Structured World Representations in Maze-Solving Transformers

Michael I. Ivanitskiy[*†1] **Alex F. Spies**[†23] **Tilman Räuker**[†]
Guillaume Corlouer **Chris Mathwin** **Lucia Quirke** **Can Rager** **Rusheb Shah** **Dan Valentine**
Cecilia Diniz Behn[1] **Katsumi Inoue**[3] **Samy Wu Fung**[1]

[1]Colorado School of Mines, Department of Applied Mathematics and Statistics
[2]Imperial College London [3]National Institute of Informatics, Tokyo

Abstract

Transformer models underpin many recent advances in practical machine learning applications, yet understanding their internal behavior continues to elude researchers. Given the size and complexity of these models, forming a comprehensive picture of their inner workings remains a significant challenge. To this end, we set out to understand small transformer models in a more tractable setting: that of solving mazes. In this work, we focus on the abstractions formed by these models and find evidence for the consistent emergence of structured internal representations of maze topology and valid paths. We demonstrate this by showing that the residual stream of only a single token can be linearly decoded to faithfully reconstruct the entire maze. We also find that the learned embeddings of individual tokens have spatial structure. Furthermore, we take steps towards deciphering the circuity of path-following by identifying attention heads (dubbed *adjacency heads*), which are implicated in finding valid subsequent tokens.

1 Introduction

In recent years, large transformer models have been applied to great effect in various domains, including language modeling, computer vision, and reinforcement learning. The proliferation of such architectures in applied settings has led to increased concern over the generality and robustness of the behaviors they learn. To this end, researchers have begun to study small transformer models on toy tasks to develop a mechanistic understanding of how transformers learn to solve varying classes of problems. The generalizability of findings from toy models to larger scales remains uncertain, but early findings in this direction have given cause for optimism [1].

The most well-known example of a mechanistic component (a *circuit*) found across many transformers models are induction heads [2], which facilitate in-context sequence completion ($A, B, [...], A \rightarrow B$) and arise in transformers with at least 2 layers. While induction heads are fairly simple, they form crucial building blocks of more complex circuits [3, 4]. Identifying complete circuits in more complex models is highly labor intensive, but other methods, such as linear probing and the TunedLens [5], allow researchers to interpret the representations learned by larger models. Indeed, recent work [6] found that a GPT-2 model trained on the game of Othello learned to (linearly [7]) represent the board state in a way which could be easily intervened upon to change the model's future actions.

Code: `github.com/understanding-search/structured-representations-maze-transformers`
[*]Corresponding Author: mivanits@mines.edu
[†]Primary Contributor

37th Conference on Neural Information Processing Systems (NeurIPS 2023). First Workshop on Unifying Representations in Neural Models (UniReps).

With the ultimate goal of better understanding how transformer models perform multi-step reasoning in search-like tasks, we apply the interpretability methods to toy models trained to solve maze tasks. In particular, we experiment with autoregressive transformers trained to solve mazes represented as a list of tokens [8], which constitutes an offline reinforcement learning task with global observations. By varying the precise configurations of these maze solving tasks, we are able to investigate the conditions under which models tend to learn representations with varying degrees of interpretability and generalizability. Additionally, while prior work has found that transformers struggle to perform complex planning tasks [9], we find that relatively small ($< 10^7$ parameters) transformers are capable of solving mazes.

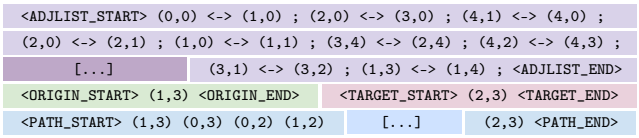
We use various interpretability techniques to study our models, finding that the geometry of their embedding space correlates with the spatial structure of the mazes (subsection 3.2). We find that our highest-performing models form a linear representation of maze connectivity structure, which can be decoded at early layers (subsection 3.4). Lastly, we identify specific attention heads that condition over valid neighbors for a given state, implicating them in path-following behavior (subsection 3.3 and subsection 3.5).

By performing these analyses across models and at different stages in training, we find evidence for grokking-like transitions during training, in which a model's ability to generalize improves rapidly [10]. These increases in generalization performance coincide with the times at which models' internal representations of the maze become more linearly decodable, suggesting that a structured internal representation improves their ability to systematically solve mazes (subsection 3.6).

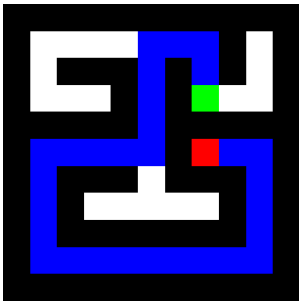
2 Experimental Setting

2.1 Datasets

We use the `maze-dataset` library [8] to generate a variety of mazes and convert them into formats suitable for a text-based autoregressive transformer. Starting with an $n \times n$ lattice, we generate paths using a variety of algorithms. The resulting mazes are converted into tokenized representations, shown in Figure 1b, which are used to train our models. The vocabulary consists of coordinate tokens and various special tokens used to connect coordinates and delimit different parts of the maze and solution description. While `maze-dataset` provides a variety of maze generation algorithms, filters, and configuration parameters, in our interpretability experiments, we focus on mazes generated via 1) Randomized Depth First Search (RDFS), which generates acyclic spanning trees; 2) "forkless" mazes consisting of a sparse tree where each node has at most two connections; 3) Randomized Depth First Search with percolation (pRDFS), which starts with RDFS, but then a `OR` is performed with a maze where adjacent connections have probability $p = 0.1$ of occurring, thus creating mazes which may have cycles.



(a) An example training sequence with four parts. **1:** The adjacency list describes the connectivity of the maze (the order of connections is randomized, ellipses represent omitted connection pairs). **2,3:** The origin and target specify where the path should begin and end, respectively. **4:** The path itself is the shortest sequence of coordinates from the origin to the target. For a "rollout," we provide everything up to (and not including) the `<PATH_START>` token and generate a sequence via `argmax` sampling until the `<PATH_END>` token is produced. For single-token tasks (see Figure 3), we provide a partially complete path and consider only the logits over the immediate next token.



(b) Visual representation of the maze defined from the tokens on the left. The origin is indicated in green, the target in red, and the correct path in blue.

Figure 1: Tokenization scheme and visualization of our shortest-path maze tasks.

2.2 Models and training

All models analyzed are autoregressive decoder-only models, identical to the GPT architecture. While extensive sweeps were performed over hyperparameters, we focused our experiments on two trained models. The first, denoted `hallway`, was trained only on "forkless" mazes and is a smaller model with approximately 1.2M parameters. The second model, `jirpy`, has approximately 9.6M parameters and was trained on sparsely connected mazes of varying sizes with multiple forking points (see subsection 2.1).[1] These two models trade off interpretability and task complexity, with the `hallway` task allowing for a simpler model, while the task for `jirpy` requires decision making at each forking point, potentially yielding a more complex maze representation. Full hyperparameters for our models can be found in the appendix (**??**).

Models were trained to perform next-token prediction on a dataset of randomly generated mazes and paths.[2] At inference time, the models are prompted with a complete adjacency list and path specification (i.e., all tokens up to `<PATH_START>`) and rolled out until they yield a `<PATH_END>` token. It is worth noting that we do not impose any constraints on the validity of a model's output, so a poorly trained model may output nonsensical paths consisting of special tokens or disconnected coordinates.

3 Experiments

To understand our trained maze-solving transformers' behavior and internal representations, we favor a post-hoc interpretability approach [11]. We begin with behavioral experiments on maze-solving trajectories and assess initial path predictions. Next, we explore the embedding space for spatial token relationships and use direct logit attribution [1, 3] to pinpoint model components sensitive to specific sub-tasks. Through linear probes on the residual stream, we decode the presence or absence of walls, revealing structured representations. Lastly, we analyze training metrics to investigate the relationship between the emergence of structured representations and improved generalization performance. Collectively, these experiments shed light on how our transformers adeptly solve mazes.

3.1 Behavioral Experiments

Although several evaluation metrics are computed during the training process, we found visual inspection of generated paths to be useful. Several example rollouts are provided in Figure 2.

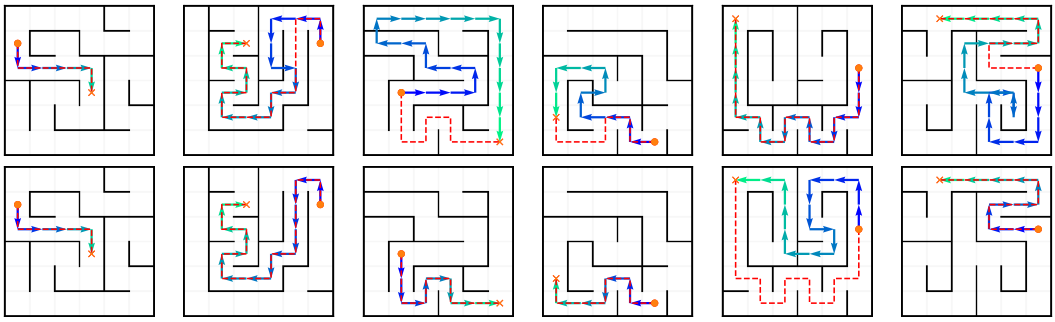


Figure 2: Example generations of `hallway` model (top row) and `jirpy` model (bottom row) on a random sample of held-out RDFS mazes (outside the training distribution of the `hallway` model). The correct path is marked as a red dashed line, with • at the starting position and `x` at the target position. For clarity, generated paths fade from blue to green. Note that both models often violate constraints, such as by passing through walls, and reach the target despite being at a dead end. Further example generations can be found with our codebase.

[1] Chosen as it was the most performant of the models trained to solve complex mazes. See Figure 12.

[2] `jirpy` received gradients only from tokens in the path (including special delimiters), while `hallway` received gradients from the entire sequence.

To facilitate the isolation of specific sub-components of our transformers, which are implicated in certain behaviors, we identify sub-"tasks" of maze solving, which consist of predicting a single token. We describe several such tasks in Figure 3. For each of these, the prompt given to the model consists of all context tokens up to and not including the targeted token. Of particular note in our experiments is the qualitative observation that the models tend to reach the goal at the conclusion of their generations but often violate the constraints in the process[3], as shown in Figure 2 and Table 1.

`<PATH_START>` `(1,3)` `(0,3)` `(0,2) [...] (2,4)` `(2,3)` `<PATH_END>`

Figure 3: Tasks used to assess model performance and their relative locations within a path prediction. From left to right, the target tokens are: `path_start`, `origin_after_path_start`, `first_path_choice`, `rand_path_token_nonend`, `final_before_path_end`, `path_end`. Notably, for hallway-type tasks, the `first_path_choice` task is the only task that requires anything other than simple following of a path and recognition of the origin and target. A `rand_path_token` task is also included, which is similar to `rand_path_token_nonend` in that one of several tokens is selected at random, but for the latter, this pool of possible tokens does not exclude endpoints. Performance on these tasks is shown in Table 1.

In Table 1, we note that on out-of-distribution pRDFS mazes, both models generalize fairly well. We observe that performance on the `first_path_choice` task is consistently the lowest. Performance of `hallway` on 6×6 mazes is slightly lower than on larger mazes (see Table 4), possibly due to the short prompt length being out-of-distribution.

dataset:	forkless		RDFS		pRDFS	
model:	`hallway`	`jirpy`	`hallway`	`jirpy`	`hallway`	`jirpy`
exactly correct rollouts	**38.3%**	**38.7%**	24.2%	82.4%	24.2%	70.7%
valid rollouts	**54.3%**	**53.5%**	37.5%	84.0%	49.6%	87.1%
rollouts with target reached	**87.1%**	**64.5%**	94.5%	99.2%	92.6%	100.0%
`path_start`	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
`origin_after_path_start`	91.0%	86.7%	100.0%	100.0%	100.0%	100.0%
`first_path_choice`	**71.5%**	**66.4%**	**67.2%**	**86.7%**	**66.4%**	**84.4%**
`rand_path_token`	93.0%	87.1%	90.2%	98.0%	84.0%	94.5%
`rand_path_token_nonend`	97.3%	89.8%	92.2%	99.2%	84.4%	97.3%
`final_before_path_end`	95.7%	85.9%	93.4%	100.0%	84.4%	100.0%
`path_end`	86.7%	71.5%	100.0%	99.6%	100.0%	100.0%

Table 1: **Setup:** Performance across tasks of the `hallway` model and `jirpy` model assessed on held-out 6×6 forkless, RDFS, and pRDFS mazes (See subsection 2.1 and [8]). All values are binary, since we perform a single rollout per maze, and score for a task is `argmax(logits) == correct_token`. **Tasks:** The first group of metrics deals with sequence generations or "rollouts," as detailed in subsection 2.1. A rollout is "exactly correct" if no deviations from the shortest path occur (pRDFS mazes may not have a unique shortest path, and thus the provided values are a *lower bound*). A rollout is "valid" if it obeys the topology of the maze (no wall jumps), but backtracking is permitted. A rollout is considered to have reached the target if the final coordinate token is the target token. The second group of single-token tasks used to assess performance are detailed in Figure 3. Data for 7×7 mazes is provided in Appendix B.

3.2 Emergent Structure in the Embedding Space

As in other language models, each token in the vocabulary corresponds to a unique orthogonal unit vector. In our experiments, each coordinate on the lattice has a single corresponding token. The embedding layer of our models maps each vocabulary vector from an input sequence to a dense vector in $\mathbb{R}^{d_{\texttt{model}}}$. Since each vocabulary vector is orthogonal, no spatial structure is encoded into the model directly; however, a spatial structure emerges after we train the model. In particular, we note that a correlation between the coordinate distance and distance between embedding vectors emerges for short distances (Figure 4). Note that proximity of tokens in the sequences alone is not enough to allow this behavior to be learned due to the randomization of the adjacency list.

[3] I.e. its output sequence is often of the form " `[...invalid path...], (goal), <PATH_END>`".

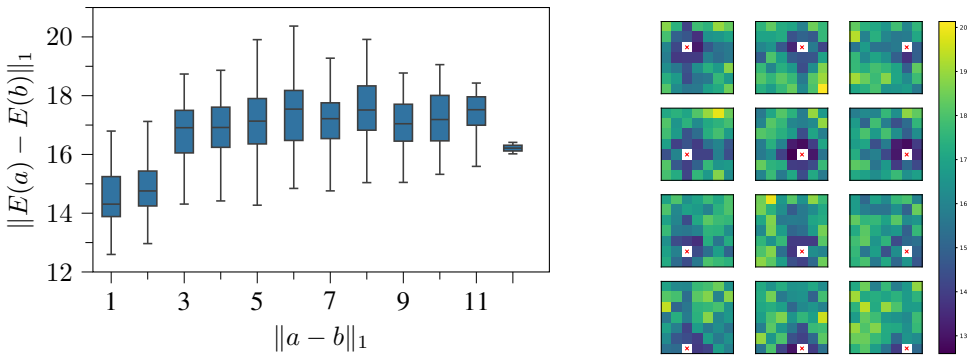


Figure 4: Structure of coordinate token embeddings for the `hallway` model. Given two coordinates a, b and the embeddings of their corresponding tokens $E(a), E(b)$ we observe the relationship between the Manhattan (1-norm) distances. Note that all coordinates have orthogonal vocabulary vectors, and the embeddings are learned. **Left:** Correlation between coordinate distance and embedding distance. **Right:** Given the embedding of the coordinate at the `x`, Manhattan distance to the other tokens on the grid is displayed. **Note:** full data for all models can be seen in Appendix C.

3.3 Direct Logit Attribution

To investigate path-following behavior, we utilize direct logit attribution (DLA) [1, 3], which measures the direct contribution of an isolated component of the network (e.g., an attention head) to a given set of forward passes. We utilize the tasks defined in subsection 3.1 for this correlational analysis (Figure 5). Specifically, we compute the contribution $C_{l,h}$ of head h at layer l to the probability assigned by the model to the correct next token. We do this by empirically estimating (over samples $(p, c) \in \mathcal{D}$) the dot product of the output[4] of that head $R_{l,h}(p)$ with the difference between the embedding of the correct token $E(c)$ and some reference embedding $r(c)$.

$$C_{l,h} = \frac{1}{|\mathcal{D}|} \sum_{(p,c)\in\mathcal{D}} \left[\texttt{LayerNorm}\left(R_{l,h}(p)\right) \cdot \left(E(c) - r(c)\right)\right] \qquad \text{where} \qquad r(c) = \frac{1}{|\mathcal{V}|-1} \sum_{t\in\mathcal{V}\setminus c} E(t)$$

Where $\mathcal{V}$ is the set of vocabulary vectors, we compute[5] the reference embedding as a mean of the embeddings of all tokens except c. In this work, the DLA analysis serves to locate attention heads of interest, which we subsequently investigate. Our future work will include ablations and other interventions on model architecture to establish causal relationships between these attention heads and path-following performance.

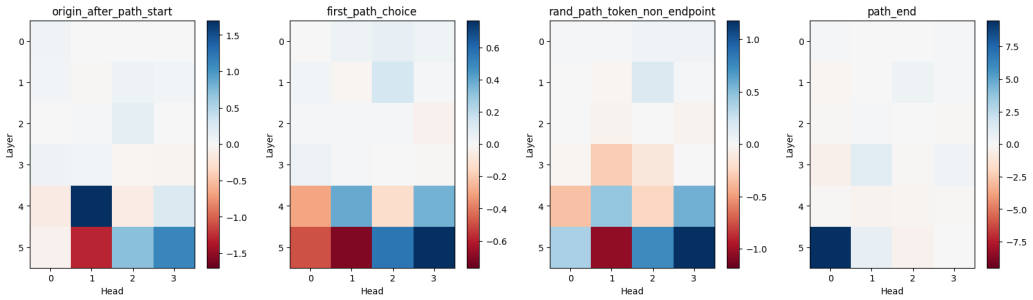


Figure 5: DLA of the `hallway` model across a subset of tasks, on held-out samples from the training distribution. The numerical value is the contribution of a given attention head to the "correct" direction in the residual stream. Note that only for `first_path_choice` must the model do anything besides path-following, and this is shown in the performance statistics of Table 1.

[4]Note that the application of LayerNorm is done to match the actual scaling at layer l, see `ActivationCache.apply_ln_to_stack()` in [12].

[5]In a manner which is not common practice, to our knowledge.

Upon investigation of attention placed on tokens as a function of their distance from the current token, we find that Layer 5, Head 0 simply places attention on the recent occurrences of the current coordinate token. This is throughout the whole sequence, but primarily between the target specification tokens. As such, its lack of involvement in `origin_after_path_start` becomes clear since the current token, in that case, would be the `<PATH_START>` token and thus not a coordinate token.

Also of interest is Layer 1, Head 2. We find that consistently across tasks, this head places attention on the `<TARGET_END>` token. We hypothesize that this head is a component of an induction head [2] but operating in reverse – a later head likely attends to the token before `<TARGET_END>` to find the target. This information may then be used to inform the model's choices of which path to take at forks, as well as identifying when the path is concluded.

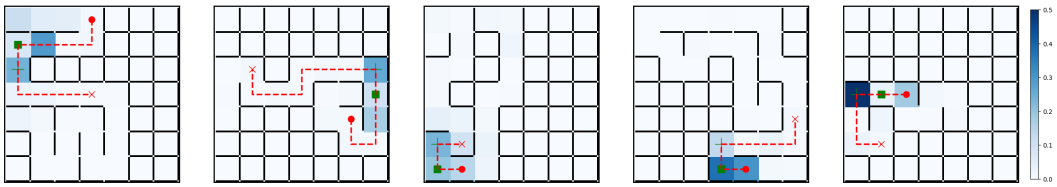


Figure 6: Attention of Layer 5, Head 3, on the `rand_path_token_nonend` task. Attention is displayed over the maze positions for five random held-out mazes. Blue shading is attention weight, true path is red dashed line from ● to x, current position is ■.

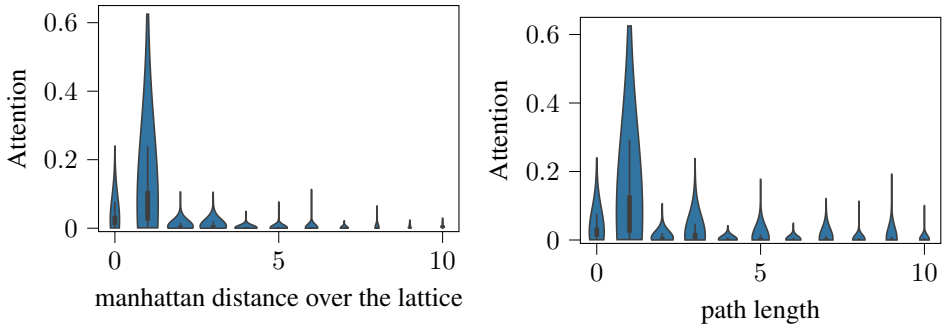


Figure 7: Attention of Layer 5, Head 3, on the `rand_path_token_nonend` task: violin plots of attention as a function of the distance between the current node and the node being attended to. x-axis on the left is the pure manhattan distance between the notes, while x-axis on the right is the path length between the nodes. Sample size $n = 200$. Note that while on the right, attention is overwhelmingly applied to nodes path length 1 away, some attention is applied to nodes at odd path lengths away because a node adjacent to the lattice will always be an odd path length away.

As observed in Figure 6 and Figure 7, the attention head at layer 5, head 3, which we term an *Adjacency Head*, consistently attends to tokens of path length 1 from the current position and thereby learns to *respect the maze's topology*. This differs from the results of subsection 3.2 in that the embedding map, since it processes each token individually, can only correlate vectors that are close on the lattice (shared across all training runs) and cannot see the topology which can only be learned in-context.

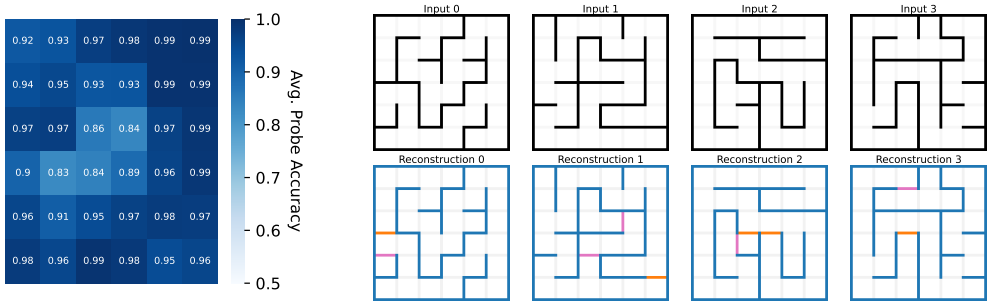
3.4 Learned internal representations

To assess whether the models learn to internally represent mazes, we follow the approach in [7] and train a set of linear probes to predict the ground-truth maze structure from a single latent vector. In particular, for a maze with $m \times m$ positions, we train $n_{\texttt{layers}} \times m \times m \times 4$ probes p on residual stream activations $R_l(t)$ collected across many rollouts, such that

$$\{[R_l(t) \cdot p_l(x, y, \texttt{dir})] > 0.5\} = \texttt{wall}(x, y, \texttt{dir})$$

where the token, t, is fixed and all layers, l, are considered. In essence, if the dot product between a particular direction `dir` probe with $R_l(t)$ exceeds 0.5, then this should reflect the presence of a wall in the input maze at that particular probing location. For all experiments, we take the token, t, to be `<PATH_START>`, as it is the final token presented in each sequence at test time and will have seen all previous tokens.

By looking at the variation in probing accuracy across layers and throughout training, we can understand how the formation of the world model occurs and potentially contributes to the model's performance. We focus our discussion here on `jirpy` as it was the most performant model, both in terms of solving mazes and yielding the best set of probes (see Figure 16 for the results on all layers). In Figure 8 we show the examples of mazes decoded at layer 2 with a set of probes that achieved the highest accuracy. Figure 9a shows that the maze representation was already learned by the second layer (more examples are shown in Figure 17 and Figure 18). Results of sweeps across different transformers are shown in Figure 12.



(a) Probe set accuracy across maze positions for layer 2 (for which `jirpy` yielded the most accurate probes).

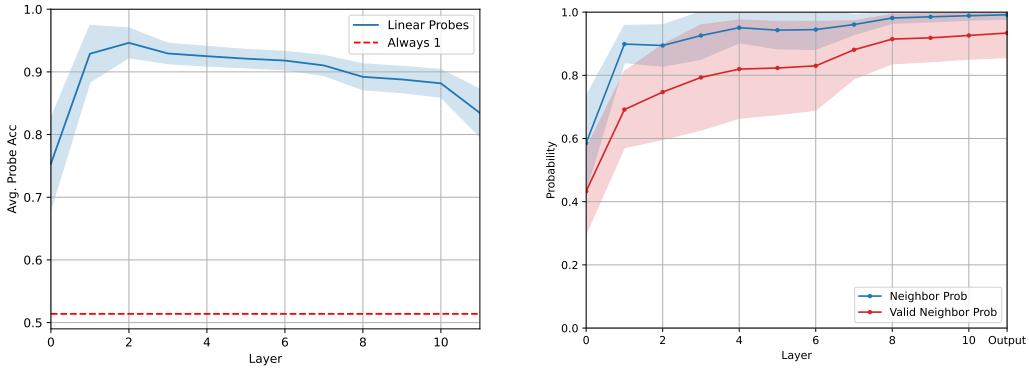
(b) Four random examples of mazes reconstructed using the probes. Reconstructions are made from a set of probes using a single latent embedding of the `<PATH_START>` token at layer 2. Wall colors indicate that thresholded probes Correctly Predicted, Omitted or Added a wall.

Figure 8: Analysis of Linear Probes applied to the `<PATH_START>` token for the `jirpy` model.

3.5 Investigating Neighbor information through Tuned Lens

To further analyze the latent representations learned by our models, we apply the Tuned Lens method introduced in [5]. The Tuned Lens provides a direct view into the information encoded at each layer, l, by learning a linear transformation $\mathbf{L}_l : \mathbb{R}^{d_{\texttt{model}}} \rightarrow \mathbb{R}^{d_{\texttt{model}}}$ (referred to as a "translator") which attempts to map embeddings to their final state (after the last layer), i.e., $\mathbf{L}_l(R_l(t)) = R_{l_{\texttt{final}}}(t)$. By applying these learned translators, we are able to unembed (into the vocabulary) embeddings from any layer in the model, thus gaining insight into what the model has captured after performing a few layers of computation.

We apply the Tuned Lens approach to see at which layers models write information about neighbors onto coordinate tokens; this includes connected neighbors (those not blocked by walls) and all neighbors (all adjacent coordinates). The resulting analysis for the `jirpy` model is shown in Figure 9b. We see that after the first layer, the residual stream already contains significant information concerning whether the next token in a path will be a coordinate, coinciding with the layer in the model where a linear representation of the maze is captured most clearly. This information is then refined gradually throughout the model, such that at later layers, the validity of the next token is enforced more strongly, perhaps owing to the effects of the heads identified in (subsection 3.3).



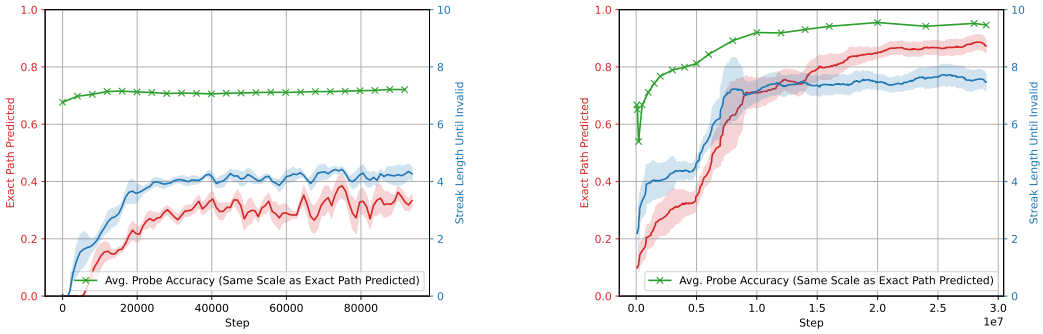
(a) Accuracy of linear probes averaged across all coordinate positions and wall directions in 15,000 mazes. In the appendix, we show that across models with high accuracy and a linearly decodable representation of the maze ($> 90\%$ accuracy), $4/5$ of these were most effective at layer 2, and $1/5$ at layer 3. Models that performed poorly often acquired such representations at later layers, but the resulting accuracies of probe sets never exceeded 80%.

(b) Results from the TunedLens. For each layer, we compute the probability mass given to connected, or merely adjacent neighboring coordinates for all coordinate tokens in the path gathered across 50 rollouts. We observe that valid neighbors become relatively more probable after layer 2, and the most significant variations in neighbor probability are aligned with the layers in the model when the maze representation is most clear.

Figure 9: Analysis of Linear Probes and Tunes Lens applied to the `<PATH_START>` token for the most performant transformer (`jirpy`). Colored regions correspond to 1σ.

3.6 When Do Models Learn to Represent the Maze?

Prior work has shown that the phenomenon of grokking [10, 13], in which the test accuracy (i.e., the generalizability of a model's learned behavior) improves abruptly during training, may be linked to the formation of structured representations over which a task can be robustly solved [13]. As we established in subsection 3.4 that models learn linearly structured representations of mazes, it is a natural question to ask when these are learned and if they co-occur with any notable changes in a model's performance during training. To this end, we trained probe sets across checkpoints for both the `hallway` and `jirpy` models, with results shown in Figure 10. We find that the `hallway` model does not learn a clear linear representation of the maze, while `jirpy` does. Furthermore, the periods during training in which these representations improve the most also correspond to the times at which the model's performance improves most sharply. This provides suggestive but incomplete evidence for the possibility that these representations play a causal role in the model's behavior.



(a) `hallway`

(b) `jirpy`

Figure 10: Layerwise analysis of maze structure captured by the model. Note that the distribution of paths for hallway mazes is shorter than those for forking mazes.

4 Related Work

Transformers' ability to solve inherently difficult tasks is increasingly being explored. In particular, the capability of transformers to process semantic information and emulate program behavior, especially with structural recursion, has been investigated [9, 14, 15].

Other research on transformers suggests that some performance may be attributed to an architectural bias towards mesa-optimization [16]. Here, it is argued that transformers employ mesa-optimization during their forward pass, constructing an internal learning objective and optimizing it. Akyurek et al. note that transformers might harness standard learning algorithms implicitly, encoding miniature models within their activations and updating these based on incoming examples [17].

Finding Meaningful Directions in Activation Space: Mechanistic interpretability seeks to reverse engineer neural networks. In the pursuit of this ambitious approach to interpretability, several techniques have been proposed in an effort to find and understand meaningful directions in a model's activation spaces. Belrose et al.'s Tuned Lens [5] involves training affine transformations that translate the basis associated with representations in any single layer's activation space with the expected basis of that of the final layer. Such transformations, when coupled with the model's unembedding layer, can be used to map the residual stream to a distribution over the model's vocabulary. Sparse Coding employs autoencoders augmented with sparsity regularization to derive disentangled representations of an activation space; this approach has been researched in works by Bricken et al. [18]. Other efforts to find meaningful directions in activation space include using k-sparse linear classifiers that map the activations of a single neuron or a collection of neurons to specific features [19].

World Models: Much recent research has been focused around finding and understanding world models, especially in planning tasks. Li et al. [20] studied world representations in the game of Othello, with Nanda et al. [7] further investigating the linearity of these representations. Turner et al. [21] focused on reinforcement learning, examining maze-solving tasks and the underlying representations. Additionally, the introduction of mechanistic interpretability for decision transformers by Bloom et al. [22] offers a new perspective on interpretability in strategic planning tasks. Together, these studies provide valuable insights into the structure and utility of world models across different contexts.

5 Conclusion

We demonstrate that transformers trained to solve mazes acquire emergent linear representations that capture maze connectivity and are encoded in a single token's latent state. The embedding layer of trained models is shown to learn an emergent spatial structure. Furthermore, we find that in simple models, some attention heads learn to respect the topology of the maze and present some evidence and hypotheses as to their function. In future work, we aim to construct a more complete mechanistic picture of how these elementary heads operate over the linear world model and ultimately form circuits responsible for solving mazes. Additionally, future work will investigate the generality of such emergent models by investigating distinct classes of neural networks trained to perform the same tasks over entirely different input representations in an attempt to provide further evidence for claims of representational "universality". With this work, we hope to inspire other researchers to investigate the seemingly systematic yet elusive internal behavior of transformer models.

Acknowledgments and Disclosure of Funding

We are grateful to AI Safety Camp for initially supporting this project and bringing many of the authors together. This work was partially funded by National Science Foundation awards DMS-2110745 and DMS-2309810.

References

[1] Tom Lieberum, Matthew Rahtz, János Kramár, et al. Does circuit analysis interpretability scale? evidence from multiple choice capabilities in chinchilla, 2023. URL http://arxiv.org/abs/2307.09458. → p 1, 3, and 5

[2] Catherine Olsson, Nelson Elhage, Neel Nanda, et al. In-context learning and induction heads, 2022. URL http://arxiv.org/abs/2209.11895. → p 1, and 6

[3] Kevin Wang, Alexandre Variengien, Arthur Conmy, et al. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small, 2022. URL http://arxiv.org/abs/2211.00593. → p 1, 3, and 5

[4] Arthur Conmy, Augustine N. Mavor-Parker, Aengus Lynch, et al. Towards automated circuit discovery for mechanistic interpretability, 2023. URL http://arxiv.org/abs/2304.14997. → p 1

[5] Nora Belrose, Zach Furman, Logan Smith, et al. Eliciting Latent Predictions from Transformers with the Tuned Lens, Mar 2023. URL http://arxiv.org/abs/2303.08112. → p 1, 7, and 9

[6] Kenneth Li, Aspen K Hopkins, David Bau, et al. Emergent world representations: Exploring a sequence model trained on a synthetic task, 2022. URL http://arxiv.org/abs/2210.13382. → p 1

[7] Neel Nanda. Actually, othello-gpt has a linear emergent world model, Mar 2023. URL https://neelnanda.io/mechanistic-interpretability/othello. → p 1, 7, and 9

[8] Michael I. Ivanitskiy, Rusheb Shah, Alex F. Spies, et al. A Configurable Library for Generating and Manipulating Maze Datasets, Sep 2023. URL http://arxiv.org/abs/2309.10498. → p 2, 4, and 11

[9] Ida Momennejad, Hosein Hasanbeig, et al. Evaluating Cognitive Maps and Planning in Large Language Models with CogEval, September 2023. URL http://arxiv.org/abs/2309.15129. → p 2, and 9

[10] Neel Nanda, Lawrence Chan, Tom Lieberum, et al. Progress measures for grokking via mechanistic interpretability, January 2023. URL http://arxiv.org/abs/2301.05217. → p 2, and 8

[11] Tilman Räuker, Anson Ho, Stephen Casper, and Dylan Hadfield-Menell. Toward transparent ai: A survey on interpreting the inner structures of deep neural networks, 2023. URL http://arxiv.org/abs/2207.13243. → p 3

[12] Neel Nanda and Joseph Bloom. Transformerlens. https://github.com/neelnanda-io/TransformerLens, 2022. → p 5

[13] Ziming Liu, Ouail Kitouni, Niklas Nolte, et al. Towards Understanding Grokking: An Effective Theory of Representation Learning, October 2022. URL http://arxiv.org/abs/2205.10343. → p 8

[14] Shizhuo D. Zhang, Curt Tigges, Stella Biderman, et al. Can Transformers Learn to Solve Problems Recursively?, June 2023. URL http://arxiv.org/abs/2305.14699. → p 9

[15] Chang Liu and Bo Wu. Evaluating large language models on graphs: Performance insights and comparative analysis, 2023. URL http://arxiv.org/abs/2308.11224. → p 9

[16] Johannes von Oswald, Eyvind Niklasson, Maximilian Schlegel, et al. Uncovering mesa-optimization algorithms in Transformers, Sep 2023. URL http://arxiv.org/abs/2309.05858. → p 9

[17] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? Investigations with linear models, 2023. URL http://arxiv.org/abs/2211.15661. → p 9

[18] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, et al. Towards monosemanticity: Decomposing language models with dictionary learning, 2023. URL https://transformer-circuits.pub/2023/monosemantic-features/index.html. → p 9

[19] Wes Gurnee, Neel Nanda, Matthew Pauly, et al. Finding Neurons in a Haystack: Case Studies with Sparse Probing, May 2023. URL http://arxiv.org/abs/2305.01610. → p 9

[20] Kenneth Li, Aspen K. Hopkins, David Bau, et al. Emergent World Representations: Exploring a Sequence Model Trained on a Synthetic Task, Feb 2023. URL http://arxiv.org/abs/2210.13382. → p 9

[21] Ulisse Mini, Peli Grietzer, Mrinank Sharma, Austin Meek, Monte MacDiarmid, and Alexander Matt Turner. Understanding and controlling a maze-solving policy network, Oct 2023. URL https://arxiv.org/abs/2310.08043. → p 9

[22] Joseph Bloom. A Mechanistic Interpretability Analysis of a GridWorld Agent-Simulator (Part 1 of N), May 2023. URL https://www.lesswrong.com/posts/JvQWbrbPjuvw4eqxv/a-mechanistic-interpretability-analysis-of-a-gridworld-agent. → p 9

A complete set of notebooks and supplementary data can be found with our codebase:
github.com/understanding-search/structured-representations-maze-transformers

A Model Hyperparameters

Model	$d_{\texttt{model}}$	$d_{\texttt{head}}$	$n_{\texttt{layers}}$	total parameters	dataset config	training dataset size
`hallway`	128	32	6	$\approx 9.64 \cdot 10^6$	`dfs(do_forks=false)`	3×10^6
`jirpy`	256	16	12	$\approx 1.24 \cdot 10^6$	varied	5×10^6 (6 Epochs)

Table 3: Hyperparameters for the two models we investigate. Both models are trained via the `AdamW` optimizer with a learning rate of 10^{-4} and batch size of 32. A single training epoch is performed.

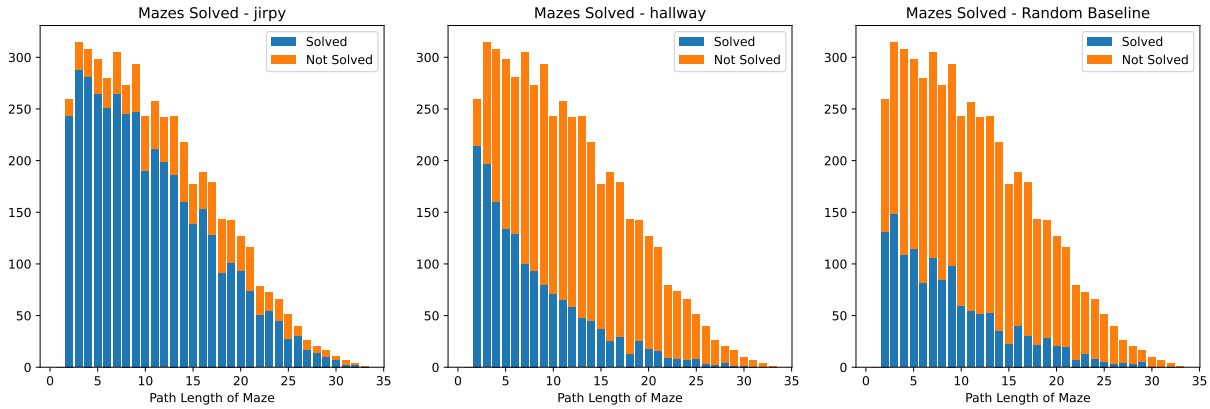
B Performance Statistics

dataset:	forkless		RDFS		pRDFS	
model:	`hallway`	`jirpy`	`hallway`	`jirpy`	`hallway`	`jirpy`
exactly correct rollouts	36.3%	52.7%	36.7%	29.3%	17.6%	21.1%
valid rollouts	50.8%	59.8%	41.4%	29.7%	74.2%	26.6%
rollouts with target reached	87.1%	80.5%	76.6%	98.4%	28.9%	98.4%
`path_start`	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
`origin_after_path_start`	94.5%	95.3%	100.0%	100.0%	97.3%	100.0%
`first_path_choice`	70.3%	70.3%	65.6%	60.2%	66.8%	54.3%
`rand_path_token`	91.0%	89.8%	91.8%	88.3%	86.7%	78.1%
`rand_path_token_nonend`	96.9%	93.8%	97.3%	88.7%	83.6%	79.7%
`final_before_path_end`	98.8%	87.5%	94.5%	95.7%	82.8%	90.6%
`path_end`	88.3%	81.6%	100.0%	100.0%	99.6%	100.0%

Table 4: Same as Table 1, but for 7×7 forkless, pRDFS, and RDFS mazes (See subsection 2.1 and [8]), also with $n = 256$ samples. The second group of single-token tasks used to assess performance are detailed in Figure 3. Note that the `jirpy` model was trained only mazes that were only dense in 6×6 subgrids, and generalizes relatively poorly to these larger mazes. Conversely, the `hallway` model was trained on only sparse mazes to start with, and performs similarly.

dataset:	forkless		RDFS		pRDFS	
model:	`hallway`	`jirpy`	`hallway`	`jirpy`	`hallway`	`jirpy`
exactly correct rollouts	**38.3%**	**38.7%**	24.2%	82.4%	24.2%	70.7%
valid rollouts	**54.3%**	**53.5%**	37.5%	84.0%	49.6%	87.1%
rollouts with target reached	**87.1%**	**64.5%**	94.5%	99.2%	92.6%	100.0%
`path_start`	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
`origin_after_path_start`	91.0%	86.7%	100.0%	100.0%	100.0%	100.0%
`first_path_choice`	**71.5%**	**66.4%**	**67.2%**	**86.7%**	**66.4%**	**84.4%**
`rand_path_token`	93.0%	87.1%	90.2%	98.0%	84.0%	94.5%
`rand_path_token_nonend`	97.3%	89.8%	92.2%	99.2%	84.4%	97.3%
`final_before_path_end`	95.7%	85.9%	93.4%	100.0%	84.4%	100.0%
`path_end`	86.7%	71.5%	100.0%	99.6%	100.0%	100.0%

Table 5: An exact repeat of Table 1, provided here for ease of comparison with Table 4.



(a) `jirpy` (b) `hallway` (c) Random Baseline

Figure 11: Performance of our models on a held-out test set of RDFS mazes. Noticeably, the performance appears roughly consistent over path length for `jirpy` which was trained to solve mazes, and similarly for the random baseline (which follows corridors and chooses a random continuation of its path when reaching a fork). The hallway model, on the other hand, is able to "solve" some of the very short mazes but struggles with longer mazes (where it is likely to encounter forking points).

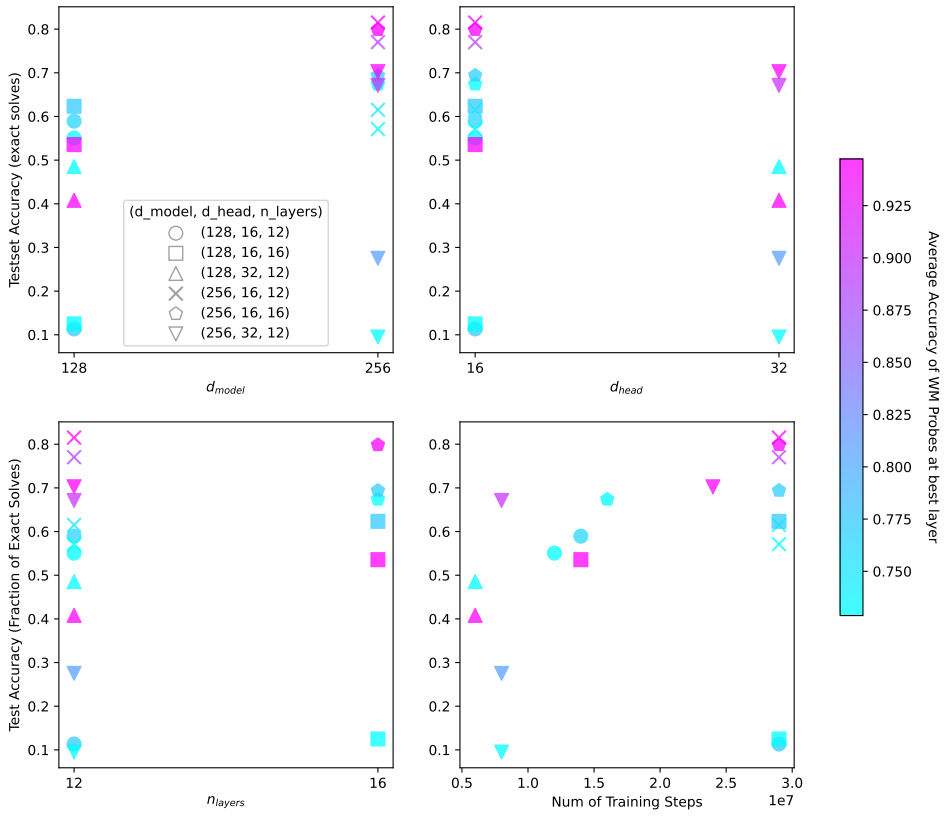


Figure 12: Results of sweeps carried out across a variety of models which were trained for varying lengths. We find that 1) Models can do well even if they don't have a linear maze representation, which we can decode, but the very best models also have a linear maze representation. 2) d_{model} and training time are the only hyperparameters that seemed to correlate with performance in the regimes we considered, but any such correlations are too weak to draw strong conclusions. The model with the highest test accuracy and probe accuracy is `jirpy`.

C Embedding Structure

Appendix of embedding structure results to Figure 4.

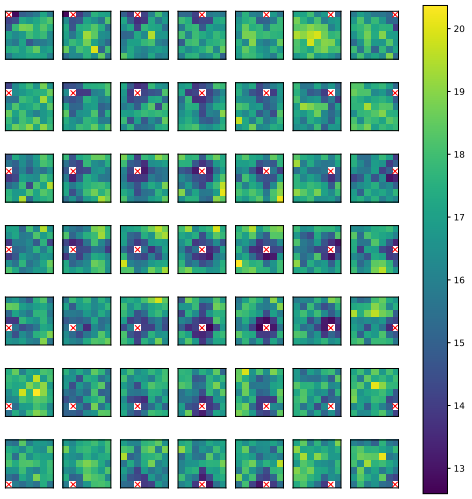


Figure 13: Given the coordinate at the x, we view the Manhattan distance to the other tokens on the grid for the `hallway` model.

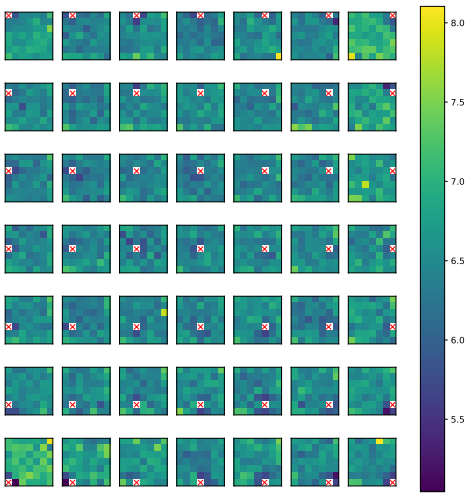


Figure 14: Given the coordinate at the x, we view the Manhattan distance to the other tokens on the grid for the `jirpy` model.

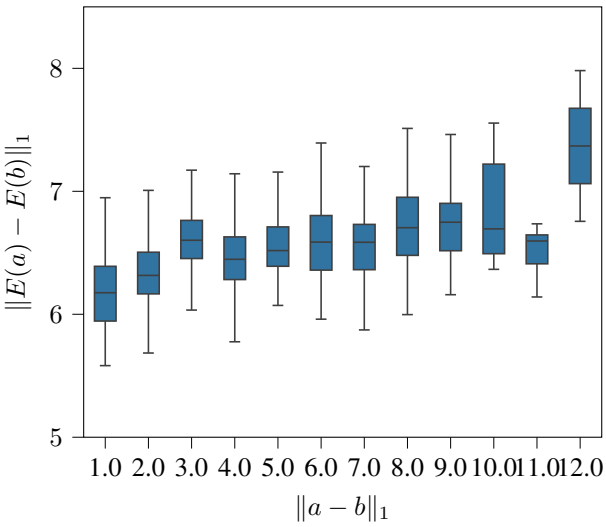


Figure 15: Correlation of coordinate and embedding distance for the `jirpy` model.

D Additional Probing Results

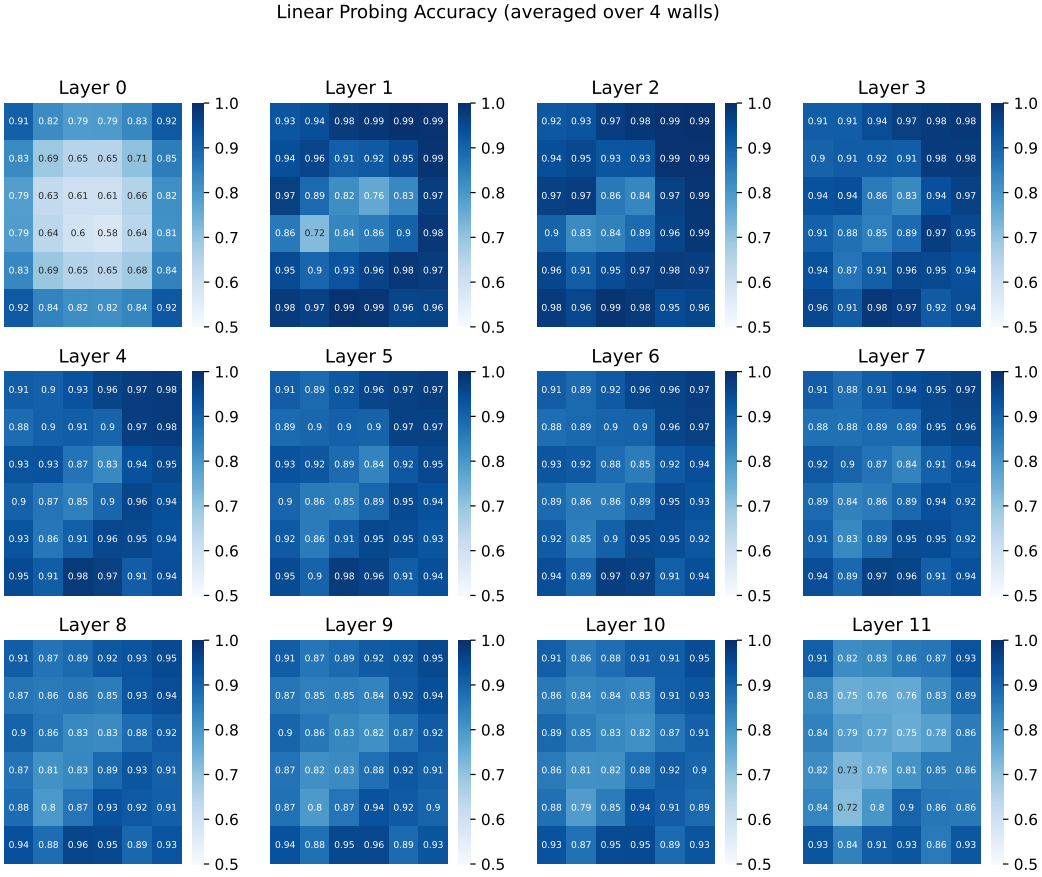


Figure 16: Accuracy of linear probes trained on the residual stream at all layers of the `jirpy` model. We see that on Layer 2 the highest average probing accuracy is achieved, and the accuracy decreases with later layers. Note that the edges have very high accuracy as outer walls are always present, meaning that 1/4 of the probes on edges (and 2/4 on corners) will achieve 100% accuracy by always predicting a wall.

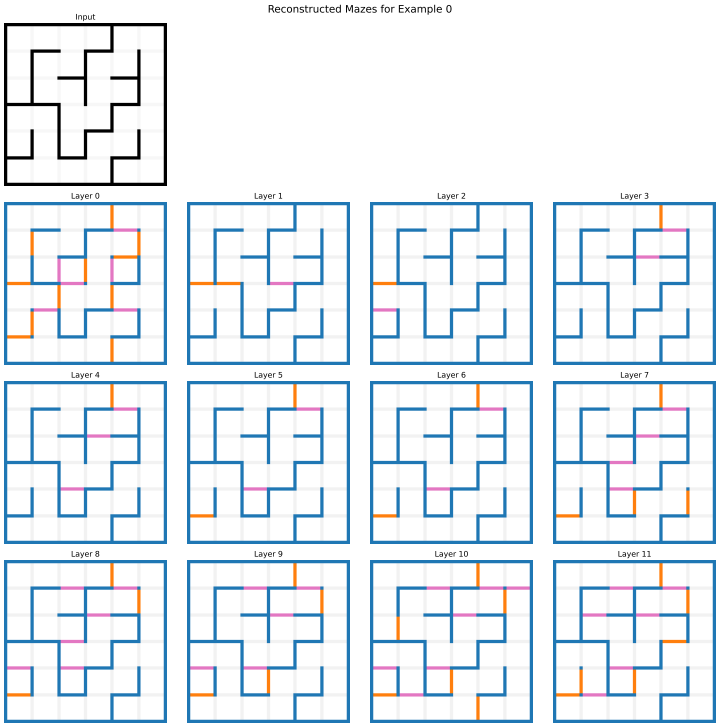


Figure 17: Decoding a maze with probe sets trained for each layer of the `jirpy` model. Wall colors indicate that thresholded probes Correctly Predicted, Omitted or Added a wall.

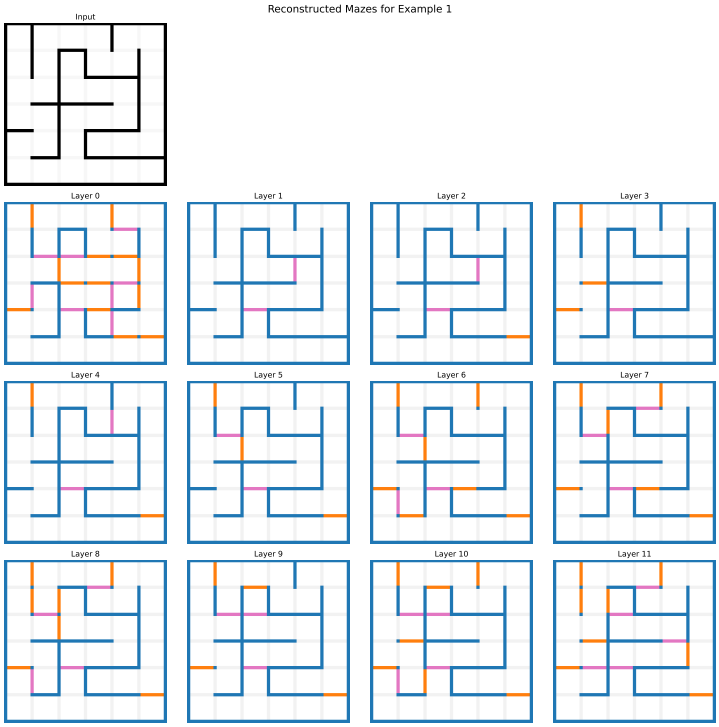


Figure 18: Decoding a maze with probe sets trained for each layer of the `jirpy` model. Wall colors indicate that thresholded probes Correctly Predicted, Omitted or Added a wall.