Interpreting Planning in Transformers[1]

Abhay Sheshadri
Georgia Institute of Technology

Victor Levoso
Independent

Neel Nanda, Esben Kran, Fazl Barez

Abstract

Whether or not current models such as large language models (LLMs) actually perform planning has been the subject of much debate. Understanding search and optimization in transformers is vital to solving the problem of *inner alignment* and avoiding potential failures such as the presence of misaligned *mesaoptimizers* or learned search mechanisms. We make the first step towards providing an example of a mechanism responsible for planning and search in toy transformer models by presenting a circuit that appears to be used for backtracking from a goal to the current state. We use a combination of attention visualizations and activation patching to provide evidence for our claims.

Keywords: Mechanistic interpretability, AI safety, Inner Alignment

1. Introduction

An ambitious goal of mechanistic interpretability research is to rigorously understand the learned internal cognition of any neural network. An important subgoal that must first be reached is uncovering the mechanisms behind reasoning and planning in current models. Whether or not current models such as large language models (LLMs) actually perform planning has been the subject of much debate. Understanding search and optimization in transformers is vital to solving the problem of *inner alignment* and avoiding potential failures such as the presence of misaligned *mesaoptimizers* or learned search mechanisms.

Since terms like planning and search are not rigorously defined, it is difficult to make progress on detecting circuits in models responsible for these functions. We operationalize search/planning as any mechanism involving an iterative procedure to decide which action to take in order to achieve a certain goal. Ideally, the mechanism should be "general-purpose" enough to be able to optimize for a variety of different goals. In this paper, we search for such mechanisms in transformers, the neural network architecture underlying current state-of-the-art language models, trained on toy settings.

[1] Research conducted at the Apart Research Alignment Jam #10 (Interpretability 3.0), 2023 (see https://alignmentjam.com/jam/interpretability)

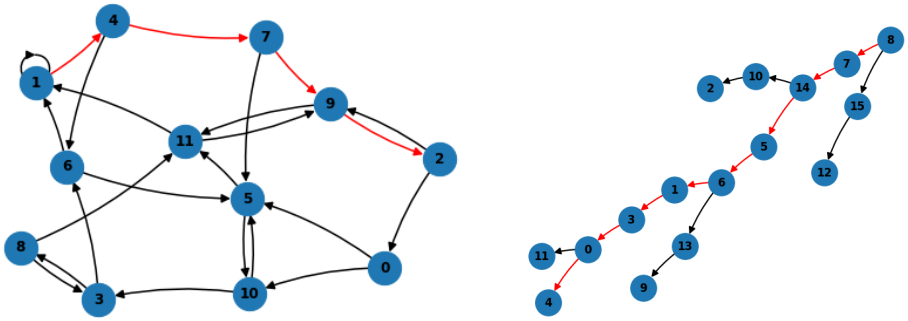

Figure 1: *Two types of graphs we train on - strongly connected graphs (left) and trees (right), both with 16 nodes*

Figure 2: *Normalized logit differences after patching residual stream from clean to corrupted (changing the goal node). Result on a 6L1H transformer trained on trees.*

We make the first step towards providing an example of a mechanism responsible for planning/search in transformers by presenting a circuit that appears to be used for backtracking from a goal to the current state. We trained autoregressive transformers on the task of finding the shortest path between two nodes in a graph when conditioned on an edge list, a starting node, and a goal node. The transformers manage to learn a series of attention heads that reveal a backtracking pattern. These heads can effectively be understood as implementing an algorithm that involves iteratively backtracking from the goal node until the model finds a path to the current state.

We trained two separate models on two types of random graphs, as shown in Figure 1. On the left are strongly connected graphs, where there exists a path from every node to every other node. On the right are trees where there exists only a single path from the starting node to the goal. Despite the different features between these classes of graphs, we find that both models have many things in common and appear to be implementing very similar algorithms. This provides preliminary evidence that similar circuits might be found in other types of transformers.

To verify our hypotheses, we perform simple activation patching experiments to determine which activations are causally linked to the outputs. Activation patching was first used to discover the indirect object identification circuit in GPT-2 small (Wang et al., 2022), and has since become a staple of mechanistic interpretability research. We use it to trace the flow of information across the residual stream, during a forward pass.

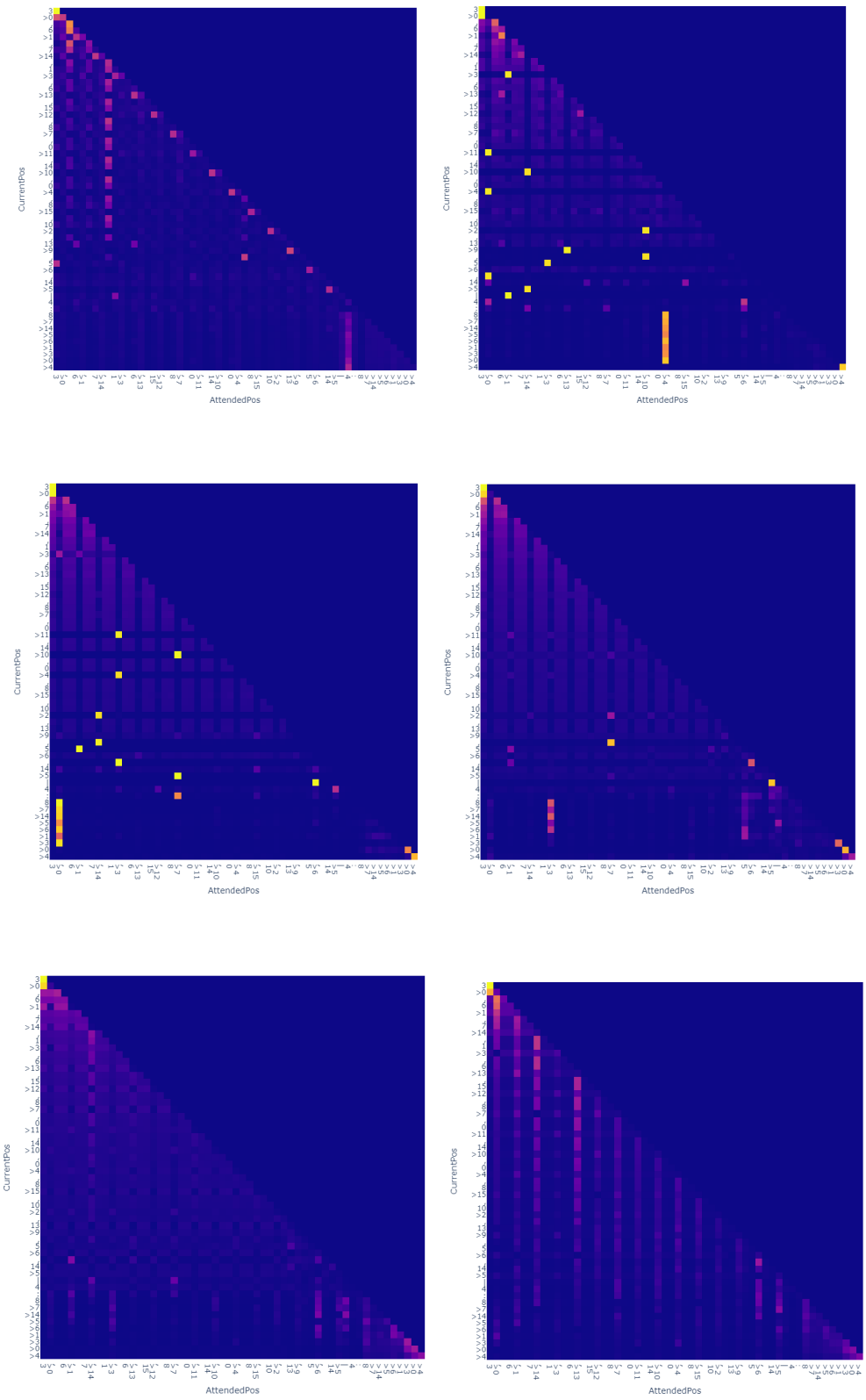

Figure 3: *Attention heads in the first four layers of a 6L1H transformer trained on trees. Includes layer 0 (top left), layer 1 (top right), layer 2 (middle left), layer 3 (middle right), layer 4 (bottom left), layer 5 (bottom right)*

2. Methods

We trained 6-layer decoder-only transformers to output the shortest path from a start node to a goal on two types of graphs: strongly connected graphs and trees. The model trained on strongly connected graphs has 4 attention heads per layer, while the model trained on tree graphs has a single attention head per layer. The models are conditioned on the edge list of the graph, the goal node, and the starting node. From there, they are trained to predict the next node in the shortest path to the goal given the path so far. At inference time, we use an autoregressive process to feed the predicted next node back into the model to output a path, just like in language models.

Our activation patching code was borrowed from Neel Nanda's Intro to TransformerLens demo. Activation patching consists of replacing the activations from a run that has some specific output with the activations on a run that has a different output and seeing which ones actually cause the output to change (by plotting the difference on some logit direction on that branch).

We also plotted attention patterns for different outputs and calculated the differences from similar graphs to highlight specific things of interest.

The code for our experiments can be found in these two colab notebooks:

- Strongly connected graphs:
  - https://colab.research.google.com/drive/1lQlc7TzT87CM-6kRwshimpVDpyxYZvYh

- Trees
  - https://drive.google.com/file/d/1R-JFxYXjzuTSatf9yRTAW-yzLFGFXNGN/view?usp=sharings
  - https://colab.research.google.com/drive/1aqk9PmB5MxV2lVXhUY3qDzm2ZdNmKnZH?usp=sharing (activation patching and other experiments)

3. Results on Tree Graph Model

Figure 3 shows the attention patterns of our transformer trained on finding paths in trees when fed the example of a tree from Figure 1. In each of the attention patterns, we can see a vertical line towards the bottom. This is the autoregressive part of the context looking back at edges in an iterative fashion, starting from the goal. In this example, the goal is to reach node 4. We can see that in Layer 0 Head 1, each position attends to the goal. In the next layer's attention head, each position (except for the last) attends to the edge 0->4 where the goal is the incoming node. In the layer after that, we see that the vertical line is at the edge 3->0, where the previous outgoing node is now the incoming node.

The pattern starts to break slightly at the next layer (Layer 3 Head 0), where the head attends to both the edges 1->3 and 5->6. Since node 6 is actually connected to node 1, this head might actually be looking back 2-4 edges at a time. Subsequent attention heads

are not as clean as the ones in Figure 3; they might be performing several steps of backtracking simultaneously or using some other trick that works on shallower subtrees. Figure 2 shows the results of performing activation patching on the residual stream of the transformer. The clean and corrupted examples share the same graph, but differ in the goal and therefore, the shortest path to the goal from the root. We compare the logit difference between the clean and corrupted example at the first branching point where the next node in the shortest path will differ. For example, with the tree from Figure 1, the clean path can be 8 all the way to 4, and the corrupted path can be 8 all the way to 11. We would then form patching at the position where node 0 is the input, and compare the logit difference between picking 4 vs 11. Patching the goal position at the 0th layer has a great effect on the logits, as it will be copied by the layer 0 attention head and used to initialize the backtracking. Patching the goal position after layer 0 has little impact on the logits since it will have already been copied over. Patching at node 0 seems to have a massive impact on the logits, most likely because it involves intervening in the intermediate computations of the backtracking process.

We also hypothesize that the MLPs are potentially being used to decide when to stop backtracking. You can see this behavior in the bottom-right corners of the attention patterns of layers 1, 2, and 3. In layer 1, the position where the model reaches the goal node attends to itself. We also see something similar in the next two layers - in the 2nd layer and 3rd layer attention patterns, we see that the second-to-last and third-to-last positions attend to themselves. The previous layer's MLP might be checking if the previous backtracking step resulted in reaching the goal read-in from the layer 0 attention head.

Another behavior that can be discovered by looking at attention patterns is nodes on some edges paying attention to nodes on other edges. In the layer 1 attention pattern from Figure 3, we can see the ">10" from edge 14 -> 10 paying attention to ">14" from edge 7 -> 14. We believe that it might be moving information about node 7 onto the node 10 position. In the layer 2 attention head, 1the ">10" in edge 14 -> 10 attends to ">7" in edge 8 -> 7. By this point, the residual stream of the token ">10" will likely contain information about what nodes are 1-2 edges back. In the layer 3 attention head, we see the ">2" in edge 10 -> 2 attend to the ">7" in edge 8 -> 7, which is looking back at 3 positions.

This separate backtracking algorithm occurring in the edges can complement the backtracking algorithm occurring in the path output positions, allowing the model to look back even further. However, this "edge backtracking" process does not work all the time since edges can not look at edges in front of them due to causal masking in the attention. This process does not occur when we reverse topologically sort the edges i.e. the edge B -> C will always be before the edge A -> B. Despite this, the model still robustly figures out the path even when edges are sorted in this way. When we perform activation patching on two graphs that differ by one edge, we can see that the patching of the position of the changed edge has a huge impact on the logits. It also appears that the attention head of layer 4 might be responsible for moving "edge backtracking" information to the solution path positions as patching the residual stream at path positions in layer 5 has a big impact on the logits.

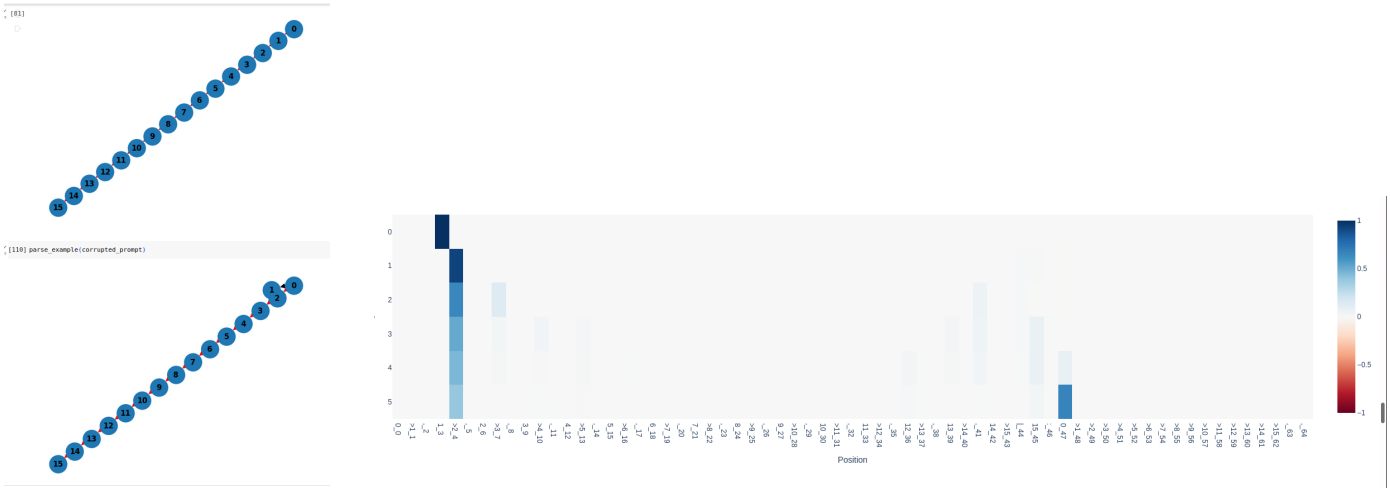

Figure 4: *Clean graph (top left), corrupted graph (bottom left), and activation patching result (right). Normalized logit differences after patching residual stream from clean to corrupted (changing one edge).*

We are still pretty confused about what's happening in layers 4 and 5, but it appears as if they have some similarities in their attention patterns. In the layer 4 attention pattern, every position pays attention to every edge except the tokens in the path output positions that pay attention to everything except the previous 2 edges. We can also see that swapping 2 edges makes all kinds of weird changes to the attention pattern of layer 5, as can be seen in Figure 4. We speculate that it might be doing something with the MLP layers.

4. Results on Strongly-Connected Graph Model

We also trained a model to predict the shortest path given an edge list of a strongly-connected graph. A key difference between strongly-connected graphs and trees is that there can now be multiple paths from the starting node to the goal. As a result, the model will need to keep track of multiple paths when backtracking from the goal.

Despite this difference, we see attention heads that appear to have a similar function to the heads we described in the previous section. The patterns in Figure 5 look similar to the patterns in the first three heads in Figure 3. The head in layer 0 copies the goal to each of the output path positions, the head in layer 1 backtracks to edges that contain nodes that are one before the goal, and the head in layer 2 backtracks to edges that contain nodes that are two away from the goal.

The other heads in this model appear to perform behaviors that are missing from the model trained on trees. For instance, there appears to be a similar “edge backtracking” process, but there also appears to be the opposite - something closer to looking ahead. Edges of the form B > C can attend to both edges A > B and C > D.

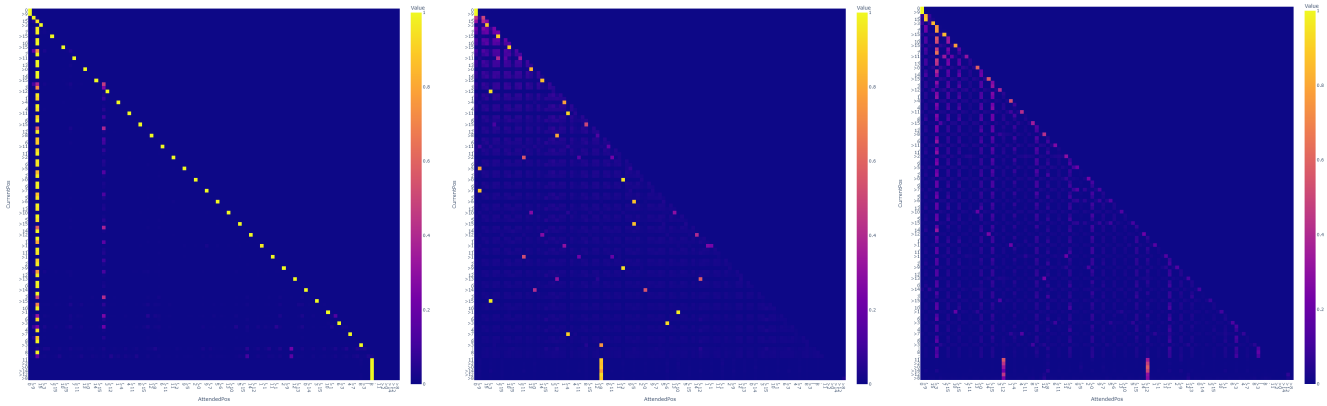

Figure 5: *Attention Patterns from the first three layers of 6L4H transformer trained on strongly-connected graphs.*
Includes layer 0 head 2 (left), layer 1 head 2 (middle), layer 2 head 3 (right)

5. Discussion and Conclusion

While in some sense, tree traversal and graph search can be seen as planning and arbitrary problems can be seen as some form of a graph, there are big disanalogies between the kind of algorithm that can solve this kind of small problem and the kind of "realistic planning" that we really care about and it seems like we are currently confused about what kind of toy models are useful to study.

However, there are plenty of insights to be gained by studying toy models. The general direction of looking into toy problems in graphs does seem like where the kind of problem that requires even a toy version of the most interesting problem lies, and starting by interpreting easier toy models like this one seems like it would be better than starting with something much more complicated. Even if it doesn't, the state of knowledge on mech interp is bad enough that reverse engineering any toy model is great progress anyway.

An interesting next step might be studying whether similar circuits form in small language models trained to perform simple *chain-of-thought* tasks. In language models, selection-inference prompting can be used to get the LM to iteratively choose steps in a proof. Training small language models like GPT-2 Small or TinyStories to resolve first-order logic queries in natural language might be a good place to start.

6. References

1. Wang, K., Variengien, A., Conmy, A., Shlegeris, B. & Steinhardt, J. Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small. Preprint at https://doi.org/10.48550/arXiv.2211.00593 (2022).