Better World Models Can Lead to Better Post-Training Performance

Prakhar Gupta
University of Michigan
prakharg@umich.edu

Henry Conklin
Princeton University
henry.conklin@princeton.edu

Sarah-Jane Leslie
Princeton University
sjleslie@princeton.edu

Andrew Lee
Harvard University
andrewlee@g.harvard.edu

Abstract

In this work we study how explicit world-modeling objectives affect the internal representations and downstream capability of Transformers across different training stages. We use a controlled 2x2x2 Rubik's Cube and ask: (1) how does explicitly pretraining a world model affect the model's latent representations, and (2) how does world-model quality affect the model's performance after reinforcement learning post-training? We compare standard next-token prediction to two explicit world-modeling strategies – (i) state-prediction pretraining and (ii) a joint state-prediction + next-token objective – and assess task performance after Group Relative Policy Optimization (GRPO) is applied as post-training. We evaluate the representation quality with linear probes and causal interventions. We find that explicit world-modeling yields more linearly decodable and causally steerable state representations. More importantly, we find that improved state representations lead to higher gains for GRPO, especially on harder cube states. Our results indicate that sharpening state representations can improve the effectiveness of post-training for sequence-planning tasks.[1]

1 Introduction

Language models have achieved impressive capabilities for various reasoning tasks. These models typically go through multiple training stages, including pre-training on generic data, fine-tuning on task-specific data, and post-training using reinforcement learning to further improve on the task. However, it is unclear how each stage affects the internal representations of the model, and in turn how such representations affect the latter stages.

We study these questions in a controlled setting. Namely, we train Transformers on a task that requires planning: solving a 2x2x2 Rubik's Cube.

Not only does the task require planning, but also requires the model to learn a "world model", by which the model must understand a latent cube state (i.e., "world"), and how its predictions (actions) affect the cube state.

Interestingly, researchers have demonstrated that language models can implicitly learn world models when trained via next-token predictions. For example, a model trained on transcripts of *moves* being played in a board game such as Othello can learn to model the latent board-state, despite never given any priors regarding the board [6, 8].

[1] Code: https://github.com/prakharg55/CubeLM-NeurIPS-MI

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Mechanistic Interpretability.

However, these works only show the emergence of a world model, but do not study its relationship with its downstream training stages nor capabilities. Thus in this work we ask two questions:

RQ1 How does explicitly pre-training a world model affect the representations of the model?

RQ2 How does the quality of world models affect the model's accuracy post-training?

From our experiments we find that (1) explicitly pre-training a world model leads to a more robust representation, in terms of higher probe accuracies as well as better steerability of the model, suggesting a higher reliance of the model on its latent cube state representations, and (2) an improved world model can also lead to improved planning accuracy after applying reinforcement-learning.

2 Setups, Notations, Data

We study a Transformer model trained to solve 2x2x2 Rubik's Cubes, which consist of 6 faces and 4 squares per face. We formulate our task as sequence modeling with the following data: $\mathcal{D} = \{(\mathcal{S}^i, \mathcal{A}^i)\}_{i=0}^N$ where $\mathcal{S}$ is a scrambled cube state representing the state of all 24 squares: $\mathcal{S} := [x_0, \dots x_{23}]$ and $\mathcal{A}$ is the *optimal* sequence of moves that solves the cube: $\mathcal{A} := [m_0, \dots, m_n]$, where n is the cube distance for the initial state S^i from being solved. Each state token x specifies a square's color ($x \in \mathcal{C}$, $|\mathcal{C}| = 6$). Each move token m specifies an action ($m \in \mathcal{M}$; using Singmaster's notations,[2] $|\mathcal{M}| = 9$). Importantly, note that intermediate cube states from applying moves m do not show up in the data, and must be implicitly learned by the model. Lastly, note that $n < 12$ (a scrambled cube can only be at most 11 moves away from being solved).

We notate the hidden states of the model $\mathbf{h}^\ell$, token embeddings $\mathcal{E}$, and token unembeddings $\mathcal{U}$. We experiment with a range of training setups, as described below.

2.1 Training Setups

Standard Fine-Tuning. In our standard setting, we train on $\mathcal{D}$ with the standard next-token prediction objective. Namely, our model is given as input a scrambled cube $\mathcal{S}$ and is trained to auto-regressively predict sequence $\mathcal{A}$:

$$\mathcal{L}_{FT} = \text{CrossEntropy}(\mathcal{U}\mathbf{h}_t^{L-1}, \mathcal{A}_t)$$

Pre-training a World Model. In some settings, we add an optional pre-training step to explicitly learn a world model first. Namely, we use a different dataset, $\mathcal{D}_{pretrain} = \{(\mathcal{S}^j, \hat{\mathcal{A}}^j)\}$, where $\hat{\mathcal{A}}$ is now a sequence of *random* moves. The goal of pre-training is to explicitly predict the latent cube state (rather than next moves) given a sequence of moves. This is done by substituting the unembedding layer $\mathcal{U}$ with 24 alternative classification heads $\mathcal{W}_i \in \mathbb{R}^{|\mathcal{C}| \times d}$, one for each square i, that each classify the correct state of the 24 squares of the cube, with the following loss:

$$\mathcal{L}_{PT} = \sum_{i=0}^{23} \text{CrossEntropy}(\mathcal{W}_i \mathbf{h}_t^{L-1}, \mathcal{S}_{t,i}^*)$$

where $\mathcal{W}_i$ is the classifier for the i-th square and $\mathcal{S}_{t,i}^*$ is the groundtruth color for the i-th square after applying the first t moves in $\hat{\mathcal{A}}$ to the initial state $\mathcal{S}$. Once pre-trained, the model is fine-tuned to predict optimal moves, re-using all the weights except for that of $\mathcal{W}$.

Joint Training. An alternative approach to explicitly train a world model is to learn the two objectives jointly in a single stage. Here the model uses both classification heads ($\mathcal{U}, \mathcal{W}$) with loss:

$$\mathcal{L}_{joint} = \mathcal{L}_{FT} + \mathcal{L}_{PT}$$

Post Training: GRPO. Once a model is trained to solve a Rubik's Cube with one of the recipes above, we are also interested in studying the effects of post-training with reinforcement learning. Namely, we use GRPO [11]. At a high level, given a training sample (scrambled cube state), GRPO samples multiple rollouts from the model and assigns a reward per rollout, which is used to compute

[2] `https://en.wikipedia.org/wiki/Rubik%27s_Cube#Solutions`

a loss term. In our setting, we simply assign a reward of 1 for rollouts that solve the cube, and a reward of 0 for all other rollouts.

Hyperparameters for our training can be found in Appendix A.

2.2 Data

We design our data splits for controlled comparisons across training strategies. A 2x2x2 cube has roughly $3.67e^6$ possible states. We split this data into two training sets and a validation set.

First, we carefully split between the train sets and validation set to minimize the number of overlaps in intermittent cube states (e.g., from applying moves from $\mathcal{A}$ on $\mathcal{S}$). We build our validation set by first constructing a set of scrambled cube states that are maximally far from being solved (11 moves away), and apply optimal moves towards the solved cube state. We collect all intermediate states from these paths, and use these states as our validation set. This results in $114,606$ cube states for validation.

All other trajectories from our data that do not cross these intermediate states are used as training data. This ensures that the intermediate cube states in the train and validation sets are disjoint.[3] This results in $3,559,560$ cube states for training. The distribution of cube complexities (number of moves away from being solved) in our train and test splits are provided in Figure 1.

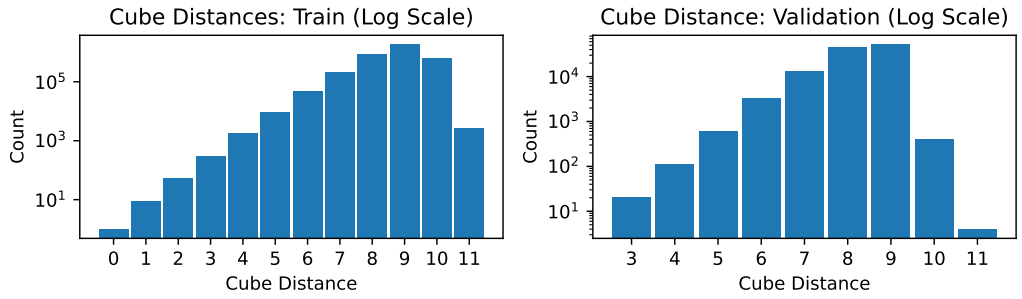

Figure 1: **Distribution of cube complexities** (i.e., number of moves away from being solved).

The remaining training data is split into two sets of equal sizes, notated $\mathcal{D}^{(1)}_{\text{train}}, \mathcal{D}^{(2)}_{\text{train}}$, where $|\mathcal{D}^{(1)}_{\text{train}}|, |\mathcal{D}^{(2)}_{\text{train}}| \approx 1,779,780$. We use the split data to study two different settings: (1) **Pre-training:** We use both splits to train models on each of the three training approaches above; (2) **Post-training:** We use the first split to pre-train models, and use the second split for post-training using GRPO. These two different setups allow us to answer our two research questions.

Table 1 summarizes our training setups and data. The first three rows are designed to answer RQ1 while the latter three rows are to study RQ2.

For each configuration, we train 5 runs with different seeds. We train 8 layer Transformer models with 8 attention heads and a dimension of 512. Hyperparameters are provided in Appendix A.

3 RQ1: Effect of Pre-Training on World Model Representations

Probing. How does explicitly training a world model affect the model's representations? We answer this question by (1) training linear probes to decode cube state information, and (2) steer the model's predictions using the linear probes.

We follow prior work [8] to train linear probes. Namely, at each timestep t, layer ℓ, we learn 24 linear decoders (one for each square i) i.e., $\mathcal{V}^{\ell}_{t,i} \in \mathbb{R}^{6\times d}$, that minimizes:

$$\mathcal{L}_{probe}(t,\ell,i) = \text{CrossEntropy}(\mathcal{V}^{\ell}_{t,i}\mathbf{h}^{\ell}_{t}, \mathcal{S}_{t,i});\ \mathcal{V}_{t,i} \in \mathbb{R}^{6\times d}.$$

[3]This disjointness does not hold for cube states less than two moves from being solved, i.e., cube states that are nearly solved, the number of such states is small.

Table 1: Summary of training configurations and datasets used. Fine-tuning is performed on one or both training splits, and GRPO post-training is always done on $\mathcal{D}^{(2)}_{\text{train}}$.

Description	**Training Setup**
Fine-Tune	$\mathcal{L}_{FT}(\mathcal{D}^{(1)}_{\text{train}} \cup \mathcal{D}^{(2)}_{\text{train}})$
Pre-Train + Fine-Tune	$\mathcal{L}_{PT}(\mathcal{D}_{\text{pre}}) \rightarrow \mathcal{L}_{FT}(\mathcal{D}^{(1)}_{\text{train}} \cup \mathcal{D}^{(2)}_{\text{train}})$
Joint Train	$\mathcal{L}_{joint}(\mathcal{D}^{(1)}_{\text{train}} \cup \mathcal{D}^{(2)}_{\text{train}})$
Fine-Tune + GRPO	$\mathcal{L}_{FT}(\mathcal{D}^{(1)}_{\text{train}}) \rightarrow \mathcal{L}_{GRPO}(\mathcal{D}^{(2)}_{\text{train}})$
Pre-Train + Fine-Tune + GRPO	$\mathcal{L}_{PT}(\mathcal{D}_{\text{pre}}) \rightarrow \mathcal{L}_{FT}(\mathcal{D}^{(1)}_{\text{train}}) \rightarrow \mathcal{L}_{GRPO}(\mathcal{D}^{(2)}_{\text{train}})$
Joint Train + GRPO	$\mathcal{L}_{joint}(\mathcal{D}^{(1)}_{\text{train}}) \rightarrow \mathcal{L}_{GRPO}(\mathcal{D}^{(2)}_{\text{train}})$

$\mathcal{V}_{t,i}$ can be thought of as six vectors that each encode the color of square i at timestep t. The projection $\mathcal{V}_{t,i}\mathbf{h}^{\ell} \in \mathbb{R}^6$ can be thought of logits for predicting each of the 6 possible colors for square i.

Figure 2 demonstrates the results. Observe that explicitly training a world model (pre-trained or joint-trained) leads to an improved cube state representation.

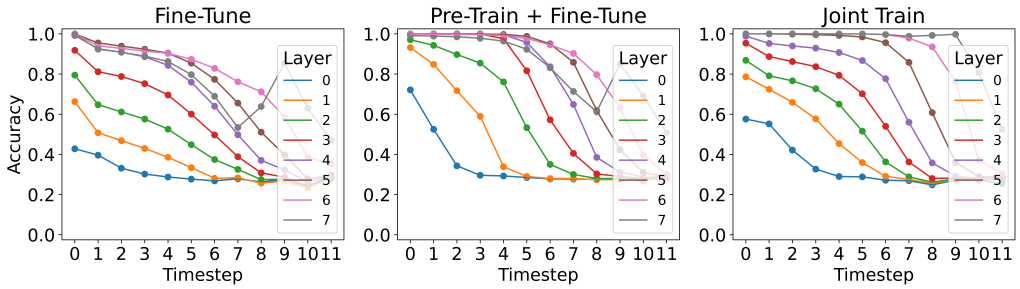

Figure 2: **Probing Accuracy.** Explicitly training a world model (i.e., Pre-Train or Joint Train) leads to improved cube state representations.

Steering. Next we study how dependent each model's predictions are on its internal cube state representations. This is done by causally intervening on the model's cube state representations, in which we use our linear probes to override the model's representations for the original cube state $\mathcal{S}$ with that of an alternative target cube state $\mathcal{T}$. A successful intervention means the model's new prediction reflects a good move for the alternative cube state $\mathcal{T}$ instead of $\mathcal{S}$, with a higher intervention success rate suggesting higher dependency on the model's cube state representations.

We construct the intervention dataset from the 114,606 cube states in our validation set. For each state, we append an optimal move sequence to the input context. At each timestep $t \in \{0, \ldots, 11\}$, we sample up to 1,000 random instances, yielding 36,440 total samples. For each instance with groundtruth cube state $\mathcal{S}$, we randomly select an alternative cube state $\mathcal{T}$ with the same optimal distance to the solved state as $\mathcal{S}$. We additionally ensure that $\mathcal{S}$ and $\mathcal{T}$ do not share any common good next moves.

We intervene on the activations to first remove information about all 24 squares of the previous cube state. This is done by first projecting out the original cube state's colors using the linear probes. We then add in information about the target cube state. More formally, let the groundtruth square colors of $\mathcal{S}$ be $[x_0, \ldots x_{23}]$. Let $\mathbf{v}_i = \mathcal{V}^{\ell}_{t,i}[x_i] \in \mathbb{R}^d$ be the x_i'th row of $\mathcal{V}_{t,i}$ which indicates the vector that represents color x_i for square i. Finally let $[y_0, \ldots y_{23}]$ indicate the colors of the target cube state $\mathcal{T}$, and $\tilde{v_i} = \mathcal{V}^{\ell}_{t,i}[y_i]$. Then our intervention is:

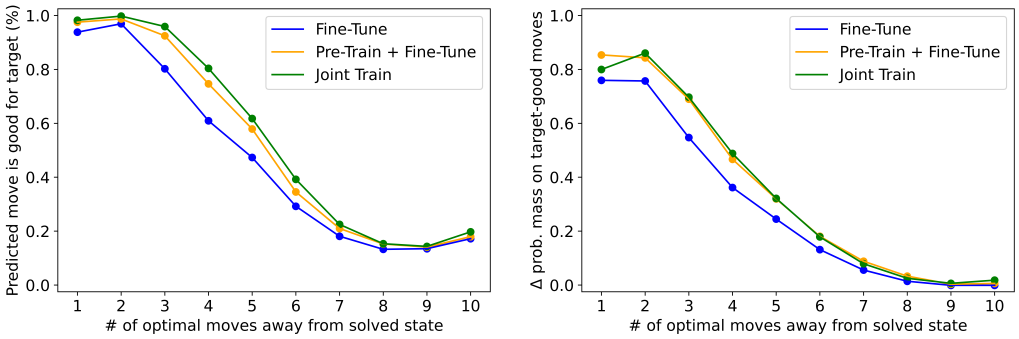

Figure 3: **Intervention success rates.** *Left:* A successful intervention means that the model predicts a good move for the alternative target cube state $\mathcal{T}$ instead of the original cube state $\mathcal{S}$. *Right:* Distributional steering measured as the change in total probability mass on the set of target-good moves (post-intervention minus pre-intervention). Higher intervention success rates suggest that the model relies more on the latent cube state representations. We see that models that were explicitly trained on world modeling have higher intervention success rates.

$$\mathbf{h}_t^\ell = \underbrace{\left(I - \sum_{i=0}^{23} \frac{v_i v_i^\top}{||v_i||^2}\right)}_{\text{project out } \mathcal{S}} \mathbf{h}_t^\ell + \underbrace{\sum_{i=0}^{23} \alpha_i \tilde{v}_i}_{\text{add } \mathcal{T}}$$

After intervening, $\mathbf{h}_t^\ell$ is re-normalized to match the original norm before the intervention. We intervene on layers 5-7 (but not the last layer, layer 8).

Note that our intervention requires hyperparameters α_i to determine how strongly to encode our target cube state. If α is too small, we may not correctly encode the desired color, while if too large it may degrade the model's performance by going off distribution. Thus we *adaptively* select each α value such that $(\mathcal{V}_{t,i}^\ell \mathbf{h}_t^\ell)_{y_i} \geq \max_{c \neq y_i} (\mathcal{V}_{t,i}^\ell \mathbf{h}_{t,i}^\ell)_c + m$, where m is a small margin to ensure that the decoded logits for the intended color y_i exceeds all alternatives ($c \neq y_i$) by at least m. Put differently, we scale α up until we guarantee that the correct board-state is encoded according to our probes.

We test on $m = 0.5, 1, 2$ and find that $m = 1$ resulted in the best results for all three models, and report our results in Figure 3. On the left, we report the fraction of interventions where the model's top predicted move reflects a good move (i.e., a move that takes the cube state one step closer to being solved) for the alternative cube state $\mathcal{T}$ instead of cube state $\mathcal{S}$. Again, note that we ensure that the original cube state $\mathcal{S}$ and target cube state $\mathcal{T}$ do not share any common good moves. On the right, we report the change in total probability mass assigned to the set of good moves for $\mathcal{T}$ after the intervention minus before the intervention (higher is better). The x-axis indicates the cube-complexity of each test sample. We see that the pre-trained and joint-trained models have higher intervention success rates, suggesting that these models rely on its underlying cube state representation more.

4 RQ2: Effect of World Model on Post-Training

Now that we know that explicitly training a world model improves cube state representations, we ask what is the relationship between the quality of a world model and post-training?

We demonstrate task accuracy of 3 different training strategies: standard fine-tuning (FT), pre-training followed by fine-tuning (Pretrain+FT), and joint-training (Joint). For each of the 3 training strategies, we study 3 variants (See Table 1 for our exact setup):

1. Apply training strategy on the first half of data: FT(D_1), Pretrain+FT(D_1), Joint(D_1)
2. Apply training strategy on the full data: FT($D_1 \cup D_2$), Pretrain+FT($D_1 \cup D_2$), Joint($D_1 \cup D_2$)

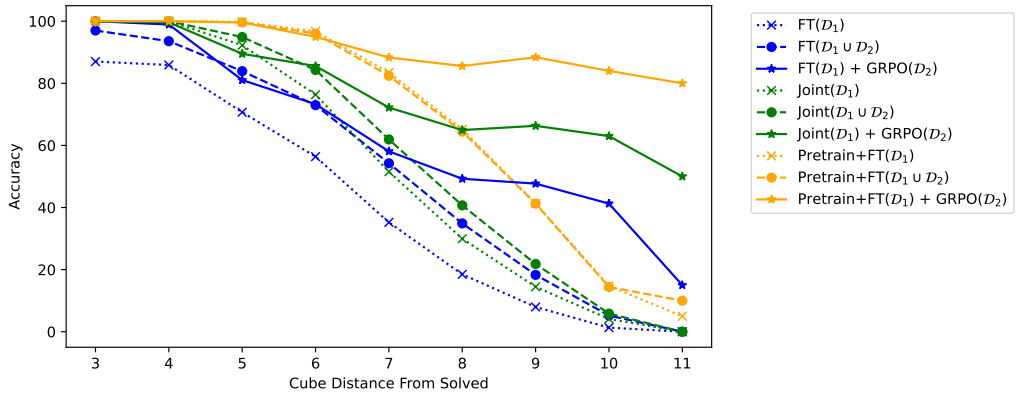

Figure 4: **Task accuracy after GRPO.** Applying GRPO instead of each training strategy (FT, Pretrain + FT, Joint Train) leads to improved results, especially for more complex cube states that require longer rollouts (solid lines vs. dashed lines). Models that were explicitly trained on world modeling also see higher gains from GRPO (orange curves vs. green curves vs. blue curves).

3. Apply training strategy on the first half of the data, use GRPO for the second half of the data: FT(D_1) + GRPO(D_2), Pretrain+FT(D_1)+GRPO(D_2), Joint(D_1)+GRPO(D_2)

This results in a total of nine models. Figure 4 reports their task accuracies, defined as the percentage of scrambles for which the model generates a valid sequence that solves the initial scrambled cube state within N+3 moves, where N is the scramble's optimal solution length. We break down the results by cube complexity, i.e., optimal distance to solve each initial cube state (x-axis), to highlight GRPO's gains for more difficult cube states.

There are numerous points of comparisons to make: Within each color, comparing dashed lines (e.g. FT($D_1 \cup D_2$)) vs. solid lines (e.g., FT(D_1) + GRPO(D_2)) demonstrates the benefit of applying GRPO over continuing to use the original training strategy.

Comparing across colors of the same line-style (e.g., FT($D_1 \cup D_2$) vs. Pretrain+FT($D_1 \cup D_2$) vs. Joint($D_1 \cup D_2$), or, FT(D_1)+GRPO(D_2) vs. Pretrain+FT(D_1) + GRPO(D_2) vs. Joint(D_1)+GRPO(D_2)) shows the difference in performance of the three training strategies. We find that explicitly training a world model (pre-train or joint-train) significantly outperforms not explicitly training a world model, with pre-training outperforming joint-training.

5 Related Work

5.1 World Modeling

Though the term "world-modeling" is often used, it is not yet precisely defined. We use the term "world model" to refer to the ability for the model to keep track of the state of its (latent) environment (i.e., "world"), and understanding how its actions affect the state of the world [3, 7].

Interestingly, [6] demonstrate that Transformers can learn world models from simple next-token prediction tasks. Namely, they demonstrate that a model trained on transcripts of game *moves* can learn to reconstruct the correct board state (i.e., "world") and make the correct move predictions that reflect the state of the board. [8] later demonstrated that such latent board state representations can be linearly decoded using linear probes.

5.2 Probing, Causal Interventions

On a similar note, a growing line of work is relying on linear probes to decode interpretable representations from the model's hidden states [1]. This idea can be extended on contemporary language models, to uncover that many human-interpretable concepts are also linearly encoded in the model's activations [4, 5, 10, 9, 2].

A common practice is to verify the role that such linear representations pay by conducting causal interventions: given the simplicity of linear representations, one can simply scale such representations smaller or larger to control the model's end behavior.

6 Conclusion

In this work we study the relationship between model representations and different training stages. First, we ask how explicitly pre-training a world model affects the model's representations, and find that doing so can lead to a more robust world model. Second, we study the relationship between the quality of model representations and post-training. We find that explicitly training for a world model can lead to better performance after post-training with reinforcement learning.

Limitations. We view this work as work in progress, and as such there are a few limitations to highlight. First, our experiments are limited to a single model, with a single task. An obvious extension we plan on is to check whether our findings generalize to additional settings. Second, we believe there are additional ways to measure how much the model relies on cube state representations in addition to our causal analyses, such as the use of mutual information between the model's predictions and cube state representations.

References

[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[2] Wes Gurnee and Max Tegmark. Language models represent space and time. *arXiv preprint arXiv:2310.02207*, 2023.

[3] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3), 2018.

[4] Andrew Lee, Xiaoyan Bai, Itamar Pres, Martin Wattenberg, Jonathan K Kummerfeld, and Rada Mihalcea. A mechanistic understanding of alignment algorithms: A case study on dpo and toxicity. *arXiv preprint arXiv:2401.01967*, 2024.

[5] Andrew Lee, Melanie Weber, Fernanda Viégas, and Martin Wattenberg. Shared global and local geometry of language model embeddings. *arXiv preprint arXiv:2503.21073*, 2025.

[6] Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*, 2023.

[7] Kenneth Li, Fernanda Viégas, and Martin Wattenberg. What does it mean for a neural network to learn a" world model"? *arXiv preprint arXiv:2507.21513*, 2025.

[8] Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. *arXiv preprint arXiv:2309.00941*, 2023.

[9] Kiho Park, Yo Joong Choe, Yibo Jiang, and Victor Veitch. The geometry of categorical and hierarchical concepts in large language models. *arXiv preprint arXiv:2406.01506*, 2024.

[10] Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. *arXiv preprint arXiv:2311.03658*, 2023.

[11] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Parameter	Value
Learning Rate	1e-5
Weight Decay	0.01
Batch Size	64
Hidden Dimension	512
Validation Size	512
Validation Patience	10
Optimizer	AdamW

Table 2: **Hyperparameters for fine-tuning.**

Parameter	Value
Learning Rate	1e-5
Per device train batch size	256
Per device eval batch size	128
Number of rollouts	8
Weight Decay	0.01
KL penalty Beta	0.01
Max generation length	13
Optimizer	AdamW

Table 3: **Hyperparameters for GRPO.**

A Hyperparameters

Here we provide hyperparameters used to train each setup.

A.1 Standard Fine-Tuning, Pretraining, Joint-Training

We train a 8 layer model with 8 attention heads. Table 2 demonstrate the hyperparameters used to train our models.

A.2 GRPO

Table 3 contains the hyperparameters used for GRPO.

A.3 Probing

Table 4 contains the hyperparameters used for training our probes.

Parameter	Value
Learning Rate	1e-3
Weight Decay	0.01
Batch size	32
Validation size	512
Validation patience	10
Epochs	1
Optimizer	AdamW

Table 4: **Hyperparameters for Probing.**