PLANNING IN A RECURRENT NEURAL NETWORK THAT PLAYS SOKOBAN

Anonymous authors
Paper under double-blind review

ABSTRACT

How a neural network (NN) generalizes to novel situations depends on whether it has learned to select actions heuristically or via a planning process. Guez et al. (2019, "An investigation of model-free planning") found that recurrent NN (RNN) trained to play Sokoban appears to plan, with extra computation steps improving the RNN's success rate. We replicate and expand on their behavioral analysis, finding the RNN learns to give itself extra computation steps in complex situations by "pacing" in cycles. Moreover, we train linear probes that predict the future actions taken by the network and find that intervening on the hidden state using these probes controls the agent's subsequent actions. Leveraging these insights, we perform model surgery, enabling the convolutional NN to generalize beyond its 10×10 architectural limit to arbitrarily sized levels. The resulting model solves challenging, highly off-distribution levels. We open-source our model and code, and believe its small size (1.29M parameters) makes it an excellent model organism to deepen our understanding of learned planning.

1 INTRODUCTION

In many tasks, the performance of both humans and sufficiently advanced neural networks (NNs) improves with more reasoning: whether by giving a human more time to think before taking a chess move, or by prompting or training a large language model to reason step by step (Kojima et al., 2022; OpenAI, 2024). Thus, understanding reasoning in such NNs is key to predicting their behavior in novel situations.

Among other reasoning capabilities, goal-oriented reasoning is particularly relevant to AI alignment. So-called "mesa-optimizers" – AIs that have learned to pursue goals through internal reasoning (Hubinger et al., 2019) – may internalize goals different from that given by the training objective, leading to harmful misgeneralization (Di Langosco et al., 2022; Shah et al., 2022).

Within goal-oriented reasoning, this work distinguishes between *plans* and *search algorithms*. A *plan* is a sequence of (perhaps abstract) actions which an agent intends to take. A *search algorithm* is an algorithm that considers many plans, and picks the best one according to some evaluation of the plan's possible outcomes. Search algorithms perform better the more plans they can evaluate, and with enough compute, generalize very well to novel problem instances (Russell & Norvig, 2009).

This work investigates a recurrent neural network (RNN) that plays Sokoban (Guez et al., 2019), a challenging puzzle game that remains a benchmark for planning algorithms (Peters et al., 2023) and reinforcement learning (Chung et al., 2024). Using linear probes, we show that its activations **represent a plan** in a way which **causes** the RNN's actions.

The behavior of the RNN suggests that it performs *search*. We find that the RNN sometimes takes actions which return to previous states of the environment. This overwhelmingly happens at the beginning of episodes and can be substituted by no-ops, which suggests the RNN **deliberately goes in cycles to get more time to think**. Giving the RNN time to think with no-ops (Guez et al., 2019) disproportionately helps with levels that are difficult (fig. 1), long, or which requires waiting many steps before the first reward (fig. 3).

Using insights from the plan probes and inspecting activations, we perform **model surgery to uncover latent generalization capabilities** of the RNN. The RNN is trained only on 10×10 levels with 4

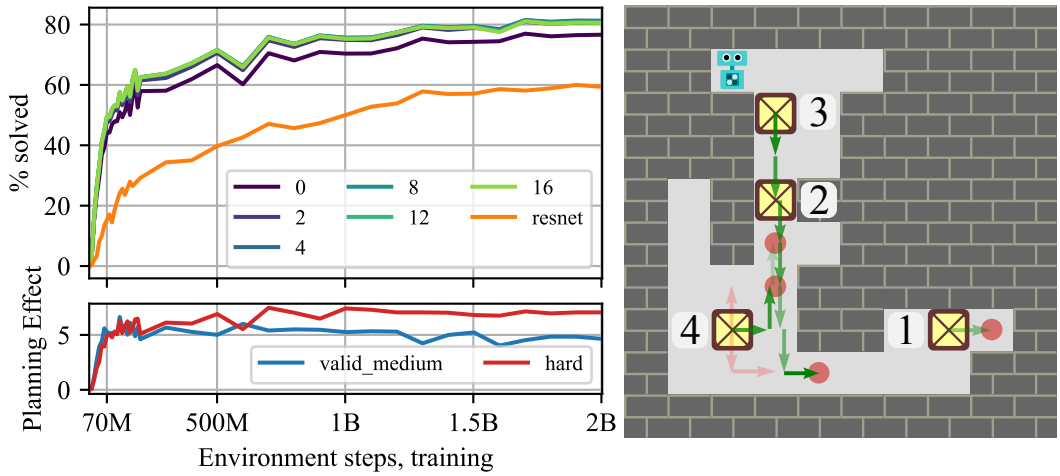


Figure 1: **Top left:** Proportion of medium-difficulty validation levels solved vs. environment steps used in training. Curves show the RNN's performance with a specific number of forced thinking steps at episode start, along with a ResNet baseline. **Bottom left:** Estimated planning effect: the 8-steps minus the 0-steps curve. We see that reasoning emerges in the first 70M steps and keeps increasing for the hard levels (red), but decreases for medium levels (blue). **Right:** Linear probe predictions (arrows) for the direction the box will move in. Opacity is proportional to the number of steps in the episode in which the probe predicted a particular arrow. Correct predictions are in green and incorrect in red, the red predictions show other plausible plans for Box 4. This probe causally affects the actions of the agent, as described in section 4. Boxes are numbered in the order in which they arrive to their final target.

boxes (and its output MLP can only handle 10×10 levels), but we are able to make it play on larger, nontrivial levels with more boxes (fig. 2).

Altogether, evidence from probes and behavior show that the RNN causally represents a *plan*. The evidence highly suggests that the RNN implements an online *search algorithm* which scales with computation, although we cannot discard multi-step heuristic algorithms.

Consequently, we believe this RNN is an excellent model organism for understanding planning in sophisticated neural networks. Its behavior is complex enough to be interesting and its size (1.29M parameters) is small enough to be tractable to reverse engineer. This work shows that supervised probes and sparse autoencoders (Huben et al., 2023; Bricken et al., 2023; Gao et al., 2024) are not enough to fully do so, but it is likely that other mechanistic interpretability techniques such as circuit discovery (Conmy et al., 2023) will reveal more about the algorithm which produces plans.

2 Setting up the test subject

We train an agent closely following the setup from Guez et al. (2019), using the IMPALA (Espeholt et al., 2018) reinforcement learning (RL) algorithm with Guez et al.'s Deep Repeating ConvLSTM (DRC) recurrent architecture. We also train a ResNet baseline. We open-source both the trained networks and training code [1] . For more details about architectures and training, see appendix A.

DRC(D, N) architecture. Most of this paper focuses on the behavior and representations of a DRC$(3, 3)$ neural network (Guez et al., 2019). The core of this network is a D-layer ConvLSTM (Shi et al., 2015), which is repeatedly applied N times per environment step. The output of the last ConvLSTM layer (the Dth layer) is fed to the input at the next tick, effectively giving the network $D \cdot N$ layers of sequential computation to compute the next action. We take $D = 3, N = 3$.

[1] URL references removed during double-blind review.

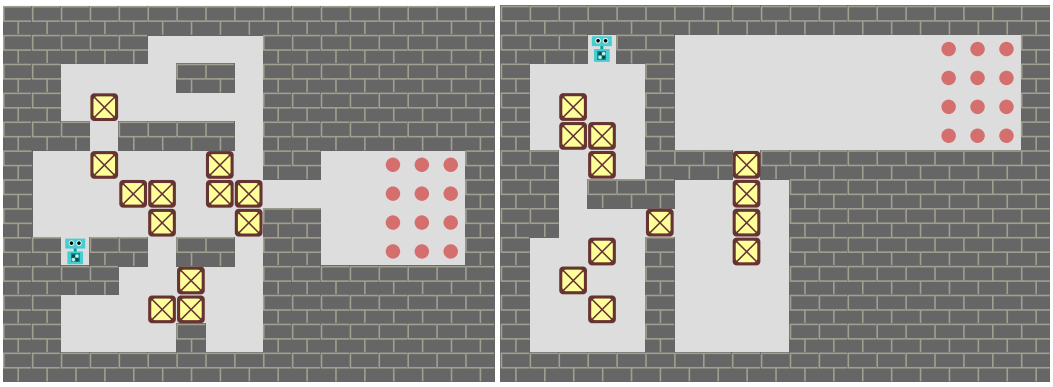


Figure 2: The DRC$(3, 3)$ is limited to 10×10 grids by a fixed-dimension MLP. With model surgery, the ConvLSTM backbone generalizes to much more challenging settings than the 10×10 training set grids. **Left:** XSokoban-31, which the DRC$(3, 3)$ solves with 0 thinking steps. **Right:** XSokoban-29, which the DRC$(3, 3)$ solves but only after 128 thinking steps. More examples in appendix B.

A linear combination of the mean- and max-pooled activations of the ConvLSTM is also fed to the next step, thus letting it communicate quickly across the receptive field (known as *pool-and-inject*). An encoder block consisting of two 4×4 convolutions processes the input, which is also fed to each ConvLSTM layer. At the end, an MLP with 256 hidden units transforms the flattened ConvLSTM outputs into the policy (actor) and value function (critic) heads.

Dataset. Sokoban is a grid puzzle game with walls, floors, movable boxes, and target tiles. The player's goal is to push all boxes onto target tiles while navigating walls. We use the Boxoban dataset (Guez et al., 2018), consisting of 10×10 procedurally generated Sokoban levels, each with 4 boxes and targets. The edge tiles are always walls, so the playable area is 8×8. Boxoban separates levels into train, validation and test sets, with three difficulty levels: unfiltered, medium, and hard. Guez et al. (2019) generated these sets by filtering levels that cannot be solved by progressively better-trained DRC networks, so easier sets occasionally contain difficult levels. In this paper, we use unfiltered-train (900k levels) to train networks. To evaluate them, we use unfiltered-test (1k levels)[2], medium-validation (50k levels), and hard ($\sim$3.4k levels), which do not overlap. To test DRC$(3, 3)$ generalization to different sizes, we use the levels collected by Þorsteinsson (2009) (see appendix B).

Environment. The observations are 10×10 RGB images, normalized by dividing each component by 255. Each type of tile is represented by a pixel of a different color (Schrader, 2018), an example is in fig. 15. The player can move in cardinal directions (Up, Down, Left, Right). The reward is -0.1 per step, 1 for placing a box on a target, -1 for taking it off, and 10 for finishing the level by placing all of the boxes. The time limit for evaluation is 120 steps, though large levels in section 5.1 use 1000 steps.

3 Behavioral evidence of search

3.1 Effects of thinking time: non-myopic network

First, we examine the effect of thinking time: introducing no-ops at the beginning of an episode, while letting the RNN process its hidden state. As in Guez et al. (2019), adding 6 extra thinking steps lets the DRC$(3, 3)$ solve an extra 4.7% of levels, slightly decreasing when going up to 16 steps (fig. 5, left).

We find that the extra thinking time disproportionately helps with more difficult levels, as measured by longer optimal solutions (fig. 3, right). We also checked whether thinking time correlates with the number of nodes expanded by an A* search, but it does not (fig. 12).

[2]We use unfiltered-test rather than unfiltered-validation so the numbers are directly comparable to Guez et al. (2019).

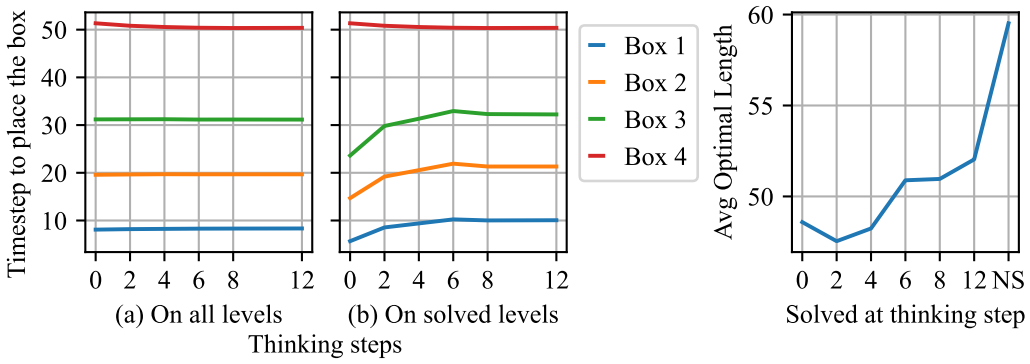


Figure 3: **Left:** Average time step to place each box B_i on target for different numbers of thinking steps. **(a)** Averages across all levels where the box B_i is placed on target by DRC with n thinking steps. **(b)** Averages for all levels solved by 6 thinking steps but not solved by 0 thinking steps. **Right:** Average optimal solution length of levels grouped by the number of thinking steps at which the level is first solved. Levels that take longer to solve tend to be harder. NS stands for "not solved".

How much thinking time helps varies over training, and over level difficulties: fig. 1 (bottom left) shows that most of the planning effect steeply appears during the first 70M steps of learning. Over the remaining training time, the RNN benefits more from thinking steps for the hard levels, but less for the medium levels; indicating perhaps that it learns better heuristics for them.

We also find that thinking time disproportionately helps with levels in which the agent can greedily put a box to target early, but where doing so is detrimental to solving the level. Figure 3 (left, b) shows this: with 0 thinking steps, the first box is pushed to target on average 4 steps too early; compared to the solution reached at 6 thinking steps. Some of this is driven by the forced thinking steps preventing the DRC$(3, 3)$ from taking a catastrophic action in the first few moves. However, the average time to first box is more than 6, even for 0 thinking steps, so there must also be many cases in which the 6 extra thinking steps give the NN time to find a better solution, rather than just preventing it from greedily pushing the first box. It must also be less greedy for boxes 2-3.

3.2 Agent "paces" to get more computation

On occasion, the DRC exhibits a curious behavior: the agent "paces" in a cycle, returning to the same location, without touching any box. The per-step penalty makes this behavior naively sub-optimal.

Since the DRC$(3, 3)$'s performance improves with thinking steps, could the agent be using pacing to gain computation time to improve its plan and so avoid irreversible actions which may block the solution? The following evidence suggests the agent has developed this meta-strategy:

The agent paces mostly at the *start* of episodes. In some levels, a single sub-optimal step can lock the level, making calculating a valid plan in early steps beneficial. Figure 4 (left) shows most cycles start in the first 5 steps of the episode, and (middle) forcing the model to think for six steps eliminates about 75% of these early cycles.

Thinking steps exactly substitute for pacing We run the DRC$(3, 3)$ on medium-validation levels and record when and where cycles occur. To avoid over-sampling very long cycles, we merge all cycles that visit overlapping sets of states into one. This gives 13702 merged cycles.

We then replace each cycle, of length N, with N thinking steps. Figure 14 shows that the DRC behavior, after the replaced cycle, is *exactly the same* for at least 60% of levels, for at least 30 steps[3]. Also, in 82.39% of cases, no new cycles start in the N steps after the replaced cycle (fig. 14).

[3]For context, the median solution length for train-unfiltered is exactly 30 steps.

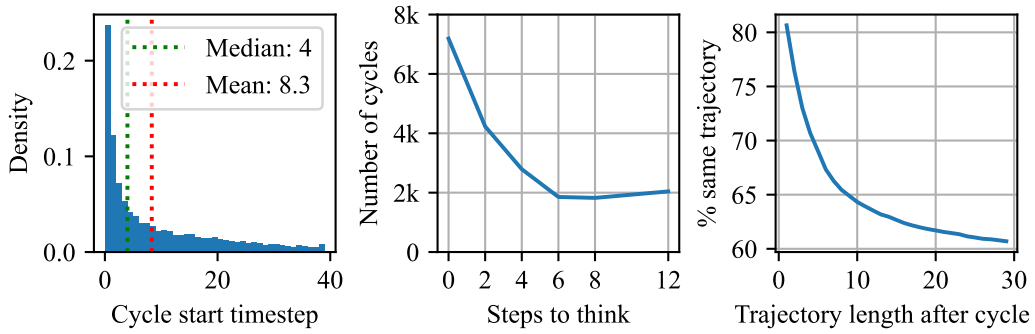


Figure 4: **Left:** Histogram of cycle start times on the medium-difficulty validation levels. **Middle:** Total number of cycles the agent takes in the first 5 steps across all episodes in medium-difficulty validation levels with n initial thinking steps. **Right:** on all levels, we replace N-length cycles with N-thinking steps and check for the same state after some timesteps. After replacing a cycle with the same length in thinking steps, the proportion of states that are the same for the next x steps.

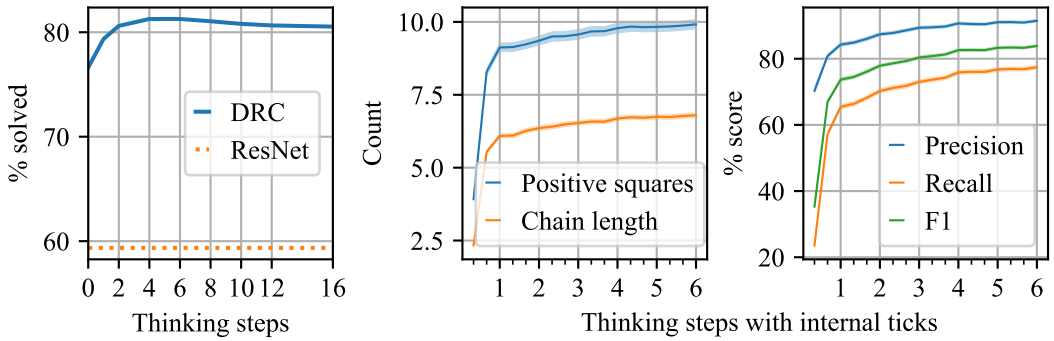


Figure 5: Success rate of the RNN, and plan quality as measured by box-direction probes (section 4.1), all increase with thinking steps. We measure plan quality by 1) summing the length of chains of probe-predicted directions, which start in boxes, and 2) simply counting the number of squares in which the box-direction probe predicts something. The F1 score of the plan probe also increases over time, suggesting that the the RNN builds the plan in that time.

Comparing DRC$(3,3)$ to ResNet The ResNet is much worse than the DRC$(3,3)$ at Sokoban. We note it almost never performs cycles for solved levels. It has a much larger receptive field per forward pass, and the *pool-and-inject* layer makes communication between locations less important than for our NN.

While it is possible that our NN is pacing because it is not optimized enough, or because it is computing a multi-step heuristic (e.g. a heuristic that depends on the convolutional receptive field), the improved performance over ResNet makes it more likely DRC$(3,3)$ is doing an iterative search during pacing.

Note that heuristics may still play a role. For example, due to the -0.1 per-step penalty and discount rate, it is rational for the training process to make the RNN take boxes greedily (hence the increasing time-to-box in fig. 3, middle). It could be that the planning effect disappearing for medium levels (fig. 1, left) is due to better heuristics about when to be greedy and when to take time to think.

4 THE DRC$(3,3)$ CAUSALLY REPRESENTS ITS PLAN

In order to improve the plan with search, as the previous section suggests, the agent must have some internal representation of the plan. The absence of such representation would force us to discard the theory that the agent is doing search.

This section presents probes which predict the future actions of the DRC$(3,3)$ and other various future features of the environment. Of these, the one which predicts the future direction *boxes* will move in has a strong causal influence in the actions of the RNN. By the standard of evidence in Li et al. (2023), we declare this conclusive evidence that the DRC$(3,3)$ represents and uses plans.

We also observe the plan grow during thinking steps and, by intervening on the probes, are able to lock the RNN into a plan when there are two options of equal value. However in most cases, if we stop intervening on the RNN activations with a suboptimal plan, the RNN comes up with a better plan online and follows it. We have some evidence for the DRC$(3,3)$ considering multiple plans: for example, in the level in fig. 1 (right) the box probe first considers moving box 4 down and right, but then rethinks that. However, we have been unable to find multiple *simultaneous* representations of plans, or anything that would decide between them such as the value of the plans. In all, this section provides very little evidence in favor of *search*, but conclusive evidence for a causal *plan*.

4.1 PROBE METHODOLOGY: TRAINING, INTERVENTION AND TARGETS

Training. We train logistic or linear regression probes with L1 decay to predict various features of the environment from the agent's activations. We train on two types of inputs. **1) Grid-wise inputs:** each square in the 10×10 grid is a different data point, potentially with a different label depending on location and timestep. The input is the 64-dimensional LSTM state (h, c) at a square, usually concatenated for the 3 layers. **2) Global inputs:** Each time step produces only one data point. The probes take as input the 64 channels of all of the squares in the 10x10 grid, resulting in a 6400-dimensional input for each layer. In both cases, the per-layer activations are concatenated for probes that are trained on multiple layers.

The dataset consists of states collected by evaluating the DRC$(3,3)$ on the hard Boxoban levels, excluding the first 5 steps of each episode because the plan is still forming. For multi-class probes, training and F1 scores are computed as one-vs-all: the presence of absence of a particular class is the probe label. We search the best learning rate and L1 decay with grid-search, evaluating the F1 on a validation set of timesteps from the hard levels.

Causal intervention. Given a probe that is predictive of future behavior, we edit the agent's activations such that the probe predicts the behavior we desire to induce. To the extent that the agent carries out the edited behavior, we can conclude that it executes a plan encoded in the linear direction of the probe.

For a linear probe on the activations h with parameter vector p, we intervene in order to increase the logits for a particular behavior, given by $p \cdot h$ (we omit a scalar intercept for brevity). Unlike previous work in steering vectors for LLMs (Turner et al., 2023; Rimsky et al., 2023; Li et al., 2024), we found that simply adding a perturbation equal to a fixed multiple of p caused the policy to degenerate and choose random actions. We instead add an adaptively-scaled multiple of p to the activations, chosen to minimize the change in the activations while still causing the probe's logit to be greater than a desired amount α. More precisely, we find $\arg\min_{h'} \|h' - h\|_2^2$, with the constraint that $h' \cdot p > \alpha$. This has the analytic solution $h' = h + \hat{p} \max(0, \alpha - (h \cdot p))$, where $\hat{p} = p / \|p\|_2^2$.

Targets. Most of these labels are for grid-wise probes. The *cycle* and *value* probes are global. The box and agent direction labels follow (Bush, 2024).

- **Agent-Directions probe**: This probe has 5 outputs: NV (No Visit), UP, DOWN, LEFT, RIGHT. The probe takes as input the 64-dim hidden state activations at a square (x, y) at timestep t_i. If the agent visits the square (x, y) at a future timestep $t > t_i$, then the probe predicts the corresponding direction that the agent takes from that square. Otherwise, the probe predicts NV. There is one single probe, for all time steps and locations (x, y)

- **Boxes-Directions probe**: This probe is similar to the above probe except that it is trained to predict the direction that *any* of the 4 boxes will move in at a given square in the future.
- **Next-Box probe**: Predicts 1 on the square of the box that the agent will move next and 0 for every other square.
- **Next-Target probe**: Predicts 1 on the square of the target that the agent will put a box in next, and 0 for every other square.
- **Pacing probe**: The global label is 1 if the agent is currently in a cycle, and 0 otherwise.
- **Value probe**: The global label is the numerical value that the critic head outputs.

Table 1: Causal and predictive probe results. Confidence is one of the mean estimator percentiles $[2.5\%, 97.5\%]$, whichever is furthest from the mean, estimated using 1000 bootstrap resamples. The AVERAGE causal probe uses 24k data points for evaluation, and the BEST-CASE probe uses 8k.

PROBE TARGET	PREDICTIVE PROBE RESULTS (A)			CAUSAL PROBE RESULTS (B)		
		SPARSE PROBES		CAUSAL EFFECT		
	BEST F1	F1	L_0-NORM	α	AVERAGE	BEST-CASE
Boxes directions	86.4 ± 0.1	73.1 ± 0.1	63	30	43.7 ± 0.6	77.8 ± 0.9
Agent directions	72.3 ± 0.1	61.3 ± 0.1	121	10	7.1 ± 0.3	20.7 ± 0.7
Next box	74.2 ± 0.4	69.7 ± 0.5	51	40	5.5 ± 1.0	15.1 ± 2.5
Next target	54.3 ± 0.5	44.0 ± 0.5	32	30	4.6 ± 0.8	13.2 ± 2.0
Pacing	31.0 ± 1.8	31.0 ± 1.5	5	—	—	—

4.2 PROBE EVALUATION AND CAUSALITY

Probe predictive power. Most probes are quite predictive, as seen in table 1. The exception is the pacing probe, with $F_1 = 31.0\%$, which is not much better than the constant 1 probe, which has $F_1 = 12.8\%$. This leads us to conclude that the DRC$(3, 3)$ does not represent whether it is in a cycle or not. See fig. 1 (right) and fig. 6 for visualizations of agent directions and box directions probe[4].

For the value function probe, we compute the fraction of variance explained R^2. If we train a global probe, R^2 is very high: 97.7%–99.7% depending on the layer. However, grid-wise probes obtain much worse but still passable results: 41.0%–79.2% depending on layer. We visually checked whether the grid-wise probe reads off the values of different plans, but could not find any such pattern.

Almost all the performance of the global probe is recovered by training on the mean-pooled inputs: 95.2%–99.5%. It is likely that the global and mean-pooled value probes are indirectly counting the number of squares the agent will step on, which almost fully determines the value, and we know is possible due to the future-direction probes.

Only the Box-Directions probe is strongly causal. We measure how causal the probes are by intervening at every step on the hidden state h, c at the layer the probe was trained on, following the procedure in section 4.1 and checking whether the agent follows that action. We report the result in table 1(b) by performing a grid-search over the strength of intervention α and picking the best value. Despite our adaptive scaling of the intervention, a high value of α can still disturb the agent such that it starts performing random actions whereas a low value of α may not be sufficient to cause the agent to follow the intended behavior. Most of the probes are not causal: only the "boxes directions" probe (used in fig. 1) is causal while the "agent directions" probe mildly causal.

Even then, we have observed that it can only change boxes directions when they *would not deadlock* the plan in naive ways. That is, if you try to make the model push a box next to another box by intervening with the boxes directions probe such that it would not be able to reach the target anymore, it does not. For this reason, we introduce the *Best-case causal effect* in table 1(b): try all three directions that are not the actual direction that the box follows, and count the probe as "causal" if it works for *any* of them.

[4]The supplementary material provides visualization videos of all the probes across several levels.

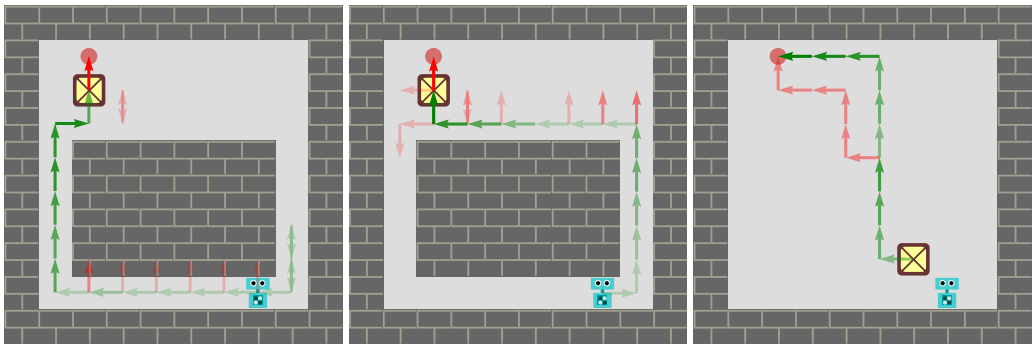


Figure 6: **Left:** A custom level where the agent has two equally good paths to follow. The arrows show the prediction of the agent directions probe with opacity proportional to the number of times an arrow was predicted across all the steps. The green and red arrows are correct and incorrect predictions, respectively. The agent behaves suboptimally by returning to the start and going left after first going right for three steps. **Middle:** The agent takes the path on the right after intervening with the corresponding arrows using the agent directions probe on the first step. The same happens on the left if we intervene on that path without the suboptimal steps right of the undisturbed agent. **Right:** An empty level with box directions probe intervening on the first four steps which are followed correctly by the agent on those four steps. When the intervention is removed on the fifth step, the agent computes the simpler path in green and doesn't follow the path laid out earlier (in red).

The box direction probe's coverage increases during thinking time. Figure 5 (middle) show that the amount of squares in which the probe predicts a direction, be they lone or linked to a box, grows over thinking steps. Its description of what the box movements will be also greatly improves after a few steps of thinking fig. 5 (right).

4.3 Bistable and unstable plans in toy environments

The causal intervention results from table 1 show that the box-directions probe is much more causal than the agent-directions probe. We further study this in an isolated setting. Figure 6 (left) shows a level with only a single box next to a target, where the agent has only two equally good paths it can follow. Without intervening, the agent first takes the path on the right by moving three steps, while the agent directions probe grows towards the box. After the first three moves, the chain connects. The agent then takes a U-turn to follow the path on left. This is very unlike the optimal behavior, which is to commit to any path and follow it.

We can get the agent to commit to one path early by intervening with the agent directions probe on any of the two paths *on the first step only*, as shown in fig. 6 (middle). When we construct a similar level where there are two equally good paths for a box to follow, we find that the agent directions probe is no longer causal; but the boxes direction probe *does* get theagent to commit. Figure 6 (right) shows an almost empty level. When we intervene with the box directions probe on the first four steps, the agent follows the laid out path in those steps, but then quickly switches to a more optimal path with fewer turns. The laid out path (not taken, shown in red) is not followed after stopping the intervention.

This analysis provides strong evidence that the network primarily builds upon the the boxes directions represented in the activations and relies on the agent directions when the boxes directions do not inform the subsequent actions.

5 Model surgery exposes latent generalization capabilities

A common feature of search algorithms is that they generalize very well to unusual problems in their domain. This was the main thrust of the argument of Guez et al. (2019): they exhibited many ways in which the DRC$(3, 3)$ generalized to levels outside of the training distribution to argue that it does search.

5.1 GENERALIZING BEYOND 10×10 INPUTS AND TRAINING EXAMPLES

This section goes beyond the capabilities of DRC$(3, 3)$ as a black-box unit. By looking for interpretable features in the activations of the ConvLSTM core, we found that layer 3, at the last tick, has four channels which represent the next action to take. Since the network up until the ConvLSTM core is completely convolutional, we can evaluate it on inputs of *any* size – unlike DRC$(3, 3)$ as a whole, which only works on 10×10 inputs, and is trained only on levels with four boxes.

Spatial aggregation. We start with the Layer 3 grid-wise next-action probes from table 3. For each spatial location, these predict the action the DRC$(3, 3)$ will take in this step. We attempt three different spatial aggregation methods: mean-pooling, max-pooling, and the proportion of locations with positive probe readouts for a particular action. We learn their relative weights and a baseline for each action by using Adam with learning rate $\ell = 10^{-3}$, annealed linearly to 0 for 10000 steps. Table 2 lists the learned parameters, which place most weight on the *mean-pooling* aggregation rule. The training set consists of 3000 levels, 1000 from each of the Boxoban unfiltered, medium, and hard training sets.

Table 2: Weight and bias for transforming direction probes to predictions

	Mean	Max	Positive proportion
Weight	1.2086	-0.0582	0.2070

	Up	Down	Left	Right
Bias	0.3337	-0.0921	-0.0632	-0.0539

Results. The aggregated features obtain 83% accuracy at predicting the MLP's action output on the training set, which is less than expected given the $90\%+$ F1 scores of probes in table 3. With only seven parameters, we did not bother collecting an i.i.d. test set and went straight to the generalization tests.

The adjusted ConvLSTM is able to solve many levels that are out-of-distribution: larger than 10×10 in both dimensions, and with more than four boxes, e.g. those in fig. 2. Out of our test set collected from Þorsteinsson (2009), it can solve $59/480$ levels with both dimensions > 10, $30/203$ levels with only one dimension > 10 and $180/482$ levels with both dimensions ≤ 10. Appendix B has a breakdown by level collection.

5.2 LOOKING FOR INTERPRETABLE FEATURES WITH SPARSE AUTOENCODERS

The way we found out actions were represented in the ConvLSTM was with sparse autoencoders (SAEs). We inspected an SAE (Huben et al., 2023; Bricken et al., 2023) trained on the last layer with the top-k activation (Gao et al., 2024; Tamkin et al., 2023), $k = 8$. However, we then noticed that the action features were also represented directly in the ConvLSTM channels.

Table 3 compares raw channels, SAE features, and linear probes for how well they match the interpretation of "the feature fires when the DRC$(3, 3)$" takes a particular action. We find that the raw channels and linear probes are similarly good, and both are considerably better than the SAE.

Thus, for these features, SAEs actually worsened interpretability. We found three other interpretable features in the SAE, which we have not checked whether they are represented in individual channels (appendix E).

Table 3: Action features scores across channels, probes, and SAE features

FEATURE	CHANNEL				SAE FEATURE				PROBE		
	NUMBER	PREC	REC	F1	NUMBER	PREC	REC	F1	PREC	REC	F1
Up	L3C29	95.7	88.1	**91.7**	L3F270	93.9	76.2	84.1	97.5	86.5	**91.7**
Down	L3C8	98.4	80.8	88.8	L3F187	98.0	79.1	87.6	97.6	86.9	**91.9**
Left	L3C27	85.5	84.6	**85.1**	L3F244	96.1	63.2	76.2	83.5	86.6	85.0
Right	L3C3	97.0	86.9	91.7	L3F385	94.6	78.5	85.8	97.6	87.4	**92.2**

6 RELATED WORK

Interpretability of agents and world models. Several works have attempted to find the mechanism by which a simple neural network does planning in mazes (Ivanitskiy et al., 2023; Mini et al., 2023), gridworlds (Bloom & Colognese, 2023), and graph search (Ivanitskiy et al., 2023). We believe the DRC we present is a clearer example of an agent than what these works focus on, and should be similarly possible to interpret.

Other works have found emergent world models in sequence-prediction (Li et al., 2023) and navigation (Wijmans et al., 2023) neural networks.

Goal misgeneralization and mesa-optimizers for alignment. From the alignment perspective, AIs optimizing monomaniacally for a goal have long been a concern (e.g. Yudkowsky, 2006; Omohundro, 2008; see the preface of Russell, 2019). In a machine learning paradigm (Hubinger et al., 2019), the goal of the training system is not necessarily optimized; instead, the NN may optimize for a related or different goal (Di Langosco et al., 2022; Shah et al., 2022), or for no goal at all.

Chain-of-thought faithfulness. Large language models (LLMs) use chain of thought, but are they faithful to it or do they think about their future actions in other ways (Lanham et al., 2023; Pfau et al., 2024)? One could hope that LLMs perform all long-term reasoning in plain English, allowing unintended human consequences to be easily monitored, as in Scheurer et al. (2023).

Reasoning neural network architectures. Many papers try to enhance NN thinking by altering the training setup or architecture (Bansal et al., 2022; Graves, 2016; Chung et al., 2024).

Ethical treatment of AIs. Do AIs deserve moral consideration? Schwitzgebel & Garza (2015) argue that very human-like AIs are conceivable and clearly deserve rights. Tomasik (2015) suggests that most AIs deserve at least a little consideration, like biological organisms of any species (Singer, 2004). But what does it mean to treat an AI *ethically*? Daswani & Leike (2015) argue that the way to measure pleasure and pain in a reinforcement learner is not by its absolute amount of return, but rather by the temporal difference (TD) error: the difference between its expectations and the actual return it obtained. If the internals of the NN have a potentially different objective (Hubinger et al., 2019; Di Langosco et al., 2022), then the TD error should come from a place *other* than the critic head. *Research into learned search algorithms* is an early step toward finding the learned-reward internal TD error, if it exists.

7 CONCLUSION

We replicated a small RL-based Sokoban agent, whose performance benefits from increased compute in the form of externally imposed no-op steps (Guez et al., 2019). We find that the agent learns to explicitly exploit this capability by pacing, which *substitutes* for the external no-op thinking steps. Extra computation helps with longer levels and those that require non-myopic thinking.

We show that the agent causally represents *plans* (sequences of actions to take). The most fundamental plan representation we find is the box direction probes, which can predict and control where the agent pushes a box. By building up a chain of box probe steps over time, the agent is able to figure out how to place boxes to solve the level. Secondarily, we find agent direction probes that take lower precedence than the box direction probes, but can control the agent direction in isolated setups. We additionally find that the box probe chains get longer and longer with more thinking steps (fig. 5).

Finally, we find that extracting a decision from the plan takes place almost entirely within the ConvLSTM, with the MLP doing little more than decoding it. To demonstrate this, we show that the agent can generalize to levels bigger than what it had ever seen during training by manually replacing its MLP decoder based on our mechanistic understanding.

We hope the model organism we trained and open-source will catalyze research into NNs that learn to plan, and to understand their inner objectives.

REPRODUCIBILITY STATEMENT

We provide the code and the trained models in the supplementary materials. We will open-source our code and trained model(s). Additionally, sections 2, 4.1 and appendices A, D, and A.1 describe the specific experimental details and settings we used to carry out our experiments.

REFERENCES

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=PPjSKy40XUB.

Joseph Bloom and Paul Colognese. Decision transformer interpretability, 2023. URL https://www.lesswrong.com/posts/bBuBDJBYHt39Q5zZy/decision-transformer-interpretability.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, et al. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2, 2023.

Tom D. Bush. Do model-free deep reinforcement learning agents actually plan? Master's thesis, University of Cambridge, 2024.

Stephen Chung, Ivan Anokhin, and David Krueger. Thinker: Learning to plan and act. *Advances in Neural Information Processing Systems*, 36, 2024. URL https://openreview.net/forum?id=mumEBl0arj.

Arthur Conmy, Augustine N. Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. In *Advances in Neural Information Processing Systems*, 2023.

Mayank Daswani and Jan Leike. A definition of happiness for reinforcement learning agents. *arXiv*, 2015. URL http://arxiv.org/abs/1505.04497v1.

Lauro Langosco Di Langosco, Jack Koch, Lee D Sharkey, Jacob Pfau, and David Krueger. Goal misgeneralization in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 12004–12019. PMLR, 2022. URL https://proceedings.mlr.press/v162/langosco22a.html.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. *arXiv*, 2018. URL http://arxiv.org/abs/1802.01561v3.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv*, 2016. URL http://arxiv.org/abs/1603.08983v6.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, Timothy Lillicrap, and Victor Valdes. An investigation of model-free planning: boxoban levels, 2018. URL https://github.com/deepmind/boxoban-levels/.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy Lillicrap. An investigation of model-free planning. *arXiv*, 2019. URL http://arxiv.org/abs/1901.03559v2.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2023. URL http://github.com/google/flax.

Shengyi Huang, Jiayi Weng, Rujikorn Charakorn, Min Lin, Zhongwen Xu, and Santiago Ontañón. Cleanba: A reproducible and efficient distributed reinforcement learning platform, 2023. URL https://arxiv.org/abs/2310.00036.

Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Evan Hubinger, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. Risks from learned optimization in advanced machine learning systems, 2019. URL https://arxiv.org/abs/1906.01820.

Michael Ivanitskiy, Alexander F Spies, Tilman Räuker, Guillaume Corlouer, Christopher Mathwin, Lucia Quirke, Can Rager, Rusheb Shah, Dan Valentine, Cecilia Diniz Behn, Katsumi Inoue, and Samy Wu Fung. Linearly structured world representations in maze-solving transformers. In *UniReps: the First Workshop on Unifying Representations in Neural Models*, 2023. URL https://openreview.net/forum?id=pZakRK1QHU.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In Francis Bach and David Blei (eds.), *International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2342–2350, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/jozefowicz15.html.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35: 22199–22213, 2022. URL https://openreview.net/pdf?id=e2TBb5y0yFf.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023. URL https://arxiv.org/abs/2307.13702.

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 *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=DeG07_TcZvT.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. *Advances in Neural Information Processing Systems*, 36, 2024.

Ulisse Mini, Peli Grietzer, Mrinank Sharma, Austin Meek, Monte MacDiarmid, and Alexander Matt Turner. Understanding and controlling a maze-solving policy network. *arXiv*, 2023. URL http://arxiv.org/abs/2310.08043v1.

Stephen M. Omohundro. The basic AI drives. In *Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, pp. 483–492, NLD, 2008. IOS Press. ISBN 9781586038335.

OpenAI. Introducing OpenAI o1-preview, 2024. URL https://openai.com/index/introducing-openai-o1-preview/.

Niklas Sandhu Peters, Marc Alexa, and Special Field Neurotechnology. Solving sokoban efficiently: Search tree pruning techniques and other enhancements, 2023. URL https://doc.neuro.tu-berlin.de/bachelor/2023-BA-NiklasPeters.pdf.

Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models. *arXiv*, 2024. URL http://arxiv.org/abs/2404.15758v1.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering Llama 2 via Contrastive Activation Addition. *arXiv*, 2023. URL https://arxiv.org/abs/2312.06681.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. ISBN 9780136042594.

Stuart Russell. *Human compatible: artificial intelligence and the problem of control*. Penguin, 2019.

Jérémy Scheurer, Mikita Balesni, and Marius Hobbhahn. Large language models can strategically deceive their users when put under pressure. *arXiv*, 2023. URL http://arxiv.org/abs/2311.07590v3.

Max-Philipp B. Schrader. gym-sokoban, 2018. URL https://github.com/mpSchrader/gym-sokoban.

Eric Schwitzgebel and Mara Garza. A defense of the rights of artificial intelligences. *Midwest Studies In Philosophy*, 39(1):98–119, 2015. doi: 10.1111/misp.12032. URL http://www.faculty.ucr.edu/~eschwitz/SchwitzPapers/AIRights-150915.htm.

Rohin Shah, Vikrant Varma, Ramana Kumar, Mary Phuong, Victoria Krakovna, Jonathan Uesato, and Zac Kenton. Goal misgeneralization: why correct specifications aren't enough for correct goals. *arXiv preprint arXiv:2210.01790*, 2022. URL https://arxiv.org/abs/2210.01790.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, 28, 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf.

Peter Singer. Animal liberation. In *Ethics: Contemporary Readings*, pp. 284–292. Routledge, 2004.

Alex Tamkin, Mohammad Taufeeque, and Noah D. Goodman. Codebook features: Sparse and discrete interpretability for neural networks. *arXiv preprint arXiv:2310.17230*, 2023. URL https://arxiv.org/abs/2310.17230.

Borgar Þorsteinsson. Sokoban: levels, 2009. URL http://borgar.net/programs/sokoban/.

Brian Tomasik. A dialogue on suffering subroutines, 2015. URL https://longtermrisk.org/a-dialogue-on-suffering-subroutines/.

Alex Turner, Lisa Thiergart, David Udell, Gavin Leech, Ulisse Mini, and Monte MacDiarmid. Activation addition: Steering language models without optimization. *arXiv e-prints*, pp. arXiv–2308, 2023.

Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, Zhongwen Xu, and Shuicheng Yan. EnvPool: A highly parallel reinforcement learning environment execution engine. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22409–22421. Curran Associates, Inc., 2022. URL https://openreview.net/forum?id=BubxnHpuMbG.

Erik Wijmans, Manolis Savva, Irfan Essa, Stefan Lee, Ari S. Morcos, and Dhruv Batra. Emergence of maps in the memories of blind navigation agents. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=lTt4KjHSsyl.

Eliezer Yudkowsky. Artificial intelligence as a positive and negative factor in global risk. In Nick Bostrom and Milan M. ĆirkoviĆ (eds.), *Artificial Intelligence as a positive and negative factor in global risk*, 2006. URL https://api.semanticscholar.org/CorpusID:2629818.

A TRAINING THE TEST SUBJECT

DRC(D, N) architecture. Guez et al. (2019) introduced the Deep Repeating ConvLSTM (DRC), whose core consists of D convolutional LSTM layers with 32 channels and 3×3 filters, each applied N times per time step. Our DRC$(3, 3)$ – or just DRC for brevity – has 1.29M parameters. Before the LSTM core, two convolutional layers (without nonlinearity) encode the observation with 4×4 filters.

The LSTM core uses 3×3 convolutional filters, and a nonstandard `tanh` on the output gate (Jozefowicz et al., 2015). Unlike the original ConvLSTM (Shi et al., 2015), the input to each layer of a DRC consists of several concatenated components:

- The encoded observation is fed into each layer.
- To allow spatial information to travel fast in the ConvLSTM layers, we apply *pool-and-inject* by max- and mean-pooling the previous step's hidden state. We linearly combine these values channel-wise before feeding them as input to the next step.
- To avoid convolution edge effects from disrupting the LSTM dynamics, we feed in a 12×12 channel with zeros on the inside and ones on the boundary. Unlike the other inputs, this one is not zero-padded, maintaining the output size.

ResNet architecture. This is a convolutional residual neural network, also from Guez et al. (2019). It serves as a non-recurrent baseline that can only think during the forward pass (no ability to think for extra steps) but is nevertheless good at the game. The ResNet consists of 9 blocks, each with 4×4 convolutional filters. The first two blocks have 32 channels, and the others have 64. Each block consists of a convolution, followed by two (relu, conv) sub-blocks, each of which splits off and is added back to the trunk. The ResNet has 3.07M parameters.

Value and policy heads. After the convolutions, an affine layer projects the flattened spatial output into 256 hidden units. We then apply a ReLU and two different affine layers: one for the actor (policy) and one for the critic (value function).

RL training. We train each network for 2.003 billion environment steps[5] using IMPALA (Espeholt et al., 2018; Huang et al., 2023). For each training iteration, we collect 20 transitions on 256 actors using the network parameters from the previous iteration, and simultaneously take a gradient step. We use a discount rate of $\gamma = 0.97$ and V-trace $\lambda = 0.5$. The value and entropy loss coefficients are 0.25 and 0.01. We use the Adam optimizer with a learning rate of $4 \cdot 10^{-4}$, which linearly anneals to $4 \cdot 10^{-6}$ at the end of training. We clip the gradient norm to $2.5 \cdot 10^{-4}$. Our hyperparameters are mostly the same as Guez et al. (2019); see appendix A.1.

A* solver. We used the A* search algorithm to obtain optimal solutions to each Sokoban puzzle. The heuristic was the sum of the Manhattan distances of each box to its nearest target. Solving a single level on one CPU takes anywhere from a few seconds to 15 minutes.[6]

A.1 TRAINING HYPERPARAMETERS

All networks were trained with the same hyperparameters, which were tuned on a combination of the ResNet and the DRC$(3, 3)$. These are almost exactly the same as Guez et al. (2019), allowing for taking the *mean* of the per-step loss instead of the sum.

Time limits. During training, we want to prevent strong time correlations between the returns so the gradient steps are not correlated over time. For this reason, the time limit for each episode is uniformly random between 91 and 120 time steps.

Loss. The value and entropy coefficients are 0.25 and 0.01 respectively. It is very important to *not* normalize the advantages for the policy gradient step.

[5] A rounding error caused this to exceed 2B (appendix A.2).

[6] The A* solutions may be of independent interest, so we make them available at `https://huggingface.co/datasets/AlignmentResearch/boxoban-astar-solutions/`.

Gradient clipping and epsilon The original IMPALA implementation, as well as Huang et al. (2023), *sum* the per-step losses. We instead average them for more predictability across batch sizes, so we had to scale down some parameters by a factor of $1/640$: Adam ϵ, gradient norm for clipping, and L2 regularization).

Weight initialization. We initialize the network with the Flax (Heek et al., 2023) default: normal weights truncated at 2 standard deviations and scaled to have standard deviation $\sqrt{1/\text{fan_in}}$. Biases are initialized to 0. The forget gate of LSTMs has 1 added to it (Jozefowicz et al., 2015). We initialize the value and policy head weights with orthogonal vectors of norm 1. Surprisingly, this makes the variance of these unnormalized residual networks decently close to 1.

Adam optimizer. As our batch size is medium-sized, we pick $\beta_1 = 0.9$, $\beta_2 = 0.99$. The denominator epsilon is $\epsilon = 1.5625 \cdot 10^{-7}$. Learning rate anneals from $4 \cdot 10^{-4}$ at the beginning to $4 \cdot 10^{-6}$ at 2,002,944,000 steps.

L2 regularization. In the training loss, we regularize the policy logits with L2 regularization with coefficient 1.5625×10^{-6}. We regularize the actor and critic heads' weights with L2 at coefficient 1.5625×10^{-8}. We believe this has essentially no effect, but we left it in to more closely match Guez et al. (2019).

Software. We base our IMPALA implementation on Cleanba (Huang et al., 2023). We implemented Sokoban in C++ using Envpool (Weng et al., 2022) for faster training, based on gym-sokoban (Schrader, 2018).

A.2 Number of training steps

In the body of the paper we state the networks train for 2.003B steps. The exact number is $2\,002\,944\,000$ steps. Our code and hyperparameters require that the number of environment steps be divisible by $5\,120 = 256$ environments $\times$ 20 steps collected, because that is the number of steps in one iteration of data collection.

However, 2B is divisible by $5\,120$, so there is no need to add a remainder. We noticed this mistake once the networks already have trained. It is not worth retraining the networks from scratch to fix this mistake.

At some point in development, we settled on $80\,025\,600$ to approximate 80M while being divisible by 256×20 and 192×20. Perhaps due to error, this mutated into $1\,001\,472\,000$ as an approximation to 1B, which directly leads to the number we used.

A.3 Learning curve comparison

It is difficult to fully replicate the results by Guez et al. (2019). Chung et al. (2024) propose an improved method for RL in planning-heavy domains. They employ the IMPALA DRC$(3, 3)$ as a baseline and plot its performance in Chung et al. (2024, Figure 5). They plot two separate curves for DRC$(3, 3)$: that from Guez et al. (2019), and a decent replicated baseline. The baseline is considerably slower to learn and peaks at lower performance.

We did not innovate in RL, so were able to spend more time on the replication. We compare our replication to Guez et al. (2019) in appendix A.3, which shows that the learning curves for DRC$(3, 3)$ and ResNet are compatible, but not the one for DRC(1,1). Our implementation also appears much less stable, with large error bars and large oscillations over time. We leave addressing that to future work.

The success rate in appendix A.3 is computed over 1024 random levels, unlike the main body of the paper. Table 5 reports test and validation performance for the DRC and ResNet seeds which we picked for the paper body.

The parameter counts (table 4) are very different from what Guez et al. (2019) report. In private communication with the authors, we confirmed that our architecture has a comparable number of parameters, and some of the originally reported numbers are a typographical error.

Table 4: Parameter counts for each architecture.

Architecture	Parameter count	
DRC(3, 3)	1,285,125	(1.29M)
DRC(1,1)	987,525	(0.99M)
ResNet:	3,068,421	(3.07M)

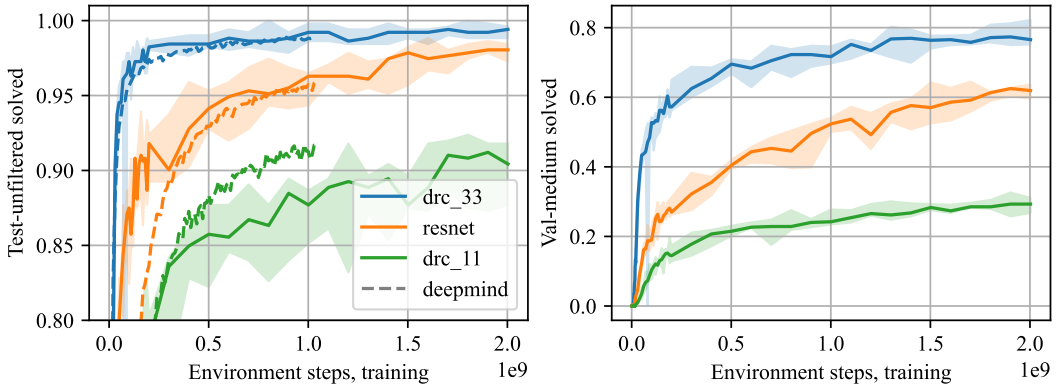


Figure 7: Success rate for Test-unfiltered and Validation-medium levels vs. environment steps of training. Each architecture has 5 random seeds, the solid line is the pointwise median and the shaded area spans from the minimum to the maximum. The dotted lines are data for the performance of architectures extracted from the (Guez et al., 2019) PDF file. The values are slightly different from what fig. 5 and section 3 report because they are calculated on a random sample of 1024 levels (24 levels are repeated for test-unfiltered, which only has 1000 levels).

Table 5: Success rate and return of DRC and ResNet on the unfiltered test set at various training environment steps.

Training Env Steps	Test Unfiltered				Valid Medium			
	ResNet		DRC(3, 3)		ResNet		DRC(3, 3)	
	Success	Return	Success	Return	Success	Return	Success	Return
100M	87.8	8.13	95.4	9.58	18.6	-6.59	47.9	-0.98
500M	93.1	9.24	97.9	10.21	39.7	-2.64	66.6	2.62
1B	95.4	9.75	99.2	10.47	50.0	-0.64	70.4	3.40
2B	97.9	10.29	99.3	10.52	59.4	1.16	76.6	4.52

B GENERALIZING THE DRC$(3, 3)$ TO LARGER LEVELS

We license the levels by Þorsteinsson (2009) as GPLv3, and make them available in the supplementary material. The performance of the DRC$(3, 3)$ on each set of levels is in table 6. The sets with higher scores are also those which humans find easier: for example, "Dimitri & Yorick" was made for children by Jacques Duthen and consists entirely of relatively small levels (largest is 12×10) with at most 5 boxes. The sets where the DRC$(3, 3)$ solves nothing are also difficult for the authors of this paper.

We tried giving the DRC$(3, 3)$ from 2 to 128 extra thinking steps for these levels, incrementing in powers of two. For most sets, we find some benefit to 2-4 thinking steps, but no more. The sole exception is XSokoban, which contains one level (fig. 2, right) which requires 128 steps of thinking to solve.

We encourage the reader to go to the website by Þorsteinsson (2009) and try solving the levels.

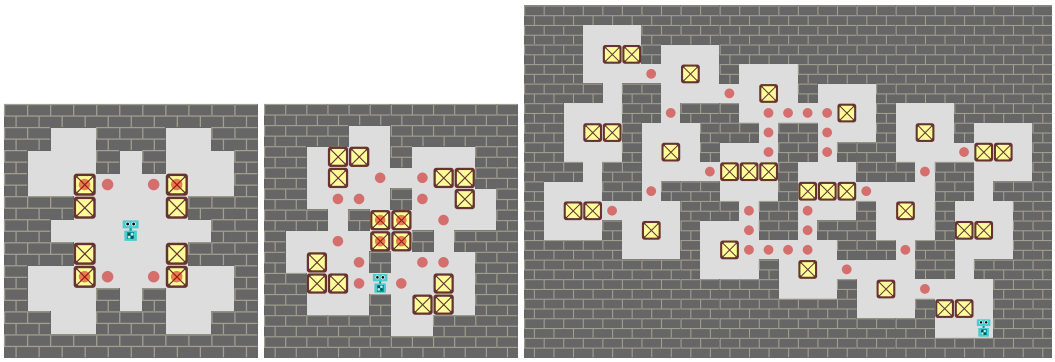


Figure 8: Thinking steps are not always useful: For the left level, the network succeeds with 0 to 16 thinking steps, but fails with 32 or more thinking steps. The middle and right levels are too difficult for the DRC$(3, 3)$, and it always fails on them independent of thinking steps, though it comes up with decent partial solutions.

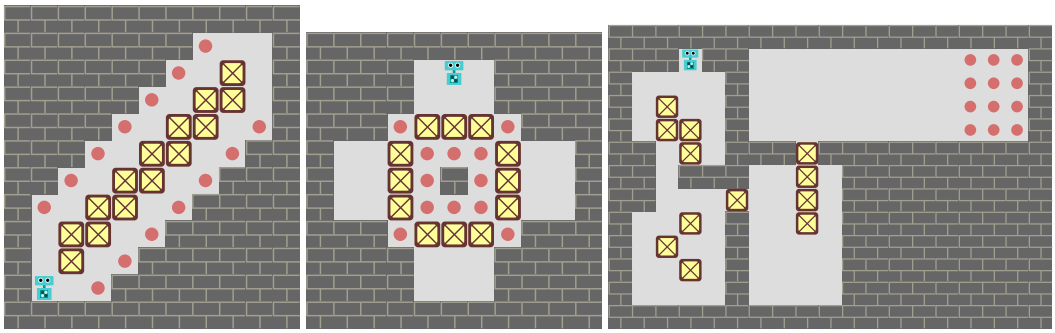


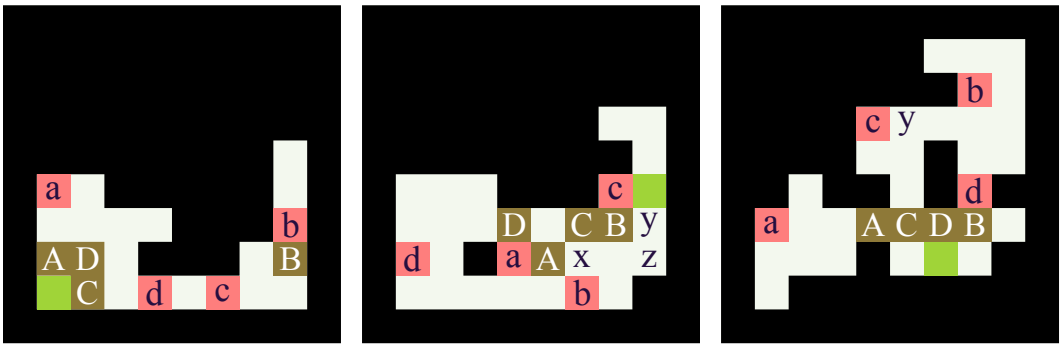
Figure 9: Some levels require thinking steps for success: For the left level, the network always succeeds. For the middle level, it succeeds at 32 or more thinking steps. For the right level, it needs 128 thinking steps to succeed.

C CASE STUDIES

Case Study: Thinking makes some levels solvable fig. 10(a). For example, thinking lets the DRC solve fig. 10(a). In the no-thinking condition, the DRC first pushes box C one square to the right. It then goes back to push A to a, but it has now become impossible to push box B onto b. In contrast, after thinking, the DRC pushes A to a first, which lets it solve the level.

Table 6: Performance of the DRC$(3, 3)$ on each set of levels by (Þorsteinsson, 2009). The "Max solved" columns represent the proportion of levels solved at the number of steps in the "at steps" column, which is the highest solved proportion for each number of thinking steps tried.

	All levels				Levels larger than 10×10			
Level collection	#	solved	Max solved	at steps	#	solved	Max solved	at steps
Dimitri & Yorick	61	83.6%	88.5%	16	0	—	—	—
Sokoban Jr. 1	60	80.0%	81.7%	2	19	73.7%	73.7%	0
Howard's 3rd set	40	60.0%	62.5%	4	1	0.0%	0.0%	0
Simple sokoban	61	54.1%	60.7%	16	51	47.1%	54.9%	16
Sokoban Jr. 2	54	48.1%	50.0%	2	40	45.0%	47.5%	2
Microban	155	23.9%	24.5%	4	17	5.9%	11.8%	2
Deluxe	55	21.8%	23.6%	2	1	0.0%	0.0%	0
Sokogen 990602	78	20.5%	23.1%	8	0	—	—	—
Yoshio Automatic	52	13.5%	17.3%	2	0	—	—	—
Sasquatch III	16	6.2%	6.2%	0	8	0.0%	0.0%	0
Howard's 1st set	100	5.0%	6.0%	2	54	0.0%	0.0%	0
Howard's 2nd set	40	5.0%	10.0%	4	22	0.0%	0.0%	0
Microcosmos	40	5.0%	7.5%	4	0	—	—	—
Still more levels	35	2.9%	2.9%	0	34	2.9%	2.9%	0
Sasquatch IV	36	2.8%	2.8%	0	20	0.0%	0.0%	0
Xsokoban	40	2.5%	5.0%	128	39	2.6%	5.1%	128
Sasquatch	49	2.0%	4.1%	4	39	0.0%	2.6%	4
David Holland 1	10	0.0%	0.0%	0	5	0.0%	0.0%	0
David Holland 2	10	0.0%	0.0%	0	9	0.0%	0.0%	0
Howard's 4th set	32	0.0%	0.0%	0	30	0.0%	0.0%	0
Mas Sasquatch	50	0.0%	0.0%	0	43	0.0%	0.0%	0
Nabokosmos	40	0.0%	0.0%	0	0	—	—	—
Sokoban	50	0.0%	0.0%	0	48	0.0%	0.0%	0



(a) file 0, level 18 unsolved $\xrightarrow{\text{think}}$ solved (b) file 0, level 53 $\xrightarrow{\text{think}}$ solved faster (c) file 0, level 153 $\xrightarrow{\text{think}}$ solved slower

Figure 10: Case studies of three medium-validation levels demonstrating different behaviors after 6 thinking steps. Colors are as in appendix F. Boxes and targets are paired in upper- and lower-case letters respectively, such that the DRC's best solution places boxes on targets in alphabetical order. Videos available at this https URL. Levels solved faster obtain higher return as the agent incurs the per-step penalty fewer times. Note that the letters used are for reference and not a property of Sokoban

Case Study: Thinking speeds up solving fig. 10(b). fig. 10(b) illustrates a scenario where the agent happens. In the no-thinking condition, the DRC takes many steps back and forth before pushing any boxes. First it goes down to y, up to c, then down onto z, back up to y and to z again. It then proceeds to solve the rest of the puzzle: push box A onto a, prepare box B on x and box C where B originally was, push in boxes B, C and finally D. In the thinking condition, the DRC makes a beeline for A and then plays the same solution.

Case Study: Thinking slows downfig. 10(c). Figure 10(c) illustrates a scenario where thinking time results in a slower solution. In the no thinking condition, the DRC starts by pushing box C into position y, then pushes boxes A, then B into place. On the way back down, the DRC pushes C onto c and finally D onto d. In contrast, in the thinking condition, the DRC goes the other way and starts by pushing B onto b. The subsequent solution (A, C, then D) is the same, but the DRC has wasted time trekking back from B to A.

D Probe Training

Probe architecture We train multiple different linear probes on the hidden states h and cell states c activations of the network. The probes are trained either on the states of the individual layers or by concatenating the states of all the three layers. We collect the activations of the model by letting it play through the hard levels. The training set for the probes are constructed by taking a random sample of cached levels and including all the timesteps in the level excluding the first five timesteps. We exclude these intial steps as we believe they can be noisy as the network comes up with the stategy to solve the levels.

The probes are trained with logistic regression with L1 decay using the Scikit-Learn library. We do a grid-search on the learning rate and the L1 weight decay to select the probe that has the highest F1-score on a validation set.

For multi-class probe targets, each potential output class l is treated like a different data point. That is: for each label l, we consider a prediction positive if the highest probe logit is for label l, and negative otherwise. We consider a data point positive iff the true label is label l. Then, we compute the confusion matrix and the F1 score from the $n \cdot l$ data points.

E Details about Sparse Autoencoders (SAEs)

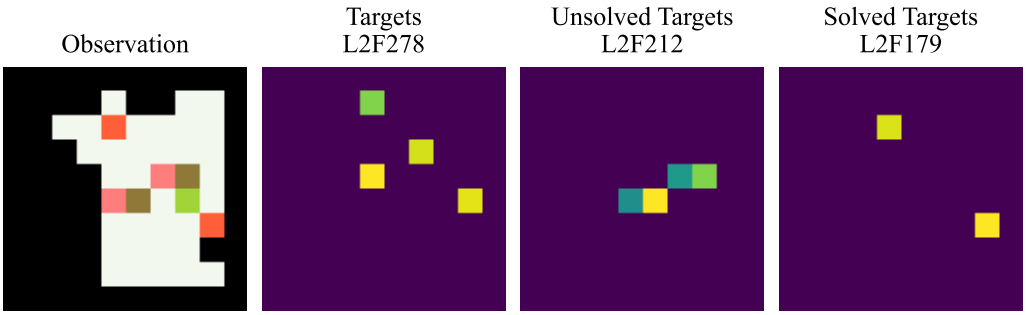


Figure 11: Visualization of some interpretable features from the SAE of last layer. The precision, recall, and F1 score for the features are reported in table 7.

In order to search for more monosemantic and interpretable features in the network, we train sparse-autoencoders (SAEs) (Huben et al., 2023; Bricken et al., 2023) on the individual squares in the h hidden state of the network consisting of 32 neurons. Thus, we get a 10×10 visualization for each SAE feature as shown in Figure 11. We use the top-k activation function (Gao et al., 2024; Tamkin et al., 2023) for the SAE as it is currently the state-of-the-art method to train SAEs and is easier to work with as it directly sets the L_0 norm of the SAE activations. The hyperparameter search space for training the SAE is provided in Table 8. We train separate SAEs for each layer with the specified

hyperparameters and pick the one that achieves greater than 90% explained variance while having interpretable features assessed through manual visual inspection.

We provide examples of some of the interpretable features from an SAE trained on the last layer with $k = 8$ in table 9 with corresponding visualization in fig. 11. For the "Target", "Unsolved Target", and "Solved Target" concepts (table 7), we observed cases where the SAE feature is offset from the ground truth by 1 square in either horizontal or vertical directions. The level setup includes a permanent one-square outer-edge wall, so this offset never results in an out-of-bounds issue. We evaluated these potential "Offset" variants for the Target, Unsolved, and Solved concepts.

Many of the interpretable features that the SAE finds are already embedded in individual channels. We find that there exists individual channels for predicting each of the 4 actions in the hidden state of the last layer. The SAE is able to dedicate a separate neuron in its hidden layer for each of these action features. However, we find that the features found by the SAE are in fact less monosemantic than the channel. Table 3 reports the precision, recall, and F1 scores for each of these features as evaluated through channels, SAE neurons, and linear probe trained against the ground truth action predictions. We find that the SAE features have lower F1 scores than channels by a margin of 5.9% on average. On the other hand, the linear probes trained across all channels of the hidden state have similar F1 scores as the specified channels indicating that the channels are monosemantic and cannot be improved upon by combining various channels by a linear probe.

Table 7: SAE feature scores

CONCEPT	FEATURE #	OFFSET (DY,DX)	F1	PRECISION	RECALL
Target	L3F278	(1, 0)	92.0	85.2	100.0
Unsolved targets	L3F212	(0, 0)	89.1	91.3	87.0
Solved targets	L3F179	(0, 0)	63.2	46.2	99.8

Table 8: Hyperparameter search space for training SAE

HYPERPARAMETER	SEARCH SPACE
k	$\{4, 8, 12, 16\}$
learning rate	$\{1e-5, 5e-5, 1e-4, 5e-4, 1e-3\}$
expansion factor	$\{16, 32, 64\}$

Table 9: SAE Feature Concepts

CONCEPT	DESCRIPTION
Target	The 4 target squares (static)
Unsolved	The (0 to 4) target squares without a block on them
Solved	The (0 to 4) target squares with blocks on them
Agent Up	The agent will move Up next step
Agent Down	The agent will move Down next step
Agent Left	The agent will move Left next step
Agent Right	The agent will move Right next step

F ADDITIONAL QUANTITATIVE BEHAVIOR FIGURES

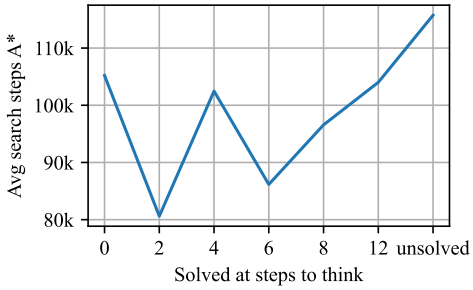


Figure 12: Number of thinking steps required to solve the level vs. number of nodes A* needs to expand to solve it. The trend is increasing at the end but very unclear, indicating different heuristics used by the DRC and A*.

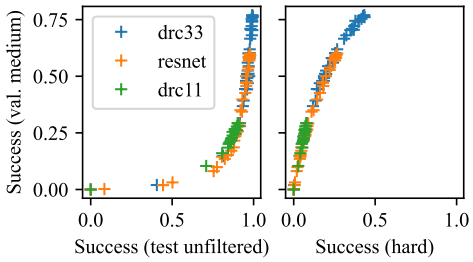
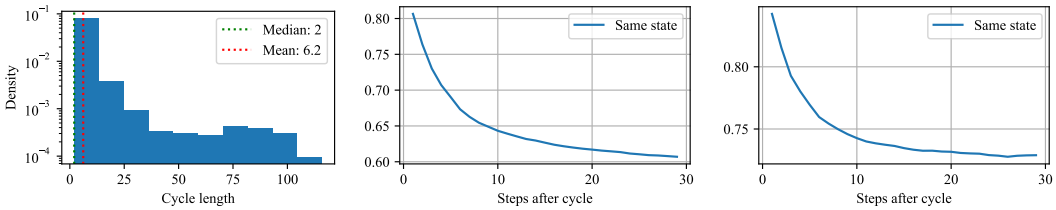


Figure 13: Success rate on datasets of various difficulty, for various checkpoints of each architecture. This deviates very little from a curve, which shows that ResNets and DRCs which are equally good at the easier sets are also equally good at the harder sets. Perhaps DRC(1,1) is a slight exception, but it also performs much worse than the others overall (see appendix A.3).



(a) Cycle length distribution

(b) with cycles from all levels

(c) with cycles from those levels that were solved by the agent

Figure 14: We replace N-length cycles with N-thinking steps and check for the same state after some timesteps. *(a)* A histogram of cycle lengths in the medium-validation set. *(b, c)* After replacing a cycle with the same length in thinking steps, are all the states the same for the next x steps?

Table 10: Breakdown of levels by category at 6 thinking steps.

LEVEL CATEGORIZATION	PERCENTAGE
Solved, previously unsolved	6.87
Unsolved, previously solved	2.23
Solved, with better returns	18.98
Solved, with the same returns	50.16
Solved, with worse returns	5.26
Unsolved, with same or better returns	15.14
Unsolved, with worse returns	1.36

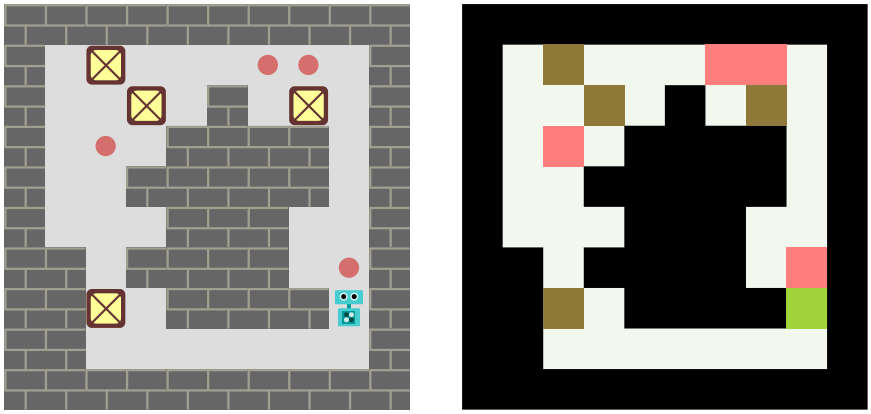


Figure 15: **Left:** A Sokoban level from the hard set. **Right:** the same level as the NN sees it, one pixel per tile. Walls are black, boxes are brown, targets are pink and the robot is green.