Circuits in Mechanistic Interpretability[1]

Frederik Callin Østern, Jonas Jørgensen Telle and Klara Liaaen Aronsen

Faculty of Mathematics and Natural Sciences, University of Oslo, Norway

Dated: 25.08.2025

Abstract: We implement and apply simple, established methods in Transformer Mechanistic Interpretability (ablation, activation patching and path patching) to localize and describe model features and circuits. Using these, we locate an attention head which seems to respond strongly to color-words. We also explore interpretability libraries like Captum and TransformerLens, as well as visualization libraries like BertViz and CircuitsVis. Using input attribution from Captum, we seek to i) find model misconceptions on physics questions, and ii) compare the embedding space structure of different embedding models, though the results are disappointing. TransformerLens, on the other hand, seems a promising library for further study, and we demonstrate finding induction heads in small-scale models, activation patching for GPT-style models and logit lens for visualizing how the residual stream evolves through a model.

[1]This report is *very* long. **We suggest reading the abstract, then skipping to the overview (Section 2), before reading the first few paragraphs under each subsection in Applications (First paragraphs of section 4.1, 4.2 and 4.3)**. This should give enough context that the reader will be able to navigate to the subsections of interest in the Theory and Applications sections. The Introduction and Discussion are written to be general and should not require that the reader is familiar with the Theory section, nor details from the Applications section.

Contents

1 Introduction

The transition from symbolic AI to trained models like neural networks (NNs) gave unprecedented advances on difficult problems like computer vision and natural language processing. However, the increased performance came at a cost of increased opaqueness; these models, which could suddenly classify tumors and fold proteins (Coudray et al., 2018; AlphaFold, 2020), did not have the inherent interpretability of human designed expert systems. This raised both immediate epistemological issues, as people hesitate to trust a model prediction without an understandable justification, and long-term safety concerns, as increasingly powerful semi-autonomous agents should not be unleashed upon the world unless we know for sure that they are benign.

To counteract these concerns, and to potentially improve existing models, the field of AI interpretability emerged, and it has become a research area also pursued out of intrinsic curiosity about information processing more generally. Optimists may hold to a variant of the universality postulate (Olah et al., 2020), regarding the extent to which learned problem-solving algorithms generalize across systems. For instance, there are slight indications that circuits found in convolutional neural networks used for image classification are also implemented in the human visual system (Agrawal et al., 2014). If certain problems have specific solution algorithms upon which both natural evolution and machine learning converge, the study of machine brains may, for example, become an interesting playground for neuroscientific hypothesis generation and understanding.

Mechanistic interpretability (MI) is one 'natural science'-inspired approach to AI interpretability, based on zooming in on parts of the network which are sufficiently small to permit concrete, falsifiable predictions. The current paradigm is loosely based on three postulates from Olah et al. (2020), assuming the existence of human-interpretable i) *features* and ii) *circuits* in models, in addition to iii) a universality claim. Briefly, a feature is the fundamental unit of the network – a single concept the model has learned – while a circuit is an algorithm using previous features to compute new ones. For example, the features 'wheel', ‘car body' and 'car window' can enter a car-detection circuit, culminating in the feature ‘car’. This suggests viewing the AI model as an idealized graph with features as the nodes and circuits as directed subgraphs. The end goal of MI then becomes to understand the circuits fully, that is, to understand the algorithms the model uses to solve the tasks we give it[2]. In other words, the goal is *to localize features and circuits and ensure a causal connection between their activation and interpretable concepts*. Our work here implements and demonstrates some simple established methods for attempting this.

We work with transformer models, with which we assume the reader is somewhat familiar. The basic architecture is displayed in Figure 1; the main differences compared to a normal NN are the attention layers, which move information between input positions, and the residual stream, which represents the fact that the transformer layer output is *added* to its input, so that the input is not overwritten like in a normal NN. Conceptually, this means that for many inputs, only a small part of the model is actually making substantial contributions to the output. This should avoid bottlenecks so that increased model depth is not wasted – empirically, *LLM scaling laws* (Kaplan et al., 2020) show that the easiest way to improve performance is to increase size and training – and might reassure us that interpretability difficulty could scale well with model size.

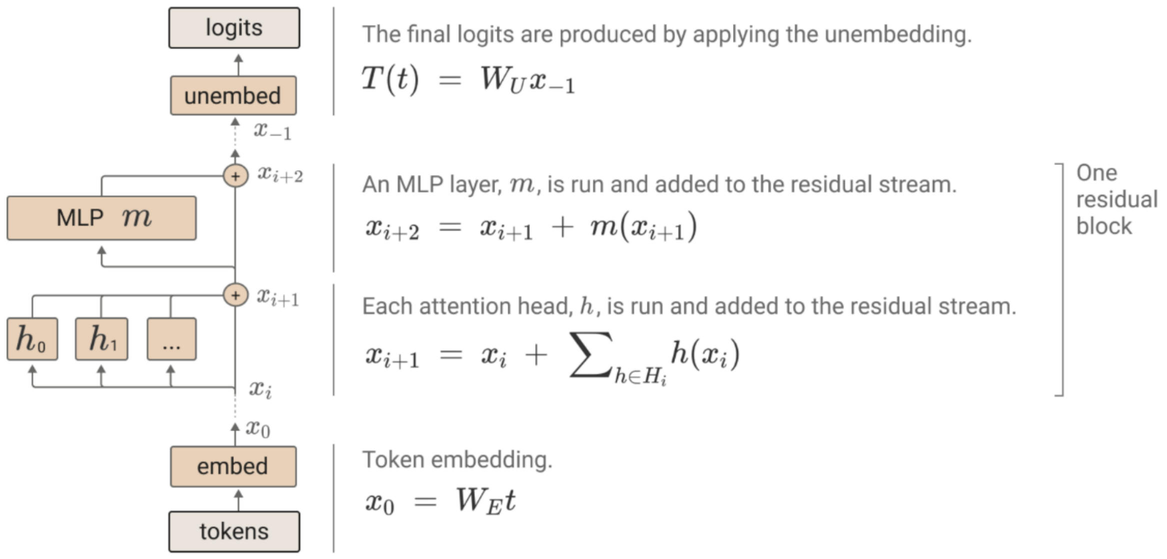

logits
unembed
x_{-1}
x_{i+2}
MLP m
x_{i+1}
h_0
h_1
...
x_i
x_0
embed
tokens
The final logits are produced by applying the unembedding.
$T(t) = W_U x_{-1}$
An MLP layer, m, is run and added to the residual stream.
$x_{i+2} = x_{i+1} + m(x_{i+1})$
Each attention head, h, is run and added to the residual stream.
$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i)$
Token embedding.
$x_0 = W_E t$
One residual block

Figure 1: Transformer architecture overview. If unfamiliar, an in-depth explanation can be found here. The image is from Olah, et al., 2021, which provides a conceptual framework for transformer interpretability.

Most MI research on transformers is done on next-token predictors like GPT, and we explore common libraries like TransformerLens and CircuitsVis for decoder interpretability and visualization. However, we also use more general libraries like Transformers and Captum to

[2] Note that it is an unverified postulate that the translation from model to feature graph is reasonable, and again a stretch to assume that all algorithms implemented can be captured by this graph. It is especially not obvious how to view attention mechanisms, which move information around in addition to processing it, in such a description.

implement MI tools for BERT-style encoders, allowing common interventions and layer-wise input attribution.

The report is structured as follows: Section 2 serves as a brief overview of the methods and libraries we have employed. In Section 3, we dig into the theory behind these methods, which will then be applied in Section 4. This section starts with a long subsection, including method, results and discussion on interventions for embedding models, corresponding to the first accompanying computational essay. Thereafter follows a short section with motivations and discussion around input attribution, corresponding to the second computational essay. Finally, Section 4.3 presents TransformerLens and corresponds to the third computational essay. In Section 5, we provide a general discussion of failed attempts and ideas for further study. We conclude with Section 6.

2 Overview

With the vast array of models in use, in addition to the countless problems one could wish to ask about them, there is a wide selection of methods to choose from when analyzing a model. What follows is an overview of the most important methods and libraries we used, as well as some others we tried. A more detailed description of the methods is provided in Section 3. Furthermore, a discussion on the limitations of certain methods and libraries is saved for Section 5.

2.1 Exploratory methods

When starting the analysis of a new model, it is difficult to know exactly where the important parts are. By using various visualization tools and other non-intervening methods, one may locate some of these parts. This is often the first step in proposing a hypothesis. In our work, visualizing attention patterns proved useful for locating active attention heads and finding induction heads in simpler examples. We also used logit lens, a technique where one accesses the intermediate layers in the residual stream and "pretends" that these are the last ones. This gives us a way to see how a model changes its most likely prediction throughout the network. Finally, we used input attribution to gauge which features in the input were most important for the model output.

While these non-intervening methods are suitable for exploring a network and locating important sections, they do not show the direct effects of certain nodes or how different parts are connected. Hence, we need stronger methods for this second step. Instead of simply looking at what the network does by itself, we intervene at predetermined components and change the activations in some way, depending on the method. This gives us more concrete indications for whether specific components contribute meaningfully.

In our work, we focused on three main intervention techniques: ablation, activation patching and path patching. Ablation works by accessing intermediate activations and setting them to zero. If this significantly changes the output, this indicates that these activations were important for the model's prediction. It is crucial to note, though, that changing an activation to such a large extent could affect other areas of the network without us noticing. We could rectify this by changing to a technique we call "soft ablation", where one changes the activations to nonzero values.

Activation patching, on the other hand, uses the calculated activation values of one prompt and patches these onto the intermediate values of another prompt. In particular, this is used for finding which nodes are necessary for a certain output. Here, spending some time to wisely choose the pair of prompts is important, for one wants to minimize the possible factors that could influence the outcome.

Lastly, path patching can be used to quantify the *strength* of a connection between two nodes in a transformer network. That is, given a change to the activation of one node, one quantifies how much the activation of a later node changes as a response. This technique is very useful in circuit analysis, as it is used to assess how strong the information flow between a pair of nodes in a circuit is.

2.2 Confirmatory methods

While the above methods are helpful for finding relevant areas of a model and generating hypotheses, they are not suited for a thorough verification of those. A systematic run-through of a method like activation patching would be expensive and prone to overlooking components. Hence, we need stronger and more rigorous methods for determining if our hypotheses hold more generally. There have been some attempts at constructing such methods, most notably causal scrubbing (Chan et al., 2022) and causal abstraction (Geiger et al., 2025). Both methods work by representing the network as a computational graph and letting the relevant hypothesis act as a subgraph. Then, by essentially removing every component we hypothesize to be unimportant, we can have a more well-defined process when iterating through the system. We tried implementing causal scrubbing for GPT-style models with the TransformerLens library, but we were unsuccessful[3]. Having said that, exploratory methods are still much more prominent in interpretability research, so the mentioned methods work more as an ideal and theoretical framework.

2.3 Libraries

With the emergence of MI research, there is a rising need for field-specific tools. Libraries built for interventions and visualizations have made exploratory analysis more approachable and convenient for a larger audience. It is also of note that many of these libraries have complementary Jupyter notebooks with detailed demonstrations of their more common use.

In our work with decoder models, we have utilized TransformerLens (Nanda & Bloom, 2022) in conjunction with CircuitsVis (Cooney & Nanda, 2023). They are created to work well together, with transferable models and architecture. TransformerLens provides an easy-to-access architecture for recovering and changing activations within the decoder model. This lays the groundwork for performing techniques such as ablation, activation patching or logit lens. CircuitsVis is, like the name suggests, a package containing visualization tools for finding circuits. It has easy-to-use functions for showing attention patterns, as well as performance and contributions of each token in a sequence with regards to the final prediction.

For encoder models, we utilized the “transformers” library from Hugging Face (Hugging Face, n.d.-a) directly. This is one of the primary machine learning libraries for accessing, analyzing and training a large collection of pre-trained transformer models. While TransformerLens is mostly adapted to decoder-only models with causal attention, the original transformers library is more robust and can be used for other NLP tasks, as well as for dealing with some multimodal models. In particular, this library was useful for our purposes, which required us to be able to access and

[3] See section 5.1

modify internal activations in encoder embedding models. Once a model had been initialized, we could use the “encoder”-method to get access to the model architecture. That allowed us to grab hold of the specific methods in each layer that are called during a forward pass, which we could use to cache and modify the activations.

While working with these libraries, we encountered some difficulties, most prominently regarding the documentation. While the documentation of TransformerLens is quite extensive (it includes, for instance, many code examples), it is missing some very useful information about the outputs of many functions. It routinely omits information about the dimensions of the outputted tensors or what, exactly, they contain. This led to many frustrating error messages in our coding process, as we were unsure what data we were working with. For Captum, the situation is similar, with extensive demo-notebooks but lacking explicit details in the documentation on tensor shapes for function inputs and outputs.

3 Theory

3.1 Transformer MI

3.1.1 Encoders and decoders

Transformer models typically fall into one of two categories, namely encoders and decoders (other similar distinctions are “BERT-style” vs. “GPT-style” transformers or “contextual transformers” vs. “next-token predictors”). Conceptually, an encoder should aim to *understand* a chunk of text, while a decoder should *predict* the next token to follow a chunk of text. In practice, the difference lies entirely in the attention masking: For an encoder, there is no masking[4], while the decoder has *causal masking,* that is, the attention layers can only look backwards. Put simply, the next-token predictor cannot use the next token to predict itself. Notice that this means that the decoders can be rerun on their own output to generate multi-token responses, which is quite different from the single run per chunk use of an encoder.

It is a common imprecise simplification that encoders output embedding vectors while decoders output tokens. In fact, all standard BERT-models will be pretrained with Masked Language

[4] The exception is masking padding tokens when using multiple batches.

Modeling (MLM), which involves outputting tokens. The model output depends only on the output layer of the model, which is only a superficial part, often changed during training and especially for fine-tuning. In our project, we use encoders to answer multiple choice questions, which is a classification task requiring an understanding of the entirety of the input. Hence classification is a *contextual one-off task* where non-causal attention makes sense, and so the encoder is most appropriate, even though the output is not an embedding vector.

Thus, the difference between encoders and decoder architecture is small, and most functions written for either should be applicable to both with minor modifications, perhaps except for attention specific visualizations. The reason that interpretability research has mostly focused on next-token predictors[5] is likely that these are, currently, the most impressive, and the output from these models is *inherently human interpretable.* For embedding models, this is not the case, and metrics must be chosen carefully to avoid misinterpretation caused by models using algorithms going against our intuition of how tasks should be done.

3.1.2 Tangent on linear feature representation and superposition

In the first postulate from Olah, et al., 2021, it is assumed that features "correspond to directions in activation space". This is a postulate on *how the model represents meaning*, which is supported by the apparent vector space structure of semantic space and, more fundamentally, by the linear nature of most model operations. Note that while there are (infinitely) many more directions than dimensions in activation space, we expect the model to minimize interference between features, and hence keep them mostly orthogonal (unless the features never co-occur in the training data). The directions corresponding to features thus form an over-complete basis which is likely "almost orthogonal" – formally, one can show that for any allowed deviation ε from orthogonality the number of such vectors grows exponentially with the dimensionality n, which ensures that a high dimensional model will always be able to fit all the features it wants with minimal interference (see for instance the Johnson-Lindenstrauss' lemma).

[5] There are exceptions, of course; see Bertology, for instance.

3.2 Interpreting attention

3.2.1 Attention patterns

First introduced in "Attention is all you need" (Vaswani et al., 2017) the attention layers of a transformer are largely responsible for moving information between different tokens in a sequence. Therefore, knowing which heads are active for a given prompt is important for understanding the in-context learning that a model does, as well as what information a prediction is based on. A single attention head will compute an attention score for each pair of tokens. These attention scores show us what connections the model deems important. Much of attention-specific MI work is based on comparing these scores for different pairs of tokens or at different layers in the network. Plotting these scores as attention patterns gives us an easy way to spot these underlying patterns. Some common patterns are seen in Figure 2. Attention head 1 emits only the diagonal, indicating that each token only attends to itself. Therefore, no information is shared between the positions at this head. For Head 0, however, all tokens attend to the first token position, corresponding to a Beginning-of-sequence token. For a more complete overview of attention heads and its implications, we refer to "A Mathematical Framework for Transformer Circuits" (Elhage et al., 2021).

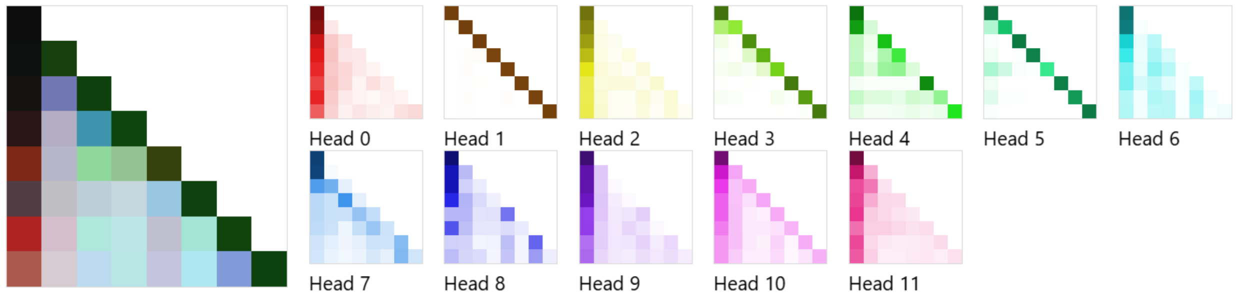


Figure 2: The attention patterns for Layer 0 for the prompt "The capital city of France is". The pattern to the left is the average of all the attention heads in the layer.

3.2.2 Induction heads

Induction heads are among the simplest circuits found in neural networks. An induction head is an attention head that detects the repetition in a sequence. For a present token, if it has occurred previously, the induction head will then copy *the next token* in the sequence. This way, the model can use prior information more efficiently and predict the next token more confidently (Elhage et al, 2021). If one were to visualize the attention patterns like mentioned in Section 3.2.1, the induction heads would be clearly visible, with the diagonal being "offset" by half of a sequence. An example of this is shown in Section 4.3.1. We also refer to "In-context Learning and Induction Heads" (Olsson et al., 2022) for a run-through of how induction heads work in different types of models and how this might scale to larger models.

3.3 Logit lens

Introduced by the user *nostalgebraist* on the blog LessWrong in 2020, the Logit Lens is a method for observing how the model prediction gradually converges toward the end prediction. By retrieving the intermediate activations from a given layer and converting them back to the corresponding logits, we get a view of what the model would have predicted at that point. An example of this method being used is shown in Section 4.3.3. It is also of note that the original method is primarily suited for smaller models like GPT2, as in our implementation. There has, however, been done work on expanding this method to larger models. One example is the open-source toolkit LogitLens4LLMs (Wang, 2025) designed for modern models.

We were also interested in applying a similar method for encoder embedding models that output embedding vectors of input text. The idea is very similar; we take the intermediate vectors of activation values from a given transformer layer and pass them into the last layer of the transformer model that computes the final embedding vector. Thus, one finds the output that the model would give if the intermediate layer was the last one. One can compute how these intermediate embedding vectors of the text input change for different intermediate layers to find how semantic meaning is being encoded in the final embedding vector.

3.4 Interventions

3.4.1 Ablation

Ablation is the simplest type of intervention. One of the main types is called *zero ablation* (Nanda, 2024), where we set certain activations equal to zero and analyze how that affects the model output. We can quantify the effect in several ways: With a next-token predictor for question-answering we can compute the logit difference between the correct answer before and after ablation, while we with an embedding model can compute the change in distance between the embedded question and the answer, with and without ablation. If the change in output is large, it tells us that the corresponding activations were important for the output. Thus, this intervention technique can be useful for identifying relevant nodes in a model or circuit.

Other types of ablation include *mean ablation* (Nanda, 2024) and *resampling ablation* (LawrenceC et al., 2022). With mean ablation, we replace the activations on certain nodes with the *average* of the activations from a bigger reference set of text prompts, while resampling ablation is based on replacing the activations of a node with its corresponding activations from a randomly selected text in a reference set. One motivation behind these alternative types of ablation is that there might be a common bias term in the activations, and so setting these activations equal to zero may have unintended consequences (Nanda, 2024; Wang et al., 2022).

To generalize zero ablation, we may multiply the activations by a nonzero number. This version of "soft ablation" has been implemented in the Circuit Tracing tutorial from Neuronpedia (Hanna, 2025). We can, for example, multiply by a negative number, to ideally *reverse* the effect of the relevant node. By analyzing how the output changes, we learn about the effect that the original node had on the output.

3.4.2 Activation patching

Activation patching is used to isolate the causal effects of chosen features in a text prompt on the model output. In this case, we replace activations on a given node from a forward pass on one text prompt with the corresponding activations from a different prompt. More specifically, we need to specify a *clean prompt* and *corrupted prompt.* The clean prompt is the original text prompt containing the features we wish to analyze. The corrupted prompt is another prompt which differs from the clean prompt by not containing the relevant features.

There are two main types of activation patching, namely *denoising* and *noising* (Heimersheim & Nanda, 2024). The denoising activation patching procedure works as follows: We do a *forward pass* on the clean prompt, and *cache* the relevant activations. That is, we send the clean prompt as input to the model and register the activation values. Then, we do a forward pass on the corrupted prompt, except that we have substituted (or *patched in*) the cached activations from the clean prompt. We can then analyze the difference in output on the patched corrupted prompt and the original clean prompt. The noising procedure works the same, only switching the clean and corrupted prompt.

Usually, we would have to make a choice of metric to compare the outputs. In the case of decoder models that predict the next token, a natural choice is the logit difference between the clean and patched corrupted prompt on a specific token. Among other places, this type of analysis is explored in the "Interpretability in the Wild"-paper (Wang et al., 2022), and a related notebook tutorial from TransformerLens (Meyer & Nanda, 2024). In the case where we use encoder models that output an embedding vector of the input text, one can for instance analyze either the l^2 or cosine distance between the clean and patched, corrupted prompt. If the distance after activation patching is very small compared to the prior distance between the embedding vectors of the clean and corrupted prompt, then this tells us that the relevant activations were important for representing the relevant feature that differentiated the two prompts. Thus, we can learn how and what information is stored in the specific activations in the network.

There is a lot of literature on this intervention technique. A summary was written by Neel Nanda and Stefan Heimersheim (Heimersheim & Nanda, 2024), which contains many useful tips on how to implement and interpret the results from activation patching.

3.4.3 Path patching

Path patching is a technique which generalizes activation patching, so that we can find how strongly a pair of nodes influence each other in the transformer model. Instead of patching in *all* activations on a *single* node (as in activation patching), we only patch in the activations on a so-called *receiver node* that have been mediated along a *direct path* from a previous *sender node*. With a suitable metric, we can measure how much the output of the model changes after patching in activations from a corrupt prompt onto a forward pass on a clean prompt. By evaluating the resulting changes on several combinations of sender and receiver nodes, we can quantify and compare the strength of the connection between any pair of nodes. This may be useful for finding how nodes couple together in a circuit.

The path patching algorithm was originally introduced in the so-called "Interpretability in the Wild"-paper (Wang et al., 2022). There, it was used for the task of identifying a circuit in the GPT-2 small decoder model implementing *indirect object identification*. In their implementation of the technique, they used logit difference as a metric for quantifying the effect of path patching. However, the algorithm can easily be used on other models (including encoder embedding models) with a different, suitable choice of metric.

The precise algorithm is described in appendix B of the paper (Wang et al., 2022), but a slightly more condensed version is also presented in a notebook by Callum McDougall (McDougall, 2023). One starts by specifying a sender and receiver node. In the "Interpretability in the Wild"-paper (Wang et al., 2022), the sender node is the output of some attention head and the receiver node is either the *input* to an attention head or the final output in the residual stream. Alternatively, we can allow the receiver node to be the output of an attention head (or perhaps the output of one or more MLP neurons in a layer). McDougall then describes two possible definitions of a direct path between the two nodes – it can either refer to a path through the transformer network that only passes along the residual stream, or one that does not pass through attention heads (but that can go through MLP layers). In order to quantify the strength of the connection between the nodes along the direct paths, we change the receiver node activations in the clean prompt to what it would be if the activations along the direct path were the same as on the corrupted prompt (but the activations along other paths are the same as on the clean prompt).

The three-step process, as given by McDougall, 2023, is as follows:

1. First, we cache all activations from a forward pass on the clean and corrupted prompt.

2. Then, we do a second forward pass on the clean prompt, with two caveats: the sender node activations are patched from the previously cached corrupted prompt activations, and all attention head activations between the sender and receiver node are patched using the cached activations from the clean prompt. If we do not want the direct paths to pass through MLP layers, we also freeze the MLP activations.

   We cache the receiver node activations on the second forward pass. This is the "heart" of the path patching algorithm: the receiver node activations could only change (compared to the first forward pass) due to the contributions that are mediated by the direct paths.

3. Finally, we do a third forward pass on the clean prompt, where we have patched the receiver node activations that were cached in step 2. With a chosen metric, we can quantify the change in output compared to the original forward pass on the clean prompt.

3.4.4 Causal scrubbing

Introduced in 2022, causal scrubbing (Chan, et al., 2022) is an ablation-based method aimed at testing a hypothesis more systematically. It involves defining a formal hypothesis about the network by looking at the transformer as a computational graph and the hypothesis as a subgraph. Then, by systematically changing all nodes that the hypothesis deems unimportant, we can test if this hypothesis indeed holds. In particular, we exchange activations based on equivalent values, which is explained below.

Following the work done by Redwood Research, we define a *hypothesis* h as a triple (G, I, c), where G is a computational graph representing the model architecture, I is a computational graph representing an "interpretation" (that is how we believe the model works) and $c : I \to G$ is a correspondence function. We here require c to be an injective graph homomorphism such that for any node in I, there is a corresponding node in G. An example of such a hypothesis is presented in Figure 3.

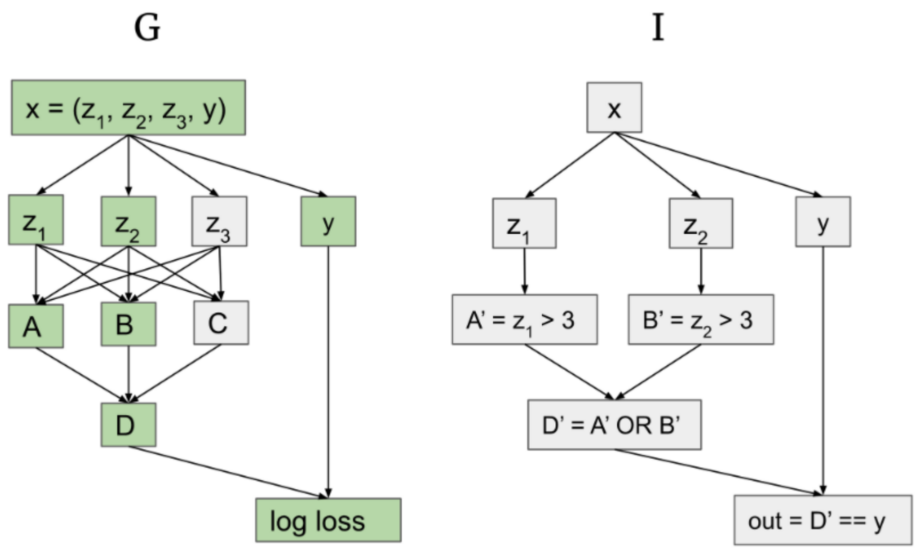


Figure 3: A graph representation of the hypothesis. The graph G is representing the entire computational graph, while the graph I represents the interpretation, that is, what components are important for the output. The image is from Chan et al., 2022.

These graphs are defined on a larger dataset D which contains all inputs to the graphs. Although the method is explained in the original article, we briefly mention the core components of the algorithm here. Given the assumptions above, causal scrubbing is performed as follows.

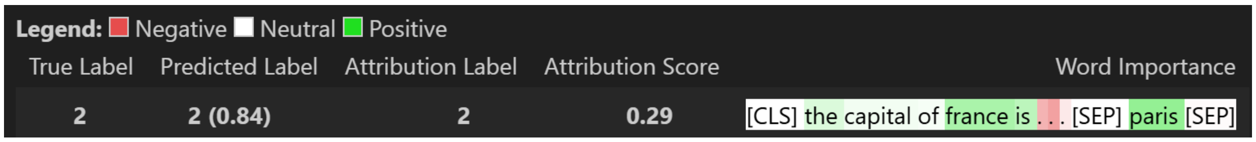
1. We first sample a random input x from the dataset D. We then

2. Then, we traverse all paths in I backwards from the output node. We do this by calling a function, "run_scrub", recursively. Given a node n_I, the function "run_scrub" will look for the corresponding node in the main graph $n_G = c(n_I)$ and consider all the nodes that feed into it. It will find the activations of n_G by looking at these "parent nodes", denoted by p_G.
3. Then, we have three possible cases for the parent node p_G
    a. If p_G is an input node, we return the input activation.
    b. If p_G is represented by a node in the interpretation graph (that is, there exists some node p_I such that $c(p_I) = p_G$), we will iterate one layer further into the network. Hence, we will call the "run_scrub" on the parent node p_G.
    c. If p_G is not represented in the interpretation graph, the function will deem this node unimportant for the model's output. Hence, we can choose an activation for this node from any other random sample input and by hypothesis, it should not change the output. We note that this random sample must be the same for all unimportant nodes, as it could otherwise throw the model off balance.
4. When all nodes have either been exchanged or checked, we perform a standard forward pass with all new activations. We compare this output with the original using some suitable loss function.

Iterating this process over multiple randomly sampled inputs, we measure the average loss change, thus obtaining a way to measure if the hypothesis holds.

3.5 Input attribution

The goal of input attribution is to determine how features in the model input contribute to the model prediction. For a CNN trained for image classification, this amounts to generating a saliency map indicating what pixels in the input image are most important for the image classification. For a transformer trained to answer multiple-choice questions, an example attribution is displayed in Figure 4.



Legend: 🟥 Negative ⬜ Neutral 🟩 Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
2	2 (0.84)	2	0.29	[CLS] the capital of france is . . . [SEP] paris [SEP]

Figure 4: Example input attribution visualization for a BERT model on a simple multiple-choice question.

To be clear, this is different from logit attribution, which seeks to explain what *parts of the model*, rather than what *parts of the input*, contribute to a given output. Input attribution only cares about the mathematical function implemented by the model, while logit attribution interacts with the specific model architecture. Hence logit attribution is more appropriate for typical MI research, focusing on the small model structures, while input attribution is appropriate for higher level analysis like finding model misconceptions. Indeed, our motivation for doing input attribution is mostly to potentially uncover model misconceptions on the FCI in a more quantitative way, though our experience indicates that the method might be inappropriate.

3.5.1 Integrated gradients

There are several axioms one might want an attribution method to satisfy. Integrated gradients is one attribution method seeking to satisfy (Sundararajan, Taly, & Yan, 2017):

i. Sensitivity: If two inputs differ only in a single feature yet gives different predictions, that feature should get a non-zero attribution.
ii. Implementation invariance: If two models have the same output for all inputs, they are said to be functionally equivalent, and the attribution should be identical for both.

The second axiom is mathematically motivated, since two functionally equivalent models trivially implement the same mathematical function. As discussed, it might still run counter to the spirit of MI, where the implementation is of essential interest; it essentially claims that two transformers producing the exact same logits must focus on the same features. This might be false, but the existence of two different transformers giving the same *logit output* is practically inconceivable either way, and we could still attribute with respect to single layers inside them to differentiate them. All this is to say that there might be better attribution methods than integrated gradients for input attribution – Integrated gradients is just one method with a decent library implementation in Captum.

The idea of the algorithm is to slowly change our input from a baseline to the real input, noting at each step how the logits, or some other continuous variable, changes with each feature. In mathematical terms, this amounts to a path integral of the model function gradient[6]. The baseline is typically a neutral input, such as a black screen for image classification or padding tokens for a transformer. The choice of baseline is often not obvious, and in the second computational essay, we give a discussion of different options on a multiple-choice dataset.

4 Applications

4.1 Circuit analysis tools for embedding models

4.1.1 Method

We implemented four main types of analysis techniques for embedding models, namely ablation, activation patching, path patching and a last logit lens-like technique. The methods were specifically implemented on the multi-qa-MiniLM-L6-cos-v1 sentence transformer model from Hugging Face. This is a relatively simple model (with roughly 22.7 million parameters) that was developed for asymmetric semantic search; in particular, it was trained on roughly 215 million question-answer pairs (Hugging Face, n.d.-b). The model outputs a single embedding vector of the input text. Thus, this model is well equipped to compare the semantic embedding vectors of questions and answers, which was the main use for our implementation. Although the methods were adapted to a specific embedding model, most of the functionality would be the same on other models, except for some important modifications mentioned in the discussion section (and elaborated on in the corresponding computational essay).

The intervention techniques were implemented in Python to allow maximum flexibility. One can perform the activations on any desired nodes in the network, including any attention head at any attention layer or any desired MLP neurons in any MLP layer. Moreover, one can provide the desired token positions whose activations one wishes to intervene.

[6] See Sundararajan, Taly, & Yan, 2017 for details.

We implemented the ablation intervention technique to allow one to specify the desired ablation type (zero, mean or resampling ablation), and to specify the desired nonzero factor α that the activations should be multiplied by in the case of soft ablation. Mean and resampling ablation were implemented by having the function taking in a list of text prompts. The activations of each text prompt were then ablated, with the whole list used as reference set. Note that mean and resampling ablation could be implemented in other ways as well. For instance, one may allow the researcher to provide a large reference dataset at the beginning, so that one could perform ablation experiments on a single text prompt by plucking activations from the specified dataset. Our method was used for ease of implementation, but it is rather trivial to expand the functionality so that the methods work for these alternative cases.

The activation patching technique was implemented according to the denoising algorithm.

For the path patching technique, we allowed bigger flexibility by generalizing the method a bit. In the original path patching algorithm, the sender node activations that are patched in for the clean prompt are cached from the corrupted prompt. In our implementation, we added the ability to also manually change sender node activations, without the use of a corrupted prompt. This can be done in two ways: one can either provide a nonzero factor α that the old sender node activations should be multiplied by, or one can manually provide all the new activation values explicitly. The point of this alternative technique is that one can use it to gain more control over how the sender node activations are modified. That allows more controlled experiments to test the strength of the connection between the sender and receiver nodes.

Our logit lens method plotted the embedding cosine distance between the intermediate embeddings of the desired text prompt and the final embedding of some specified reference text prompt, as a function of sublayer (i.e. attention layer or MLP layer) index. This allows us to track how the semantic embedding vector is nudged by each layer in the transformer. In this way, we learn how the model we are using embeds the input text, and the results could potentially be useful for identifying relevant layers in a circuit.

To visualize the intervention results, we quantified the effect of an intervention by computing the following quantity:

$$\frac{d(ref,\ i) - d(ref,\ f)}{d(ref,\ i)}. \tag{1}$$

Here, d represents some metric quantifying the distance between two embedding vectors. In our case, it is the cosine distance. The variable ref represents the embedding vector of a reference text, i refers to the initial embedding of some text prompt *prior* to any intervention on its activations. f refers to the final embedding of the prompt *after* we have done an intervention on its activations in one of the nodes in the network. In other words, we quantify the effect of the interventions by finding the *relative* change in distance between the reference text and the text prompt whose activations we modify.

The choice of ref , i and f for the three types of interventions were as follows:

- For path patching, ref represented the embedding of a question, i represented the embedding of the correct answer, and f was the embedding after having patched in activations from a *wrong* answer to the question. If the relative distance change is negative, it means the distance between the question and answer *increased* after path patching (which we might expect).
- For activation patching, ref represented the embedding of a clean prompt, and i corresponded to the embedding of a corrupted prompt. f then represented the embedding of the corrupted prompt after patching in activations from the clean one. If the relative distance change is positive, then the corrupted prompt got closer to the clean one after the activations were patched in, which we might expect.
- Finally, for ablation, ref represented the embedding of a question and i corresponded to the embedding of the correct answer. f was then the embedding of the answer after its activations were ablated. If the relative distance change is negative, then the distance between the question and answer embeddings increases after ablation, which we might expect.

The methods that we implemented were tested for different question-answer combinations. After trial and error applying the intervention techniques on different questions and answers, we ended up doing the main analysis on three "color sentences", i.e. sentences containing a "color word", like "red", "blue" or "yellow". We analyzed how the concept of color was represented in the model by performing intervention experiments. The corrupted prompts were formed by replacing the color word with a different word, and for the path patching and ablation experiments, we made a question that one of the color sentences was the answer to. A summary of the sentences used can be found in Table 1.

Table 1. Text prompts that were used to do ablation, activation patching and path patching experiments on the BERT encoder model multi-qa-MiniLM-L6-cos-v1. The main sentences were called "color sentences" (since they contain a color word), and they were numbered as shown.

	1	**2**	**3**
Color sentence/clean prompt	"The skies here are so blue"	"It is clearly red"	"I am looking at the yellow flowers over there"
Corrupted prompt	---	"It is clearly mine"	"I am looking at the majestic flowers over there"
Question	---	"What is the color of this house?"	---

Moreover, we imported 250 mathematics question-answer pairs from an online json file. We also used the first and third of the question-answer combinations, together with wrong answer options, from the Force Concept Inventory (FCI).

First, we visualized the activations in all attention heads and MLP nodes for the three color sentences. We did so by calculating the norm of the activation vectors at each attention head and token position, and then we averaged the result over all token positions. The MLP neurons were divided into eight batches or MLP nodes, and we computed the norm of the activation vectors in each batch. We compared these results with denoising activation patching experiments on every attention head and MLP node, for the second and third color sentences (with the corrupted sentences shown in table 1). We quantified the effect of each activation patching experiment using eq. 1. Next, we performed ablation experiments on the second color sentence by using the question shown in table 1. We checked both ordinary zero ablation, as well as soft ablation with $\alpha = -1$. We did ablation on every attention head and MLP node in the network, and visualized all the resulting values of the relative distance change from eq. 1. In the final part of the analysis on the color sentences, we performed path patching experiments on the second color sentence. We chose a relevant attention head as receiver node, and computed the relative distance change from all possible choices of sender nodes. We tested both the ordinary path patching algorithm, using the question, clean and corrupted second color prompts from table 1, as well as the soft version with $\alpha = -1$.

With the color sentences, we tested our intervention techniques on small, controlled changes to the sentences (in particular, by changing the single color word). In the final part of the analysis, we also applied the methods on the more complicated mathematics and physics questions and answers from the mathematics dataset and the FCI. The reason we do this is so that we can get a sense of what the results of the various analysis tools can be when there is a bigger difference between the prompts that we compare. First, we chose a random question-answer pair from the mathematics dataset. For all the attention heads in the network and the eight MLP nodes in each MLP layer, we visualized averaged activations in the answer and calculated relative distance changes between the answer and question due to soft ablation with $\alpha = -1$. Furthermore, we chose the answer to a *different* mathematics question and used it as corrupted prompt (with the original answer being the clean prompt) in order to do activation patching experiments on the activations in all nodes in the network. We compared these results to similar ablation and activation patching experiments on a question-answer pair from the FCI, except that we used one of the wrong answer options to the *same* question as the corrupted prompt. The idea is that one might get more interesting activation patching results when the answer corresponds to the same question, since much of the semantic meaning in the two answers will be the same.

Using the mathematics dataset, we could also test the mean and resampling ablation techniques. We used a text prompt dataset consisting of 50 of the mathematics answers and performed ablation on all the attention heads in the last attention layer. To quantify the effect of these ablations, we computed the percentage of questions and corresponding answers in the text dataset whose embedding vectors were *nearest neighbors* before and after the ablation experiments on the answer activations. By "nearest neighbors", we mean that compared to the embedding vectors of all 50 answers and their corresponding questions, the embedding vector of the answer was the vector closest in distance to the embedding of its corresponding question. If the question and answer are close in semantic meaning (which they should be prior to ablation), then we might expect most questions and answers to be closest neighbors. This might change, however, after the answer embeddings have been modified due to ablation.

Finally, we tested the “logit lens”-method. We visualized how the distance between the intermediate embedding vectors of five of the answers and the final embedding vector of the corresponding questions in the mathematics dataset changed as a function of sublayer index. We compared these results to the case when we instead used the intermediate embeddings of the answers to *different* questions in the dataset. Lastly, we tested the case of visualizing the embedding cosine distance between the final embedding of a question in the FCI and the intermediate embeddings of all the answers, including the *correct* and the four *wrong* answers.

4.1.2 Results

Before we present the results of the concrete intervention experiments, we summarize some of the general findings after the various trial and error attempts using different question and answer text prompts. The main bullet points were the following:

- Some words/concepts seemed to fire the same attention head activations, *independent of their locations in the sentence*. Sometimes, a single word/concept fired a single attention head. For example, we attempted various physics answers and stumbled upon the fact that using the word "friction" in a sentence usually activated head H7L5 (i.e. attention head 7 at the last layer, namely layer 5). Other times, a single word activated several attention heads in the last attention layer. For instance, the word "gravity" activated both head H7L5 and head H4L5. However, the most striking example we stumbled upon was that of *color* - we found that almost all colors (whether it was represented by the words "green", "red", "blue" etc.) caused a strong activation in head H1L5. Thus, it appears that this attention head responded strongly to the *feature* color. Since this attention head appears in the last layer, this could perhaps indicate that it might represent the last node in a circuit that is involved in representing the semantic meaning of text involving color.

- Activation patching seemed to be really useful for highlighting such behavior in single words/concepts. In fact, we used activation patching to find the properties mentioned above in the first place. While it often worked to visualize the activations of the original prompt in order to find which attention heads or MLP neurons were most active, activation patching was more useful for extracting the source of these activations (as well as finding *exactly* which nodes the individual words/concept activated).

- It was also hard to interpret the relative distance changes (given by eq. 1) from interventions on the MLP nodes - those did not appear to be consistent between different types of interventions. This might be expected, as MLP neurons might process information about the detailed structures of a text and hence could be sensitive to the type of interventions on the activations of those texts. We did see one interesting result though, which was that some specific MLP nodes activated most strongly for many different text prompts with varying content. In particular, when the neurons were split into 8 batches or nodes, the N4L0 node was most active, while the nodes N1L2, N3L1 and N5L2 also had high activations. Although there were a lot of sentences with high activations in these neurons, it was not the case for all sentences - this could indicate that there is some subtle similarity between sentences that have these activations. On the other hand, it could also be that these MLP nodes do not have much meaning at all, and that they simply respond to generic text prompts. It would be interesting to investigate this further.

Next, we present the analysis results. We start by visualizing the activations from the color sentences in Table 1. See Figures 5, 6 and 7. As mentioned above, we see that for a variety of color sentences, the head H1L5 has the highest activations. The heatmaps below also clearly indicate that the color sentences we used are examples of sentences that lead to high activations in the MLP nodes mentioned above.

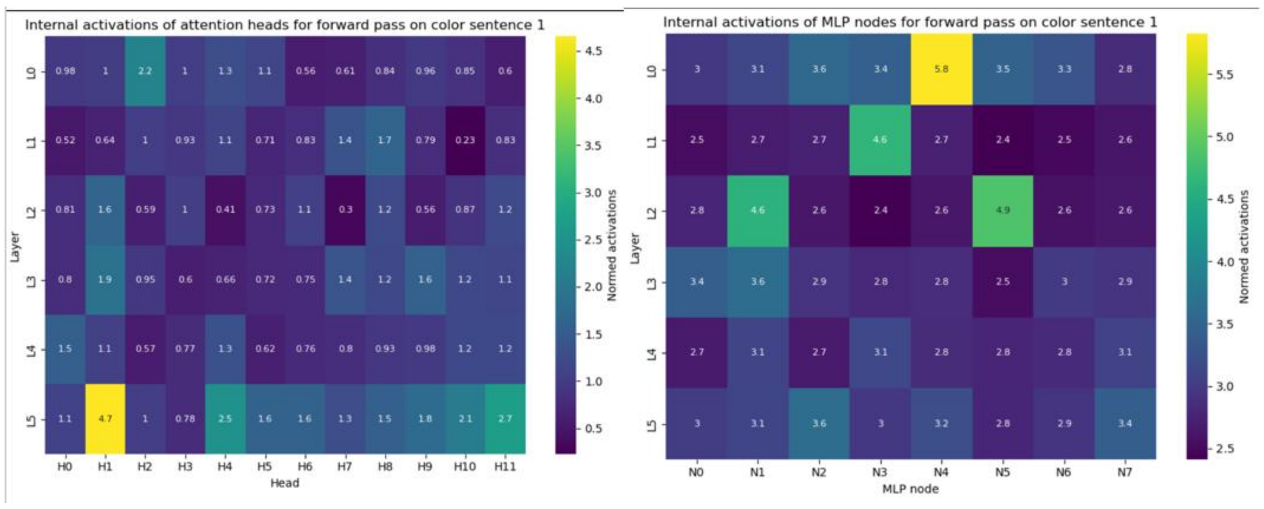


Figure 5: Average normed activations of attention heads and MLP nodes during a forward pass on color sentence 1 (given in table 1).

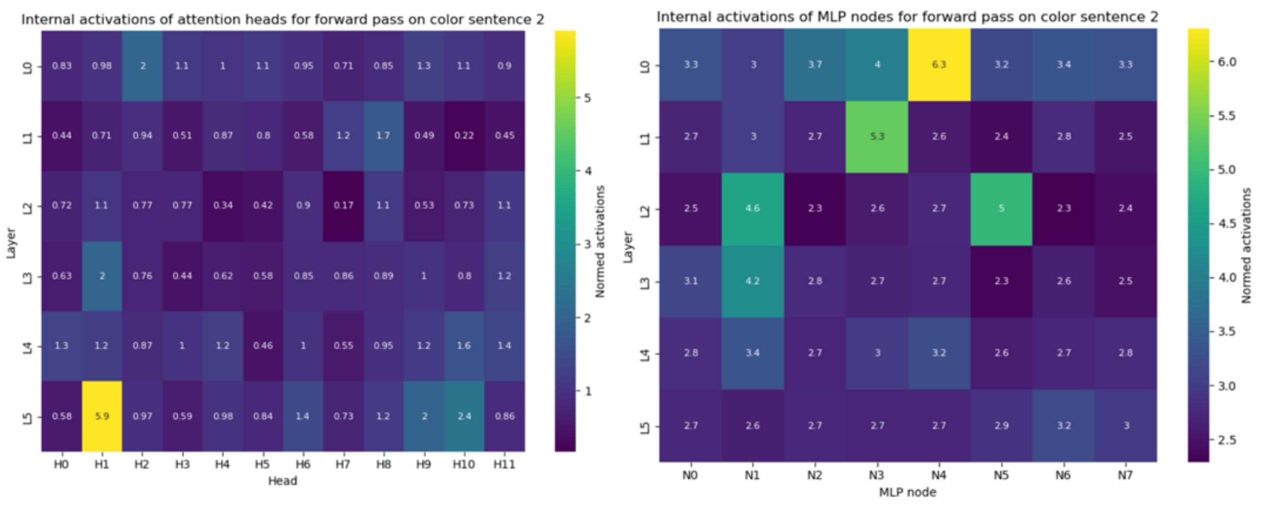


Figure 6: Average normed activations of attention heads and MLP nodes during a forward pass on color sentence 2 (given in table 1).

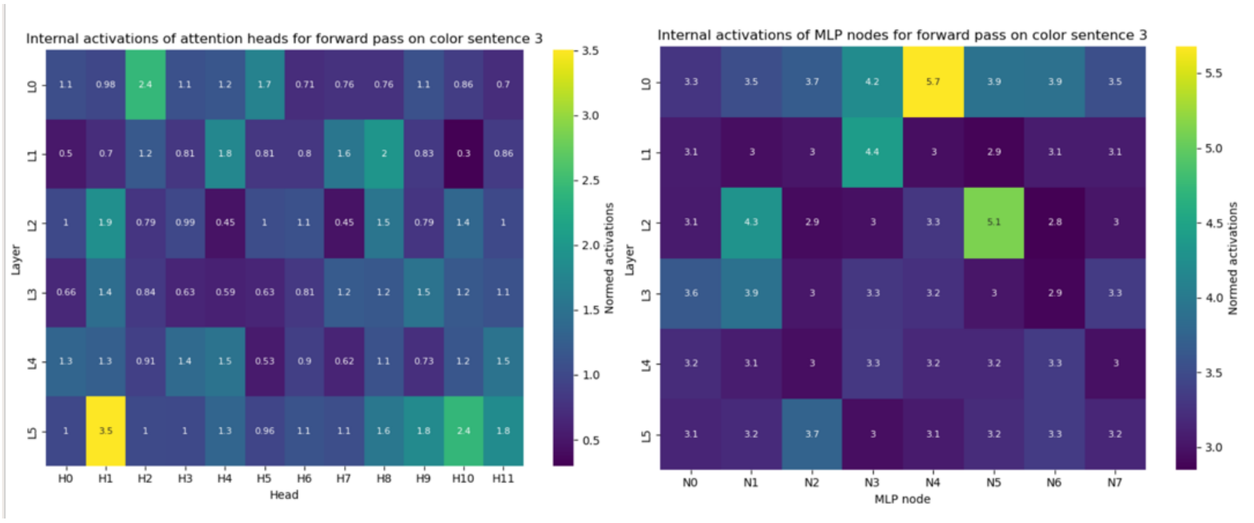


Figure 7: Average normed activations of attention heads and MLP nodes during a forward pass on color sentence 3 (given in table 1).

In Figures 8 and 9, we present the resulting relative cosine distance changes after performing activation patching experiments on the second and third color sentences. The corrupted sentences (made by replacing the color word) that were used are shown in Table 1. It is found that patching the activations of the head H1 at the last layer causes the distance between the clean and corrupted prompt to shrink appreciably. In fact, the distance shrinks by approximately 50 % (48 and 53 % in the two cases shown below), which is the majority of the original distance between the embedding vectors of the clean and corrupted color sentences - a clear sign that this attention head provides the majority of the representation of the color word.

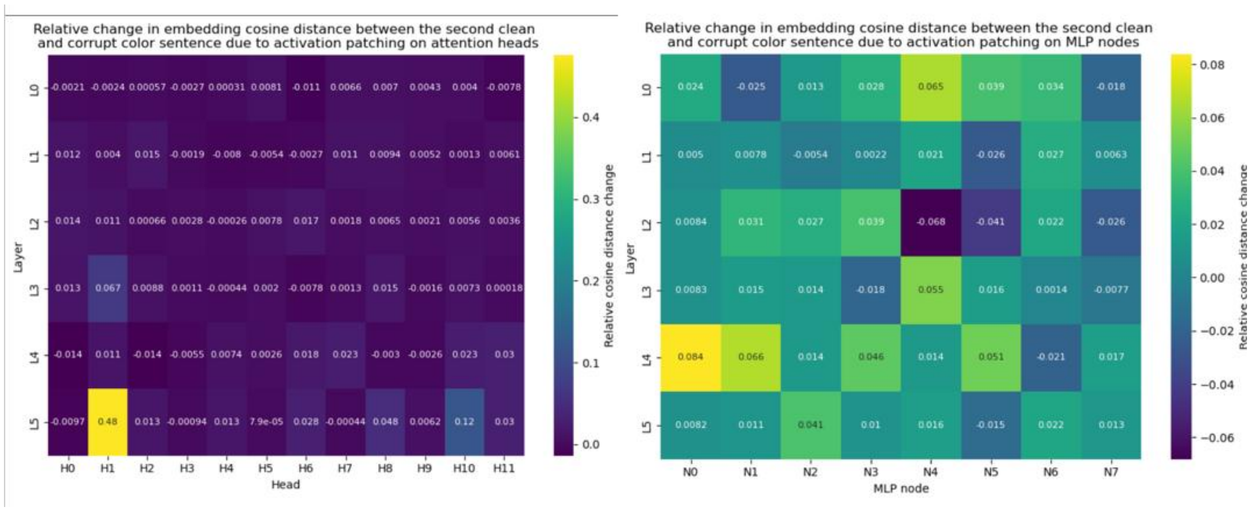


Figure 8: Relative cosine distance changes (given by eq. 1) between the second clean and corrupted color sentences in table 1 due to activation patching on attention heads and MLP nodes.

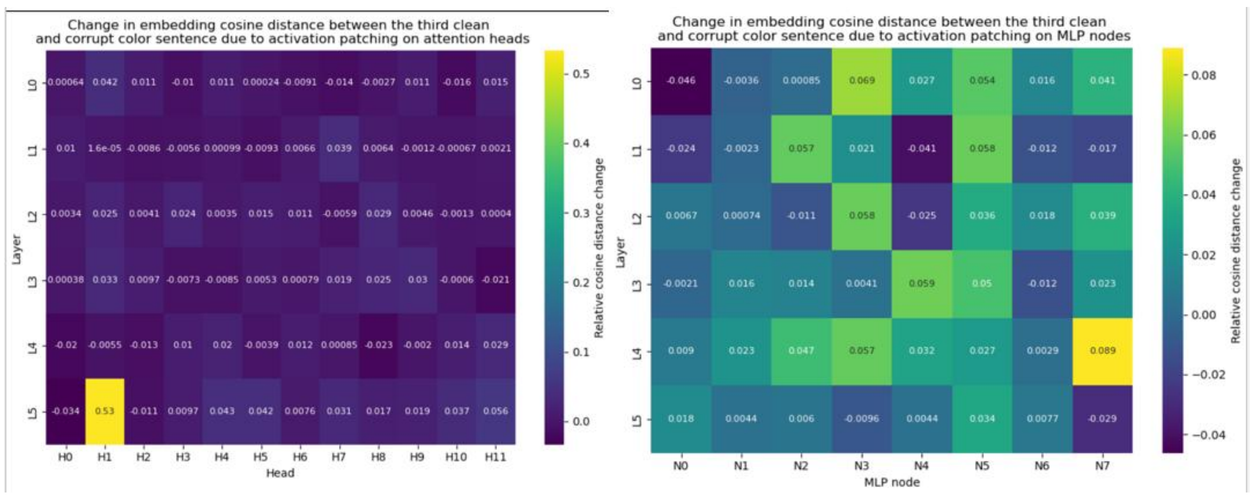


Figure 9: Relative cosine distance changes (given by eq. 1) between the third clean and corrupted color sentences in table 1 due to activation patching on attention heads and MLP nodes.

Note the relative distance changes due to activation patching on the MLP nodes. There appears to be no consistency between the two sentences in the pattern of relative distance changes for these nodes. Moreover, the pattern of relative distance changes due to activation patching on the MLP nodes seem to have no correlation with the internal activations of the clean sentences in Figures 6 and 7. This shows how MLP node results may sometimes be harder to interpret.

In Figure 10, we show the results of the ablation experiment on the attention head activations of the second color sentence in table 1. We show the results from both zero ablation ($\alpha = 0$) and soft ablation with $\alpha = -1$.

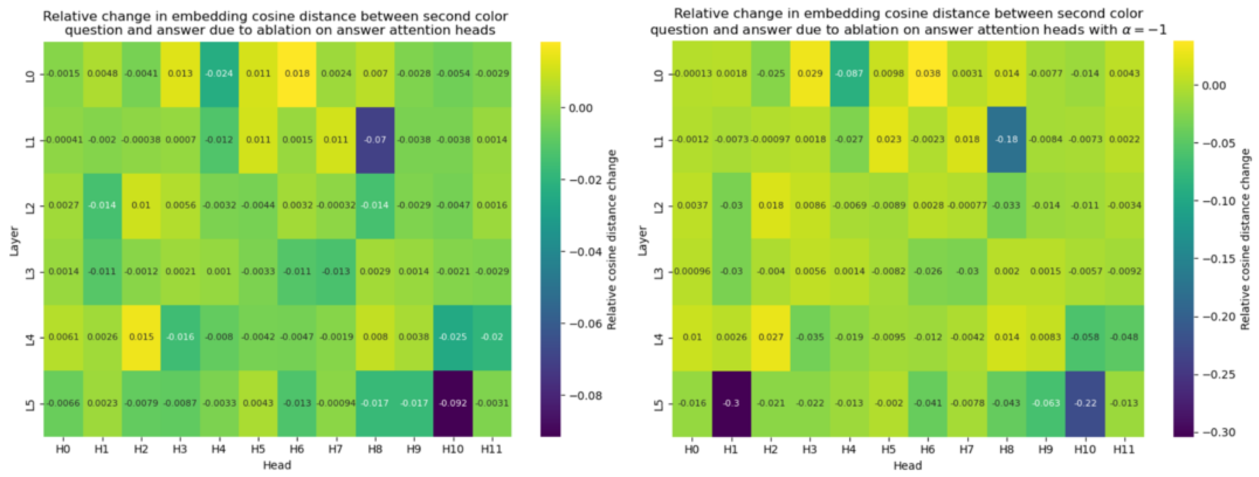


Figure 10: Relative distance changes between the second color sentence and question (from table 1) due to ablation on the attention heads of the second color sentence. The first plot shows the results due to zero ablation, and the second shows the results from soft ablation with $\alpha = -1$.

Notice that zero ablating the activations at attention head H1L5 has nearly no effect. It appears that ablating activations on attention head H10L5 leads to a much bigger increase in distance between the question and answer (the increase being roughly 9 %). Ablating attention head H8 at layer L1 also seems to cause a large increase in distance. Interestingly, these attention heads did not show up in the activation patching results.

Though the results are not shown here, it should also be mentioned that there were no obvious interesting features in the MLP ablation results, and once again, there was little correlation with the results in Figures 5 through 9.

In the soft ablation results in Figure 10, we do see that ablating attention head H1L5 causes the biggest increase in distance between the question and answer. Thus, the attention head that we may expect to see did show up in the soft ablation results. We got similar results when we tried other question and answer combinations where a single word (such as a color) in the answer sentence provided the answer to the question. Generically, it appeared that many extra attention heads (compared to the ones that showed up from activation patching or from visualizing activations) had a significant effect when performing zero ablation. When we used a negative α , however, we often got that the attention heads one would expect (from activation patching and activation visualization) had an important effect too.

Finally, we show the path patching results on the second color sentence in Figure 11. Previously, we found that attention head H1L5 could be significant for representing the color feature. Thus, we perform the path patching algorithm with head H1L5 as receiver node, and compute the relative distance changes with all the previous attention heads as sender nodes. This enables us to analyze which previous nodes it may connect to. We show the results from both the ordinary path patching algorithm, as well as our alternative algorithm, where we multiply the sender node activations by $\alpha = -1$ and perform the path patching algorithm with the modified sender node activations.

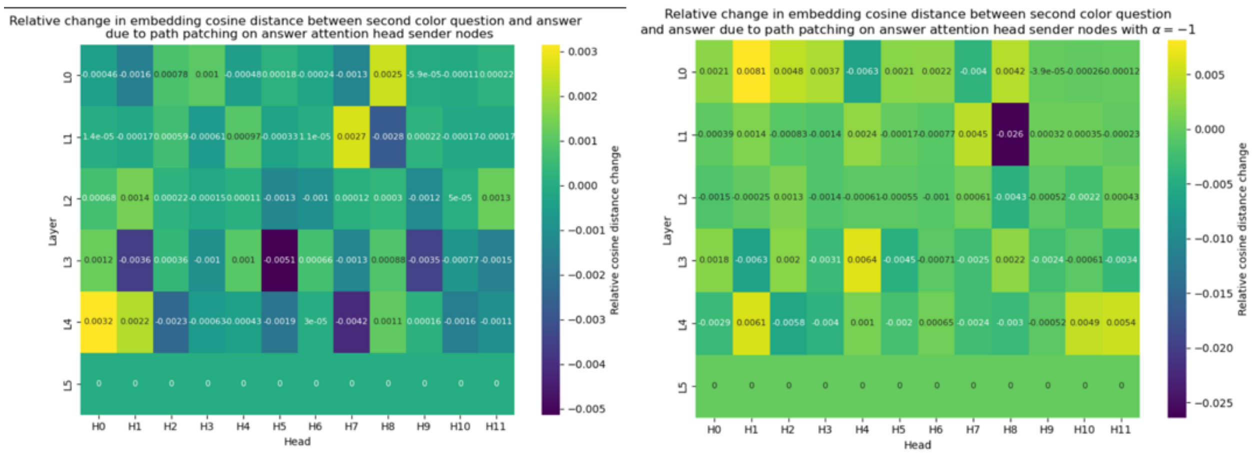


Figure 11: Relative cosine distance changes between the second color question and sentence (from table 1) due to path patching with attention head H1L5 as receiver node and the previous attention heads as sender nodes. The first figure shows the case of the ordinary path patching algorithm, with the corrupted second color sentence from table 1. The second figure shows the case of modifying the sender node activations with a factor of $\alpha = -1$.

We see that in the ordinary path patching algorithm, no nodes seem to stick out - whether we look at the attention heads or MLP neurons, it appears that the relative embedding cosine distance has similar order of magnitude for all nodes, which would imply that the weight of the direct paths from previous nodes are all quite similar. The behavior we see here was rather generic when I tried to do path patching for different combinations of question-answer pairs, corrupted anwers and receiver nodes. There were rarely any nodes with a particularly large change in relative embedding distance compared to the other nodes. This would seem to imply that there were hardly any noticeable circuits in the examples I tried (though that does not imply that no circuits would be found in a more careful analysis), or that the usual path patching algorithm may not be so useful.

We get a result from the alternative path patching algorithm that might be somewhat more interesting. We see that attention head H8 at layer L1 has a larger relative embedding distance (in absolute value) compared to other nodes. More precisely, its absolute value is roughly four times as large compared to the nodes with next highest values. Note also an important detail, which is that this node has a negative value; this means that the distance between the question and answer *increases* after path patching - this is what we would expect, since we patch activations in the receiver node *from* the corrupt answer *to* the clean one. In general, the sign of the relative distance change could be used as an indication of whether the corresponding value is just noise, or whether it might be an indication of a real connection between the sender and receiver nodes.

Although we will not show the results here, it should be noted that when we performed the soft path patching algorithm with head H10L5 as receiver node, we also found that we got the largest absolute value of the relative distance change with attention head H8L1 as sender node. More precisely, the absolute value was roughly twice as large compared to any other sender node. When we tested the path patching algorithm with the other attention heads in the last layer as receiver nodes, however, the head H8L1 typically did not give the largest relative distance change. All in all, our path patching results are consistent with the ablation results that indicated that attention head H8L1 was also important for encoding color information.

The point of the examples is not that they are very realistic - it might be that some of our results are just noise, rather than real relationships. But if one does path patching experiments, and it is found that certain sender and receiver nodes are strongly connected (due to causing a large change in embedding distance between the question and answer), then such results could serve as evidence that the two nodes are connected in a larger circuit. Our results above could serve as weak evidence that attention head H1L5 and attention head H8L1 are part of some "color circuit" that identifies text with "color words" and embeds their semantic meaning based on the presence of such words.

Lastly, we show the results from the physics and mathematics questions and answers. In Figure 12, 13 and 14, we show the results from activation visualization, as well as activation patching and ablation experiments on a mathematics question-answer pair. We used the third pair from the mathematics dataset. For the activation patching results in Figure 13, we used the answer to the fourth mathematics answer-question pair as the corrupted prompt. Thus, we see the results of activation patching in the case when the corrupted prompt is unrelated to the clean one.

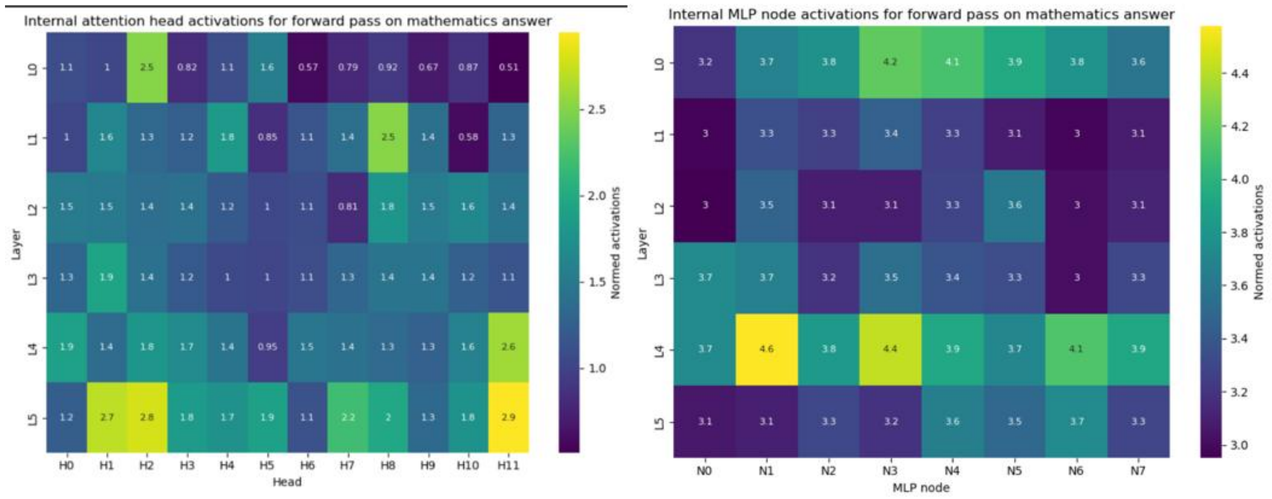


Figure 12: Internal activations from forward pass on the answer of the third mathematics answer in the mathematics dataset. The first figure shows the activations from the attention heads, and the second figure shows the activations from the MLP nodes.

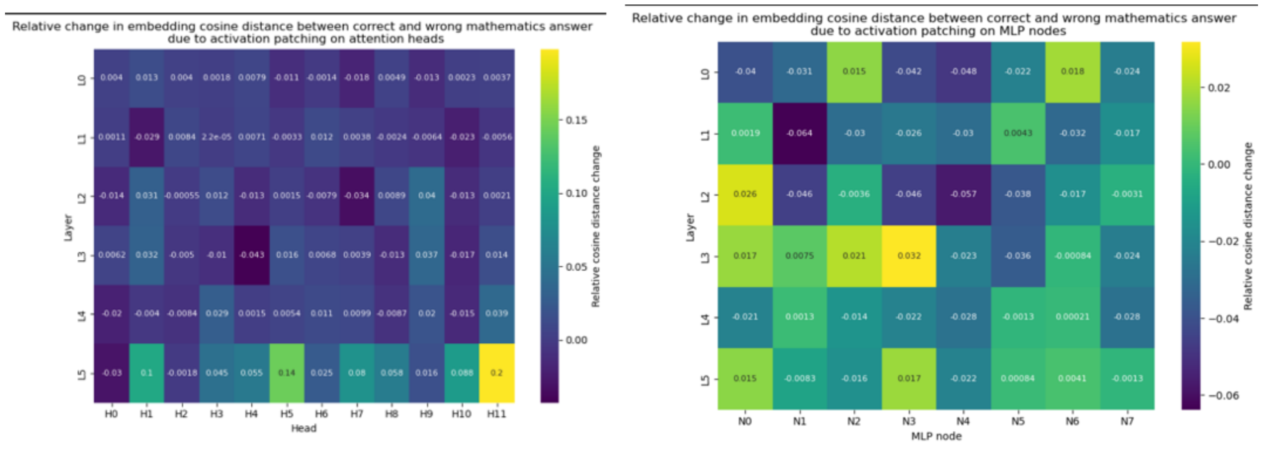


Figure 13: Relative cosine distance changes (given by eq. 1) between the correct and wrong answer to third mathematics question due to activation patching on attention heads (left figure) and MLP nodes (right figure).

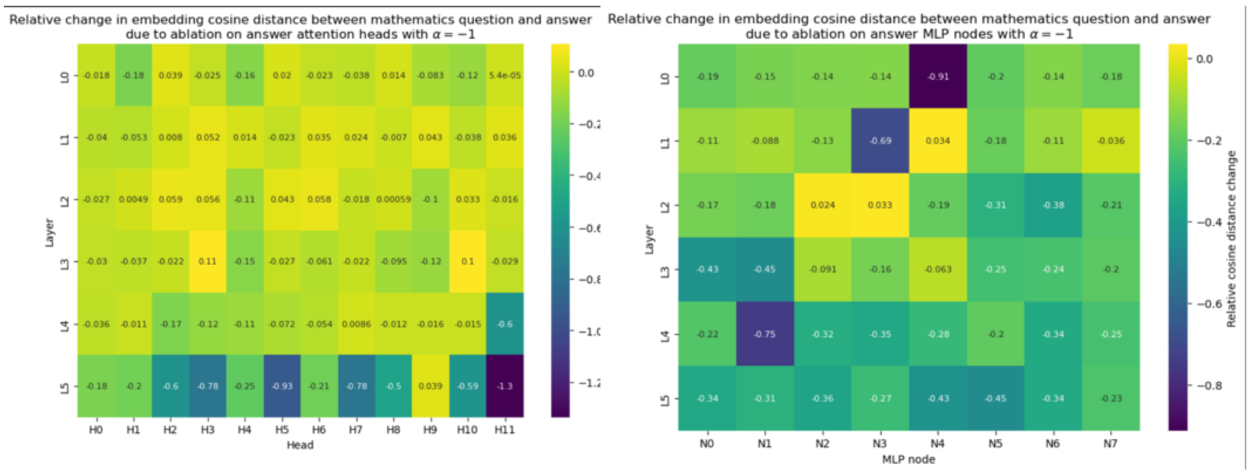


Figure 14: Relative distance changes (from eq. 1) between the question and answer of the third mathematics question-answer pair due to soft ablation on with $\alpha = -1$ on the attention heads (left figure) and MLP nodes (right figure) of the answer.

We see that with the more complicated prompts, there are more attention heads with relatively high activations. For instance, we see in Figure 12 that attention heads H2L0, H8L1, H11L4, as well as heads H1, H2 and H11 in layer L5 are most active. Not all of these attention heads show up during activation patching in Figure 13. But the most intense activation (namely, the head H11L5) nonetheless shows up with the biggest relative distance change of 0.2. This head is also associated with the biggest relative distance change after ablation in Figure 14 - in fact, after ablation on this attention head, the distance increases by more than 100 % (130 %, to be exact). Once again, we find that new attention heads show up during ablation (in this case, the heads H3, H5 and H7 at layer L5 give large relative distance changes, for example).

Thus, a similar general pattern as before emerges; when we do an activation patching on the most active attention heads, then the relative distance change becomes large. Ablation (with a negative α) also has a large effect on the most active heads. However, unlike the previous cases, we see a much bigger variety in the attention heads that give a significant contribution with the various analysis tools. New attention heads show up from the activation patching and ablation results, as we might expect (since the difference between the clean and corrupt text is much more than a difference of a single word).

As usual, we also find no correlation in MLP node values between the activations, activation patching results and ablation results in Figures 12-14.

In Figure 15, we show the corresponding analysis results with a physics-answer pair from the FCI. We only show the attention head results (as the MLP results were uninteresting, once again). We modified the questions and answers slightly to fit them for our purposes; in particular, we strived to adapt the answer options so that they had similar syntax, differing only in their main content. The question ended up being "What happens to a stone dropped from the roof of a single story building to the surface of the earth?”. The correct answer option that was used was “The stone speeds up because of an almost constant force of gravity acting upon it”. The wrong answer option used as the corrupted prompt in the activation patching experiments was “The stone falls because of the combined effects of the force of gravity pushing it downward and the force of the air pushing it downward”.

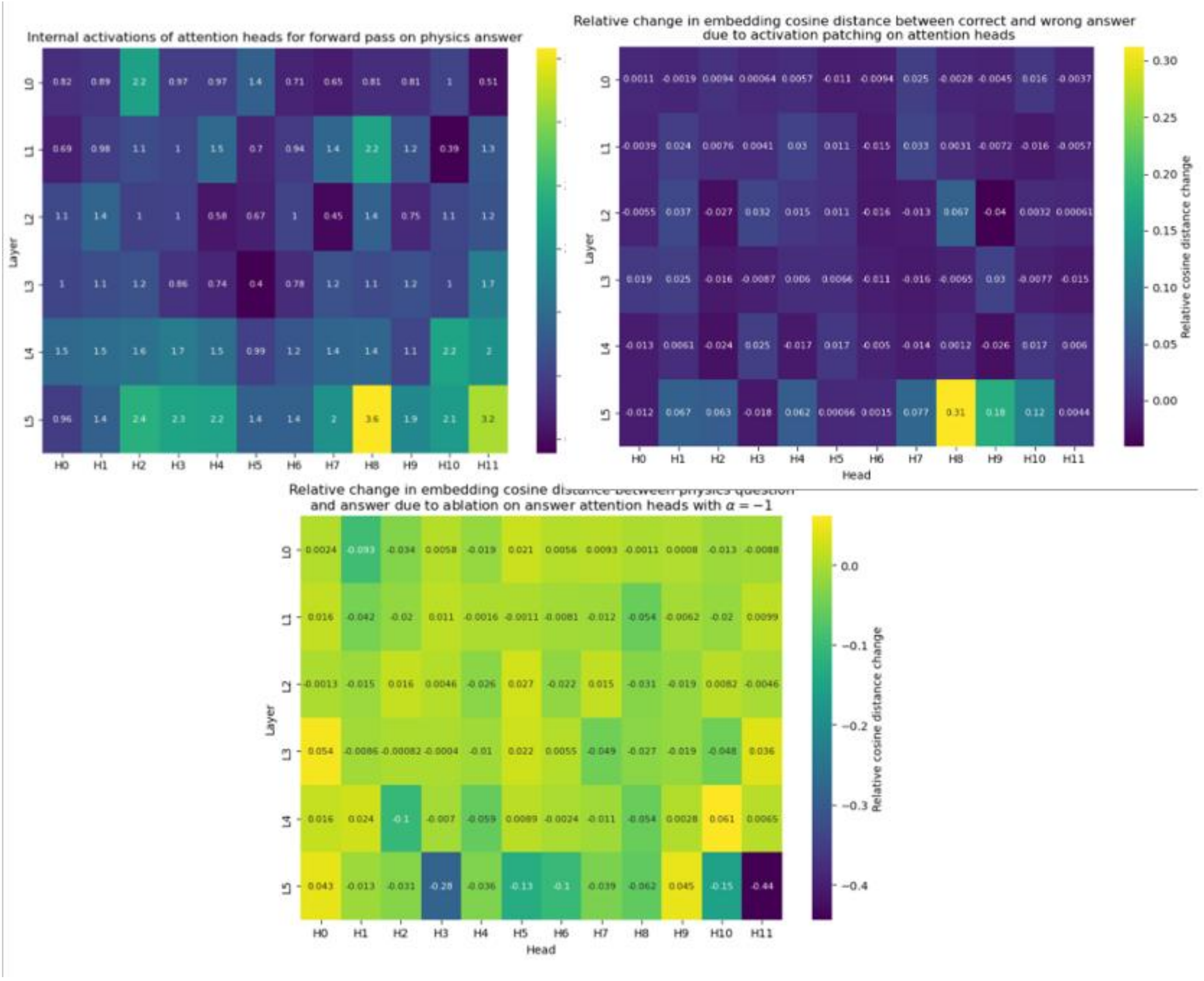


Figure 15: Analysis results from physics question-answer pair. Top left figure: Internal activations of attention heads from forward pass on the correct physics answer. Top right figure: Relative cosine distance changes (given by eq. 1) between the correct and

wrong answer option due to activation patching on attention heads. Bottom figure: Relative cosine distance changes (given by eq. 1) between the correct and wrong answer option due to soft ablation with $\alpha = -1$ on attention heads.

Again, it is found that a similar pattern as with the mathematics question-answer pair emerges. Figure 15 shows that activation patching has the strongest effect for head H8L5, and this was also the most active attention head in the correct answer. However, we see that the activation patching results are significant for a fewer number of attention heads compared to the mathematics question-answer pair. This may be expected on the grounds that there are fewer semantic differences between the two answer prompts in this case. Moreover, we find that the attention head H11L5 gives the biggest increase in distance between the question and answer after ablation, which again shows that using $\alpha = -1$ gives some consistency with the activation visualization method. But once again, many other attention heads show up in the ablation experiments. For instance, we find that attention head H3L5 has a disproportionally high value in the ablation experiment compared to the activation visualization results.

When we tested mean and resampling ablation on the mathematics questions, we found that resampling ablation was much more effective for reducing the percentage of question-answer pairs that were nearest neighbors. In particular, after performing a mean ablation on all attention heads in the last layer, we found that the percentage of the 50 mathematics question-answer pairs that were nearest neighbors was 60.7 %, compared to a percentage of 82.4 % without ablation. However, after performing a resampling ablation on the same attention heads, we found a percentage of a few percent (one of the values being 1.9 %) of mathematics question-answer pairs who were nearest neighbors. Thus, resampling ablation seemed to be far more effective for changing the semantic meaning of the answer prompts. A possible reason for this effect could be that the questions were quite varied, meaning that the difference in semantic meaning (and thus the activations at the last layer) of two arbitrary answer prompts were, on average, quite large.

Finally, we can present the encoder model logit lens results. In Figure 16, we plot the results. Here, we show the cosine distance between various questions and the intermediate embeddings of various answer prompts, as a function of sublayer. By sublayer, we refer to *either* an attention layer or MLP layer. Thus, the first attention layer is sublayer 0, the first MLP layer is sublayer 1 etc. Since the model had 6 transformer blocks, there are 12 sublayers in total. In the bottom plot in Figure 16, we used the first physics question from the FCI and all its answer options. That is to say, we used the question “Two metal balls are the same size but one weighs twice as much as the other. The balls are dropped from the roof of a single story building at the same instant of time. What is the time it takes the balls to reach the ground?”.

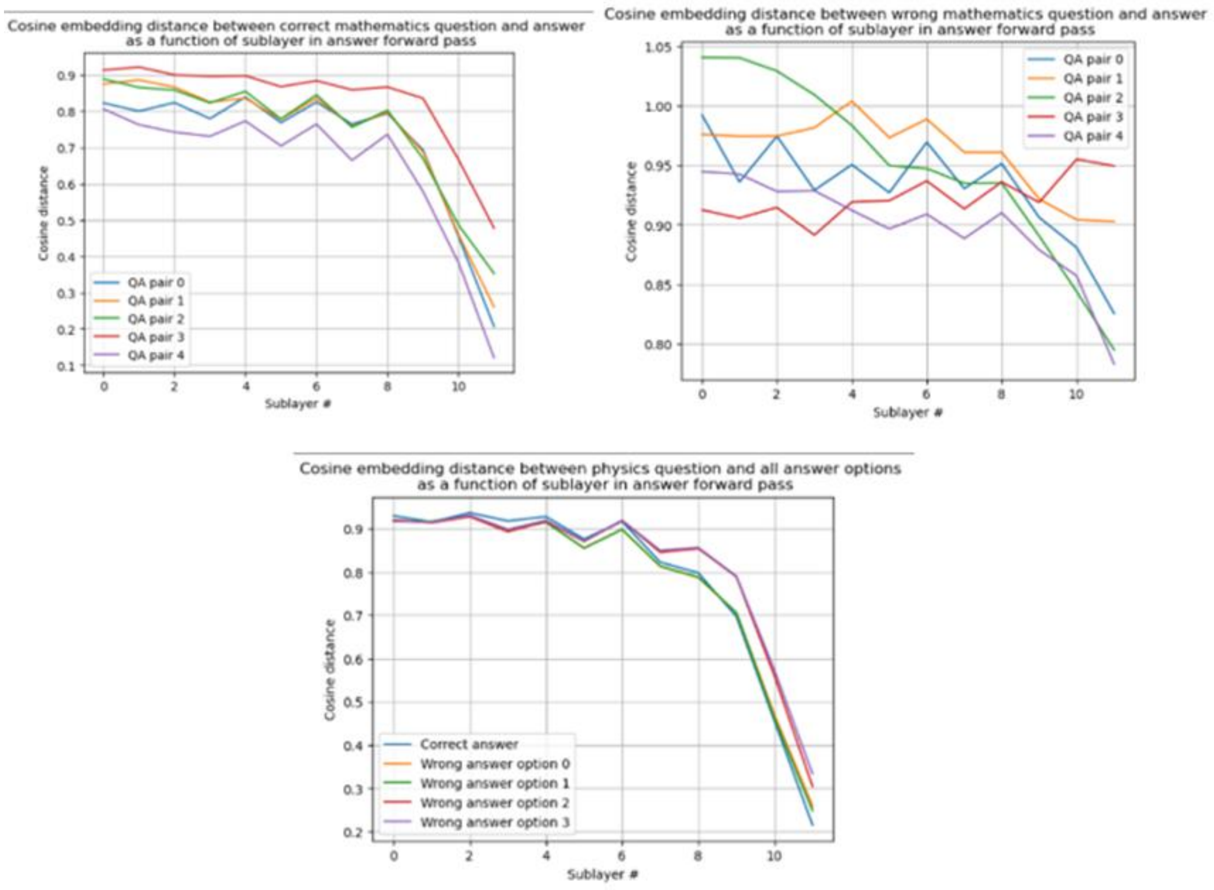


Figure 16: Logit lens for encoder embedding models. Here, we plot the cosine distance between question embeddings and intermediate embedding vectors of answers, as a function of sublayer. In the top left plot, we have picked five question-answer pairs from the mathematics dataset and compared the questions with the correspondingly right answers. In the top right plot, we have the same five questions, but the five answers are wrong (since they are answers to other questions). In the bottom plot, we compare the embedding of a physics question from the FCI with the intermediate embeddings of all the answer options, including the right and wrong answers.

In the top left plot in Figure 16, we see that when the questions and answers match, the intermediate distance consistently drops sharply at the two *last* transformer layers (corresponding to sublayers 8-11). However, when the questions and answers do not fit (as in the top right plot in Figure 16), the cosine distance varies more wildly as a function of sublayer number, without any sharp drop in the last transformer layers.

These results suggest that semantic meaning is encoded in the residual stream at the last two layers in the transformer. This is consistent with the fact that the most interesting attention layers in the previous analysis were often at the last layer. Moreover, this result has consequences for how we interpret early nodes, and thus any circuits that utilize them. In particular, early nodes may not directly correspond to some part of the semantic meaning of features in input texts. That makes interpreting them far more subtle. For example, while attention head H8L1 could have some important connection to the "color" attention head H1L5, it may not, by itself, represent the semantic meaning of color. This is broadly consistent with the fact that it was only attention head H8L1 that primarily distinguished the clean and corrupt color sentences in the activation patching experiments.

Lastly, we see that there is indeed a very tiny difference in distance between the different answer options for the FCI question. It seems the model does slightly better on the right answer option, since it started with the biggest distance to the question at the first layer and ended up with the smallest distance in the last layer. However, the differences between the different options are so small here that it seems the model does not have a high enough resolution to be able to distinguish different answer options clearly.

4.1.3 Limitations of encoder embedding model methods and suggestions for future study

There was a stark difference between the results from our experiments on color sentences on the one hand, and the mathematics and physics questions on the other hand. The activation patching experiments on the color sentences gave clean results, with only a single attention head with a significant relative distance change. This made it possible to reliably interpret the meaning of this attention head. The fact that we got such clean results enhances our confidence that our methods generally work as they should.

However, we were left off having to make more speculative conclusions from the experiments on the mathematics and physics questions. The highest relative distance changes that we got from the analysis results were typically only larger than the next highest ones by a factor of order unity. There were also often several attention heads that had roughly the same relative distance changes. When we tried different, but similar question-answer combinations, we could also sometimes get very different patterns in the relative distance changes. This means that it was much harder to reliably distinguish between interesting behavior and noise.

The contrast between these results should make it clear that in a more serious study, it will be important to have properly trained the model one is using for the specific questions and topics that one is interested in. The model we used appeared to behave poorly for questions about mathematics and physics. This was clear both from the poor results mentioned above, but also from the fact that the model could not clearly distinguish the correct and wrong answers in the last logit lens results. Moreover, it might be a good idea to use a larger model, so that it would have a higher resolution in its ability to reliably differentiate the semantic meaning in text. The logit lens results indicated that our model encoded semantic meaning in the residual stream only in the last two transformer layers, meaning that it did not utilize all the layers for this purpose. Moreover, the model we used had relatively few transformer layers and a low dimensionality in the embedding space. With a higher dimensionality, there would be more room for it to store semantic meaning, and with more layers, there would also be more capacity for the model to implement various circuits.

If you decide to use a different embedding model, then there is an important point to be made. In order to access and modify the activations of attention heads and MLP neurons, we had to access the methods that our particular embedding model calls every time it computes the activations. These methods were specific to our model, and so if you desire to use a different one, then you would have to find the corresponding methods in your model and use them instead. Moreover, our embedding model used mean pooling to compute the final embedding vector at the last layer. We implemented mean pooling in a function and used it to compute the final embedding. Thus, if the last transformer layer of your model uses a different algorithm, then you would have to replace the mean pooling function. Other than these replacements, our code should work for other embedding models as well.

We were not able to find reliable results for any of the intervention techniques on the MLP nodes. The patterns of activations and relative distance changes were rarely consistent for different techniques. This may suggest that our methods are not applicable to analyze MLPs. Thus, in a future study, it may be more appropriate to focus attention on the attention heads.

In our project, we mostly used activation patching and ablation to find relevant nodes in the transformer network, as well as for interpreting what they represented. This had to be done by trial and error by testing different prompts. Once we found such nodes, we could use path patching to quantify the strength of connection between a pair of them. However, finding circuits this way would be highly inefficient, nor would it let us quantify how reliable any candidate circuits might be. The next logical step would therefore be to implement *automatic* algorithms for finding circuits. We have already mentioned causal scrubbing as a highly relevant algorithm for quantifying the evidence of a candidate circuit. In the literature, researchers have also implemented some algorithms for finding circuits automatically. In Section 5.2, we mention *Automated Circuit Discovery* (Conmy et al. 2023), or ACDC, as a notable example of such a technique. In combination with activation patching, ablation and path patching, one would then have a powerful set of tools for finding and analyzing circuits. Once a circuit has been found through a combination of techniques like causal scrubbing and ACDC, one can then use activation patching, ablation and path patching in a similar way as we did in this project, to analyze the meaning of the nodes in the circuit and the connections between them.

We should also note another very important point: most research in the area of mechanistic interpretability is based on decoder-only models that output text (such as the gpt-2 model). Thus, if one wishes to use BERT models that output embedding vectors, then one has to appropriately modify the analysis techniques to fit such models. That is what we had to do to implement activation patching, ablation and path patching. This would also have to be done in order to implement the automatic circuit techniques.

4.2 Input Attribution with Captum on Encoders

While this project introduces two interesting problems for future study, the specific methods employed turn out to be somewhat of a dead end, and the results are disappointing and inconsistent. Hence, we refer to the second computational essay for details on methods and results, focusing instead on motivations and discussion.

4.2.1 Motivation and introduction

The overarching motivation behind this subproject was to develop tools which will highlight a model's physical misconceptions on the Force Concept Inventory. Specifically, we want to know which words in the question and answers are most important for the model prediction and compare with the foci of students who failed the same questions, whose misconceptions have been documented in earlier qualitative research. This might provide a hypothesis for a misconception in the model, which might be verified/falsified by modifying the question/answers and seeing how the attribution changes.

The essential tool, then, is an *input attribution method*, and we detailed our method of choice (Integrated Gradients) in section 3.5. In the accompanying project notebook, we showcase both layer-wise and full-model attribution with Integrated Gradients from Captum on an encoder fine-tuned for multiple choice. Layer-wise attributions could be useful for rough feature localization[7] – for example, it is sometimes claimed that facts are stored in the MLP-layers of the network (Sanderson, 2024; Nanda et al., 2023), in which case we might use layer-wise attribution and a relevant question ("What is the capital of France?") to find the correct layer ("The capital of France is Paris") – while full-model attribution may be more relevant to the overarching subproject goal of uncovering model misconceptions.

The second interesting application of input attribution could be to compare the embedding spaces of different models. It is particularly interesting to see whether the words in a sentence which matter most for the embedding placement are consistent across models, and further, if we were to fine-tune one model for increased resolution on physics related topics, could we use attribution to see increased sensitivity to physics related words? We demonstrate one approach for assigning a relative importance weight to each word in the text-embedding for its placement in space.

4.2.2 Discussion and difficulties

The first application turns out to be difficult for several reasons. Firstly, the models employed do not have any understanding of physics whatsoever, and are not trained for long physics questions like the FCI. We cannot compute the gradients for large models with long inputs on my laptop CPU, so this rules out this dataset immediately. However, the code should be applicable to any appropriate PyTorch model (potentially self-fine-tuned for the purpose) with minimal adjustments.

Still, even with an improved model, We are skeptical that one can expect progress to be made, since the results with simple models are highly unstable even on toy examples. Semantically trivial changes, like adding spaces, exchanging synonyms or reordering words causes sometimes

[7] This particular application is less promising, as there are much better methods for doing this, namely logit attribution.

dramatic changes. Additionally, the attributions often do not make sense, sometimes placing emphasis on separator tokens or common words or giving a negative overall attribution score for the chosen index. This might be due to a bad choice of baseline (though preliminary tests indicate that this is not the issue) or an implementation error. In short, the results are not trustworthy and cannot reasonably be regarded as more objective than a thorough qualitative analysis.

We suspect that this instability issue might persist for the second application, though we have not observed this directly as the embedding output is not interpretable and hence, we have no intuition for what the outputs *should* be. However, we would be a bit more optimistic here, as the attribution scores at least always seem to have the right sign, and common words usually get less weight.

Overall, we would advise against using Captum Integrated Gradients for Mechanistic Interpretability on Transformers, first and foremost because the reduction of a model to the mathematical function it represents disregards the model architecture, which runs counter to the paradigm of MI (the section on Integrated Gradients in the Report gives context on this point). This does not mean that input attribution is not interesting, nor that it cannot be used for probing LLM brains, but that it likely does not fall into the subfield of MI. Generally, more custom-made libraries like TransformerLens might show greater promise, as discussed in the following section.

4.3 Decoder models with TransformerLens

We implemented three main methods for interpretability of decoder models, namely induction scores, activation patching and logit lens. Here, the first and last methods are non-intervening, while the second is an intervention method[8]. We also unsuccessfully attempted to implement causal scrubbing for decoders. In this section, we will outline our method of implementation, as well as our findings.

While working with decoder models in TransformerLens, we chose to use the model GPT2. To speed up our computations, we often resorted to using GPT2-small, which is a smaller model with 124 million parameters. This model is included in the HookedTransformer subclass of TransformerLens, originally imported from HuggingFace, with all the necessary architecture

[8] See section 2 for an overview of the different types of methods.

already included. GPT2-small is a next-token predictor, giving us an interpretable answer to a given prompt. As a result, these are more accessible to early exploration and intuition than encoders, where we are reliant on a suitable metric. In the cases where this smaller model did not perform well enough to solve the tasks, we used the larger versions of GPT2 instead.

4.3.1 Induction heads

For our first implementation, we calculated induction scores from the attention scores of running a forward pass on a sequence with repeated tokens. For our implementation, we used the prompt “The capital city of France is called Paris. The capital city of France is called Paris.”. To compute the induction scores, we used the hook functionality of the TransformerLens package to access the intermediate attention scores. For each layer in the transformer, and each attention head in that layer, we retrieved the value for how much a token in the second half of the sequence attended to its corresponding twin in the first half. We then computed the average attention score of these values for each attention head. Hence, a high average signals that a head computes high attention scores for repeated token-pairs, indicating that it is an induction head.

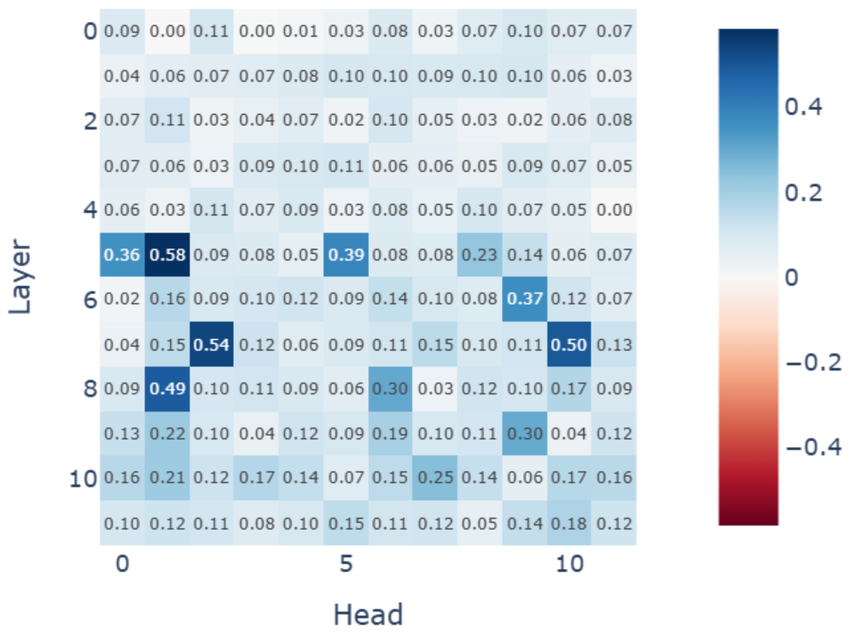


Figure 17: 2 Induction scores by head and layer. Note that these are zero-indexed.

As we can see from Figure 17, the average induction score of attention head 1 of layer 5 is higher than all other scores. With reference to Figure 18 we see that the diagonal for the second head is shifted down half a sequence. This means that a token strongly attends to another token if it is a repetition of itself. Contrasting this with attention head 6 of the same layer, the difference is clear. The induction score for this head is low and the corresponding attention pattern shows little deviance from the diagonal. Thus, all tokens only attend to themselves, and the head does not compute any new information.

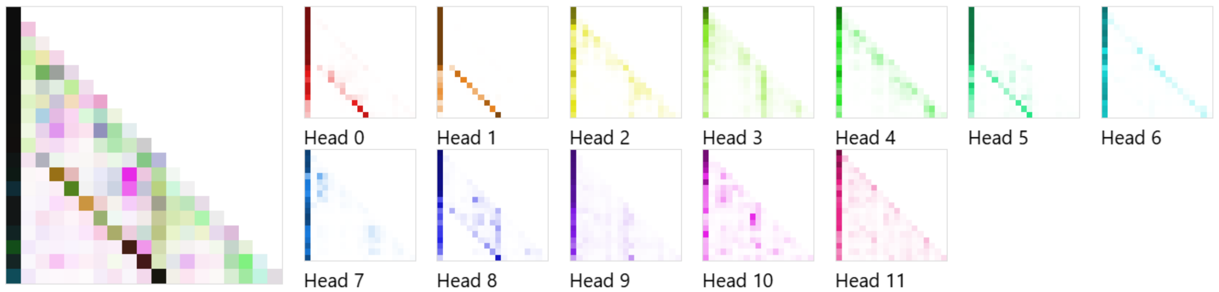


Figure 18:3 The attention patterns of the attention heads of the fifth layer of the model, with the average head pattern being shown on the top left. Note that these are zero-indexed.

4.3.2 Activation patching

After the induction scores, we implemented activation patching for decoder models[9]. This method closely follows the same algorithm as for encoder models explained in Section 4.1.1. The main difference lies in the output of the model and how we interpret the results. First, we chose suitable clean and corrupted prompts. We continued with the "Paris"-example (this time not repeated) and added a corrupted prompt. This gave us the prompts "The capital city of France is called" and "The capital city of Italy is called" with answers " Paris" and " Rome", respectively. We verified that the prompts had the same number of tokens and were spaced out in the same way. We also note the spaces in front of the answers. This is to adhere to the way the model tokenizes the words and ensures that both answers are a single token. We then run the clean prompt through a forward pass while we cache the activations. This was easily done using functions readily available in TransformerLens.

The next step was to patch the activations of the clean run onto the activations of a corrupted run. We patched them one at a time, and thus did not get any noise from prior patching. To patch an activation, we looped through all layers and token positions and then defined a hook function that

[9] We refer to https://colab.research.google.com/github/neelnanda-io/TransformerLens/blob/main/demos/Main_Demo.ipynb for further details.

exchanged the corrupted activations with the clean activations for that particular position in the model. To measure the change in the model's output, we calculated the logit difference between the corrupted run with and without patched activations. To calculate this logit difference, we find the difference between the clean and corrupted answer for both the unpatched corrupted run and the patched run. Then, we find their difference and normalize by difference between the clean and corrupted run.

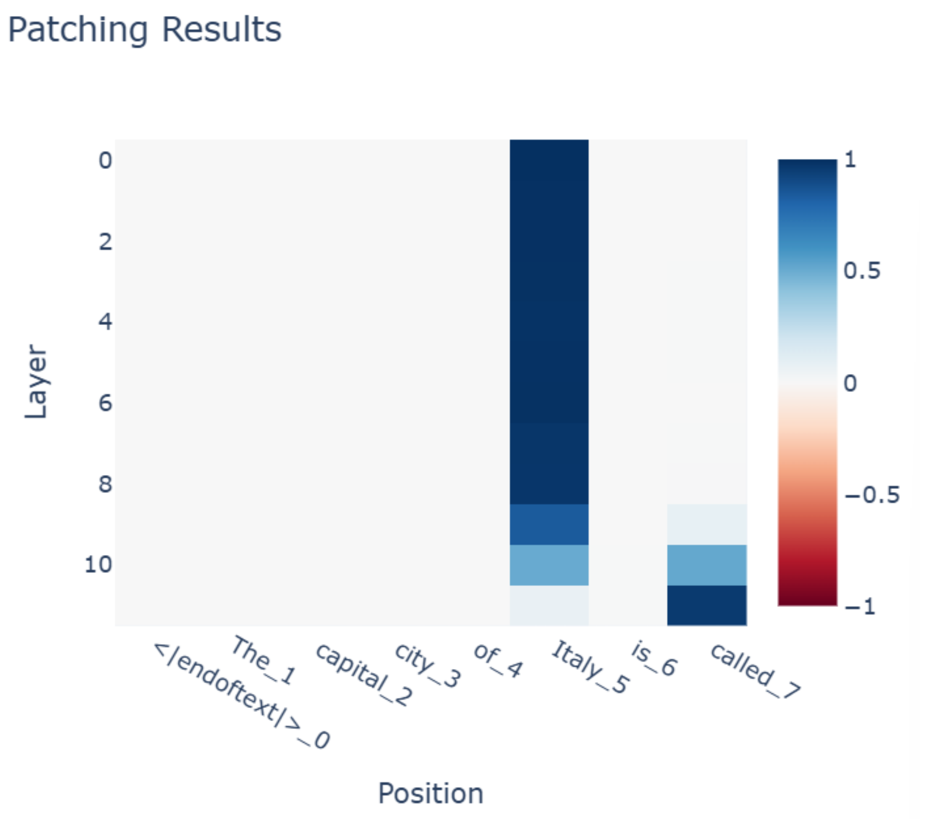


Figure 19: 4The calculated logit difference by layer and position. As the prompt is the same for the first positions, there is no measurable difference in output either. Then, when the change in prompt happens, the results are sudden and local.

The results of the given activation patching are plotted in Figure 19 From these results, we can see a few things. First, we note that since the prompts are equal for the first few tokens, the logit difference is zero. More importantly, however, we see that the transportation of information in the network is very localized. The information of the exact capital is not registered and passed on until around layer 9, where it then quickly corrupts the final prediction of the model.

While our study of this method only included this short example, this clearly shows the use of such a method. A thorough investigation of activation patching for a particular model involves generating a large set of similar clean and corrupted prompts, as well more computing power. The layer of information exchange in the example above would be a very interesting layer of further study. Performing a proper large-scale experiment might uncover other interesting layers as well.

4.3.3 Logit lens

The next method we implemented was logit lens, a technique used to see how a prediction changes through the model[10]. That is, if we were to stop a model run before the end, what would the prediction been then? This method consists mainly of retrieving the relevant information and converting this to interpretable tokens. There is, unlike activation patching, no intervention at any intermediate values.

First, we chose a suitable prompt for the task. We continued with the repeated prompt from Section 4.3.1, to see if there is any relation to the induction heads found in that section. We ran the prompt and retrieved the accumulated residual stack using methods found in the TransformerLens library. Then, we applied a layer norm to the values and multiplied the vectors by the unembedding matrix usually reserved for the end of the network. By computing the logit values of each position at each layer, we get the plot in Figure 20.

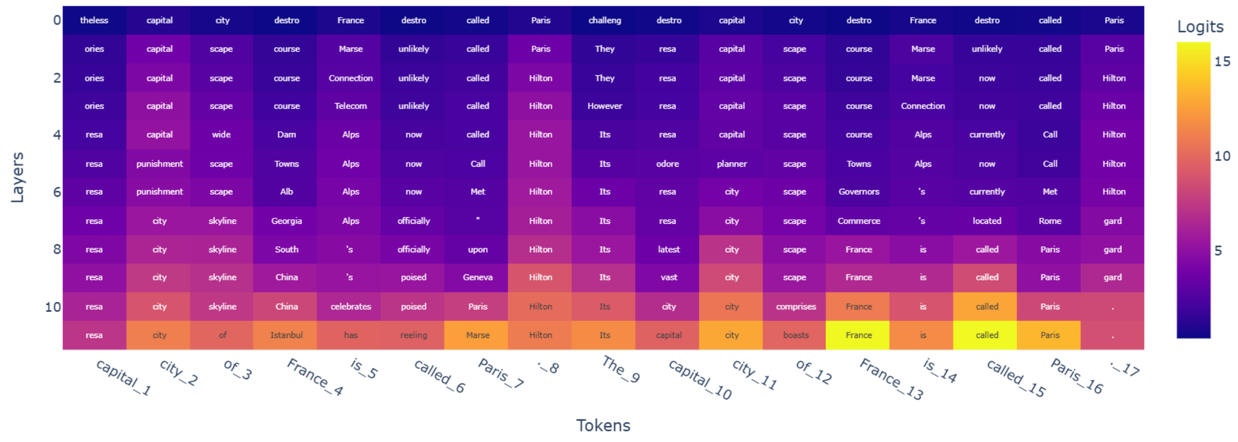


Figure 20: 5The results from Logit Lens. The cells show the top token for each position after a given number of layers. The color signifies the logit value for the intermediate "prediction", with higher values signaling that the model is more confident that it is correct. The labels at the bottom are the correct tokens that we wish the model to predict.

[10] We refer to https://github.com/TransformerLensOrg/TransformerLens/blob/main/demos/Exploratory_Analysis_Demo.ipynb for further details and other interesting methods.

From this, we can see that although the model is quite small, it gives sensible predictions in most cases. Some fun observations are the model's insistence on " Hilton" after " Paris" and "Theresa" instead of "The capital". This indicates that these sequences kept coming up in the training data, and that the model is too small to fully incorporate the context of the prompt into its predictions. However, more interestingly, the predictions get significantly better for the second half. That is, the predictions improve after the model has already seen a copy of the sequence. Importantly, this improvement occurs most in the layers which correspond to the induction heads found in Section 4.3.1. This further confirms the validify of those findings.

Also, we can see that the predictions are quite bad in the first few layers, until they suddenly improve in the last layers. This is similar to what we saw in Section 4.3.2, where the transferal of information was a local operation, with only a few layers contributing. We suggest that this method might be used for exploring which layers might be important in exchanging information between positions and as a starting point for further investigations.

4.3.4 Causal scrubbing

Lastly, we tried implementing causal scrubbing, but as mentioned in Section 2.2, we were unsuccessful. We are therefore brief in our exposition of this method and instead refer to Part 3 of the computational essay for more details on the implementation, as well as the original algorithm from Redwood Research (Chan et al., 2022).

Following the pseudocode provided, we decided to implement the concept of a single node and the hypothesis object as classes, as these need multiple methods and attributes each. We also tried implementing the "run_scrub" function, but this was also where the problems arose. To look at the nodes in the paths, we want to access multiple internal activation at the same time. That proved very difficult, as the available methods in TransformerLens get complicated when more than one hook is involved. As our code did not manage to access these activations, the scrub function also never worked properly. Although the results were not as we hoped for, the incomplete code is provided in Part 3 of our computational essay.

5 Discussion

5.1 Unsuccessful attempts

As mentioned in Section 2.2 and Section 4.3.4, we tried implementing causal scrubbing for a GPT2 model, but we were unable to do so. The algorithm (Chan et al., 2022) relies on recursively iterating through the network, referencing the hypothesis subgraph and collecting predetermined activation s from the forward passes. The architecture of a decoder model is quite complicated, and the list of possible helper functions from TransformerLens so extensive, that we did not manage to complete a working example from the pseudocode provided in the source. In particular, we struggled to access multiple hook points in the same forward pass. As TransformerLens contains many ways to access and change activations on hook points, this made it difficult to know which method to use. After much trial and error, we had yet to succeed by the end of the project. We still think causal scrubbing has potential as a method of verification of own work, and theory provides interesting insights into how one can imagine working with hypotheses in future work.

Furthermore, we tried using the CircuitTracer library (Hannah et al., 2025) to help locate and visualize circuits. This is a very promising library which offers a wide range of methods. However, while working with this library, we quickly discovered some major limitations. The main flaw of CircuitTracer is its lack of support for a diverse set of models. It exclusively supports two full scale sets of models, namely Gemma and Llama (Hannah et al., 2025). Even the smallest of these models, cause problems while running them in our systems. As locating circuits in a transformer model involves running through a model multiple times, as well as accessing and caching intermediate values in the system, it requires significant computational expenses. We also tried running experiments a Google Collab notebook provided by the creators of the library[11], but experienced the similar issues there. Even so, CircuitTracer could prove interesting for future work, but it exceeded our computational capacity and current expertise.

Moreover, we spent some time attempting to understand and possibly implement the *attribution graphs* developed by Anthropic (Ameisen et al., 2025), which was implemented in the CircuitTracer GitHub (Safety Research, 2025). An attribution graph is a graph whose nodes represent individual interpretable features, and for a given input text prompt, it shows which

[11] See https://github.com/safety-research/circuit-tracer/blob/main/demos/attribute_demo.ipynb

features influence which other features in order to eventually construct the output. Attribution graphs are therefore closely related to circuits (though, unlike circuits, attribution graphs are specific to a given prompt), and thus they may be highly relevant for the topic of this project. However, we found out that it was a too difficult task to understand exactly how these graphs worked, let alone the code implementing them. They also relied on other technology we were unfamiliar with, including transcoders and local replacement models. Thus, we eventually decided to focus our attention on other concepts. With that said, this is clearly a technique that could be useful in other, more ambitious projects.

5.2 Future work

In our reading, we encountered numerous methods and principles for work in MI research, some of which we did not have capacity to implement. We suggest that these may be explored further in later work and include a summary.

Attribution patching (Nanda, 2023) is an algorithm for patching activations at large, sweeping through the entire network in only a few, complete passes. This method is significantly cheaper than a full iteration of activation patching, for instance. This could help narrow the search for interesting parts of the network, locating specific activations to patch or ablate. The method assumes local linearity and uses gradients to take linear approximations of the model locally. As is the nature of approximations, the results of this method are thus slightly imprecise; approximating the larger components, like the residual stream, leads to more noise in the output. Hence, this is mostly intended as a rough, exploratory technique used as a first step in an analysis of a model.

Another method, Automated Circuit Discovery (ACDC) (Conmy et al. 2023), aims to automate the flow and structure of mechanistic interpretability research. Interpretability work centered on finding circuits exhibiting some behavior is time consuming, as one must manually sieve through much of the network or iterate with some method, like activation patching, to locate the components of such a circuit. ACDC is an algorithm automating the step of finding such a circuit. Given a certain behavior and suitable dataset, one can perform the algorithm in order to locate the exact circuit displaying the behavior for the given dataset.

As mentioned in Section 2.2, causal abstraction (Geiger et al., 2022) is another confirmatory method for verifying hypotheses about models. Conceptually, this method is fairly similar to causal scrubbing, but it differs slightly in implementation. As there was an easily accessible pseudocode for causal scrubbing, this was the method we tried implementing in our work. Causal abstraction is also modelled on quite a rigorous mathematical model, which formalizes the language of MI research. This method would be interesting to study further.

Lastly, as mentioned in Section 2.3, we used the TransformerLens library (Nanda & Bloom, 2022) in our work with decoder models. We note that this library also offers functionality for encoder models, with corresponding hook methods. Although we did not use this in our work, we suppose, based on our experience with our decoder work, that this could prove useful in future work if one were to work with encoders.

6 Conclusion

Simple, established methods in Transformer Mechanistic Interpretability – including ablation, activation patching and path patching – for localizing and describing model features and circuits were implemented and applied. An attention head which seems to respond strongly to color-related words was found. Several libraries, like BertViz and CircuitsVis for visualization, and Captum and TransformerLens for interpretability, were explored. With Captum, unsuccessful attempts were made to i) find model misconceptions on physics questions, and ii) compare the embedding space structure of different embedding models. TransformerLens was considered a promising library for further study, and it was demonstrated how to use it for finding induction heads in small-scale models, do activation patching for GPT-style models and logit lens for visualizing how the residual stream evolves through a model.

Literature

Agrawal, P., Stansbury, D., Malik, J., & Gallant, J. L. (2014). *Convolutional neural networks mimic the hierarchy of visual representations in the human brain*. https://people.csail.mit.edu/pulkitag/data/cnn_mimics_brain.pdf

AlphaFold. (2020, November 30). *AlphaFold: A solution to a 50-year-old grand challenge in biology*. DeepMind. Retrieved 25.08.2025 from https://deepmind.google/discover/blog/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology/

Ameisen, et al. (2025). *Circuit Tracing: Revealing Computational Graphs in Language Models*, Transformer Circuits. https://transformer-circuits.pub/2025/attribution-graphs/methods.html

Ameisen, E., Lindsey, J., Pearce, A., Gurnee, W., Turner, N. L., Chen, B., Citro, C., Abrahams, D., Carter, S., Hosmer, B., Marcus, J., Sklar, M., Templeton, A., Bricken, T., McDougall, C., Cunningham, H., Henighan, T., Jermyn, A., Jones, A., … Batson, J. (2025, March 27). *Circuit Tracing: Revealing Computational Graphs in Language Models*. Anthropic. https://transformer-circuits.pub/2025/attribution-graphs/methods.html

Chan, et al., "Causal Scrubbing: a method for rigorously testing interpretability hypotheses", AI Alignment Forum, 2022. https://www.lesswrong.com/posts/JvZhhzycHu2Yd57RN/causal-scrubbing-a-method-for-rigorously-testing

Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., & Garriga-Alonso, A. (2023). *Towards automated circuit discovery for mechanistic interpretability* (arXiv:2304.14997). arXiv. https://doi.org/10.48550/arXiv.2304.14997’

Cooney, A., & Nanda, N. (2023). *CircuitsVis* [Computer software]. GitHub. https://github.com/TransformerLensOrg/CircuitsVis

Coudray, N., Ocampo, P. S., Sakellaropoulos, T., Narula, N., Snuderl, M., Fenyö, D., Moreira, A. L., Razavian, N., & Tsirigos, A. (2018). Classification and mutation prediction from non–small-cell lung cancer histopathology images using deep learning. *Nature Medicine, 24*(10), 1559–1567. https://doi.org/10.1038/s41591-018-0177-5

Geiger, A., Wu, Z., D’Oosterlinck, K., Kreiss, E., Goodman, N. D., Icard, T., & Potts, C. (2022, October 31). *Faithful, interpretable model explanations via causal abstraction*. Stanford AI Lab Blog. https://ai.stanford.edu/blog/causal-abstraction/

Geiger, A., Ibeling, D., Zur, A., Chaudhary, M., Chauhan, S., Huang, J., Arora, A., Wu, Z., Goodman, N., Potts, C., & Icard, T. (2025). *Causal abstraction: A theoretical foundation for mechanistic interpretability*. *Journal of Machine Learning Research*, 26, 1–63. https://doi.org/10.48550/arXiv.2301.04709

Hanna, M. (2025). *Circuit Tracing Tutorial* [Jupyter notebook]. GitHub. circuit-tracer/demos/circuit_tracing_tutorial.ipynb at main · safety-research/circuit-tracer · GitHub

Hanna, M., Piotrowski, M., Lindsey, J., & Ameisen, E. (2025). *circuit-tracer* [Software]. GitHub. https://colab.research.google.com/github/safety-research/circuit-tracer/blob/main/demos/attribute_demo.ipynb#scrollTo=Qa5r1-7RmS8j

Heimersheim, S. & Nanda, N. (2024). How to use and interpret activation patching. arXiv preprint arXiv:2404.15255

Hugging Face. (n.d.-a). *Transformers documentation*. *Hugging Face*. https://huggingface.co/docs/transformers/index

Hugging Face (n.d.-b). *multi-qa-MiniLM-L6-cos-v1. Hugging Face.* https://huggingface.co/sentence-transformers/multi-qa-MiniLM-L6-cos-v1

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020, January 23). *Scaling Laws for Neural Language Models* (arXiv:2001.08361) [Preprint]. arXiv. https://arxiv.org/abs/2001.08361

McDougall, C. (2023, June 19). *Path Patching & Activation Patching* [Jupyter notebook]. GitHub. https://colab.research.google.com/drive/15CJ1WAf8AWm6emI3t2nVfnO85-hxwyJU#scrollTo=Ja036OpBmBKb

Meng, K., Bau, D., Andonian, A., & Belinkov, Y. (2022). *Locating and Editing Factual Associations in GPT* (arXiv:2202.05262). arXiv. https://doi.org/10.48550/arXiv.2202.05262

Meyer, B. & Nanda, N. (2024, May 10). *Exploratory Analysis Demo* [Jupyter notebook]. GitHub. https://colab.research.google.com/github/neelnanda-io/TransformerLens/blob/main/demos/Exploratory_Analysis_Demo.ipynb#scrollTo=WlGcRWRLHYe1

Nanda, N. (2022, March 11). *A longlist of theories of impact for interpretability*. LessWrong. https://www.lesswrong.com/posts/uK6sQCNMw8WKzJeCQ/a-longlist-of-theories-of-impact-for-interpretability

Nanda, N. (2023, February 4). *Attribution patching: Activation patching at industrial scale*. Neel Nanda. https://www.neelnanda.io/mechanistic-interpretability/attribution-patching

Nanda, N. (2024, December 21). *A Comprehensive Mechanistic Interpretability Explainer & Glossary*. Neel Nanda. https://www.neelnanda.io/mechanistic-interpretability/glossary

Nanda, N., & Bloom, J. (2022). *TransformerLens* [Computer software]. GitHub. https://github.com/TransformerLensOrg/TransformerLens

Nanda, N., Rajamanoharan, S., Kramár, J., & Shah, R. (2023, December 20). *Fact finding: Attempting to reverse-engineer factual recall on the neuron level*. AI Alignment Forum. https://www.alignmentforum.org/posts/iGuwZTHWb6DFY3sKB/fact-finding-attempting-to-reverse-engineer-factual-recall

nostalgebraist. (2020, August 31). *Interpreting GPT: The logit lens*. LessWrong. https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., & Carter, S. (2020). *Zoom In: An Introduction to Circuits*. Distill. https://distill.pub/2020/circuits/zoom-in/

Olah, C., Goh, G., Cammarata, N., Voss, C., Schubert, L., Petrov, M., & Carter, S. (2021). *A mathematical framework for transformer circuits*. Distill. https://transformer-circuits.pub/2021/framework/index.html

Olsson, C., Carter, S., Batson, J., & Henighan, T. (2022). *In-context learning and induction heads*. Transformer Circuits. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html

Safety Research. (2025, August 11). *attribute.py* [Source code]. In *circuit-tracer*. GitHub. https://github.com/safety-research/circuit-tracer/blob/main/circuit_tracer/attribution/attribute.py

Sanderson, G. (2024, August 31). *How might LLMs store facts | Deep Learning Chapter 7* [Video]. YouTube. https://www.youtube.com/watch?v=9-Jl0dxWQs8

Sundararajan, M., Taly, A., & Yan, Q. (2017). *Axiomatic attribution for deep networks* (arXiv:1703.01365). arXiv. https://doi.org/10.48550/arXiv.1703.01365

Wang, K., Variengien, A., Conmy, A., Schlegeris, B. & Steinhardt, J. (2022). *Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small*. arXiv preprint arXiv:2211.00593

Wang, Z. (2025). *LogitLens4LLMs: Extending Logit Lens analysis to modern large language models* (arXiv:2503.11667). arXiv. https://doi.org/10.48550/arXiv.2503.11667