

Stable Planning through Aligned Representations in Model-Based Reinforcement Learning

Misagh Soltani, Forest Agostinelli

Keywords: Model-Based Reinforcement Learning, Planning, Reinforcement Learning, Alignment Model, Aligned Representation Learning

Summary

World models that reliably predict state transitions can facilitate the integration of planning with reinforcement learning to solve sparse-reward, long-horizon sequential decision-making problems. However, enabling such learned world models to generalize to noise for which the underlying Markov decision process is invariant (e.g., lighting variations, blurring, or rotations) requires training the world model on a dataset with such noise. Unfortunately, this does not explicitly capture these invariances and requires that the world model, and any functions trained with data generated from it, be re-trained if it fails to generalize to new kinds of noise. To solve this problem, we introduce Stable Planning through Aligned Representations (SPAR), an algorithm that explicitly captures invariances by training an alignment network to map noisy states to the latent representation of clean states while keeping the learned world model fixed. SPAR first trains a world model with a discrete latent space on clean states. SPAR then trains an alignment network to map noisy states to the same latent representation as the clean states, exploiting the discrete latent structure by rounding the output of the alignment network after training so that it exactly matches the clean latent representation. As a result, SPAR preserves the ability to use a discrete world model across thousands of timesteps without accumulating any error by rounding the predicted discrete latent states. Our experiments show that SPAR can be used to successfully solve over 89% and 96% of problem instances from the Rubik’s Cube and Sokoban domains across 16 and 18 different visual transformations respectively. Furthermore, SPAR can be used to solve the Rubik’s Cube based on real-world pictures taken by a phone, without ever having seen real-world Rubik’s Cubes during training. Our experiments show that continuous representations lead to model degradation, indicating that the discrete latent component is crucial to SPAR’s success.

Contribution(s)

1. We introduce Stable Planning through Aligned Representations (SPAR), an algorithm to adapt noisy observations to a discrete world model trained on clean observations. The discrete world model remains fixed and does not need to be re-trained.
Context: While discrete world models have been shown to accurately predict transitions over long horizons, as we will show, they are not able to adapt to noise. This hinders their application to real-world scenarios.
2. We show that SPAR can be generalized to real-world observations of the Rubik’s Cube, despite not being trained on such data. In particular, we show that, in many cases, it can accurately solve the Rubik’s Cube based on pictures from a phone.
Context: These results show that SPAR can be used for applications beyond simulation.

Stable Planning through Aligned Representations in Model-Based Reinforcement Learning

Misagh Soltani¹, Forest Agostinelli¹

msoltani@email.sc.edu, foresta@cse.sc.edu

¹Department of Computer Science and Engineering, University of South Carolina, SC, USA

Abstract

World models that reliably predict state transitions can facilitate the integration of planning with reinforcement learning to solve sparse-reward, long-horizon sequential decision-making problems. However, enabling such learned world models to generalize to noise for which the underlying Markov decision process is invariant (e.g., lighting variations, blurring, or rotations) requires training the world model on a dataset with such noise. Unfortunately, this does not explicitly capture these invariances and requires that the world model, and any functions trained with data generated from it, be re-trained if it fails to generalize to new kinds of noise. To solve this problem, we introduce Stable Planning through Aligned Representations (SPAR), an algorithm that explicitly captures invariances by training an alignment network to map noisy states to the latent representation of clean states while keeping the learned world model fixed. SPAR first trains a world model with a discrete latent space on clean states. SPAR then trains an alignment network to map noisy states to the same latent representation as the clean states, exploiting the discrete latent structure by rounding the output of the alignment network after training so that it exactly matches the clean latent representation. As a result, SPAR preserves the ability to use a discrete world model across thousands of timesteps without accumulating any error by rounding the predicted discrete latent states. Our experiments show that SPAR can be used to successfully solve over 89% and 96% of problem instances from the Rubik’s Cube and Sokoban domains across 16 and 18 different visual transformations respectively. Furthermore, SPAR can be used to solve the Rubik’s Cube based on real-world pictures taken by a phone, without ever having seen real-world Rubik’s Cubes during training. Our experiments show that continuous representations lead to model degradation, indicating that the discrete latent component is crucial to SPAR’s success. An implementation of SPAR is publicly available at <https://github.com/misaghsoltani/SPAR>.

1 Introduction

Many real-world problems require long-horizon planning to find solutions. Planning requires a state transition function (also known as the “world model”), which maps states and actions to next states. It has been shown that world models can be learned, in both simulation and the real world, from data obtained by interacting with the environment (Racanière et al., 2017; Hafner et al., 2019; Bagatella et al., 2021; Tian et al., 2021; Agostinelli & Soltani, 2024). The world model can then be used to generate data to train a function (i.e., a policy, value, or heuristic function) with reinforcement learning that can be used to guide a planner. However, these approaches are not robust to noise. For example, if irrelevant distractors that were not seen during training are part of a state at test time, the learned world model’s prediction may no longer be useful for planning. As a result, these approaches will most likely fail to generalize beyond very carefully controlled settings. Furthermore, when specifying goals with goal images, noise present in the environment can make goal state

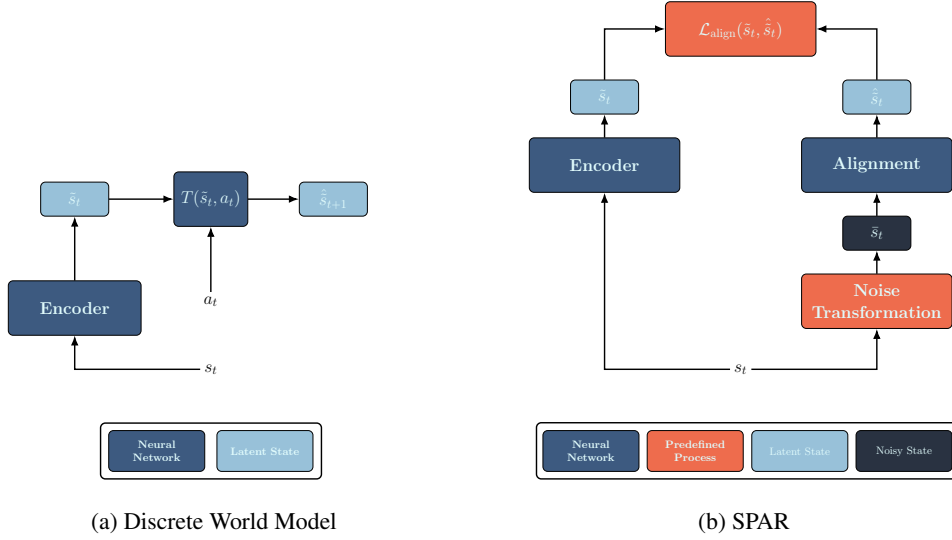


Figure 1: Overview of SPAR. First, a discrete world model is trained to encode clean states to a discrete latent state and map discrete latent states and actions to next discrete latent states (left). Then, an alignment model is trained to map noisy states to the same latent representation as the clean states (right).

identification difficult. For example, if a goal image is specified in a bright room and the room is currently dark, unless the action space includes controlling the lights, there is no sequence of actions that can reach the goal image.

One approach to solving this problem of noise would be to train the world model on noisy states. However, this suffers from the following drawbacks: 1) the world model is forced to capture noisy transitions, which may require more parameters, hinder performance, and does not capture the underlying invariances; 2) the world model and any functions trained from data generated by it must be re-trained if it fails to generalize to new noise; 3) goal state identification still depends on the noise present in the state, which means different goal images must be given based on the noise present in the start state. To address these issues, we introduce Stable Planning through Aligned Representations (SPAR). SPAR trains a discrete world model that encodes states in a clean environment to a discrete latent state and predicts next discrete latent states given current latent states and actions. SPAR then adds noise to the clean states used to train the world model and trains an alignment model that maps the noisy states to the same latent representation as the clean states. Since the latent representation is discrete, after training, the output of the alignment model is rounded with the aim of exactly matching the true discrete latent state. Given that discrete world models have been shown to accurately predict next states across thousands of timesteps without model degradation (Agostinelli & Soltani, 2024), this also enables planning over long horizons without model degradation in the presence of noise. Our approach is summarized in Figure 1.

2 Related Work

Model-based reinforcement learning (RL) and planning. Model-based reinforcement learning (MBRL) improves sample efficiency by learning a dynamics model and exploiting that model for policy improvement or planning (Sutton, 1990). The Dyna architecture interleaves real experience with model-generated rollouts to update value functions and policies (Sutton, 1990). More recent methods couple learned neural models with powerful search procedures. AlphaZero learns value and policy networks and performs Monte Carlo Tree Search (MCTS) in known perfect-information games (Silver et al., 2018), while MuZero achieves superhuman performance without access to the true simulator by learning a latent dynamics model for planning with MCTS (Schrittwieser et al.,

2020). Deep Planning Network (PlaNet), Dreamer, and DreamerV2 learn latent world models and optimize policies via latent imagination, but they do not perform explicit test-time search to arbitrary goal states (Hafner et al., 2019; 2020; 2021).

Learning state representations for search. A body of work learns symbolic or latent state spaces that enable classical search from pixels. LatPlan learns a discrete binary latent representation with an autoencoder and induces a classical planning model solvable by off-the-shelf planners (Asai & Fukunaga, 2018; Asai et al., 2022). PPGS (Bagatella et al., 2021) learns a continuous latent space with forward and inverse models, constructs a latent-state graph using threshold-based state re-identification, and then uses breadth-first search to solve combinatorial puzzles. However, continuous latent rollouts can accumulate error and force frequent re-planning, while breadth-first search scales poorly to large state spaces. A complementary line learns heuristics to guide search directly from experience. DeepCubeA trains a deep neural network to approximate the cost-to-go via deep approximate value iteration and uses the learned heuristic with A* search to solve the Rubik’s Cube and other puzzles, training against a single fixed goal (the canonical solved state) (Agostinelli et al., 2019). DeepCubeAI (Agostinelli & Soltani, 2024) instead learns a discrete world model and a goal-conditioned heuristic defined over binary latent states. Provided errors remain below 0.5, rounding each bit eliminates compounding errors in long rollouts and enables state re-identification, supporting efficient search in latent space (Agostinelli et al., 2024b). Our work builds on this discrete-latent planning foundation by learning a lightweight alignment network to handle noisy observations, leaving the world model and heuristic unchanged.

Handling noisy observations. Domain shift in observations (e.g., lighting, background, camera pose) challenges visual planning systems. Domain randomization broadens the training distribution in simulation so real observations appear as another randomized variant (Tobin et al., 2017; Sadeghi & Levine, 2017). In robotics, sim-to-real pipelines augment randomization with image-to-image translation. For example, Randomized-to-Canonical Adaptation Networks (RCAN) maps randomized renders to a canonical domain to improve transfer (James et al., 2019). In RL specifically, data augmentation improves policy robustness (Laskin et al., 2020; Yarats et al., 2021). Disentangled Representation Learning Agent (DARLA) learns disentangled representations with a modification of the variational autoencoder (β -VAE) (Higgins et al., 2017a) pretraining to achieve zero-shot transfer across visual changes before learning to act (Higgins et al., 2017b). Beyond training-time robustness, test-time adaptation methods update perception or policies during deployment without labels or rewards. For example, Policy Adaptation during Deployment (PAD) for self-supervised policy adaptation (Hansen et al., 2021) and invariance through latent alignment (ILA), which aligns target-domain features to the source distribution without paired data (Yoneda et al., 2022). These approaches, however, generally target policy performance and short-horizon control rather than long-horizon heuristic search that requires exact state re-identification. Denoising autoencoders (Vincent et al., 2008) are also related, as they map corrupted observations to clean ones. However, they operate in pixel space and do not guarantee that the denoised output encodes to exactly the same discrete latent code as the clean state, which is the property required for exact state re-identification and goal testing during search. SPAR builds on this work by mapping noisy observations to clean observations, but does so in a discrete latent space that has already been learned on clean observations. This preserves the long-horizon stability and exact state re-identification of discrete-latent planning (Agostinelli & Soltani, 2024) while avoiding retraining the world model and heuristic function for each new visual change.

3 Preliminaries

3.1 Pathfinding

The particular kind of planning problems SPAR addresses are pathfinding problems. A pathfinding problem is defined as a weighted directed graph (Pohl, 1970), where nodes represent states, edges represent actions that transition between states, weights on the edges represent transition costs, a

given state represents the start state, and a given set of states represents the goal. The transition function, T , defines how actions transform states, where $s_{t+1} = T(s_t, a_t)$ for action a_t , if and only if there exists an edge connecting state s_t to s_{t+1} . The transition-cost function, $c(s_t, a_t)$, is the cost of taking action a_t in state s_t . While we will use the aforementioned notation throughout the rest of the paper, from a reinforcement learning perspective, a pathfinding problem can also be defined as a deterministic undiscounted Markov decision process (MDP) (Puterman, 2014). Given a pathfinding problem, the objective is to find a sequence of actions that transforms the start state into a goal state with a preference for paths with the lowest cost, where the cost of a path is the sum of transition costs. The cost-to-go is the cost of a lowest cost path, also known as a shortest path.

3.2 DeepCubeAI

Training. DeepCubeAI (Agostinelli & Soltani, 2024) learns a discrete world model represented as a deep neural network (DNN) (Schmidhuber, 2015) that maps states to binary latent states from an offline dataset of state, action, and next state tuples. The world model is learned using an encoder to map states to discrete latent states, a decoder that maps discrete latent states to states, and a world model that maps discrete latent states and actions to next discrete latent states. The encoder and decoder are trained to minimize the reconstruction error between the input to the encoder and the output of the decoder. The world model is trained simultaneously with the encoder and decoder to accurately predict the next latent state.

After learning, the outputs of the discrete world model are rounded when predicting next discrete latent states, which prevents model degradation when errors are less than 0.5. Using the learned world model, DeepCubeAI then learns a heuristic function represented as a deep Q-network (DQN) (Mnih et al., 2015), using Q-learning (Watkins & Dayan, 1992) to estimate the cost-to-go to a given goal when starting from a given state and taking a given action. Training data is obtained by using the world model to generate new experiences from the offline dataset. As a result, the heuristic function takes discrete latent states as input.

Solving Pathfinding Problems. After training, a particular pathfinding problem is given to DeepCubeAI in the form of an image representing the start state and an image representing the goal state. DeepCubeAI then encodes the start state and goal state into the discrete latent space and uses the learned world model to perform heuristic search in the latent space, guided by the learned heuristic function, to find a path from the latent start state to the latent goal state. The heuristic search algorithm checks whether a latent state encountered during search is a goal state by checking if all bits match the latent goal state. Since the world model is represented as a DNN, using it for heuristic search can be computationally expensive. Therefore, Q* search (Agostinelli et al., 2024b), a variant of A* search (Hart et al., 1968), is used to find paths. Q* search uses a DQN as the heuristic function to ensure the number of applications of the transition function (in this case, the world model) and number of applications of the heuristic function remain constant with respect to the size of the action space for each iteration of search as opposed to growing linearly, as is the case for A* search.

4 Methods

SPAR assumes the existence of a trained encoder that maps states to discrete latent states and a trained world model that maps discrete latent states and actions to next discrete latent states. Given a trained world model, our goal is to train an alignment model that maps noisy states to their corresponding discrete latent states obtained from the clean state. We obtain a dataset of noisy states paired with the discrete latent representations of their corresponding clean states by adding noise to the clean states used to train the world model and encoding those clean states with the trained encoder. As a result, SPAR does not require that any additional real-world training data be collected. We train the alignment model by minimizing the mean squared error (MSE) between the predicted latent states and the target latent states.

4.1 Discrete World Model

The discrete world model maps pixel observations into a discrete latent space and learns to predict latent state transitions. Let E be the encoder, D the decoder, and T the transition model. The encoder maps a state, s , to a discrete latent state $\tilde{s} = E(s)$. To ensure the output is discrete, the output of the encoder is passed through a logistic activation function and rounded to $\{0,1\}$. During training, a straight-through estimator (Bengio et al., 2013) is used to enable gradient-based training. The decoder maps the discrete latent state back to the original state space $\hat{s} = D(\tilde{s})$. Given an action, a , the transition model maps latent states at timestep, t , to latent states at timestep $t + 1$, $\hat{\tilde{s}}_{t+1} = T(\tilde{s}_t, a)$.

Given an offline dataset of transitions $\{(s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)})\}_{i=1}^N$, we optimize two losses. First, a reconstruction loss $\mathcal{L}_{\text{rec}}$ that encourages meaningful encodings and decodings, as given in Equation 1.

$$\mathcal{L}_{\text{rec}}(\theta) = \frac{1}{N} \sum_{i=1}^N \left[\frac{1}{2} \|s_t^{(i)} - D(E(s_t^{(i)}))\|_2^2 + \frac{1}{2} \|s_{t+1}^{(i)} - D(E(s_{t+1}^{(i)}))\|_2^2 \right]. \quad (1)$$

Second, we define a transition loss $\mathcal{L}_m$ that couples the encoder and transition model. Let $\tilde{s}_t^{(i)} = E(s_t^{(i)})$ and $\tilde{s}_{t+1}^{(i)} = E(s_{t+1}^{(i)})$ be the continuous encoder outputs, and let $r(\cdot)$ denote element-wise rounding to $\{0,1\}$. The transition model predicts the next latent state $\hat{\tilde{s}}_{t+1}^{(i)} = T(\tilde{s}_t^{(i)}, a_t^{(i)})$. We use a symmetric loss with a stop-gradient operator $\text{sg}(\cdot)$ applied to one side of each term to stabilize encoder-transition training, as given in Equation 2.

$$\mathcal{L}_m(\theta) = \frac{1}{N} \sum_{i=1}^N \left[\frac{1}{2} \|r(\tilde{s}_{t+1}^{(i)}) - \text{sg}(r(\hat{\tilde{s}}_{t+1}^{(i)}))\|_2^2 + \frac{1}{2} \|\text{sg}(r(\tilde{s}_{t+1}^{(i)})) - r(\hat{\tilde{s}}_{t+1}^{(i)})\|_2^2 \right]. \quad (2)$$

The two losses are combined using a weighted sum, as shown in Equation 3. The parameter, ω , is initially set to a small value to weight the reconstruction loss higher and is gradually increased until both are weighted equally.

$$\mathcal{L}(\theta) = (1 - \omega)\mathcal{L}_{\text{rec}}(\theta) + \omega\mathcal{L}_m(\theta). \quad (3)$$

4.2 Alignment Model

Given trained components of the discrete world model, (E, T, D) , learned on clean states, we train an alignment network A that maps a visually transformed observation $\bar{s}$ to the discrete latent state of the corresponding clean observation s . Training data is obtained by adding noise to the data that was used to train the world model. Supervised learning is used to train A by minimizing the MSE between its real-valued outputs and the discrete latent targets produced by the encoder. The cost-to-go of a state to any possible goal should be invariant to the noise added to that state. For example, for the Rubik’s Cube, the cost-to-go is invariant to changes in background color, but not to drastic changes in color of the stickers on the Rubik’s Cube.

Symbolically, given a clean state, s_t , noise is added from some stochastic function, $\mathcal{N}$, to obtain $\bar{s}_t = \mathcal{N}(s_t)$, and the discrete latent encoding of s is obtained using the encoder, $\tilde{s}_t = E(s_t)$. Next, the alignment model, A , attempts to predict $\tilde{s}_t$ from $\bar{s}_t$, $\hat{\tilde{s}}_t = A(\bar{s}_t)$. A is trained with gradient descent to minimize the loss in Equation 4, where ϕ denotes the parameters of A . At inference time, we obtain discrete alignment codes by rounding the outputs $A(\bar{s}_t)$ element-wise to $\{0,1\}$.

$$\mathcal{L}_{\text{align}}(\phi) = \frac{1}{N} \sum_{i=1}^N \|\hat{\tilde{s}}_t^{(i)} - \tilde{s}_t^{(i)}\|_2^2 \quad (4)$$

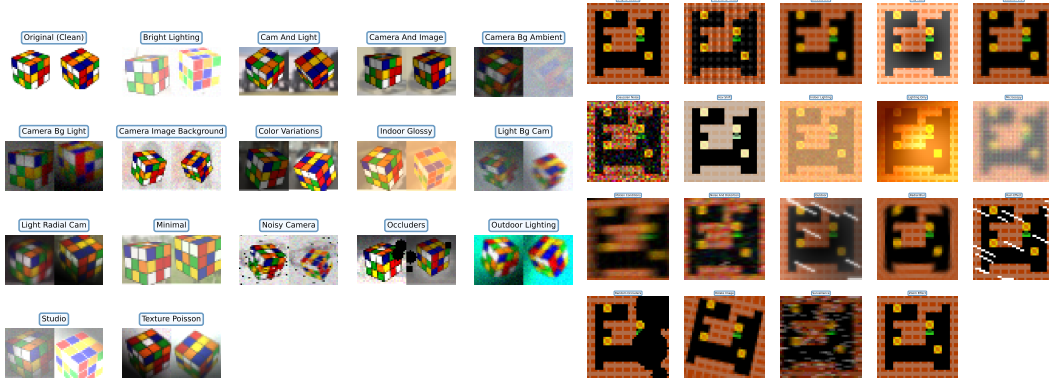


Figure 2: Visualization of sample data from the datasets used for training the alignment model for the Rubik’s Cube (left) and Sokoban (right).

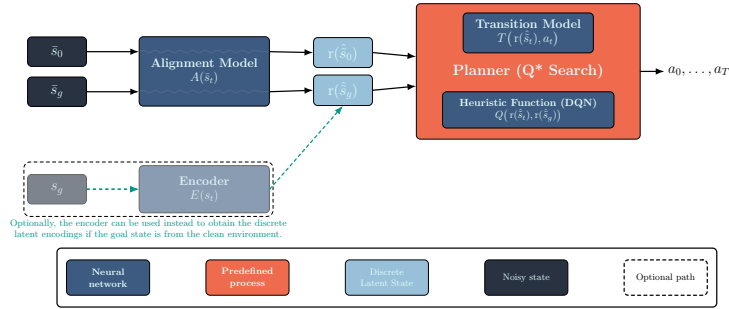


Figure 3: Overview of the planning procedure at inference time starting from a noisy state.

Visual variations. The stochastic noise function, $\mathcal{N}$, is comprised of several visual variations to which the cost-to-go is invariant. These include pre-render (background), object-render (lighting, color, geometry), and post-render stages. Additionally, we apply object-level and background transformations to the Rubik’s Cube, such as color shifts and calibrations, sticker wear, material finish, camera viewpoint changes and zoom, and a physically motivated directional lighting model (Tobin et al., 2017; Zhong et al., 2020; Shorten & Khoshgoftaar, 2019; Hendrycks & Dietterich, 2019; Da et al., 2025). Backgrounds used for Rubik’s Cube include randomly selected images from the CIFAR dataset (Krizhevsky et al., 2009). These transformations were applied with parameters sampled per-frame to cover a wide range of observation changes. The stochastic noise function combines several of these individual transformations to create new visual transformations. Examples of the images produced by our noise function are shown in Figure 2 and visualizations and detailed explanations of all transformations and their combinations are given in Supplementary Materials 11.

4.3 Finding Paths Given Noisy States

Given a noisy start state, $\bar{s}_0$ and a noisy goal state $\bar{s}_g$, the alignment model is used to map them to the latent encodings for their corresponding clean states, $\hat{s}_0$ and $\hat{s}_g$, respectively. The output of the alignment model is then rounded so that, if the approximation error for all bits is below 0.5, the approximated clean latent state will exactly match the true latent state. This rounded output is then given to the latent planner. The latent planner performs Q* search in latent space with the learned world model as the transition model and the learned DQN as the heuristic function. An overview of this process is shown in Figure 3.

5 Experiments

We evaluate SPAR by isolating the role of discrete latent representations and the alignment network in long-horizon planning. Below we detail the experimental setup, models, training procedures, and evaluation framework. For world model training and heuristic learning, we use a setup similar to DeepCubeAI, with the same underlying logic and implementation as its publicly available code-base (Soltani, 2025). We compare to a model with a continuous latent space as well as a continuous model with no latent space that directly predicts the next state.

5.1 Offline Datasets

We evaluate our approach on the $3 \times 3 \times 3$ Rubik’s Cube and Sokoban. A Rubik’s Cube state is represented as two 32×32 RGB images, each showing three faces of the cube, and a Sokoban state is represented as a single 40×40 RGB image. For training the world model and heuristic function, we generate an offline dataset of 10,000 episodes using action sequences of 30 steps with a random uniform policy. Then we use 90% of the dataset for training and 10% for validation. Start states are sampled by applying 100–200 inverse moves from the canonical solved configuration, ensuring diverse but reachable states. Moreover, we generate a dataset of 100 episodes of 10,000 steps each for comparing rollout stability between discrete and continuous models. For training the alignment model, we generate a dataset of 10,000 episodes of 30 steps each for every visual variant of the environment by applying random transformations to states in the original dataset, and we pair each transformed state with its corresponding clean state.

5.2 Model Architectures

Encoder, decoder, transition model, and DQN heuristic function follow DeepCubeAI’s architecture. Details of these models are given in Supplementary Materials 8. For the alignment model, we use a convolutional residual architecture for both the Rubik’s Cube and Sokoban. For the Rubik’s Cube, the two views are concatenated along the channel dimension, giving an input with 6 channels. The input tensor first passes through a batch normalization layer, followed by two convolutional layers. The first layer maps $6 \rightarrow 64$ channels with a 3×3 kernel, stride 1, and padding 1. The second layer maps $64 \rightarrow 38$ channels with a 5×5 kernel, stride 2, and padding 2. Both layers use batch normalization and ReLU non-linearities. The convolutional stem is followed by 12 residual blocks of 38 channels, each with batch normalization and ReLU activations. The output is then flattened and passed through a two-layer MLP that maps $9728 \rightarrow 10000$ and then to a vector matching the size of the discrete latent space, with batch normalization and ReLU after the first layer and a sigmoid activation at the final layer. For Sokoban, the input has 3 channels and the alignment model is fully convolutional. Two convolutional layers with 2×2 kernels and stride 2 map $3 \rightarrow 64 \rightarrow 64$ channels, each with batch normalization and ReLU. These are followed by 8 residual blocks of 64 channels with batch normalization and ReLU. A final 1×1 convolution with a sigmoid activation maps the 64-channel feature map to the 16-channel discrete latent feature map, which is flattened to match the discrete latent space.

5.3 Training

We train E , T , and D jointly, minimizing the combined reconstruction and discrete world model objectives in Equation 3 with a scheduler that increases the weight, ω , from 1×10^{-3} to 0.5. We use the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 10^{-3} , weight decay of 10^{-5} , and a batch size of 100. For the model with the continuous latent state, we do not round the outputs of the encoder and the transition model. We train a goal-conditioned DQN over discrete latent states using Q-learning. States are generated by encoding states used to train the world model and then taking a random walk in latent space using T . Goal states are generated by taking another random walk from the resulting state of the first walk. For each training instance, the length of the random walk is uniformly sampled between 0 and 30. All transition costs are 1. Similar to prior work, we

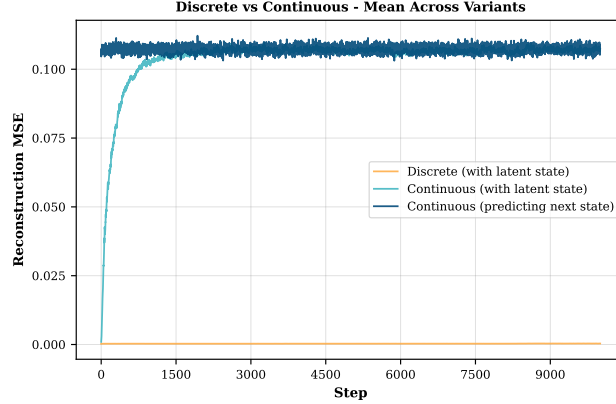


Figure 4: MSE for discrete vs. continuous alignment models across Rubik’s Cube noise variants over 10,000 steps. The discrete world model has lower reconstruction error and does not accumulate error over time.

maintain a target network (Mnih et al., 2015). The target network is updated when the performance of the greedy policy of the DQN being trained improves. Finally, we train an alignment network A to map a noisy observation $\bar{s}$ to the discrete latent state of the corresponding clean observation, $r(E(s))$. We use the same optimizer as the discrete world model training and the loss defined in Equation 4.

5.4 Evaluation

Model Performance. To determine the performance of the alignment model, we use the alignment model to map noisy states to clean latent states and then use the world model to predict next states. We do this for 100 sequences of 10,000 steps each, with actions selected uniformly at random. We compare against a model with a continuous latent space and a model that directly predicts the RGB values for the next state. The continuous and discrete alignment models are trained on the same paired data of noisy observations and clean latent targets, with the only difference being whether rounding is applied to the outputs of the encoder and the transition model. For the continuous model, the alignment network outputs continuous real values, which are passed to a transition model that also operates without rounding. In both cases, the rollout is started from a noisy observation, and the output of the alignment model is used to initiate the latent rollout. We compare the predicted next states with the ground truth next states to determine model accuracy. For the models with latent states, we use the decoder at each step to obtain the predicted next state. Results show that, while both continuous and discrete world models perform well on Sokoban, the discrete world model performs significantly better on the Rubik’s Cube. Figure 4 shows the mean squared error averaged across the visual transformations for the Rubik’s Cube. The continuous models perform relatively poorly, while the discrete model remains accurate and does not accumulate error. Figure 5 shows the decoded state from the output of the alignment model right after noise is applied and then after taking 1,000 steps using the world models with continuous and discrete latent states. Visual degradation can be seen for the continuous model over time, but not for the discrete world model. The complete reconstruction MSE plots for the Rubik’s Cube and Sokoban are available in Supplementary Materials 10.

Planning performance. We now seek to verify that SPAR can be used to solve pathfinding problems when states are given as noisy images. The alignment model maps noisy start and goal states to discrete latents. Then, Q^* search is performed in the discrete latent space, with the learned world model predicting next states and the learned DQN serving as the heuristic function. We use exact bitwise comparisons in the latent space to do goal tests and to reidentify states that have already been

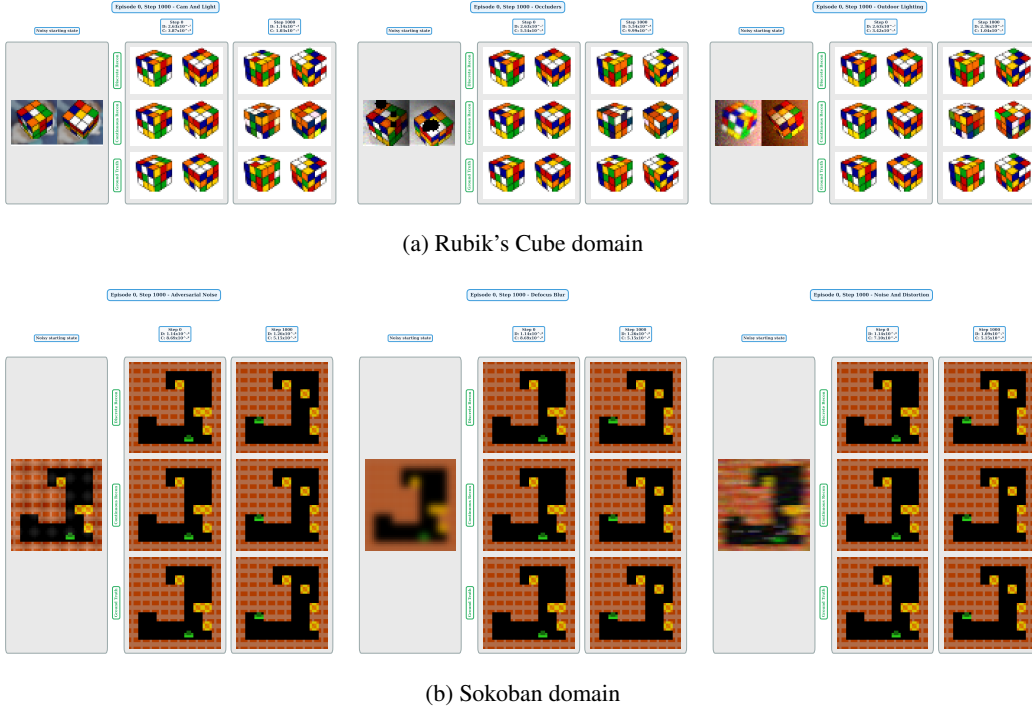


Figure 5: Reconstruction of the alignment model’s outputs starting from a noisy observation.

seen during search. We compare SPAR to DeepCubeAI, which does not use an alignment model, on both clean and noisy states. We note that DeepCubeAI was not designed for noisy observations. It is included to illustrate the brittleness of a state-of-the-art latent-space planner in the absence of alignment, not as a baseline tuned for noisy observations. We also compare the results to using a greedy policy after the alignment, which uses the learned DQN to select the action with the lowest estimated cost-to-go at each step for 100 iterations, instead of Q^* search.

We report the percentage of problems solved and the cost of solutions found in Table 1, and the results for each visual transformation in Supplementary Materials 12. Each visual variant is evaluated on 1,000 test instances, and every reported solve rate is accompanied by a 95% Wilson score confidence interval (Wilson, 1927). Results show that, while SPAR and DeepCubeAI perform similarly on clean states, only SPAR is able to solve any of the noisy states. While SPAR solves the majority of noisy states, there are failure cases, as shown in Figure 6 and Supplementary Materials 13.

Training the world model on noisy data. A natural alternative to SPAR is to train the world model directly on noisy observations. This alternative, however, conflicts with the algorithmic requirements of planning in a discrete latent space. First, heuristic search requires exact state re-identification. The planner must recognize when a latent state has already been visited and when the goal has been reached, and both operations rely on exact bitwise equality in the discrete latent space. When the world model is trained on noisy observations, the same underlying state produces different latent encodings under different noise, which breaks these operations. Second, because the noise at timestep t differs from the noise at timestep $t + 1$, transitions in a deterministic MDP appear stochastic during training, which hinders learning reliable transition dynamics. Third, whenever a new kind of noise appears, the world model, along with the heuristic function trained with data generated from it, must be re-trained, which SPAR avoids by design. To quantify these effects, we trained a discrete world model on the same noisy dataset, with the same architecture and hyperparameters, used to train SPAR’s alignment model. This model predicted the discrete latent representation of the next state with roughly 50% accuracy on held-out data. In contrast, the encoder trained on clean obser-

Table 1: Comparison of SPAR (ours) to DeepCubeAI and to using SPAR with a greedy policy instead of Q^* search when solving clean and noisy states. Metrics are length of paths, number of nodes generated, number of seconds to find a solution, nodes per second, and percentage of problems solved with 95% Wilson score confidence intervals. Each visual variant is evaluated on 1,000 test instances. The clean settings use 1,000 instances per domain, the noisy settings aggregate all visual variants (16,000 instances for the Rubik’s Cube and 18,000 for Sokoban), and the real-world setting uses 10 instances.

DOMAIN	OBSERVATION	SOLVER	LEN	NODES	SECS	NODES/SEC	SOLVED	95% CI
Rubik’s Cube	Clean	SPAR	22.85	2.00E+05	6.21	3.22E+04	100%	[99.6, 100.0]
		DeepCubeAI	22.85	2.00E+05	6.21	3.22E+04	100%	[99.6, 100.0]
		Greedy Policy	—	—	—	—	0%	[0.0, 0.4]
	Noisy	SPAR	22.85	2.00E+05	6.21	3.22E+04	89.92%	[89.4, 90.4]
		DeepCubeAI	—	—	—	—	0%	[0.00, 0.02]
		Greedy Policy	—	—	—	—	0%	[0.00, 0.02]
	Real-World	SPAR	23.61	1.90E+05	6.21	3.06E+04	50%	[23.7, 76.3]
		DeepCubeAI	—	—	—	—	0%	[0.0, 27.8]
		Greedy Policy	—	—	—	—	0%	[0.0, 27.8]
Sokoban	Clean	SPAR	32.32	2.39E+04	20.58	1.16E+03	100%	[99.6, 100.0]
		DeepCubeAI	32.32	2.39E+04	20.58	1.16E+03	100%	[99.6, 100.0]
		Greedy Policy	—	—	—	—	41.9%	[38.9, 45.0]
	Noisy	SPAR	32.30	2.39E+04	20.55	1.16E+03	96.31%	[96.0, 96.6]
		DeepCubeAI	—	—	—	—	0%	[0.00, 0.02]
		Greedy Policy	—	—	—	—	0%	[0.00, 0.02]

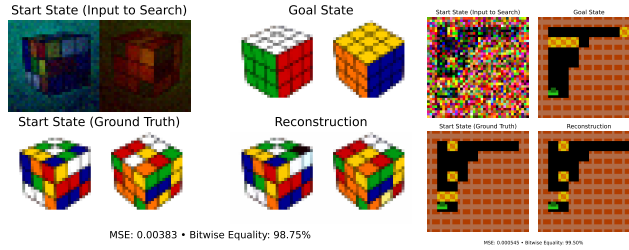


Figure 6: Failure cases for using SPAR to find solution paths.

variations achieves 100% accuracy with a simpler neural network, and the alignment model, trained to map noisy observations to the discrete latent states produced by this encoder, reaches about 90% accuracy on the held-out noisy dataset.

Generalization to Real Images. To assess the generalization capabilities of SPAR, we evaluate SPAR on photos of a Rubik’s Cube taken in the real world in different lighting conditions and backgrounds. We note that no real-world images were seen during training. We take two photos of the Rubik’s Cube with a phone, each one containing three faces of the cube, in a similar orientation to the offline data. These two photos are then resized to 32×32 and concatenated. The prediction of the alignment model is then passed to the pretrained decoder to get the reconstruction of the discrete encoding. For a qualitative analysis, the original images, the resized images, and the reconstruction obtained after applying SPAR are shown in Figure 7. Despite never seeing real-world images of the Rubik’s Cube, SPAR successfully reconstructs the cube in several examples, while failing to reconstruct some stickers in others. For a quantitative analysis, as we did for the other noisy images, we use SPAR to find a solution path based on the real-world images and, if one is found, we validate it on the real-world cube. This evaluation uses 10 real-world instances, so the corresponding confidence interval in Table 1 is wide. Table 1 shows that SPAR is able to find a solution path for half of the real-world examples.

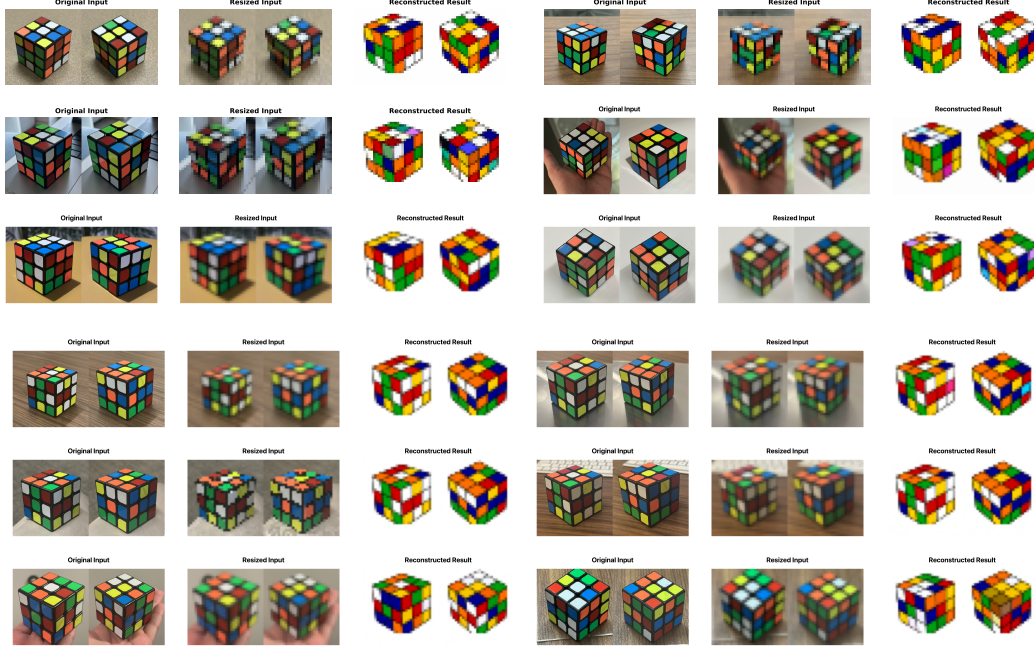


Figure 7: Applying SPAR to a real-world Rubik’s Cube image with differing lighting and background.

6 Discussion and Future Work

Failure cases and robustness to increasing noise. Figure 6 and Supplementary Materials 13 show that, in many failure cases, information relevant to the clean state was occluded by noise. While a lack of information may make it impossible to find a path with SPAR, future work could use re-planning when the observations expected under the plan that SPAR executes diverge from the actual observations. This could be integrated by treating particularly noisy examples as partially observable MDPs (POMDPs), where the belief state is updated as actions are taken, or by modeling sequences of observations to recover the ground-truth state. A structured stress test that gradually increases the severity of each transformation, characterizing the noise threshold below which exact latent reconstruction is possible, is another direction for future work. For failure cases where enough information is present, but the examples are out of distribution and generating similar training data is difficult, as is the case for the real-world photos of the Rubik’s Cube, large vision models could be used to generate such out-of-distribution data.

Large vision models could also be used to relate salient features in the images to the discrete latent space. As a result, bits in the discrete latent space could be used to identify the presence or absence of certain objects. Building on this information, formal logic languages capable of describing complex goals, such as answer set programming (Brewka et al., 2011), could be used to specify goals (Agostinelli et al., 2024a) instead of a goal image.

Continuous latent performance on Sokoban. While the continuous alignment models degrade quickly on the Rubik’s Cube, they perform comparatively well on Sokoban, where the difference in reconstruction MSE between the continuous and discrete models is on the order of 10^{-3} (Supplementary Materials 10). A possible explanation is that boxes in Sokoban quickly get pushed up against walls and become immovable. After this occurs, only the location of the agent changes between transitions, making small real-valued errors less likely.

Assumption of clean training data. SPAR assumes access to pairs of noisy and clean observations of the same underlying state, which we obtain by adding noise to the clean states used to train the world model. This is a strong assumption, and obtaining such pairs may be difficult for real-world tasks that are hard to simulate. However, SPAR does not require a perfectly clean image for every noisy frame. Approximate base views can come from simulator renderings or from complementary acquisition setups. For example, pairs of blurry long-exposure and noisy short-exposure images can provide practically feasible supervision (Zhang et al., 2022), and paired noisy observations alone can supervise restoration under standard noise assumptions (Lehtinen et al., 2018).

Beyond discrete domains and planning. Our experiments focus on inherently discrete tasks, but discrete latent world models learned from high-dimensional continuous observations have proven effective for continuous control from pixels (Hafner et al., 2019; 2021; Okada & Taniguchi, 2022). From the perspective of state abstraction theory, a discrete latent world model induces an abstract MDP whose states correspond to equivalence classes of behaviorally similar continuous states, and planning in such an abstraction can preserve near-optimal behavior (Li et al., 2006). Extending SPAR to continuous domains would then amount to learning a discrete latent world model from continuous observations and training an alignment model to map corrupted observations into the induced latent space. Moreover, the impact of alignment is not limited to planning. A policy trained with reinforcement learning on clean discrete latent states could be deployed zero-shot in noisy environments by passing observations through the alignment network, without any fine-tuning of the policy itself.

Finally, SPAR could be applied to real-world sequential decision-making problems. In robotics, an agent often encounters varying lighting, backgrounds, etc. SPAR could provide a fast way to recalibrate the robot’s perception without retraining its policy or dynamics model each time the environment changes (Longhini et al., 2024; Nutalapati et al., 2022). Fast adaptation of perception without full retraining aligns with test-time adaptation results in robotics and perception (Park et al., 2024; Piriyajitakonkij et al., 2024). Coupling SPAR with classic robot planning algorithms or continuous control techniques may be necessary to handle continuous action spaces, but the concept of a discrete latent adapter remains useful.

7 Conclusion

Discrete world models, which offer the promise of long-horizon planning with no model degradation, must be able to plan in the presence of noise. SPAR trains a discrete world model and heuristic function on clean states and then trains an alignment model to map noisy states to their corresponding clean discrete latent state while keeping the learned world model and heuristic function fixed. This approach completely sidesteps the complexities of training a world model on noisy data and the prospect of having to re-train the world model and heuristic function when presented with new out-of-distribution noise. Results show that SPAR solves all clean test instances and the vast majority of noisy test instances while, without SPAR, no noisy instances are solved. Furthermore, results show that SPAR can generalize to noise not seen during training. In particular, for real-world pictures taken from a phone, SPAR was able to solve several problems despite never seeing a real-world image of the Rubik’s Cube during training.

References

- Forest Agostinelli and Misagh Soltani. Learning discrete world models for heuristic search. *Reinforcement Learning Journal*, 4:1781–1792, 2024.
- Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the rubik’s cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363, 2019.

- Forest Agostinelli, Rojina Panta, and Vedant Khandelwal. Specifying goals to deep neural networks with answer set programming. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pp. 2–10, 2024a.
- Forest Agostinelli, Shahaf S Shperberg, Alexander Shmakov, Stephen McAleer, Roy Fox, and Pierre Baldi. Q* search: Heuristic search with deep q-networks. In *ICAPS Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning workshop*, 2024b.
- Masataro Asai and Alex Fukunaga. Classical planning in deep latent space. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 6094–6101, 2018.
- Masataro Asai, Hiroki Yamasaki, and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *Journal of Artificial Intelligence Research*, 75: 645–704, 2022.
- Marco Bagatella, Miroslav Olšák, Michal Rolínek, and Georg Martius. Planning from pixels in environments with combinatorially hard search spaces. *Advances in Neural Information Processing Systems*, 34:24707–24718, 2021.
- Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.
- Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
- Longchao Da, Justin Turnau, Thirulogasankar Pranav Kutralingam, Alvaro Velasquez, Paulo Shakarian, and Hua Wei. A survey of sim-to-real methods in rl: Progress, prospects and challenges with foundation models. *arXiv preprint arXiv:2502.13187*, 2025.
- Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019.
- Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.
- Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021.
- Nicklas Hansen, Hao Su, and Xiaolong Wang. Self-supervised policy adaptation during deployment. In *International Conference on Learning Representations*, 2021.
- Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.
- Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017a. URL <https://openreview.net/forum?id=Sy2fzU9gl>.
- Irina Higgins, Arka Pal, Andrei A. Rusu, Loic Matthey, Christopher P. Burgess, et al. Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *PMLR*, pp. 1480–1490, 2017b.

- Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, et al. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12627–12637, 2019.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- Michael Laskin, Kimin Lee, Aravind Srinivas, Emile Mathieu, Zhaoping He, et al. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems*, 2020.
- Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018.
- Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. *AI&M*, 1(2):3, 2006.
- Alberta Longhini, Yufei Wang, Irene Garcia-Camacho, David Blanco-Mulero, Marco Moletta, Michael Welle, Guillem Alenyà, Hang Yin, Zackory Erickson, David Held, et al. Unfolding the literature: A review of robotic cloth manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, 8, 2024.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Belle-mare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- Mohan Krishna Nutalapati, Lavish Arora, Anway Bose, Ketan Rajawat, and Rajesh M Hegde. A generalized framework for autonomous calibration of wheeled mobile robots. *Robotics and Autonomous Systems*, 158:104262, 2022.
- Masashi Okada and Tadahiro Taniguchi. Dreamingv2: Reinforcement learning with discrete world models without reconstruction. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 985–991. IEEE, 2022.
- Hyoungseob Park, Anjali Gupta, and Alex Wong. Test-time adaptation for depth completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20519–20529, 2024.
- Maytus Piriyajitakonkij, Mingfei Sun, Mengmi Zhang, and Wei Pan. Tta-nav: Test-time adaptive reconstruction for point-goal navigation under visual corruptions. *arXiv preprint arXiv:2403.01977*, 2024.
- Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.
- Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
- Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in neural information processing systems*, pp. 5690–5701, 2017.

- Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. In *Robotics: Science and Systems*, 2017.
- Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2020.
- Connor Shorten and Taghi M Khoshgohar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Arthur Guez, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
- Misagh Soltani. DeepcubeAI: Learning discrete world models for heuristic search, 2025. URL <https://github.com/misaghsoltani/DeepCubeAI>. Commit 48a88e23fd3cfb9858552021e02545e1583aa044.
- Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216–224. Morgan Kaufmann, 1990.
- Stephen Tian, Suraj Nair, Frederik Ebert, Sudeep Dasari, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Model-based visual planning with self-supervised functional distances. In *International Conference on Learning Representations*, 2021.
- Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.
- Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.
- Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
- Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.
- Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021.
- Takuma Yoneda, Ge Yang, Vincent Sitzmann, Phillip Isola, Tsung-Yi Lin, Alexei A. Efros, and Lerrel Pinto. Invariance through latent alignment. In *Robotics: Science and Systems*, 2022.
- Zhilu Zhang, RongJian Xu, Ming Liu, Zifei Yan, and Wangmeng Zuo. Self-supervised image restoration with blurry and noisy pairs. *Advances in Neural Information Processing Systems*, 35:29179–29191, 2022.
- Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 13001–13008, 2020.

Supplementary Materials

The following content was not necessarily subject to peer review.

8 Model Architecture

Here, we detail the neural network architectures used for encoder, decoder, transition model, and heuristic model.

For the Rubik’s Cube, we use fully connected networks for the encoder and decoder with one hidden layer, and the encoder outputs a 400-dimensional vector. The decoder uses a fully connected network with a linear (identity) function as the final activation. For the transition model, we use four fully connected layers of size $500 \rightarrow 500 \rightarrow 500 \rightarrow 400$ with batch normalization and ReLU, except for the last layer, which uses a logistic output without batch normalization. Actions are one-hot encoded, concatenated to the input state, and passed to the neural network. For the heuristic function, we use a deep Q-network (DQN) (Mnih et al., 2013) with a fully connected residual network. The input is the concatenation of the discrete latent representations of current state and goal state. A two-layer MLP maps $800 \rightarrow 5000 \rightarrow 1000$ with batch normalization and ReLU after each layer. This is followed by 4 residual blocks of width 1000, each block having batch normalization and ReLU activation. At the final layer, a linear activation outputs Q-values for all actions ($1000 \rightarrow 12$).

For Sokoban, we use a two-layer convolutional encoder with 2×2 kernels, stride 2, and zero paddings, mapping channels $\text{chan_in} \rightarrow 16 \rightarrow 16$. Batch normalization and ReLU are applied after the first layer. The second layer uses a sigmoid activation. The resulting feature map has spatial size 10×10 with 16 channels and is flattened to a 1600-dimensional vector ($10 \times 10 \times 16$). For the transition model, actions are one-hot encoded as channel planes and concatenated with the current latent feature maps along the channel dimension. The transition model is a 3-layer 3×3 convolutional network with stride 1 and padding 1 ($\text{number of actions} + 16$) $\rightarrow 32 \rightarrow 32 \rightarrow 16$. The first two layers use batch normalization and ReLU, and the final layer omits batch normalization and uses a sigmoid output. The decoder mirrors the encoder with two transposed 2×2 convolutions of stride 2, mapping $16 \rightarrow 16 \rightarrow 16$ channels (batch norm + ReLU on the first, sigmoid on the second) to upsample back to the original spatial resolution. A final 1×1 convolution maps $16 \rightarrow \text{chan_in}$ with a linear (identity) activation to reconstruct the observation. Similar to Rubik’s Cube, the DQN input is the concatenation of the encoded current state and goal state, giving a $2 \times 1600 = 3200$ -dimensional vector. A two-layer MLP maps $3200 \rightarrow 5000 \rightarrow 1000$ with batch normalization and ReLU after each layer. This is followed by 4 fully connected residual blocks of width 1000, each using batch normalization and ReLU. A final linear head outputs Q-values for all actions ($1000 \rightarrow \text{number of actions}$).

9 Additional Rollouts and Visualizations

In this appendix, we provide supplementary rollout visualizations for the discrete and continuous alignment models for the Rubik’s Cube and Sokoban domains. Each figure below shows (a) the reconstructions of the discrete latent representation (first row), (b) the reconstructions of the continuous latent representation (second row), and (c) the ground-truth clean observations (third row). All images show reconstructed frames from the alignment model’s latent predictions, with the inputs given in the left panels. The output of the alignment model is given to the decoder to visualize the reconstruction.

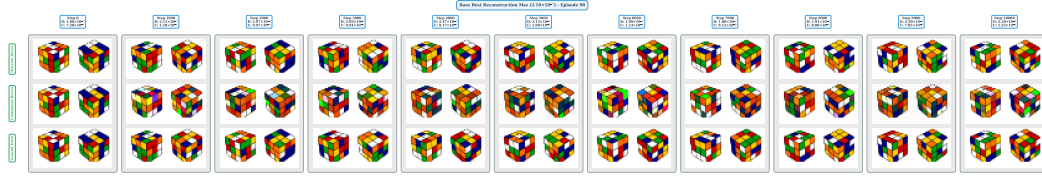


Figure 8: Reconstructs of rolling out the discrete and continuous alignment models for the base (clean) setup for Rubik's Cube.

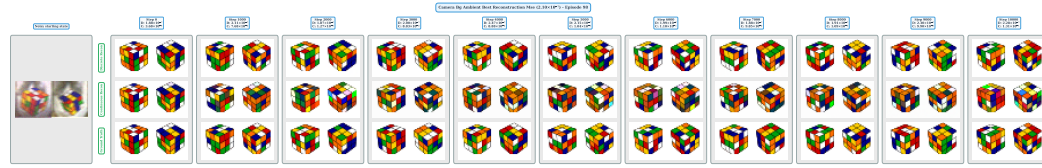


Figure 9: Reconstructs of rolling out the discrete and continuous alignment models for the Ambient Background + Camera perturbation for Rubik's Cube.

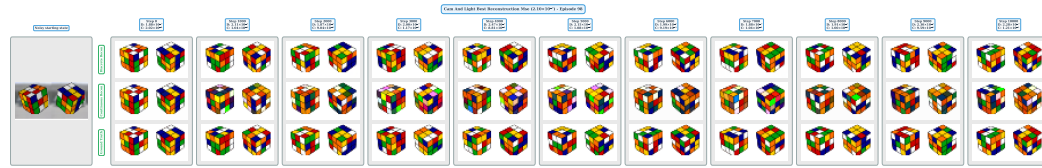


Figure 10: Reconstructs of rolling out the discrete and continuous alignment models for the Camera + Lighting perturbation for Rubik's Cube.

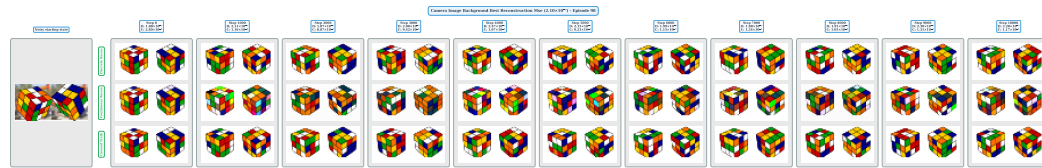


Figure 11: Reconstructs of rolling out the discrete and continuous alignment models for the Image Background perturbation for Rubik's Cube.

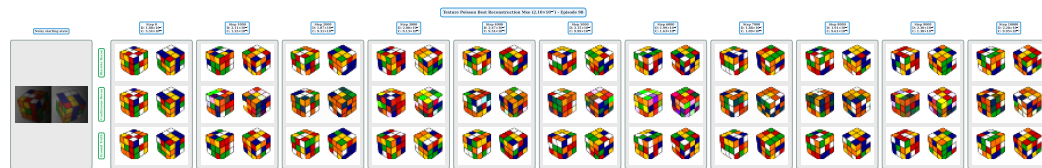


Figure 12: Reconstructs of rolling out the discrete and continuous alignment models for the Textured Background + Poisson Noise perturbation for Rubik's Cube.

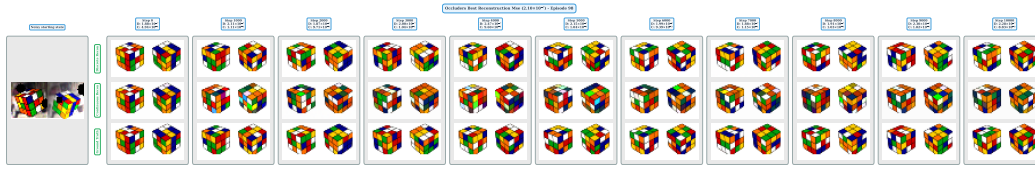


Figure 13: Reconstructions of rolling out the discrete and continuous alignment models for the Occlusions perturbation for Rubik's Cube.

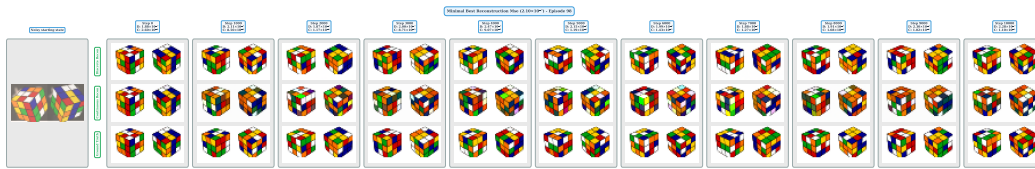


Figure 14: Reconstructions of rolling out the discrete and continuous alignment models for the Minimal Augmentations perturbation for Rubik's Cube.

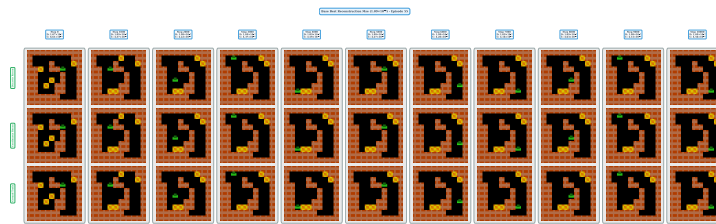


Figure 15: Reconstructions of rolling out the discrete and continuous alignment models for the base (clean) setup for Sokoban.

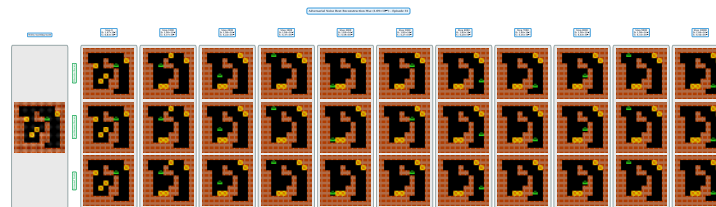


Figure 16: Reconstructions of rolling out the discrete and continuous alignment models for the Adversarial Noise perturbation for Sokoban.

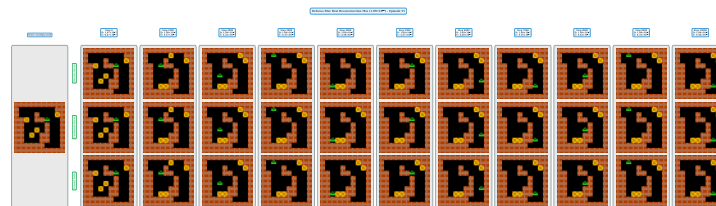


Figure 17: Reconstructions of rolling out the discrete and continuous alignment models for the Defocus Blur perturbation for Sokoban.

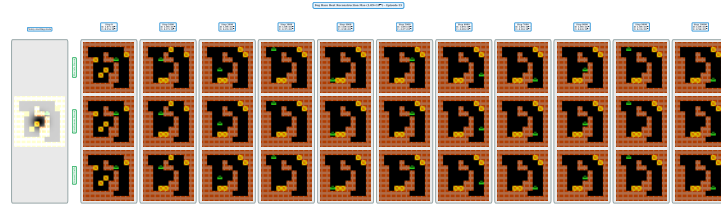


Figure 18: Reconstructions of rolling out the discrete and continuous alignment models for the Fog Haze perturbation for Sokoban.

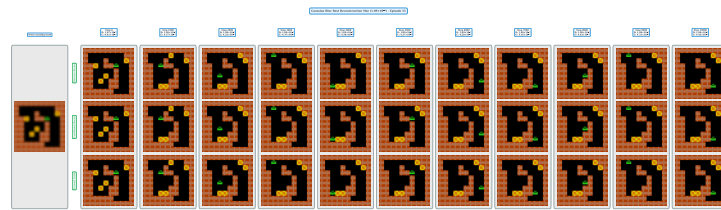


Figure 19: Reconstructions of rolling out the discrete and continuous alignment models for the Gaussian Blur perturbation for Sokoban.

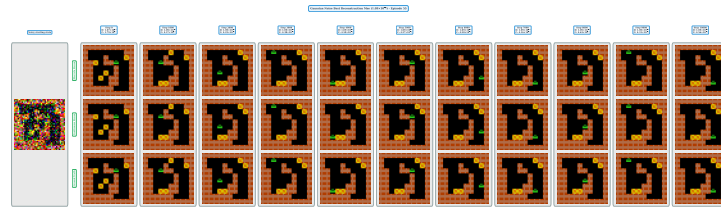


Figure 20: Reconstructions of rolling out the discrete and continuous alignment models for the Gaussian Noise perturbation for Sokoban.

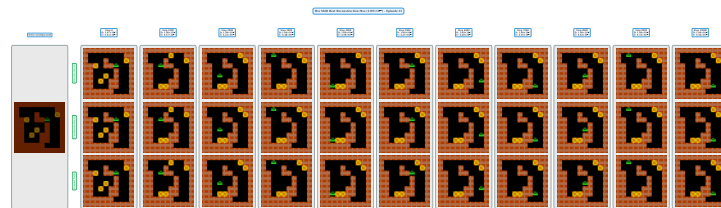


Figure 21: Reconstructions of rolling out the discrete and continuous alignment models for the HSV Shift perturbation for Sokoban.

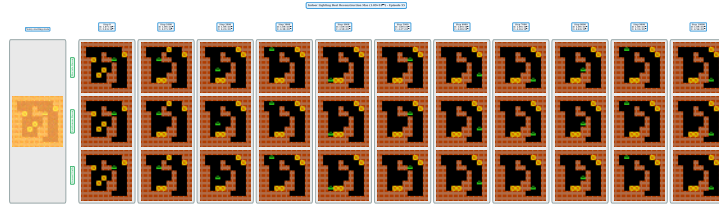


Figure 22: Reconstructions of rolling out the discrete and continuous alignment models for the Indoor Lighting perturbation for Sokoban.

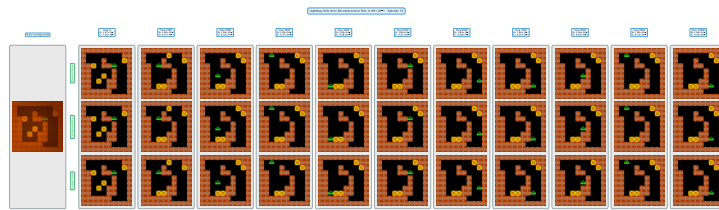


Figure 23: Reconstructions of rolling out the discrete and continuous alignment models for the Lighting Only perturbation for Sokoban.

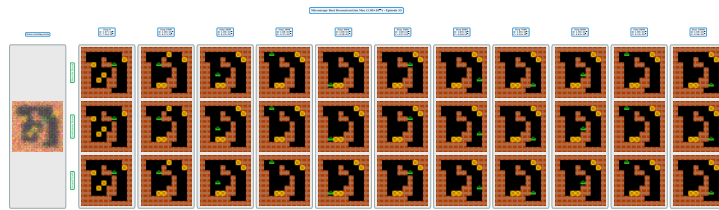


Figure 24: Reconstructions of rolling out the discrete and continuous alignment models for the Microscopy perturbation for Sokoban.

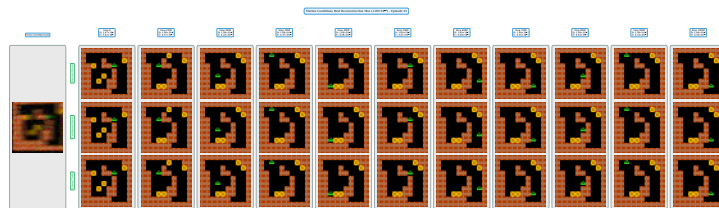


Figure 25: Reconstructions of rolling out the discrete and continuous alignment models for the Motion Conditions perturbation for Sokoban.

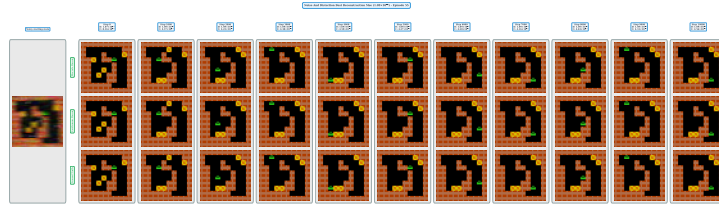


Figure 26: Reconstructions of rolling out the discrete and continuous alignment models for the Noise and Distortion perturbation for Sokoban.

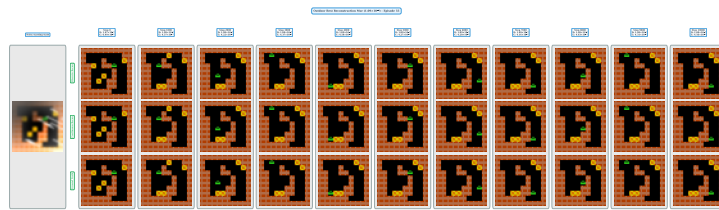


Figure 27: Reconstructions of rolling out the discrete and continuous alignment models for the Outdoor perturbation for Sokoban.

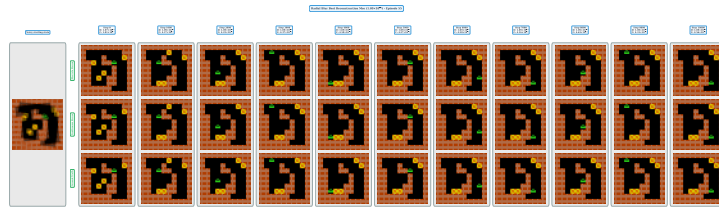


Figure 28: Reconstructions of rolling out the discrete and continuous alignment models for the Radial Blur perturbation for Sokoban.

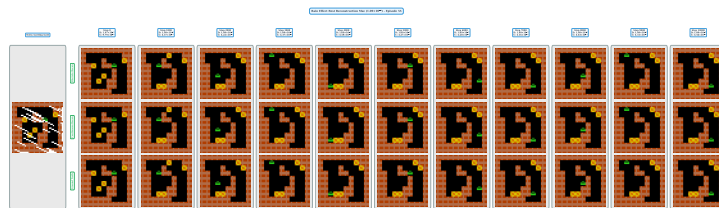


Figure 29: Reconstructions of rolling out the discrete and continuous alignment models for the Rain Effect perturbation for Sokoban.

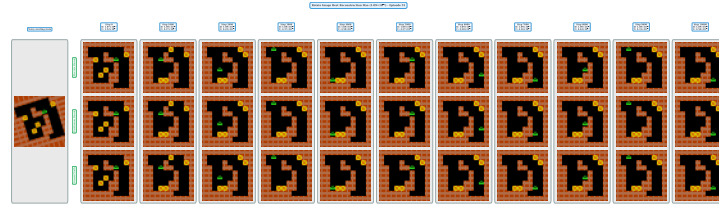


Figure 30: Reconstructions of rolling out the discrete and continuous alignment models for the Rotate Image perturbation for Sokoban.

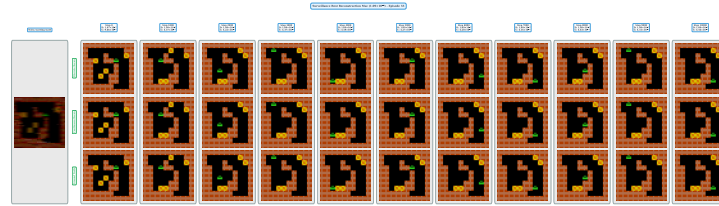


Figure 31: Reconstructions of rolling out the discrete and continuous alignment models for the Surveillance perturbation for Sokoban.

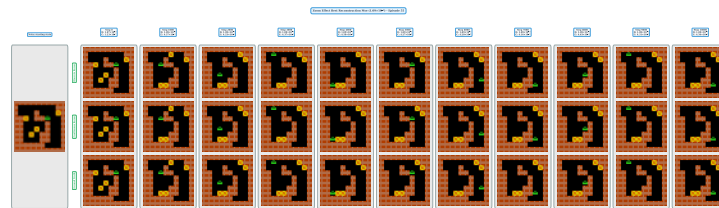


Figure 32: Reconstructions of rolling out the discrete and continuous alignment models for the Zoom Effect perturbation for Sokoban.

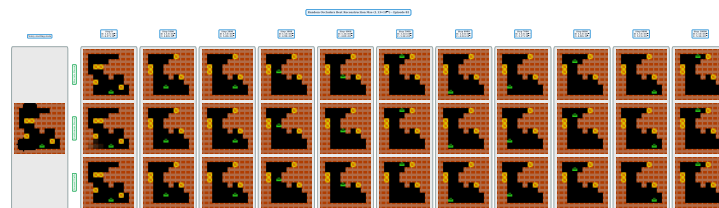


Figure 33: Reconstructions of rolling out the discrete and continuous alignment models for the Random Occluders perturbation for Sokoban.

10 Model Comparison Plots

We compile every reconstruction MSE model-comparison plot generated for the Rubik’s Cube and Sokoban benchmarks. Each subfigure contrasts the discrete world model, its continuous counterpart, and the aligned variants under a specific appearance change or aggregation. Figure 34 shows the Rubik’s Cube plots and Figure 35 shows the Sokoban plots.

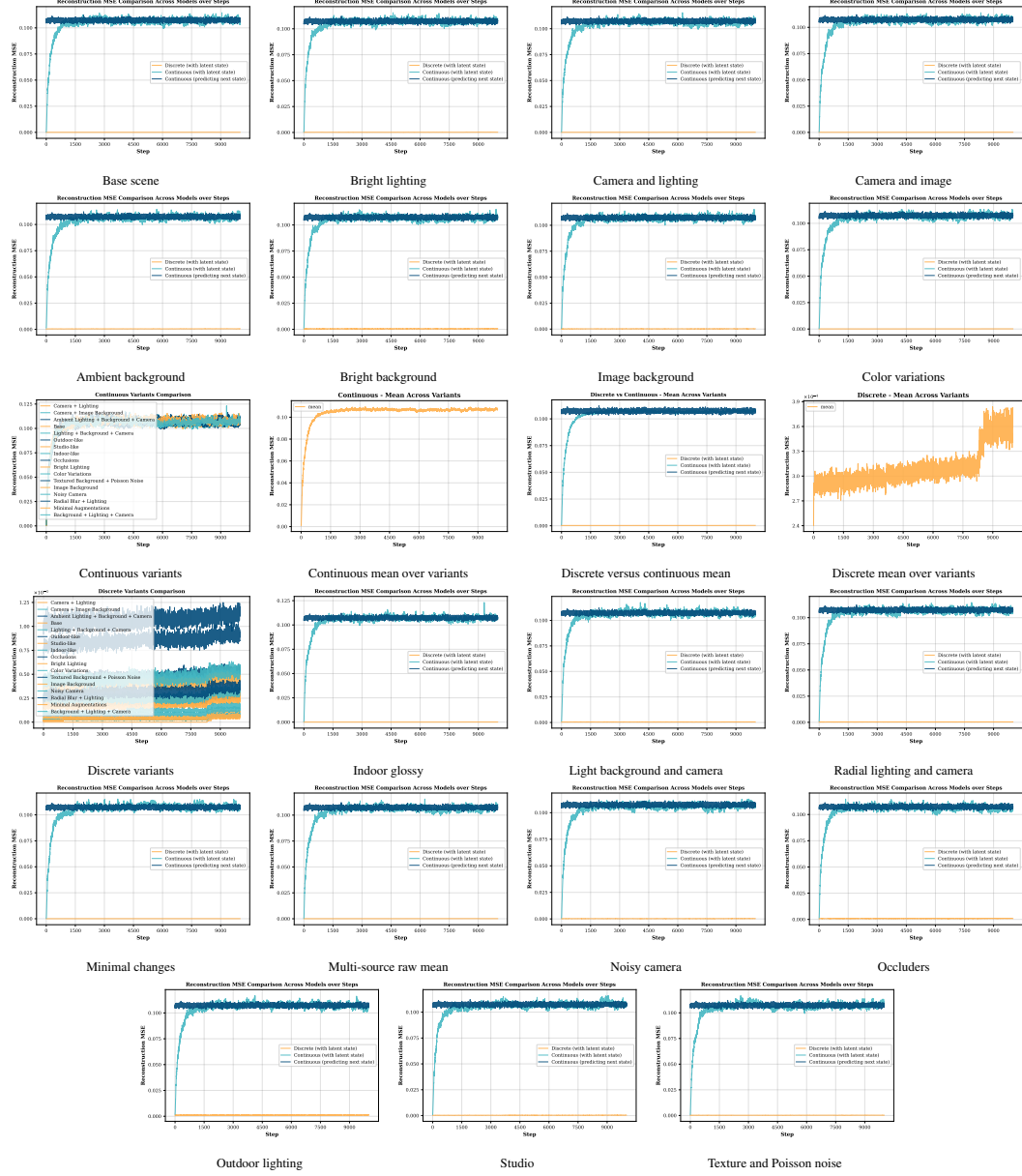


Figure 34: Rubik's Cube reconstruction MSE comparisons across lighting, camera, background, texture, and occlusion variations, showing discrete, continuous, and aligned model curves where lower values reflect better reconstruction fidelity.

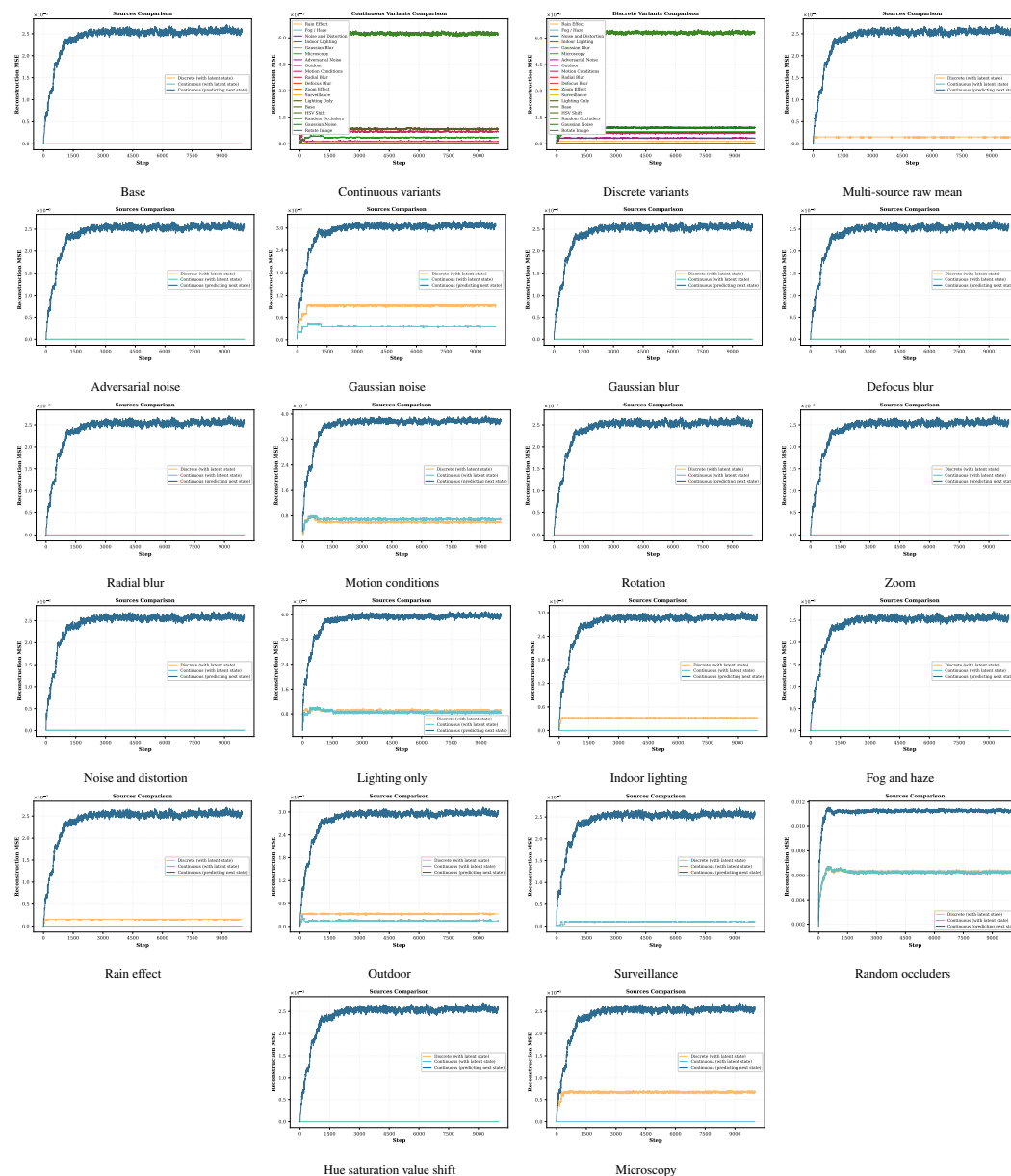


Figure 35: Sokoban reconstruction MSE comparisons covering lighting, blur, motion, noise, view-point, weather, and occlusion changes, presenting discrete, continuous, and aligned model curves where lower values indicate stronger reconstruction.

11 Environmental Perturbation Augmentations

This appendix formalizes the augmentations used to simulate environmental perturbations in the Rubik’s Cube and Sokoban environments. For each augmentation, we state the governing principle, outline how it is applied, and specify the controlling parameters and their qualitative effects. We denote an image by $I : \Omega \subset \mathbb{R}^2 \rightarrow [0, 1]^3$ with per-channel intensities in $[0, 1]$. Elementwise operations act per pixel and per channel unless noted. The operator $\text{clip}(\cdot)$ truncates values to $[0, 1]$. Convolution is $*$. Homogeneous image coordinates are $\tilde{x} = (x, y, 1)^\top$.

Directional Surface Lighting (Lambertian) Per-surface intensity follows a Lambertian model augmented with ambient and back-face floor terms. Let $\mathbf{n}$ be the surface normal and ℓ a unit light direction. The shading field is $S = \text{clip}(A + (1 - A) \max(0, \mathbf{n} \cdot \ell) + s)^\gamma$, and the observed color is $\text{clip}(S \odot \mathbf{c})$, where A is an ambient term, s is a subsurface/back-face term, $\gamma > 0$ is a contrast control, and $\mathbf{c}$ is the base reflectance. Azimuth and elevation set ℓ . This augmentation represents directional illumination, self-shadowing, and highlight contrast on faceted objects.

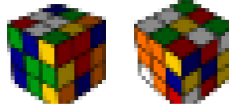


Figure 36: $A = 0.2, \kappa = 0.6, \gamma = 1.5, s = 0.1, \text{azdeg} = 315, \text{altdeg} = 45$

Random Background Chromaticity To diversify global backgrounds across frames, the constant color is drawn from a distribution, typically $\mathbf{c} \sim \mathcal{U}([0, 1]^3)$. The rendering pipeline applies the sampled $\mathbf{c}$ before foreground compositing, inducing broad variation in global tone and contrast. Parameters are the sampling law and any constraints on hue or saturation. This models uncontrolled ambient lighting and incidental background coloration across capture sessions.



Figure 37: $\mathbf{c} \sim \mathcal{U}([0, 1]^3)$

Procedural Background Textures A parametric texture $B(\mathbf{x})$ (e.g., grids, dots, stripes, noise, crosshatch, sinusoids) is synthesized in image coordinates and alpha-composited behind the object, $I' = (1 - \alpha)I + \alpha B$. The construction of B governs spatial frequency, orientation, regularity, and contrast. Parameters include the texture family, spatial density, tint, and opacity $\alpha \in [0, 1]$. This augmentation reproduces structured backgrounds such as patterned walls or tabletops that introduce periodic distractors and clutter.

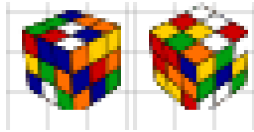


Figure 38: $B(\mathbf{x}) = \text{grid}, \text{color} = \text{gray}, \text{density} = 1, \alpha = 0.3$

Hue-Saturation-Value Adjustment Chromatic attributes are modified by mapping RGB to HSV, applying a hue rotation and multiplicative scaling of saturation and value, then mapping back. Writ-

ing $(h, s, v) \mapsto (h', s', v')$ with $h' = (h + \Delta_h) \bmod 1$, $s' = \text{clip}(\lambda_s s)$, and $v' = \text{clip}(\lambda_v v)$ alters perceived colorfulness and brightness while preserving relative luminance structure. The parameters are $\Delta_h, \lambda_s \geq 0$, and $\lambda_v \geq 0$. This simulates camera white-balance shifts, colored illumination, and post-processing tints.



Figure 39: $\Delta_h = 0.12, \lambda_s = 1.25, \lambda_v = 1.12$

Contrast and Brightness A global affine photometric transform, $I' = \text{clip}(\alpha I + \beta)$, adjusts contrast via the gain $\alpha > 0$ and brightness via the offset $\beta \in \mathbb{R}$. The method is applied uniformly per pixel and channel, followed by clipping. This models exposure settings and tone-curve changes typical of automatic image pipelines.



Figure 40: $\alpha = 1.25, \beta = 0.1$

Exposure via Gamma Nonlinear tone mapping with gamma correction redistributes mid-tones while preserving black/white after clipping: $I' = \text{clip}(I^{1/\gamma})$. The parameter $\gamma > 0$ brightens ($\gamma < 1$) or darkens ($\gamma > 1$) mid-range intensities. The augmentation captures low-light and short exposure regimes.



Figure 41: $\gamma = 1.5$

Color Temperature Shift A chromatic bias along the red-blue axis emulates warm/cool illumination. In RGB, a simple model biases red and blue in opposite directions, e.g., $I'_R = \text{clip}(I_R + \delta)$, $I'_B = \text{clip}(I_B - \delta)$, leaving green largely unchanged. The scalar δ controls the shift magnitude. This represents tungsten (warm) versus daylight (cool) lighting and mixed color temperatures.



Figure 42: $\delta = 0.12$

Ambient Color Overlay Uniform ambient illumination is modeled by alpha-blending with a constant color: $I' = (1 - \alpha)I + \alpha c$. The overlay color c and opacity $\alpha \in [0, 1]$ define the effective

fill light. The result approximates global tints arising from wall reflections, skylight, or translucent enclosures.



Figure 43: $\alpha = 0.2, c = \#ffd1a4$

Directional Light with Optional Vignetting After gamma-based exposure adjustment, a directional lighting field modulates intensity according to a cosine-like pattern aligned with a unit vector $\mathbf{d}$ at angle θ . With normalized coordinates $\hat{\mathbf{x}} \in [-1, 1]^2$, the mask $m(\mathbf{x}) = \text{clip}(\frac{1}{2}\hat{\mathbf{x}} \cdot \mathbf{d} + \frac{1}{2})$ yields $I' = \text{clip}\{I^{1/\gamma} \odot (A + \kappa m)\}$. A radial vignette $v(r) = 1 - \eta r$ optionally attenuates corners. Parameters are $\gamma > 0$, direction θ , directional strength $\kappa \geq 0$, ambient $A \in [0, 1]$, and vignette strength $\eta \in [0, 1]$. This emulates off-axis window light, desk lamps, and lens vignetting.

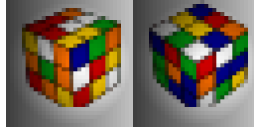


Figure 44: $\gamma = 1, \theta = 0, \kappa = 1, A = 0.5, \eta = 0.5$

In-Plane Rotation A rigid planar transform rotates the image about its center, $\mathbf{x}' = \mathbf{R}_\theta(\mathbf{x} - \mathbf{c}) + \mathbf{c}$, with resampling at inverse-warped coordinates to preserve grid spacing. The angle θ controls orientation. The augmentation models camera roll, tripod tilt, and handheld orientation drift.

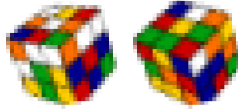


Figure 45: $\theta = 15$

Isotropic Zoom Uniform scaling by a factor f about the image center modifies the field of view. $f > 1$ crops the center region and upsamples, while $f < 1$ downsamples and pads to the original size. The single parameter $f > 0$ governs magnification. This simulates focal changes, reframing, and subject distance variation.

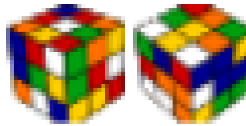
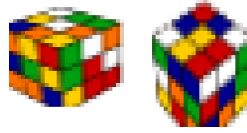


Figure 46: $f=1.2$

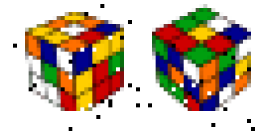
Edge Cropping with Rescaling A fraction κ of one edge is removed and the remaining content is resized back to the original dimensions, conserving aspect ratio. Parameters are the crop fraction $\kappa \in (0, 0.5)$ and the selected edge. The effect models partial occlusions, misframing, and sensor readout loss.

Figure 47: $\kappa = 0.2$

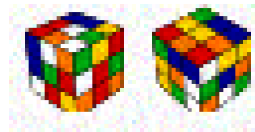
Additive Gaussian Noise Zero-mean i.i.d. noise is added per pixel, $I' = \text{clip}(I + \varepsilon)$, $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. The standard deviation $\sigma > 0$ controls grain strength. This simulates sensor read noise and compression residue at moderate bitrates.

Figure 48: $\sigma = 0.05$

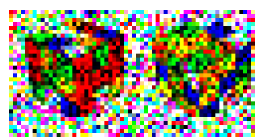
Salt-and-Pepper Noise Impulsive corruption sends a proportion p of pixels to intensity extremes, with salt ratio ρ governing the fraction set to white versus black. Parameters are the amount $p \in (0, 1)$ and salt ratio $\rho \in [0, 1]$. This reproduces stuck pixels, transmission dropouts, and packet loss artifacts.

Figure 49: $p = 0.01, \rho = 0.5$

Poisson (Shot) Noise Photon-counting statistics are modeled by $I' = \frac{1}{K} \text{Poisson}(KI)$, where K is an exposure scale. As K decreases, variance grows relative to mean, especially in dark regions. This augmentation captures low-light and short exposure regimes.

Figure 50: $K=256$

Multiplicative Speckle Noise Granular fluctuations modulate intensity multiplicatively: $I' = \text{clip}(I \odot (1 + n))$ with $n \sim \mathcal{N}(0, \sigma^2)$. The variance σ^2 shapes speckle contrast. The effect characterizes coherent imaging artifacts (e.g., ultrasound, radar) and rough surface scatter.

Figure 51: $n \sim \mathcal{N}(0, 1)$

Color Quantization (Banding) Uniform scalar quantization reduces the number of displayable intensities to L levels, $Q_L(I) = \frac{\lfloor LI \rfloor}{L}$. The integer $L \in [2, 32]$ sets the severity. This simulates reduced bit-depth, posterization, and banding from aggressive compression or low-quality displays.



Figure 52: $L=8$

Gaussian Defocus Blur Convolution with an isotropic Gaussian kernel G_σ yields $I' = G_\sigma * I$, approximating defocus from finite aperture. Separability provides efficient realization. The standard deviation $\sigma > 0$ determines blur radius. The augmentation models out-of-focus capture and depth-of-field limits.



Figure 53: $\sigma = 1.5$

Horizontal Motion Blur Apparent lateral motion is approximated by 1D convolution along the horizontal axis with a box kernel of width k , effectively averaging shifted replicas: $I' = b_k *_x I$. The kernel size k (odd) increases streak length. This simulates camera pan or fast object motion.

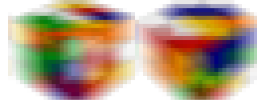


Figure 54: $k=9$

Radial (Spin) Blur Small-angle rotations around the image center are averaged: $I' = \frac{1}{N} \sum_{i=1}^N \mathcal{R}_{\theta_i}(I)$, $\theta_i \in [-\theta_{\max}, \theta_{\max}]$. Parameters are the number of rotations N and maximum angle $\theta_{\max}$. The augmentation represents rotational shake and platform yaw jitter.

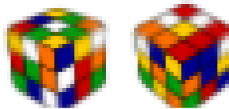


Figure 55: $N = 7, \theta_{\max} = 5$

Radial Lens Distortion For normalized coordinates $\mathbf{u}$, a radial warp $\mathbf{u}' = \mathbf{u} (1 + k \|\mathbf{u}\|^2)$ models first-order barrel ($k < 0$) or pincushion ($k > 0$) distortion. Inverse mapping with interpolation produces the output grid. The distortion strength k controls severity. This reproduces wide-angle lens distortion and optical calibration error.

Figure 56: $k=0.0001$

Projective (Perspective) Transform A homography $\mathbf{H} \in \mathbb{R}^{3 \times 3}$ maps homogeneous coordinates via $\tilde{\mathbf{x}}' \sim \mathbf{H}\tilde{\mathbf{x}}$. Choosing $\mathbf{H}$ from perturbed corner correspondences induces keystone deformation. Inverse warping with interpolation preserves sampling density. Parameters are corner displacements or direct specification of $\mathbf{H}$. This models oblique viewpoints and camera pose changes relative to the image plane.

Figure 57: $\Delta = 0.1$

Object Albedo Channel Offset Object face colors are shifted additively in RGB, $\mathbf{a}' = \text{clip}(\mathbf{a} + \boldsymbol{\delta})$, applied to each semantic region prior to shading. The channel-wise offsets control tint magnitude and direction. This emulates illumination casts and sensor biases producing uniform tints on object surfaces.

Figure 58: $\boldsymbol{\delta} = [0.1, 0, 0]$

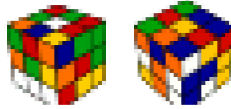
Object Albedo Channel Scaling A diagonal color calibration $\mathbf{D} = \text{diag}(\lambda_R, \lambda_G, \lambda_B)$ scales albedo channels, $\mathbf{a}' = \text{clip}(\mathbf{D}\mathbf{a})$. Per-channel gains λ . modulate relative color balance. The augmentation simulates white-balance and per-channel gain mismatches across cameras.

Figure 59: $\lambda_R = 1, \lambda_G = 1, \lambda_B = 1$

Material Fading (Sticker Wear) Chromatic content blends toward local luminance $y = \frac{1}{3}(a_R + a_G + a_B)$: $\mathbf{a}' = (1 - \rho)\mathbf{a} + \rho y \mathbf{1}$. The fade factor $\rho \in [0, 1]$ governs desaturation. The effect captures sun bleaching, aging, and pigment loss from surface wear.


 Figure 60: $\rho = 0.5$

Micro-Texture Perturbations High-frequency albedo perturbations are injected as bounded noise $\epsilon \in [-\nu, \nu]^3$ added to each surface region, followed by clipping. The amplitude ν shapes scratch visibility. This simulates fine scratches and scuffs that locally modulate reflectance.


 Figure 61: $\nu = 0.05$

Material Finish (Matte vs. Glossy) The base plastic color of the cube is switched between a dark, absorptive value for a matte finish and a light, reflective silver value for a glossy finish, which changes highlight prominence and perceived reflectance under fixed lighting. A categorical parameter selects matte (more absorptive) or glossy (more reflective). This models manufacturing differences in plastics and surface polishing.



Figure 62: finish=matte

Polygonal Imperfections in Stickers Vertices of sticker polygons receive bounded spatial jitter Δv with $\|\Delta v\| \leq \epsilon$, preserving polygon closure. The piecewise-linear boundary undergoes small shape changes. The jitter radius ϵ sets deviation magnitude.


 Figure 63: $\epsilon = 0.05$

Camera Pose Perturbation (Yaw-Pitch-Roll) The extrinsic rotation is varied by Euler angles (ψ, ϕ, φ) , and the pinhole projection maps 3D points $\tilde{X}$ to pixels via $\tilde{x} \sim \mathbf{K} [\mathbf{R}(\psi, \phi, \varphi) \mid \mathbf{t}] \tilde{X}$. Small perturbations in the angles capture realistic viewpoint drift while maintaining object visibility. This simulates handheld motion and mount misalignment.

Figure 64: $\psi = -20, \phi = -10, \varphi = -8$

3D Zoom and Viewport Translation Scene geometry is uniformly scaled by s , optionally combined with a small translation within the image plane. The projection window adjusts to avoid cropping. Parameters are scale $s > 0$ and planar offsets. The effect simulates effective focal changes and subject recentering without altering the background.

Figure 65: $s = 1.2, offset_x = 0, offset_y = 0$

Photographic Background Composite A natural photograph B sampled from the CIFAR-10 dataset (Krizhevsky et al., 2009) is resized to the canvas and alpha-composited behind the rendered object, so that for background pixels $I' = (1 - \alpha)I + \alpha B$. The parameters are the opacity $\alpha \in [0, 1]$ and a placement mode among *fill*, *contain*, *cover*, and *center* that governs how B is fit to the frame. This introduces realistic scene context and clutter behind the object, in contrast to the constant or procedural backgrounds above.

Figure 66: $\alpha = 0.8, mode = fill$

Random Occluders A set of n opaque shapes, each a rectangle or circle whose size is drawn as a fraction of the image dimensions within $[f_{min}, f_{max}]$, is rasterized into a binary mask $M \in \{0, 1\}^\Omega$, and the image is masked multiplicatively, $I' = I \odot M$, so that covered regions are set to zero. The parameters are the number of shapes n and the size-fraction range $[f_{min}, f_{max}]$. This models partial occlusion of the object by foreground clutter.

Figure 67: $n = 3, [f_{min}, f_{max}] = [0.1, 0.25]$

Affine Spatial Perturbation A single random affine map combines translation, rotation, and anisotropic scaling, $\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{t}$, realized by inverse warping with interpolation. The translation is drawn uniformly up to $\tau \min(H, W)$ pixels, the rotation uniformly in $[-\theta_{max}, \theta_{max}]$, and the per-axis scale as $1 + \mathcal{U}(-\zeta, \zeta)$. The parameters are the maximum translation fraction τ , maximum

rotation $\theta_{\max}$, and maximum scale deviation ζ . This simulates combined camera translation, roll, and distance changes.

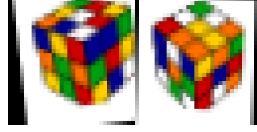


Figure 68: $\tau = 0.05, \theta_{\max} = 10, \zeta = 0.1$

Adversarial Sinusoidal Noise A structured sinusoidal pattern is added to the image, $I' = \text{clip}(I + \frac{a}{2}(\sin(fx) + \cos(fy)))$, where the coordinates (x, y) span $[0, 2\pi]$ across the width and height. The frequency f sets the pattern density and the amplitude a its strength. This injects a coherent, patterned perturbation reminiscent of adversarial interference.



Figure 69: $f = 10, a = 0.05$

Fog and Haze A radial transmission mask $g(\mathbf{x}) = \exp(-\|\mathbf{x}\|^2/\varsigma^2)$ over normalized coordinates $\mathbf{x} \in [-1, 1]^2$ reduces contrast away from the image center, giving $I' = \text{clip}((I + \iota(1 - g))(1 + 0.05) + 0.02)$. The spread ς sets the falloff radius and the intensity ι the haze strength. This reproduces atmospheric scattering and veiling glare.

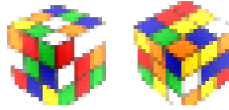


Figure 70: $\varsigma = 0.5, \iota = 0.3$

Rain Streaks A total of N bright diagonal streaks are drawn at random positions, each of random length ℓ , advancing one pixel horizontally and half a pixel vertically per step to form a roughly 2:1 slope, and rendered with thickness t by setting the covered pixels to white. The parameters are the number of streaks N and the streak thickness t . This simulates rain captured under a short exposure.



Figure 71: $N = 20, t = 1$

Resampling and Codec Degradation Perceptual capture loss is emulated by downsampling to a fraction ρ of the original size and resampling back to the original dimensions with a chosen interpolation kernel, followed by an image-codec round-trip $\mathcal{C}$ that is either a lossless PNG pass with 8-bit quantization or a lossy JPEG pass with quality q and chroma subsampling. Writing one pass as $\mathcal{D} = \mathcal{C} \circ \mathcal{R}_{\uparrow} \circ \mathcal{R}_{\downarrow}$, the effect applies r passes and blends the result with the source,

$I' = (1 - \sigma) I + \sigma \mathcal{D}^r(I)$, while preserving the original dimensions. The parameters are the resize fraction ρ , the interpolation kernel, the save format, the JPEG quality q , the number of round-trips r , and the blend strength $\sigma \in [0, 1]$. This models resolution loss and compression artifacts introduced by imaging pipelines.



Figure 72: $\rho = 0.5, \sigma = 1, r = 2, q = 50$

For Rubik's Cube, we combine these augmentations into the following variants:

- Directional lighting with controllable ambient, shadow, gradient, and subsurface components, a background image composited behind the cube, and camera yaw, pitch, and roll variation.
- Hue, saturation, and value shifts, RGB channel offsets, global contrast and brightness scaling, zooming in and out with positional offsets, camera rotation, and a background image.
- A lightweight mix for regularization involving directional lighting, zoom and vertical offset, camera angle variation, background image, and ambient light tint to simulate global illumination.
- Warm look with glossy material finish for the cube, slight sticker-vertex jitter, per-channel color calibration, warm ambient light and color-temperature shift, zoom and camera jitter, and a background image.
- Harsher directional light with optional vignette, color-temperature shifts in both directions, Gaussian blur and exposure gamma variation, background randomization from the image pool, camera jitter, and low to moderate additive Gaussian noise.
- High-key lighting with white ambient fill, slightly increased contrast and brightness, glossy finish, per-channel scaling, and Gaussian noise.
- Noisy surveillance look with higher Gaussian noise, salt-and-pepper noise, reduced color bit-depth (banding), edge cropping, conservative contrast tweaks, mixed matte or glossy finish, and sticker jitter.
- Controlled illumination using a white ambient fill plus directional light, glossy material, sticker jitter, low noise, slightly boosted contrast, and a background image.
- Camera angle variation, a background image, and Gaussian noise to mimic sensor grain.
- Background image, camera jitter, Gaussian blur, directional light with gamma and vignette controls, and light Gaussian noise.
- Background image, directional light combined with a colored ambient wash, motion blur, and Gaussian noise.
- Similar to the above but with lighter ambient, motion blur, a milder vignette, and small Gaussian noise.
- Filled background image, Gaussian blur, radial motion blur, colored ambient light at low opacity, small Gaussian noise, and directional light with vignette.
- Procedural background texture patterns (grid, dots, stripes, crosshatch, waves, or noise) added in the background, Gaussian blur and radial blur, Poisson noise, and directional light on the cube.
- Random opaque shapes placed in front of the scene to partially occlude the cube, combined with Gaussian noise, directional light, camera jitter, zoom, and a background image.
- A simple composition of a background image, directional lighting on the cube, and camera pose variation.

12 Planning Result Tables

Table 2: SPAR’s performance on the Rubik’s Cube across augmentation variants, showing solution length, nodes generated, time (seconds), nodes per second, and percentage solved with 95% Wilson score confidence intervals (1,000 test instances per variant).

DOMAIN	OBSERVATION	LEN	NODES	SECS	NODES/SEC	SOLVED	95% CI
Rubik’s Cube	Bright Lighting	22.85	2.00E+05	6.21	3.22E+04	96.8%	[95.5, 97.7]
	Camera and Lighting	22.86	2.00E+05	6.22	3.22E+04	97.6%	[96.5, 98.4]
	Camera and Image	22.84	2.00E+05	6.21	3.22E+04	93.9%	[92.2, 95.2]
	Camera and Background and Ambient Light	22.86	2.00E+05	6.22	3.22E+04	82.0%	[79.5, 84.3]
	Camera and Background and Lighting	22.84	2.00E+05	6.21	3.22E+04	84.9%	[82.5, 87.0]
	Camera and Image Background	22.86	2.00E+05	6.21	3.22E+04	95.4%	[93.9, 96.5]
	Color Variations	22.83	2.00E+05	6.21	3.22E+04	93.7%	[92.0, 95.0]
	Indoor and Glossy	22.86	2.00E+05	6.22	3.22E+04	94.6%	[93.0, 95.8]
	Lighting and Background and Camera	22.85	2.00E+05	6.21	3.22E+04	84.3%	[81.9, 86.4]
	Lighting and Radial Blur and Camera	22.87	2.00E+05	6.22	3.22E+04	84.0%	[81.6, 86.1]
	Minimal	22.86	2.00E+05	6.21	3.22E+04	97.0%	[95.7, 97.9]
	Noisy Camera	22.86	2.00E+05	6.22	3.22E+04	80.7%	[78.1, 83.0]
	Occluders	22.87	2.00E+05	6.22	3.22E+04	94.1%	[92.5, 95.4]
	Outdoor Lighting	22.83	2.00E+05	6.21	3.22E+04	80.2%	[77.6, 82.6]
	Studio	22.87	2.00E+05	6.22	3.22E+04	88.2%	[86.1, 90.1]
	Texture and Poisson Noise	22.85	2.00E+05	6.21	3.22E+04	91.3%	[89.4, 92.9]

Table 3: SPAR’s performance on Sokoban across augmentation variants, showing solution length, nodes generated, time (seconds), nodes per second, and percentage solved with 95% Wilson score confidence intervals (1,000 test instances per variant).

DOMAIN	OBSERVATION	LEN	NODES	SECS	NODES/SEC	SOLVED	95% CI
Sokoban	Adversarial Noise	32.32	2.39E+04	20.58	1.16E+03	99.9%	[99.4, 100.0]
	Defocus Blur	32.32	2.39E+04	20.58	1.16E+03	100.0%	[99.6, 100.0]
	Fog Haze	32.32	2.39E+04	20.58	1.16E+03	100.0%	[99.6, 100.0]
	Gaussian Blur	32.32	2.39E+04	20.58	1.16E+03	100.0%	[99.6, 100.0]
	Gaussian Noise	32.27	2.39E+04	20.54	1.16E+03	97.6%	[96.5, 98.4]
	Hsv Shift	32.32	2.39E+04	20.58	1.16E+03	100.0%	[99.6, 100.0]
	Indoor Lighting	32.31	2.39E+04	20.58	1.16E+03	99.9%	[99.4, 100.0]
	Lighting Only	32.27	2.38E+04	20.54	1.16E+03	97.6%	[96.5, 98.4]
	Microscopy	32.31	2.39E+04	20.58	1.16E+03	98.1%	[97.1, 98.8]
	Motion Conditions	32.17	2.38E+04	20.44	1.16E+03	92.4%	[90.6, 93.9]
	Noise And Distortion	32.30	2.39E+04	20.59	1.16E+03	98.4%	[97.4, 99.0]
	Outdoor	32.27	2.38E+04	20.52	1.16E+03	99.3%	[98.6, 99.7]
	Radial Blur	32.32	2.39E+04	20.58	1.16E+03	99.8%	[99.3, 99.9]
	Rain Effect	32.34	2.39E+04	20.59	1.16E+03	99.7%	[99.1, 99.9]
	Random Occluders	32.46	2.41E+04	20.25	1.19E+03	52.2%	[49.1, 55.3]
	Rotate Image	32.32	2.39E+04	20.58	1.16E+03	100.0%	[99.6, 100.0]
	Surveillance	32.25	2.38E+04	20.46	1.16E+03	98.9%	[98.0, 99.4]
	Zoom Effect	32.33	2.39E+04	20.59	1.16E+03	99.8%	[99.3, 99.9]

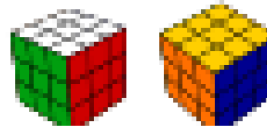
13 Unsolved Examples

In this section, we include examples of test instances that were not solved during planning. These examples show the significant changes that augmentation can introduce into the observations.

Start State (Input to Search)



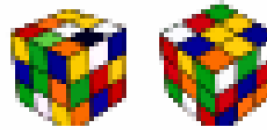
Goal State



Start State (Ground Truth)

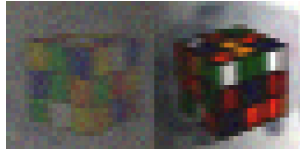


Reconstruction

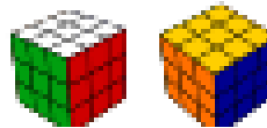


MSE: 0.00254 • Bitwise Equality: 99.00%

Start State (Input to Search)



Goal State



Start State (Ground Truth)



Reconstruction

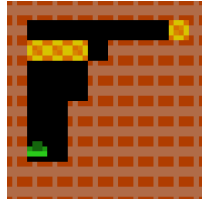


MSE: 0.000584 • Bitwise Equality: 99.25%

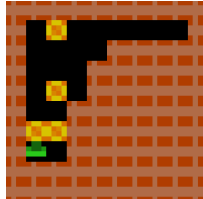
Start State (Input to Search)



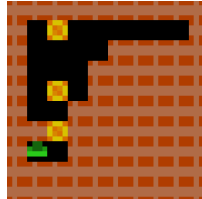
Goal State



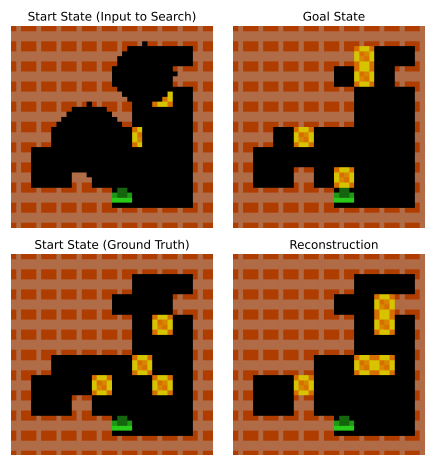
Start State (Ground Truth)



Reconstruction



MSE: 0.000545 • Bitwise Equality: 99.50%



MSE: 0.0229 • Bitwise Equality: 94.81%