UNIVERSITY OF AMSTERDAM

MECHANISTIC INTERPRETABLITY PROJECT

Interpreting Transformers on a Sorting Task

September 21, 2024

Supervisor:
Leonard Bareska

Examinor:
Rick Quax

Student:
Amir Sahrani
12661651

Course:
Individual Project

Course code:
5284INPR6Y

Abstract. Mechanically understanding transformer models might be an important tool for evaluating the behavior of current Large Language Models. In this report, we investigate a transformer trained on the task of sorting. Sparsity was introduced through an L1-penalty, the effectiveness of which was measured using the Local Learning Coefficient. Through the use of benchmarks, activation patching, logit reconstruction, and visualization of attention, we find that the model performs most of the sorting in the first layer, while the other layers act to refine the outputs of the first layer. The results of this study demonstrate how benchmarks and logit reconstruction can provide insights into the process of generating outputs.

All code can be found on this Repository.

1 Introduction

Transformer models have demonstrated remarkable success in various natural language processing tasks. With this success come many concerns about safety and societal implications. Models are often 'fine-tuned' to not fulfill certain requests deemed to be unsafe or unethical. This fine-tuning often results in models that look safe but can be made to fulfill these requests through specific prompts.

Transformers are typically optimized to maximize a proxy reward signal meant to approximate some desired high-level behavior. Yet an inherent gap exists between this proxy objective and the true intended goals envisioned by the human developers. This leads to the problem of "reward hacking" [3] - where models become incentivized to maximize the specified objective in a narrow and potentially misaligned way, neglecting other important factors comprising the intended overall utility. While substantial work aims to better formulate objectives matching human values, a complementary approach is understanding how models behave when trained on imperfect or misspecified objectives. Such insights could help validate whether a model is indeed optimized for its intended purpose, even when assuming the original objective was well-designed.

The investigation of a neural network's inner workings is called mechanistic interpretability. This work tries to find ways in which models perform the tasks they are trained on, usually through visual exploration, activation patching, which, in some cases, leads to a mathematical description of the algorithms performed by the model [9].

2 Background

2.1 Superposition

Many models can represent far more 'features' than they have individual neurons. A feature can generally be thought of as an abstract pattern, and depending on the location in the neural network, this might represent something fundamental, like a horizontal edge for a vision model or the sentiment of a sentence for a language model. One proposed mechanism by which models can accomplish this cramming of features is the theory of Superposition [1]. This theory supposes that although there might be many more features than the model has parameters, the inputs to these models are sparse, meaning an input contains far fewer features than the total dataset. This allows the model to shift from orthogonal storing of the features to "quasi" orthogonal storing of the features. In the former, each feature activates a particular pattern of neurons that is not shared with any other feature, while in the second, some features might share a pattern. Still, given that they rarely co-occur, the model can ascribe different meanings to the pattern based on the context received from other features in the input. Here, contexts refer to the other features in the input space.

This superposition poses an additional challenge when investigating the inner workings of these models. When a model is in superposition, it is not sufficient to understand what circuits activate when a feature is present; the interaction between a feature and its context adds another layer of complexity. In this paper, we will consider two methods of mitigating superposition.

Motivated by the benefits of mechanistic transparency, our work aims to shed light on the inner workings of transformer models trained for the fundamental algorithmic task of sequence sorting. We explore two complementary approaches for this objective. Firstly, using the Tracr library [5] to construct interpretable "idealized" transformers with known sorting mechanisms specified directly through a programming language. Secondly, applying sparsity-inducing regularization to encourage simpler computation paths when training standard transformer models for sorting.

Through analysis and comparison of these constructed and trained sorting models, we aim to develop a clearer understanding of the algorithms and processes involved in this foundational task. While sorting itself has well-known traditional solutions, dissecting how transformers learn to approach this problem can yield valuable insights and serve as a stepping stone toward interpreting more complex model behaviors. Ultimately, such mechanistic transparency will be crucial for ensuring the safety and trustworthiness of these powerful language models as their capabilities continue growing.

2.1.1 Tracr

Tracr [5] is a library developed by Google DeepMind. This library allows for direct specification of the behavior of a model through the Rasp (*Restricted Access Sequence Processing Language*) programming language [8]. Rasp defines the fundamental operations transformers can perform, not limited to toy models. These operations are performed on entire sequences and are functional; this means that transformers can only perform operations on the entire input space and cannot make in-place changes to the sequences they operate on. The language consists of two types of variables and three types of instructions. Sequence operations and selectors are the variables available; Sequence operations can be seen as a function $f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{n \times n}$. Selectors result from the `Select` operation performed on two sequences; selectors are matrices of all pairwise comparisons performed on two input sequences. The `selector` can be reduced back to a sequence using the `Aggregate` function, which can take in a matrix and a sequence operation and return a new sequence operation. Finally, the `selector_width` function takes in a sequence and generates a sequence of the same length, with the count for each token at each position.

Using Rasp, Tracr can construct models with orthogonal activations, meaning each feature on the input space has a unique activation pattern. The models produced by Tracr are idealized Transformers, which are straightforward to interpret. Another benefit of these models is the fact that they allow for a proof of concept, meaning that given a particular task, if a set of Rasp instructions can be formulated for Tracr to generate a model with, we can know this task can be performed on a model of the same size, without making use of superposition.

2.1.2 Sparsity

We looked into regularization as a method for encouraging sparsity. One method of regularization is L1-regularization, defined as

$$E = \ell(w) + \lambda \sum_i |w_i|. \tag{1}$$

ℓ is the model's loss function and w_i is the model's weight. Through L1-regularization, the model is incentivized to keep the absolute value of the parameters low, with the goal of only necessary parameters getting a non-zero value. L1-regularization only applies to the weights in the network; in the case of a regular neural network, this might result in cleaner activations, but in the case of transformers, the residual stream accumulates activations after each attention or MLP block. The penalty term is modified to incentivize the model to keep the residual stream sparse.

$$E = \ell(w) + \lambda \sum_i |\alpha_i^{\text{post}}|. \tag{2}$$

This results in a loss term with a penalty for all values in the residual stream after each layer.

2.2 Complexity

In this context, sparsity is a proxy for the goal of having a simpler model that takes steps that are easier for human evaluators to understand. Many metrics attempt to explain a model's complexity, including the Local Learning Coefficient (LLC) [4]. This is a local approximation of the learning coefficient. The Learning Coefficient is a theoretical measure of model complexity; however, it is computationally infeasible. The main idea behind LLC is that most models are singular, meaning that given a local minimum, changes to the parameters in small amounts do not lead to a change in the loss of the model. Hoogland et al. [2] have used this complexity estimate to investigate the development of a model throughout its training, arguing that inflection points in the derivative of the LLC correspond to phase transitions in the model.

2.3 Algorithmic Tasks

Understanding the entirety of large language models is not feasible with current knowledge and techniques. Not only do we not have the tools and frameworks to understand the computations performed by these models, but in many cases, the tasks for which we train the models are not fully understood, as is the case with, for example, large language models or protein folding.

Nanda et al. [6] have taken another approach, taking steps to understand simple models trained on an algorithmic task. This approach is hoped to let researchers develop tools, intuition, and knowledge on the behavior of models in simpler tasks, which can be used in investigations in more complex models.

In this paper, we will look into sorting a sequence of integers. A Tracr model will be constructed and analyzed as a demonstration of Rasp and to understand one approach to sorting in a transformer. Furthermore, a model will be trained to perform the same task; regularization will make the model activations more sparse. Model complexity and performance on benchmarks will be measured after each epoch. Finally, an attempt will be made to understand some internal computations performed by the model to perform the sorting task.

3 Methods

3.1 Transformer Model

The transformer model studied in this work is tasked with sorting sequences of ten unique numbers ranging from one to ten. We use the Tracr library [5] to first construct an "idealized" transformer architecture with built-in sorting capabilities specified through a custom programming language called Rasp. The configuration of this Tracr model is then copied to initialize the weights of a standard transformer model, which is trained from scratch for the sorting task.

The Rasp program we use to define the sorting procedure is shown in Algorithm 1. It operates by first selecting the "smaller" elements from the input keys (token IDs) compared to themselves. This creates a mask highlighting the minimum value. The width of this mask is then computed to determine the target output position for the minimum value. A new mask is generated, marking the indices where the target position equals the current position. Finally, the values (embeddings) at these marked indices are aggregated to produce the output. In essence, this program implements a simple selection sort by repeatedly picking out the smallest remaining element to output.

Algorithm 1 Rasp Program for Unique Sorting

```
Require: keys, values                                          ▷ Token IDs and embeddings
Ensure: out                                                    ▷ Output embeddings
1: $smaller \leftarrow \text{Select}(keys, keys, smaller)$                      ▷ Get mask for min value
2: $target\_pos \leftarrow \text{SelectorWidth}(smaller)$                ▷ Compute target output position
3: $new\_selection \leftarrow \text{Select}(target\_pos, indices, equal)$              ▷ Get mask for target
4: $out \leftarrow \text{Aggregate}(new\_selection, values)$             ▷ Aggregate values at target
5: return $out$
```

The transformer consists of 3 layers, each containing an attention block followed by a multi-layer perceptron (MLP) block. Within each attention block, there is a single attention head. The vocabulary size is set to 12 to account for the begin-sequence and part-of-sequence tokens required by Tracr. The dimension of the residual stream (i.e., embedding size) is 50.

Three separate instances of this 3-layer transformer are trained from scratch, each with a different L1 weight decay coefficient of $\lambda \in 0, 10^{-6}, 10^{-5}$ to induce varying levels of sparsity, these values were chosen to result in a loss of the same order of magnitude as the loss of a completely untrained model. This results in the penalty not dominating the loss function. The AdamW optimizer [7] is used with a learning rate of 10^{-3} for 300 epochs and a batch size 256 and the Cross-Entropy loss function. During the training process, each element of a batch was randomly generated. An element was defined as a sequence of 10 integers, where each integer was selected from the set $0, 1, \ldots, 9$, as shown in the following equation:

$$(x_0, \ldots, x_9) \mid x_i \in \{0, \ldots, 9\}. \tag{3}$$

Due to the vast number of possible sequences (10^{10}), the model was trained on only 300×256 inputs, which represents a small fraction of the possible sequences. As a result, there was no meaningful separation between the training and test sets, as the trained data covered only a limited portion of the potential input space.

4 Results

4.1 Performance Metrics

After 300 training epochs, all models achieved 100% accuracy on the sorting task for sequences of 10 unique numbers from 1 to 10. Figure 1 shows the loss curves, sorting evaluation metrics, and logit-level logistic circuit capacity (LLC) estimates over training for the three models with different L1 weight decays ($\lambda \in \{0, 10^{-4}, 10^{-3}\}$).

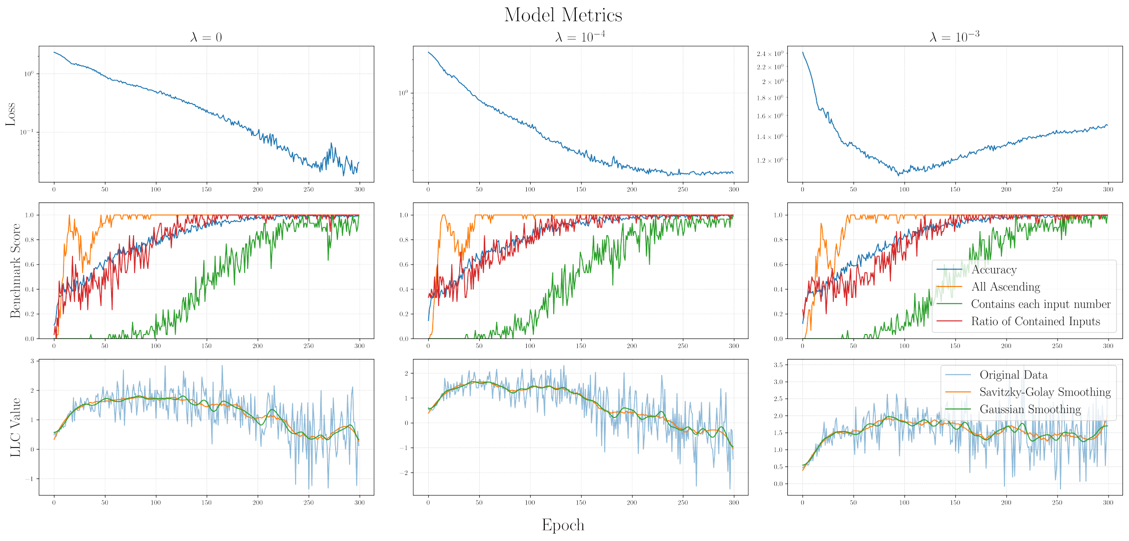

Figure 1: The loss, benchmarks, and LLC value for the model with three values for λ, we can see that the model with $\lambda = 10^{-4}$ is the least complex model according to the LLC value

The model with $\lambda = 10^{-4}$ exhibits the lowest estimated LLC, suggesting a less complex solution. While all models eventually satisfy all metrics, we can identify three key transition points in their progression:

1) Around epoch 25-50, the “all ascending” metric is first satisfied, indicating the models begin outputting sequences in ascending (though not necessarily correct) order.

2) By epoch 50, the ratio of input tokens present in the output becomes high enough to satisfy the “contains all inputs” metric.

3) Between epochs 150 and 250, the models achieve high accuracy as the remaining metrics level off, with a subsequent decrease in LLC.

These transition points illustrate how the models decompose the sorting task into logical steps of first outputting ascending sequences, then representing all input tokens, before finally producing the correct sorted output.

4.2 Sorting Mechanism Analysis

4.2.1 Tracr Compiler Visualization

To analyze how the transformer architecture implements the sorting functionality, we first inspect the “idealized” model constructed by the Tracr compiler based on our defined Rasp sorting program 1.

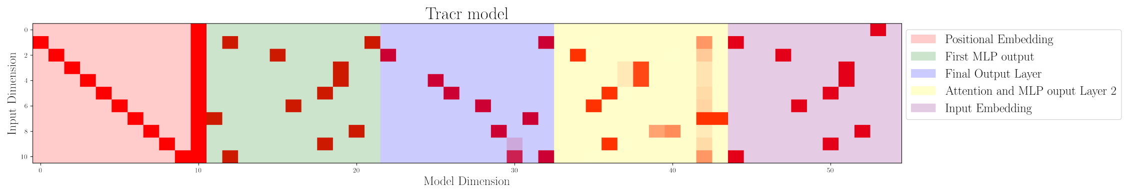

Figure 2: Final residual Layer of the Tracr model, each color represents a Rasp Operation.

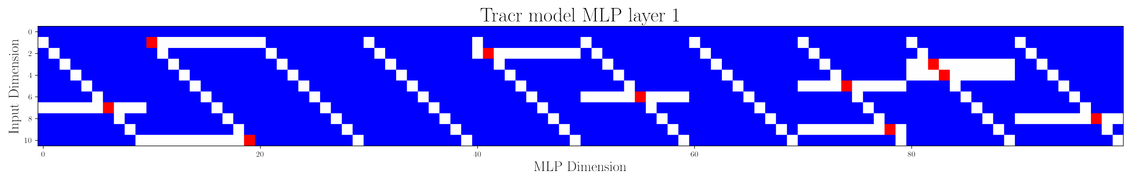

Figure 3: The MLP activations in the first Layer

Figure 2 shows the final residual stream activations of this Tracr model. The outputs are highly sparse, with different segments corresponding to each of the Rasp operations. The first segment performs the token encoding, while the last contains the scrambled input tokens. In the first layer's MLP (Figure 3), a section exists for each input value to compute the `select_width` operation, counting its occurrences.

The second layer's attention then calculates the indices associated with each value. Its MLP appears to "spread out" the values according to their counts from the previous layer. Finally, the last attention mechanism sorts the values by these indices and handles duplicates. This sparse implementation precisely mirrors the procedural Rasp sorting routine.

4.2.2 Trained Transformer Interpretation

Analyzing the trained transformer model reveals several insightful patterns:

Attention Analysis (Figure 4): For a scrambled input, the first layer exhibits almost uniform attention, attending to all tokens equally. The second layer shows an alternating broad attentional pattern, while the third layer demonstrates more concentrated attention. This progression from broad to focused attention suggests the model first encodes the global input context before specializing to more localized token relationships in later layers.

Logit Reconstruction (Figure 6): Visualizing the "logit reconstructions" (unembedded residual stream activations treated as logits) shows the model's intermediate sorting estimates. A key observation is that the general form of the sorted logits is already established after the first layer, with subsequent layers refining this initial estimate.

For scrambled, sorted, and reversed inputs, the reconstructions follow similar trajectories - first exhibiting two distinct logit clusters (tokens 0-2, 5-8), which gradually merge into the diagonal sorted form. When all tokens are identical (e.g. all 1s), we interestingly see "antithetic" logit patterns where the model is highly confident the values are not only the correct token but also not other tokens.

Epoch 50 Analysis (Table 1, Figure 7): Examining the model's behavior at epoch 50 aligns with the transitions observed in the metric curves. The model outputs are partially sorted, correctly handling simple cases like all 1s or sorted inputs. However, it struggles with more complex reorderings like scrambled or reversed sequences. The logit reconstructions show a less refined but qualitatively similar trajectory as the final model.

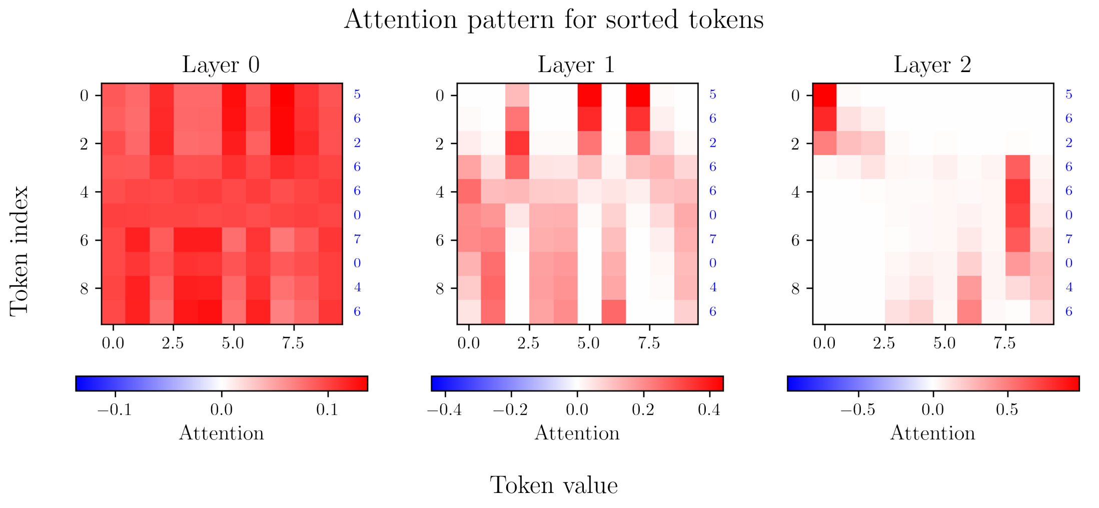

Figure 4: The attention at all three layers for a scrambled input

Table 1: Predicted values for different token scenarios after

	Original Tokens	Model Output
Sort tokens	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Reverse tokens	[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]	[0, 2, 2, 3, 4, 5, 6, 8, 8, 9]
Only ones	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Only Nines	[9, 9, 9, 9, 9, 9, 9, 9, 9, 9]	[9, 9, 9, 9, 9, 9, 9, 9, 9, 9]
Scrambled tokens	[8, 3, 8, 8, 6, 0, 0, 5, 9, 2]	[0, 1, 2, 2, 5, 6, 7, 8, 9, 9]

4.3 Activation Patching

To probe the model's learned representations, we perform activation patching - systematically replacing activations from a "clean" run with activations from a corrupted run (e.g. reversed token order) at different layers and positions. We then measure the KL divergence between the original and patched output distributions (Figure 5).

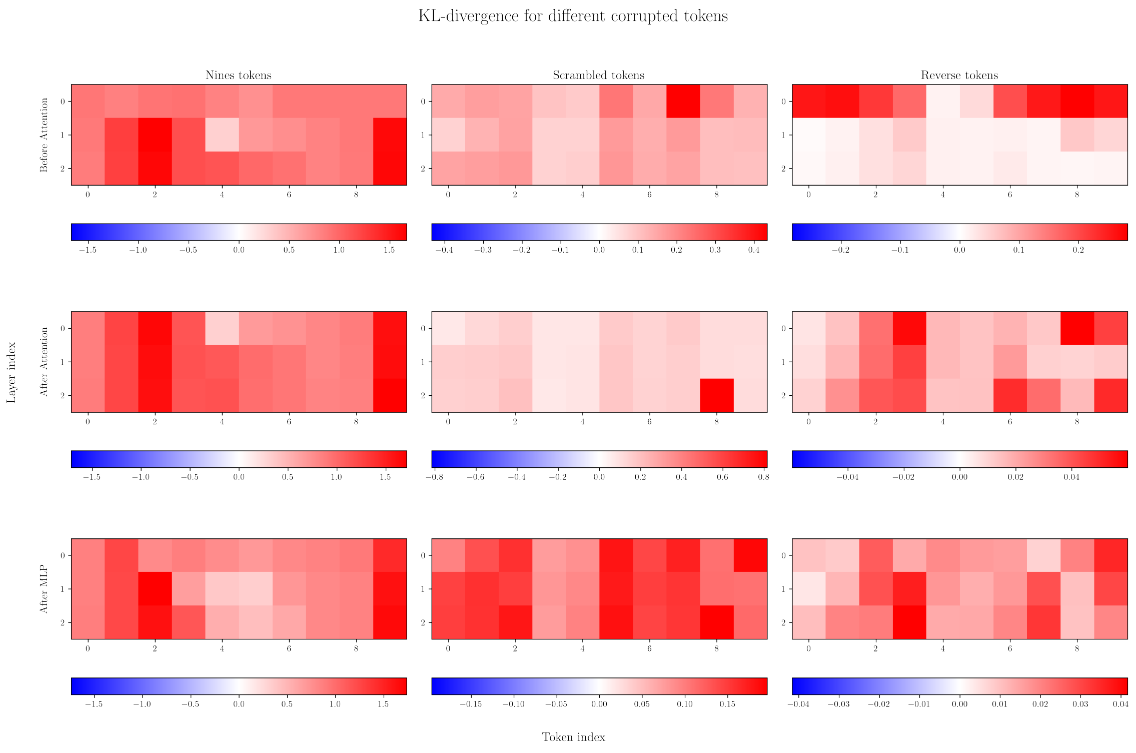

Figure 5: KL divergence as a result of activation patching different tokens into a 'clean' input at various layers

Patching reversed token activations into the first layer before the initial self-attention shows high divergence for tokens 0-3 and 6-9, effectively reversing just those token positions. In contrast, patching all 1s or all 9s into the first layer yields uniform high divergence across all tokens.

For patching at other layer positions, the divergence patterns more closely resemble the (normalized) logit distributions for the respective constant inputs. This suggests the early self-attention layers play a crucial role in establishing the basic token ordering, while later layers operate on this coarse ordering signal.

In summary, the interpretation analysis provides insights into how the transformer model deconstructs and solves the sorting task: establishing a coarse global context in early layers, iteratively refining the token estimates, and ultimately converging on the correct output order - with attention sparsity induced by L1 regularization leading to a less complex solution.

5 Discussion

Although no mathematical description of these findings can be given, multiple insights have been gained from investigating this toy model. LLC seems to be a useful measure for model complexity; it allows for comparison between different models on the same task and between one model at different points of training. The benchmarks proposed for the model show that it is possible to find specific points at which a model can perform certain actions. These benchmarks accurately represent what kind of capabilities to expect from this model. We have also found the first attention layer to show uncertainty about the distribution of tokens, showing uniform attention between all tokens. This attention narrows to a diagonal band as the model progresses through its layers, representing the decrease in uncertainty.

5.1 Limitations

Little can be said about the MLPs in this model, although we can see how they relate to the model output through the logit investigation and activation patching.

The architecture of this model might also limit the generalization of these findings. Firstly, the fact that the model's vocabulary size and context window are roughly the same size might mean that the model can more accurately use heuristics based on the vocabulary range. In contrast, these heuristics might not be possible with larger vocabularies. Secondly, the use of learned positional embeddings might enable the model to perform some sorting tasks outside the attention and MLP blocks, which this investigation has not examined, this could be prevented by fixing the embeddings to one-hot-encoded representations. In typical use cases, an efficient representation of the data might be desired; in that sense, this model is more comparable to real-world models. However, in the context of real-world (Large Language) models, many other aspects of this model are both idealized and non-representative. When applying this work to these settings, it might be beneficial to consider the level of understanding human evaluators have regarding the task at hand. Evaluating a task through benchmarks is only viable when researchers have a clear understanding of what constitutes a meaningful improvement towards the goal, as well as being able to formulate the goal itself.

5.2 Future work

As transformers are inherently probabilistic and parallel, traditional sorting algorithms are necessarily unable to capture the mechanism by which a transformer would perform a sorting operation. One way to potentially find a well-posed framework of the mechanism by which a transformer sorts a sequence could be through probabilistic integer sorting, as these algorithms tend to perform well in cases where the range of inputs in known and limited. Transformers are probabilistic models that assign probabilities to different possible output sequences based on the input sequence. This is similar to the randomization aspect employed in probabilistic integer sorting algorithms. Transformers also operate on a finite number of discrete elements (tokens) with a fixed vocabulary size, which is analogous to the limited and known input range that probabilistic integer sorting algorithms excel at handling.

Benchmarks are readily available for large language models; these could be used to find developmental stages during training; this process can be combined with the LLC to find stages of capability gains in these models. Most current benchmarks test the number of correct answers a model achieves on a specific set of tasks. At the same time, this might be useful for estimating the capabilities of a fully trained model. Using the right combination of benchmarks, or benchmarks specifically crafted to show incremental improvement, might help in understanding the capabilities of larger models.

Finally, investigating toy models is often limited to the investigation of 1 task. However, large language models are often tasked with numerous tasks, which they can all perform to different extents. An approach to emulate this would be to train a toy model on a set of n tasks, which can then be to disentangled. For example, a model could be trained to do sorting, addition and filtering depending on the input form. Even if the algorithms individually cannot be understood, separating the circuitry for each algorithm might reveal task delegation within a model. This process can be aided through the use of benchmarks and the LLC to find points at which specific tasks can be performed and relate those to the model at stages at which it was not able to perform the task.

6 Appendix

6.1 Scope

Not all that was investigated has made it into this report; two particular streams of investigations have been omitted. Firstly, an investigation into compression was done. The model was compressed by multiplying the inputs and outputs of the model's intermediate representation with a lower-rank matrix. Upon further inspection, multiplying this matrix with itself resulted in an approximation of an identity matrix. This approximation, however, was of poor resolution, making the model perform worse while increasing its complexity.

Secondly, the Query Key and output Value circuits were investigated. The results were, however, difficult to interpret and present.

6.2 Figures

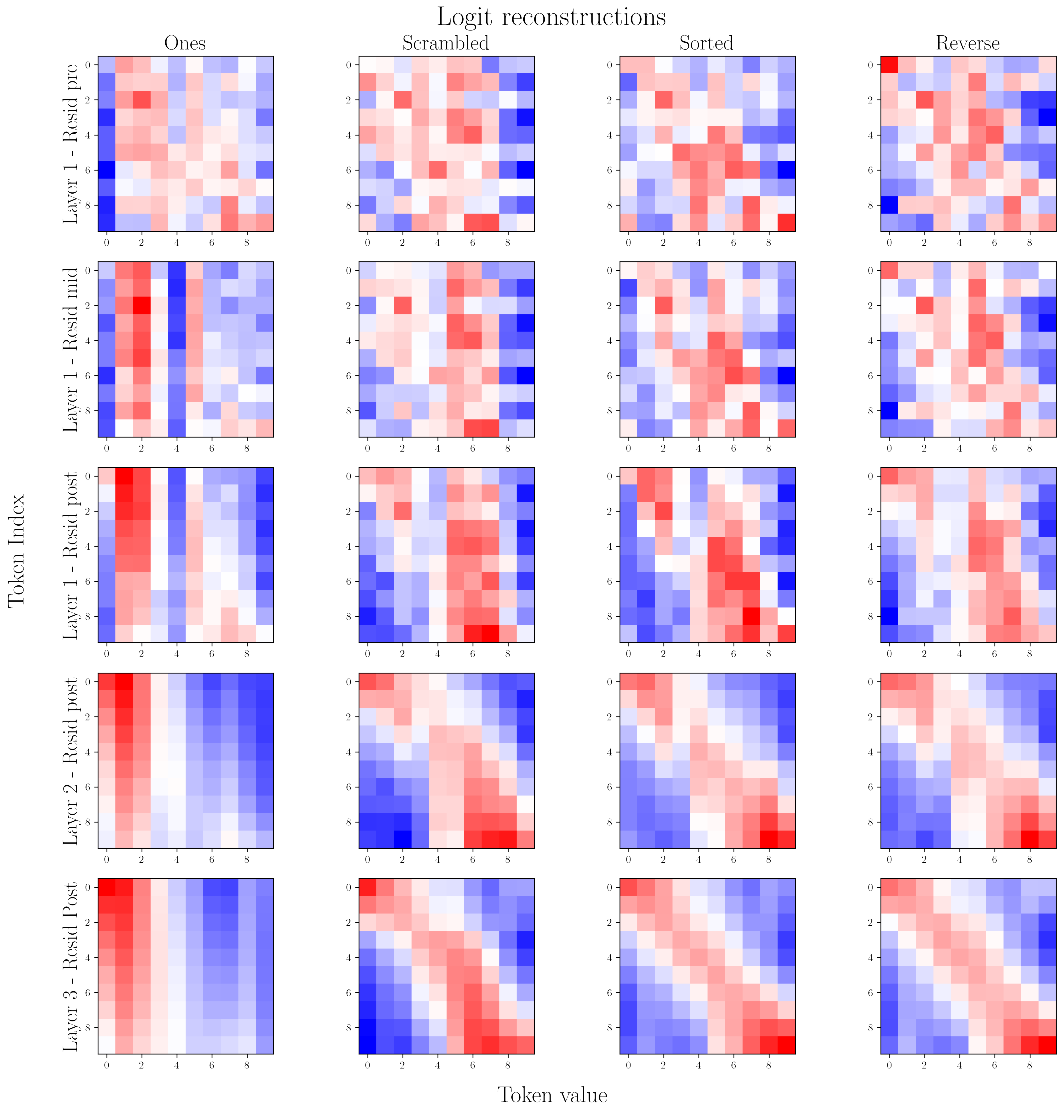

Figure 6: The logit reconstruction generated by unembedding the residual stream at different points.

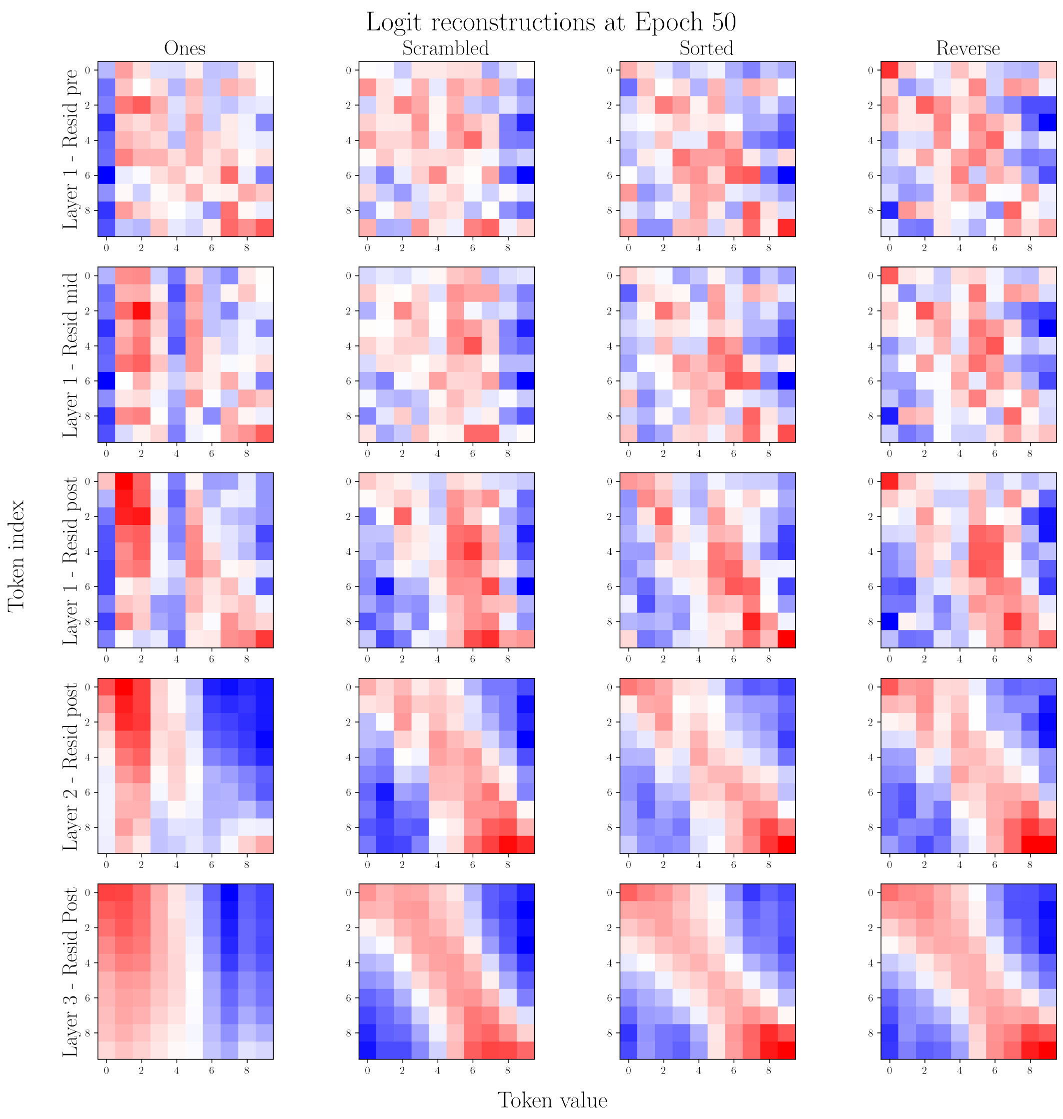

Figure 7: The logit reconstruction generated for the model at checkpoint 50

6.3 Transformer Architecture Details

The transformer architecture used in this work follows the standard setup with the following configuration:

- Number of Layers: 3
- Embedding Dimension: 50
- Feedforward Dimension: 100
- Number of Attention Heads: 1 (per layer)
- Activation: ReLU

Each layer contains a multi-head attention block followed by a position-wise feedforward network. Let $\mathbf{x} \in \mathbb{R}^{n \times d}$ be the input to a layer, where n is the sequence length and d is the embedding dimension.

The attention block first computes query, key, and value projections of the input:

$$\mathbf{Q} = \mathbf{x}\mathbf{W}^Q \ \mathbf{K} \qquad = \mathbf{x}\mathbf{W}^K \ \mathbf{V} = \mathbf{x}\mathbf{W}^V$$

Where $\mathbf{W}^Q \in \mathbb{R}^{d \times d_k}$, $\mathbf{W}^K \in \mathbb{R}^{d \times d_k}$, $\mathbf{W}^V \in \mathbb{R}^{d \times d_v}$ are learnable projection matrices, and d_k, d_v are the dimensions of the query/key and value vectors respectively.

The attention scores are then computed as follows:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

In our case, with a single head, $d_k = d_v = 50$. The attention output is then passed through a feedforward network.

Finally, the outputs of the attention and feedforward networks are combined via a residual connection to produce the layer's output activations.

References

[1] Nelson Elhage et al. *Toy Models of Superposition.* 2022. arXiv: `2209.10652 [cs.LG]`.

[2] Jesse Hoogland et al. "The Developmental Landscape of In-Context Learning". en. In: arXiv:2402.02364 (Feb. 2024). arXiv:2402.02364 [cs]. URL: `http://arxiv.org/abs/2402.02364`.

[3] Leon Lang et al. *When Your AIs Deceive You: Challenges with Partial Observability of Human Evaluators in Reward Learning.* 2024. arXiv: `2402.17747 [cs.LG]`.

[4] Edmund Lau, Daniel Murfet, and Susan Wei. "Quantifying degeneracy in singular models via the learning coefficient". en. In: arXiv:2308.12108 (Aug. 2023). arXiv:2308.12108 [cs, stat]. URL: `http://arxiv.org/abs/2308.12108`.

[5] David Lindner et al. "Tracr: Compiled Transformers as a Laboratory for Interpretability". In: *arXiv preprint arXiv:2301.05062* (2023).

[6] Neel Nanda et al. "Progress measures for grokking via mechanistic interpretability". In: arXiv:2301.05217 (Oct. 2023). arXiv:2301.05217 [cs]. DOI: `10.48550/arXiv.2301.05217`. URL: `http://arxiv.org/abs/2301.05217`.

[7] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[8] Gail Weiss, Yoav Goldberg, and Eran Yahav. "Thinking Like Transformers". en. In: arXiv:2106.06981 (July 2021). arXiv:2106.06981 [cs]. URL: `http://arxiv.org/abs/2106.06981`.

[9] Ziqian Zhong et al. *The Clock and the Pizza: Two Stories in Mechanistic Explanation of Neural Networks.* 2023. arXiv: `2306.17844 [cs.LG]`.