and especially poorly when the last two digits are very high (e.g. 1789), as remarked towards the end of Section 3.

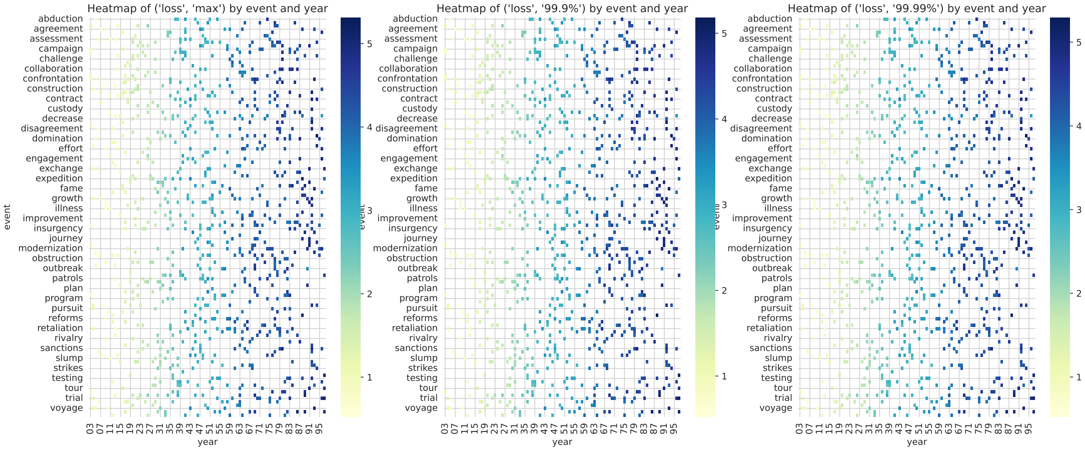

Figure 10. Greater-Than: Three heatmaps, showing different percentiles (the max, the 99.9th percentile, and the 99.99th percentile) of the distribution of KL divergences between the circuit's output and the model's output on a sample of 1 million input-corrupted-input pairs, as in Section 3, plotted against the event and the year in the clean input prompt.