Sliding Puzzles Gym: A Scalable Benchmark for State Representation in Visual Reinforcement Learning

Bryan L. M. de Oliveira[1,2], **Luana G. B. Martins**[1], **Bruno Brandão**[1,2], **Murilo L. da Luz**[1,2], **Telma W. de L. Soares**[1,2], **Luckeciano C. Melo**[1,3]
bryanlincoln@discente.ufg.br

[1]**Advanced Knowledge Center for Immersive Technologies – AKCIT**
[2]**Institute of Informatics, Federal University of Goiás, Brazil**
[3]**OATML, University of Oxford**

Abstract

While effective visual representation learning is critical for reinforcement learning (RL) agents to generalize across diverse environments, existing benchmarks cannot evaluate how different inductive biases affect this capability in isolation. To address this, we introduce the Sliding Puzzles Gym (SPGym), a benchmark that isolates the challenge of visual representation learning. SPGym transforms the classic sliding puzzle into a visual RL task where visual complexity can be scaled by adjusting grid sizes and the pool of images used for tiles, while environment dynamics, observation, and action spaces remain fixed. Our experiments with model-free and model-based algorithms reveal how different architectural and algorithmic biases affect an agent's ability to handle visual diversity. As the image pool grows, all algorithms exhibit performance degradation both in- and out-of-distribution, with sophisticated representation techniques often underperforming simpler approaches like data augmentation. These findings expose critical gaps in visual representation learning and establish SPGym as a valuable tool for developing more robust and generalizable agents.

1 Introduction

The ability to learn meaningful representations from raw sensory inputs like images is crucial for reinforcement learning (RL) agents to generalize in complex, open-world environments (Bengio et al., 2013; Lesort et al., 2018). In visual RL, agents must process high-dimensional pixel data to extract features for decision-making, a process fundamentally shaped by the inductive biases of their architecture and learning algorithms (Mnih et al., 2015; Yarats et al., 2021a). However, measuring an agent's representation learning capabilities independently of other learning tasks, such as policy optimization or dynamics modeling, remains a key challenge in RL benchmarks.

Traditional RL benchmarks like Atari (Bellemare et al., 2013) and the DeepMind Control Suite (Tassa et al., 2018) are valuable for assessing overall agent performance, but they conflate representation learning with policy optimization and environment dynamics. Even recent benchmarks designed for visual learning and generalization have limitations. For instance, ProcGen (Cobbe et al., 2020) varies visual and task difficulty simultaneously, making it hard to isolate the effect of representation learning. The Distracting Control Suite (Stone et al., 2021) adds visual distractors that are irrelevant to the task. RL-ViGen (Yuan et al., 2023) tests generalization across different types of modifications to the Markov decision process but doesn't fix visual complexity as the controlled variable. As a result, current benchmarks cannot systematically assess how well an agent learns task-relevant visual representations in isolation.

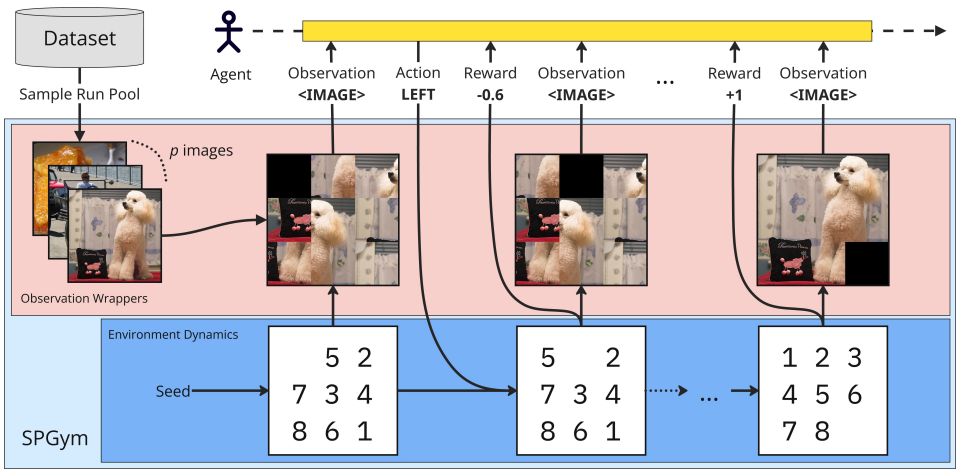

Figure 1: **Overview of SPGym.** The framework extends the 8-tile puzzle by replacing numbered tiles with image patches. At each training run, SPGym samples a pool of images and, at each episode, it randomly selects one of those images to form the observations. While we scale visual diversity by adjusting the pool size, the task and environment dynamics remain fixed.

To address this gap, we introduce the *Sliding Puzzles Gym (SPGym)*[1], an open-source benchmark for evaluating how agents handle visual diversity. As illustrated in Figure 1, SPGym turns the classic sliding puzzle into a discrete visual RL task. Its design is guided by three principles: (1) consistent environment dynamics, so the task itself does not change; (2) precise control over visual complexity via adjustable grid sizes and image pools; and (3) a clear success metric based on puzzle completion. SPGym provides a controlled setting to systematically measure how an agent's performance is affected by visual diversity, revealing the limits of its representation learning capabilities.

Our experiments show that SPGym effectively distinguishes agents by their out-of-the-box representation learning abilities, which strongly correlate with task performance. While pretraining and data augmentation are beneficial, many advanced methods underperform with standard configurations, suggesting either a need for task-specific tuning or fundamental mismatches with SPGym's visual-structural dynamics. More importantly, our generalization analysis reveals a critical gap in current methods: on SPGym, agents that master the training images fail to transfer to unseen ones, even when trained on larger and more diverse image pools. In fact, performance often degrades as visual diversity increases, indicating that current methods struggle to learn truly generalizable representations. We further demonstrate SPGym's extensibility through experiments with procedurally generated images and larger puzzles, maintaining fixed observation/action spaces while expanding state space and visual diversity. These findings expose critical gaps in current visual RL methods and establish SPGym as a valuable tool for advancing robust, generalizable decision-making systems.

Contributions. Our main contributions are: (1) SPGym, a novel benchmark for systematically evaluating representation learning by scaling visual complexity while holding environment dynamics constant; (2) an extensive empirical analysis of state-of-the-art methods that uncovers their limitations in handling visual diversity; and (3) fundamental insights into the challenges of scaling visual RL that suggest new directions for research.

2 Related Work

Traditional RL benchmarks. Foundational visual RL benchmarks like the Atari Learning Environment (Bellemare et al., 2013) and the DeepMind Control Suite (Tassa et al., 2018) spurred the development of representation learning techniques like pretraining (Higgins et al., 2017; Stooke

[1] Available at https://github.com/bryanoliveira/sliding-puzzles-gym.

et al., 2021; Schwarzer et al., 2021), contrastive learning (Laskin et al., 2020a), self-supervised prediction (Schwarzer et al., 2020), and world models (Hafner et al., 2025). However, these benchmarks assess overall agent performance, intertwining representation learning with policy optimization and dynamics modeling, which obscures the specific impact of representation learning. SPGym addresses this by fixing the environment dynamics while varying visual complexity, enabling a focused evaluation of representation learning.

Specialized benchmarks for visual RL. Recent benchmarks have tried to better assess visual generalization, but with certain limitations. ProcGen (Cobbe et al., 2020) scales visual and task difficulty simultaneously, making it hard to disentangle their effects. The Distracting Control Suite (Stone et al., 2021) adds visual distractors that are irrelevant to the task and can be ignored. RL-ViGen (Yuan et al., 2023) tests generalization across different MDP modifications but doesn't control visual complexity scaling. In contrast, SPGym makes visual understanding essential for solving the task. By scaling the visual diversity of the puzzle tiles while keeping the task complexity fixed, SPGym creates a representation learning challenge that is directly tied to task completion.

Puzzle-based benchmarks and solving methods. Puzzle-based environments have proven valuable for evaluating neural algorithmic reasoning, primarily with discrete states (Estermann et al., 2024). While classical approaches to sliding puzzles employ heuristic search algorithms like A* with Manhattan distance heuristics (Korf, 1985; Burns et al., 2012; Lee & See, 2022), these methods require access to internal states and can be computationally expensive. Deep RL offers a scalable alternative that learns strategies without handcrafted heuristics (Agostinelli et al., 2019; Moon & Cho, 2024; Estermann et al., 2024), though prior work has focused on discrete states rather than visual inputs (Agostinelli et al., 2019; Moon & Cho, 2024). Similarly motivated by puzzle simplicity and scalability, Wang et al. (2025) proposed Jigsaw-R1 for evaluating multimodal reasoning in MLLMs, though our focus is on RL representation learning sample efficiency. While Estermann et al. (2024) demonstrated that agents struggle with even basic pixel-based inputs, their work revealed the potential of puzzles for visual learning and the need for systematic evaluation. SPGym advances this research direction by incorporating rich visual observations into a controlled puzzle environment, where agents must learn exclusively from pixel observations without access to internal states, thereby creating a benchmark for assessing visual representation learning and providing insight into RL algorithms' performance on this challenging visual task.

A preliminary version of this work appeared at the Open-World Agents Workshop at NeurIPS 2024 (de Oliveira et al., 2024). The present version includes extended related work and formalization, representation learning methods for SAC, out-of-distribution analysis, solution optimality analysis, experiments with one-hot encoding, 4x4 grids, linear probing evaluation, and experiments with an alternative base dataset.

3 The Sliding Puzzles Gym

SPGym is an open-source benchmark that extends the classic sliding puzzle to evaluate visual representation learning in RL agents. It supports configurable $H \times W$ grids and various observation modalities, and adheres to the Gym interface (Brockman, 2016). In our experiments, we use a 3×3 grid where tiles are patches of an image, and the agent observes the composite image of the grid (Figure 1). SPGym adheres to the Gym (Brockman, 2016) interface, promoting modularity between environment and agent.

Formalization. SPGym is a partially observable Markov decision process (POMDP) defined by $(\mathcal{S}, \mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{S}_0)$, where $\mathcal{S}$ is the state space of tile configurations, $\mathcal{X}$ is the image-based observation space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the deterministic transition function, $\mathcal{R}$ is the reward function, and $\mathcal{S}_0$ is the initial state distribution.

State space and observations. The agent does not see the underlying state $s \in \mathcal{S}$ and must learn a policy $\pi : \mathcal{X} \to \mathcal{A}$ from observations $x \in \mathcal{X}$. For each training run, a pool $\mathcal{I}$ of p images is sampled from a dataset. At the start of an episode, an image $i \in \mathcal{I}$ is randomly selected, split into

$H \times W$ patches, and overlaid on the puzzle. The agent's goal is to reassemble the shuffled image. This formulation provides two mechanisms for controlling complexity: (1) varying p adjusts the diversity of the observation space $\mathcal{X}$ by changing the pool of available images, and (2) modifying H and W alters the state space complexity by changing the grid dimensions and number of puzzle pieces. Both mechanisms operate while keeping the underlying dynamics $\mathcal{P}$, the action space $\mathcal{A}$, and the reward function $\mathcal{R}$ fixed.

Action space and dynamics. The action space $\mathcal{A}$ consists of four discrete actions: *UP*, *DOWN*, *LEFT*, or *RIGHT*, which move a tile into the adjacent empty space. Once the agent selects a tile to move, the dynamics $\mathcal{P}$ define the next puzzle state in a predictable and deterministic way.

Reward function. We use a dense reward function based on the normalized Manhattan distance of tiles from their goal positions, a common metric in sliding puzzle solvers (Korf, 1985; Burns et al., 2012; Lee & See, 2022; Moon & Cho, 2024). The reward at each step is:

$$r_t = \begin{cases} -D, & \text{if action is valid} \\ -1, & \text{if action is invalid} \\ +1, & \text{if puzzle is solved} \end{cases}, \text{ with } D = \frac{\sum_{i=1}^{H}\sum_{j=1}^{W} |u_{i,j} - u^*_{i,j}| + |v_{i,j} - v^*_{i,j}|}{\sum_{i=1}^{H}\sum_{j=1}^{W} \max(i, H-i) + \max(j, W-j)}. \quad (1)$$

Here, $(u_{i,j}, v_{i,j})$ is the current tile position and $(u^*_{i,j}, v^*_{i,j})$ is its target. This provides a learning signal between $[-1, +1]$ that encourages solving the puzzle in the fewest steps. Invalid moves are penalized with -1, and don't alter the puzzle state. For example, in Figure 1, the *DOWN* and *RIGHT* actions would be invalid for the first state. Solving the puzzle yields $+1$ and ends the episode.

Initial state distribution. To start an episode, we generate a random, solvable puzzle state by uniformly placing tiles on the grid and ensuring solvability by adjusting the puzzle's parity if needed (Johnson & Story, 1879). We also support curriculum learning by starting from a solved state and applying random moves, but we do not use this method in our experiments due to its computational cost at larger grid sizes.

Scalability and extensibility. SPGym provides two orthogonal mechanisms for scaling task difficulty: visual diversity and grid size. While our primary experiments use 3×3 image-based puzzles, SPGym supports larger grids and other observation modalities, such as one-hot encodings (Figure 2), to isolate specific research questions (see Section 9.1).

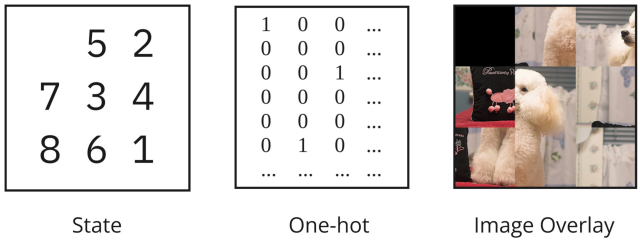

Figure 2: **Different observation modalities in SPGym.** Each modality presents a unique challenge for representation learning, while corresponding to the same puzzle state. We focus our experiments on image-based observations.

The primary mechanism isolates the challenge of representation learning by increasing visual diversity while keeping all other task components fixed. This is achieved by expanding the image pool from which puzzles are generated (Figure 3). As the pool grows, agents require more samples to solve the task (Table 1), even though the state space, action space, and transition dynamics remain unchanged. This increased difficulty stems directly from the representation learning challenge. We confirm this link through linear probe analysis (Section 9.1.2), which shows a strong correlation between representation quality and task performance. This controlled scaling provides a stress test for visual learning that is not possible in settings with fixed visual inputs or state-based observations (see Section 9.1.1).

The secondary mechanism scales the exploration and planning challenge by adjusting grid dimensions. Increasing the grid from 3×3 to 4×4 expands the state space exponentially, increasing sample complexity (Table 3). This also makes representation learning harder, as the agent must correctly position more patches. Crucially, since the observation and action spaces remain identical,

this setup isolates the impact of state-space complexity. While this mechanism is useful for testing exploration, our experiments show that the 3×3 grid is sufficient to assess the representation learning capabilities of different agents.

By independently controlling these two axes of difficulty, SPGym creates a controlled experimental setting where performance variations can be more directly attributed to an agent's representation learning capabilities.

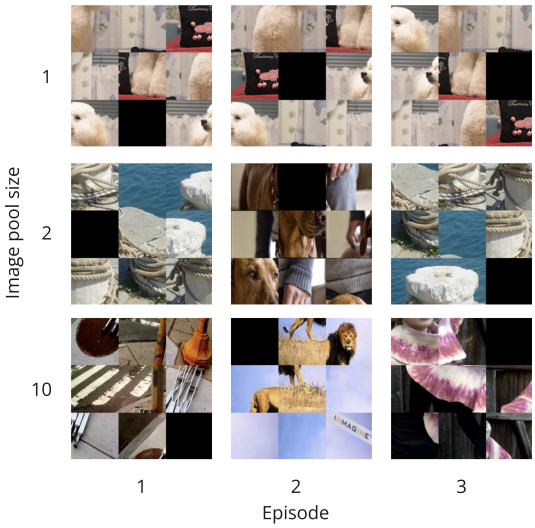

Figure 3: **Scalable visual diversity in SPGym.** Each row shows the first observation of 3 different episodes of the same training run. Crucially, we keep the grid size fixed, ensuring difficulty comes solely from handling the larger variety of visual inputs controlled by the size of the image pool.

4 Methods

We evaluate various RL agents within the SPGym framework to assess their ability to handle increasing visual diversity.

Experimental setup. We construct visual observations using images from ImageNet-1k's validation split (Russakovsky et al., 2015), resized to 84×84 pixels. To isolate visual diversity, we fix the puzzle size to 3×3 and vary only the image pool size p. For each training run, we randomly sample p distinct images to create a fixed pool, then randomly select one image per episode to generate puzzle observations. We cap environment steps at 10M and limit episodes to 1,000 steps.

Our primary metric is sample efficiency: the number of environment steps required to reach 80% success rate. We terminate runs early when agents maintain 100% success rate for 100 consecutive episodes, enabling out-of-distribution evaluation before extreme overfitting. Each experiment uses 5 random seeds, reporting mean ± 95% confidence intervals. We evaluate all agent variants on pools of 1, 5, and 10 images, with additional scaling experiments up to 100 images for baseline methods.

Algorithms and variants. We explore three algorithmic approaches representing different learning strategies: Soft Actor-Critic (SAC) (Haarnoja et al., 2018a), Proximal Policy Optimization (PPO) (Schulman et al., 2017), and DreamerV3 (Hafner et al., 2025). For SAC, we begin with the standard implementation for discrete action spaces (Christodoulou, 2019) and examine several representation learning variants: RAD (Laskin et al., 2020b) (data augmentation), CURL (Laskin et al., 2020a) (contrastive learning), SPR (Schwarzer et al., 2020) (self-supervised prediction), DBC (Zhang et al., 2021) (state metric learning), SAC-AE and SAC-VAE (Yarats et al., 2021b) (reconstruction-based learning), and Simple Baseline (SB) (Tomar et al., 2023) (reward and transition prediction). For PPO, we evaluate three encoder configurations: standard with random initialization, pretrained on the same image distribution (in-distribution, ID) to provide an upper bound on expected pretraining performance, and pretrained on a different image distribution (out-of-distribution, OOD) to establish the potential benefits of generally pretrained encoders. For DreamerV3, we compare the standard version against a variant without decoder gradients to evaluate the impact of the reconstruction objective on performance. Detailed descriptions for all agents are provided in Section 8.

Implementation details. We adopt established neural architectures and hyperparameters from visual discrete control implementations, using three-layer CNN encoders with deconvolutional decoders and MLP components. For data augmentation, we apply a two-step pipeline consisting of grayscale conversion (20% probability) followed by channel shuffling, selected through systematic augmentation searches. To assess out-of-the-box performance, we use default hyperparameters for PPO and DreamerV3, with SAC requiring minimal tuning of the temperature parameter ($\alpha = 0.05$). Comprehensive details on algorithm implementations, architectural specifications, hyperparameter

settings, augmentation strategies, and gradient flow patterns are provided in Section 7. Our code [2] extends CleanRL (Huang et al., 2022) and official DreamerV3 implementations.

5 Results

Our analysis reveals three fundamental tensions in visual RL: between method assumptions and environment structure, sample efficiency and solution optimality, and training diversity versus generalization capability. We organize findings through six research questions.

Can SPGym distinguish agents on representation learning capabilities?

Table 1 demonstrates SPGym's ability to differentiate agents based on the sample efficiency of their representation learning variants. For PPO, in-distribution pretraining (PT (ID)) significantly boosts sample efficiency, with limited gains from OOD. For SAC, data augmentation via RAD is the most effective variant, while many sophisticated auxiliary methods underperform. This result aligns with findings from Tomar et al. (2023) and Yuan et al. (2023), suggesting that current methods may be overfitting to the benchmarks they were developed on. DreamerV3 demonstrates strong, stable performance, with its discrete reconstruction objective proving crucial for success. These results highlight SPGym's diagnostic value in assessing how different methods handle visual diversity and structured dynamics. For detailed performance analysis, see Section 9.2.1.

Table 1: **Million steps to reach 80% success rate across pool sizes.** Lower is better. Best performing variant for each algorithm and pool size is highlighted in bold.

Agent	Pool 1	Pool 5	Pool 10
PPO	1.75±0.44	7.80±1.08	9.73±0.36
PPO + PT (ID)	**0.95±0.21**	**5.55±1.22**	**9.17±1.10**
PPO + PT (OOD)	1.34±0.42	7.03±1.07	9.70±0.41
SAC	0.33±0.07	0.91±0.12	2.03±0.38
SAC + RAD	**0.24±0.03**	**0.42±0.06**	**0.82±0.18**
SAC + CURL	0.46±0.10	1.56±0.31	5.24±1.92
SAC + SPR	2.09±0.81	3.68±1.68	10.00±0.00
SAC + DBC	0.99±0.25	1.12±0.22	2.13±0.41
SAC + AE	1.04±0.24	1.02±0.19	2.01±0.38
SAC + VAE	1.13±0.14	5.30±0.68	10.00±0.00
SAC + SB	0.98±0.88	2.08±0.30	10.00±0.00
DreamerV3	**0.42±0.06**	**1.23±0.20**	**1.44±0.58**
DreamerV3$_{\text{w/o dec.}}$	1.13±0.12	1.79±0.61	2.57±0.91

How does visual diversity affect performance and generalization?

Figure 4 reveals that increasing visual diversity leads to distinct failure modes across algorithms. PPO degrades significantly at pool size 10, SAC at 30, while DreamerV3 is most robust, showing learning even at pool size 100. These patterns suggest agents memorize features rather than learning generalizable ones, exhausting network capacity as diversity increases. DreamerV3's world model appears to foster more compressed representations, leading to more graceful degradation. Preliminary DreamerV3 experiments with very large image pools (10,000-20,000 images) support this interpretation: agents failed to learn useful policies as each observation became vir-

Table 2: **Success rate of PPO and SAC agents on Easy OOD across different pool sizes.** Higher is better.

Algorithm	Pool 1	Pool 5	Pool 10
PPO	0.49±0.13	0.53±0.14	0.34±0.08
PPO + PT (ID)	0.33±0.09	0.53±0.16	0.27±0.07
PPO + PT (OOD)	0.49±0.12	0.52±0.14	0.34±0.08
SAC	0.45±0.12	0.58±0.12	0.46±0.12
SAC + AE	0.78±0.11	0.64±0.16	0.55±0.12
SAC + VAE	0.64±0.15	0.30±0.08	0.12±0.03
SAC + SPR	0.65±0.13	0.21±0.09	0.07±0.04
SAC + DBC	0.44±0.13	0.34±0.13	0.13±0.04
SAC + CURL	0.76±0.09	0.44±0.10	0.37±0.11
SAC + RAD	0.62±0.15	0.42±0.13	0.30±0.11
SAC + SB	0.89±0.08	0.65±0.12	0.06±0.02

[2] Available at https://github.com/bryanoliveira/spgym-experiments

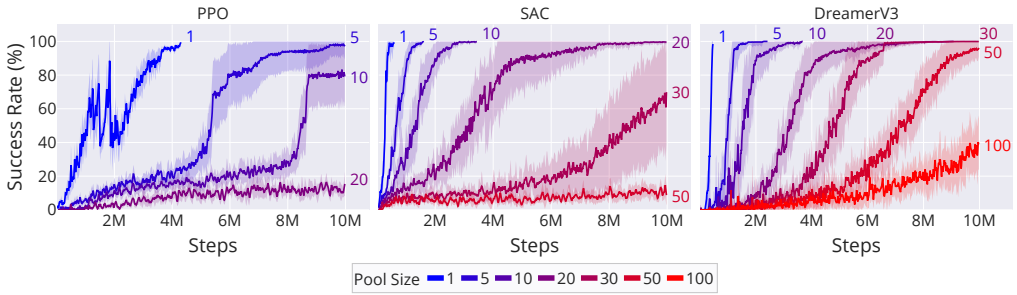

Figure 4: **Success rate as a function of environment steps.** The gradual increase in visual diversity affects the sample efficiency of standard PPO, SAC, and DreamerV3 agents at different rates. Each line represents a different pool size, from 1 to 100 images.

tually unique. This suggests that increased dataset scale alone is insufficient when the RL signal is too sparse for encoder training amidst SPGym's high visual diversity.

This reliance on memorization is confirmed by out-of-distribution (OOD) evaluation on 'Easy' (augmented training images) and 'Hard' (unseen images) difficulties. For Hard OOD, agents almost universally fail, with near-zero success rates. This complete inability to generalize suggests agents primarily memorize visual patterns. Counter-intuitively, even on Easy OOD (Table 2), performance often *decreases* as the training pool size grows, suggesting greater visual diversity hinders, rather than helps, learning robust invariances. However, we found a strong correlation (Pearson r=-0.81, p=2.5e-12) between Easy OOD success and in-distribution sample efficiency, suggesting that representations robust to simple transformations are tied to faster learning. These findings reveal that current methods may not learn sufficiently abstract representations, and that, for performance in SPGym, simply increasing training image diversity is insufficient. Full OOD performance data for both Easy and Hard settings across all configurations can be found in Section 9.2.

Is representation quality linked with performance?

To measure the quality of learned representations, we performed linear probing on frozen encoders from trained PPO and SAC agents, using a single-layer MLP to predict one-hot puzzle states. We find a statistically significant correlation between probe accuracy and sample efficiency (Pearson r=-0.81, p=1.1e-13), indicating that encoders capturing more task-relevant spatial information are strongly linked with faster learning. As image pool size increases, both probe accuracy and task performance systematically degrade, with standard SAC maintaining high accuracy (100% at pool size 1, 97.63% at pool size 5) mirroring its strong efficiency, while less efficient methods like SAC+VAE (78.21% at pool size 5) and SPR (dropping from 94.31% to 75.48% from pool size 5 to 10) show reduced probe performance. These consistent trends across algorithms demonstrate SPGym's ability to identify learning procedures that develop representations better aligned with its spatial reasoning needs. Full results are provided in Section 9.1.2.

Does performance generalize across image sources?

To validate that our findings generalize beyond ImageNet, we evaluated agents on DiffusionDB (Wang et al., 2023), a dataset of procedurally generated images. Our analysis reveals that performance scaling patterns on DiffusionDB closely mirror those observed on ImageNet across PPO, SAC, and DreamerV3 (see Section 7.6 for detailed performance curves). This consistency across different image sources, comparing real photographs with synthetic generations, demonstrates that visual diversity rather than semantic content drives the representation learning challenge in SPGym. The similarity in degradation patterns as pool size increases indicates that our algorithmic insights reflect fundamental properties of the tested methods rather than dataset-specific artifacts. Procedurally generated images also offer practical advantages for future research, including eliminating storage requirements through on-demand generation, enabling fine-grained control over visual similarity, and providing unlimited training diversity.

How does puzzle size affect learning performance? Another direction is increasing puzzle size. As shown in Table 3, the complexity increase from 3×3 to 4×4 grids significantly impacts learning. On the simpler 3×3 puzzle, PPO solved the puzzle in 1.75M steps, while SAC and DreamerV3 were more efficient, requiring 0.33M and 0.42M steps, respectively. For the 4×4 puzzle, PPO's sample requirements surged to 24.46M steps, far exceeding the 10M step training budget, while SAC and DreamerV3 still solved the puzzle within budget at 8.14M and 2.26M steps, respectively. This shows that while larger state spaces pose a major challenge requiring more exploration and complex visual representations, sample-efficient algorithms can still scale to such tasks.

Table 3: **Million steps to reach 80% success rate across grid sizes, with pool size 1.** Lower is better.

Grid Size	PPO	SAC	DreamerV3
3×3	1.75±0.44	0.33±0.07	0.42±0.06
4×4	24.46±7.58	8.14±3.64	2.26±0.29

How optimal are the learned solutions?

While our primary focus is on sample efficiency for task completion, we also analyze solution quality by examining the average number of steps agents take to solve puzzles. Our experimental design uses early termination when agents achieve 100% success rate to enable out-of-distribution evaluation before extreme encoder overfitting and to save computational resources. However, this approach may prevent agents from discovering more optimal solutions through continued training. To investigate this trade-off, we trained PPO, SAC, and DreamerV3 on pool size 1 for the full 10M steps without early termination across 5 seeds. Comparing episode lengths between when these agents first achieve 100% success rate (where early termination would occur) and after completing the full training reveals substantial improvements in solution efficiency. For PPO, the first 100 successful episodes average 214.30 ± 16.52 steps, while the last 100 episodes average 31.35 ± 6.59 steps. SAC shows improvement from 64.16 ± 9.81 to 57.27 ± 12.29 steps, and DreamerV3 improves from 126.02 ± 17.25 to 23.48 ± 0.71 steps. Notably, DreamerV3 with continued training on pool size 1 approaches the theoretical 22-step optimal solution for the 3x3 puzzle (Reinefeld, 1993), confirming that early termination may prevent agents from discovering more optimal solutions despite being needed for our experimental objectives.

6 Conclusion

We introduce the Sliding Puzzles Gym (SPGym), a benchmark that isolates visual complexity from environment dynamics to evaluate representation learning in discrete RL settings. Our analysis reveals three key challenges: (1) sophisticated representation learning techniques often underperform simpler approaches like data augmentation on SPGym's combination of visual diversity and spatial structure; (2) agents trained on smaller image pools show better robustness to simple perturbations, suggesting they learn more task-specific invariances; and (3) current methods fail on novel visual contexts, achieving poor or near-zero success rates despite strong in-distribution performance, indicating reliance on memorization rather than genuine visual understanding. These findings highlight the need for algorithms that better balance sample efficiency with visual generalization.

We identify two main limitations in this work: (1) evaluating methods with minimal tuning may not reveal their full potential, and (2) computational constraints limited us to 5 runs per configuration, while more seeds would improve statistical robustness given the stochasticity in image sampling.

SPGym enables systematic investigation of training diversity versus generalization (ID & OOD), integration with image generation systems for controlled continual learning experiments, and incorporation of multiple modalities for studying pretrained model encoders. These directions, along with the study of stronger inductive biases, could lead to more robust, transferable learning approaches.

Broader Impact Statement

This work advances understanding of visual representation learning in RL, potentially enabling more robust systems for real-world applications like robotics and autonomous systems. Our benchmark exposes current limitations while promoting rigorous evaluation practices. However, risks include overfitting to specific evaluation criteria, misuse for surveillance, and potential job displacement through automation. To mitigate these concerns, we have open-sourced our benchmark and protocols, encouraging researchers to consider both performance metrics and societal implications while implementing safeguards against dual-use applications.

Acknowledgments

The authors gratefully acknowledge the valuable insights and constructive discussions provided by Professors Marcos R. O. A. Maximo and Flávio H. T. Vieira.

This work has been partially funded by the project Research and Development of Digital Agents Capable of Planning, Acting, Cooperating and Learning supported by Advanced Knowledge Center in Immersive Technologies (AKCIT), with financial resources from the PPI IoT/Manufatura 4.0 / PPI HardwareBR of the MCTI grant number 057/2023, signed with EMBRAPII.

Luckeciano C. Melo acknowledges funding from the Air Force Office of Scientific Research (AFOSR) European Office of Aerospace Research & Development (EOARD) under grant number FA8655-21-1-7017.

References

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. DOI: 10.1038/s42256-019-0070-z.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Greg Brockman. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Ethan Burns, Matthew Hatem, Michael Leighton, and Wheeler Ruml. Implementing fast heuristic search code. *Proceedings of the International Symposium on Combinatorial Search*, 3(1):25–32, 2012.

Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2048–2056. PMLR, 13–18 Jul 2020.

Bryan L. M. de Oliveira, Bruno Brandão, Murilo L. da Luz, Luana G. B. Martins, Telma W. de L. Soares, and Luckeciano C. Melo. Sliding puzzles gym: A scalable benchmark for state representation in visual reinforcement learning. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Benjamin Estermann, Luca A. Lanzendörfer, Yannick Niedermayr, and Roger Wattenhofer. Puzzles: A benchmark for neural algorithmic reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

P. Ghosh, M. S. M. Sajjadi, A. Vergari, M. J. Black, and B. Schölkopf. From variational to deterministic autoencoders. In *8th International Conference on Learning Representations (ICLR)*, April 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *35th International Conference on Machine Learning (ICML)*, pp. 1861–1870. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, G. Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, P. Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018b.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, April 2025. ISSN 1476-4687. DOI: 10.1038/s41586-025-08744-2.

Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *34th International Conference on Machine Learning (ICML)*, pp. 1480–1490. PMLR, 2017.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Wm. Woolsey Johnson and William E. Story. Notes on the '15' puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879. DOI: 10.2307/2369492.

Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *37th International Conference on Machine Learning (ICML)*, pp. 5639–5650. PMLR, 2020a.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in Neural Information Processing Systems*, 33:19884–19895, 2020b.

Siaw Chong Lee and Tyan Her See. Comparing the hamming and manhattan heuristics in solving the 8—puzzle by a* algorithm. In Aida Binti Mustapha, Suhadir Shamsuddin, Syed Zuhaib Haider Rizvi, Saliza Binti Asman, and Siti Suhana Jamaian (eds.), *Proceedings of the 7th International Conference on the Applications of Science and Mathematics 2021*, pp. 189–195, Singapore, 2022. Springer Nature Singapore. ISBN 978-981-16-8903-1.

Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

Seong Uk Moon and Youngwan Cho. Improving the training performance of dqn model on 8-puzzle environment through pre-training. *International Journal of Applied Engineering Research*, 6(1): 63–67, 2024.

Alexander Reinefeld. Complete solution of the eight-puzzle and the bene t of node ordering in ida*. In *International Joint Conference on Artificial Intelligence*, pp. 248–253. Citeseer, 1993.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron C. Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2020.

Max Schwarzer, Nitarshan Rajkumar, Michael Noukhovitch, Ankesh Anand, Laurent Charlin, Devon Hjelm, Philip Bachman, and Aaron C. Courville. Pretraining representations for data-efficient reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite–a challenging benchmark for reinforcement learning from pixels. *arXiv preprint arXiv:2101.02722*, 2021.

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9870–9879. PMLR, 18–24 Jul 2021.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Manan Tomar, Utkarsh Aashu Mishra, Amy Zhang, and Matthew E. Taylor. Learning representations for pixel-based control: What matters and why? *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856.

Zifu Wang, Junyi Zhu, Bo Tang, Zhiyu Li, Feiyu Xiong, Jiaqian Yu, and Matthew B Blaschko. Jigsaw-r1: A study of rule-based visual reinforcement learning with jigsaw puzzles. *arXiv preprint arXiv:2505.23590*, 2025.

Zijie J. Wang, Evan Montoya, David Munechika, Haoyang Yang, Benjamin Hoover, and Duen Horng Chau. DiffusionDB: A large-scale prompt gallery dataset for text-to-image generative models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 893–911, Toronto, Canada, July 2023. Association for Computational Linguistics. DOI: 10.18653/v1/2023.acl-long.51.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *9th International Conference on Learning Representations (ICLR)*, 2021a.

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *Proceedings of the aaai conference on artificial intelligence*, 35(12):10674–10681, 2021b.

Zhecheng Yuan, Sizhe Yang, Pu Hua, Can Chang, Kaizhe Hu, and Huazhe Xu. Rl-vigen: A reinforcement learning benchmark for visual generalization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 6720–6747. Curran Associates, Inc., 2023.

Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021.

Supplementary Materials

The following content was not necessarily subject to peer review.

7 Experimental Setup and Configuration

7.1 Detailed Experimental Protocol

This section provides comprehensive details on the experimental setup described in Section 4.

Dataset preparation and preprocessing. We sample images from ImageNet-1k's validation split (Russakovsky et al., 2015) to construct visual observations, resizing each image to 84×84 pixels and normalizing values to $[0, 1]$. For each training run, we randomly sample p distinct images to create a fixed image pool. At episode start, we randomly select one image from this pool to generate the puzzle observations.

Training configuration. We cap the number of environment steps to 10M and limit episodes to 1,000 steps. Our analysis focuses on sample efficiency, measured by the number of environment steps required to solve the puzzle (lower is better). We calculate this metric by averaging the steps needed to reach 80% success rate across all parallel environments in a run, then average this number across seeds. We terminate training runs early when an agent maintains 100% success rate for 100 consecutive episodes, indicating task completion. This early termination serves two purposes: it enables out-of-distribution evaluation before extreme encoder overfitting occurs, and it reduces computational costs for running the comprehensive set of experiments. Each experiment is conducted 5 times with different random seeds, and we report the mean ± 1.96 standard errors (95% confidence interval).

Evaluation protocol. We evaluate sample efficiency using pools of 1, 5, and 10 images across all agent variants. For standard agents, we additionally test with progressively larger pools up to 100 images or until the final performance is less than 80% success rate after the full 10M training steps. This protocol allows us to analyze both the effectiveness and scalability of different algorithms and representation learning methods.

We also evaluate out-of-distribution (OOD) performance on 'Hard' and 'Easy' image distributions, comprised respectively of unseen images and the training images augmented with the methods described in Section 7.4. For Easy OOD, we run evaluations for each augmentation type individually across all 5 seeds for 100 episodes, average the success rate for each augmentation, and then report the average success rate across all augmentation types.

Algorithm implementation details. We explore three distinct algorithmic approaches: Soft Actor-Critic (SAC) (Haarnoja et al., 2018a), Proximal Policy Optimization (PPO) (Schulman et al., 2017), and DreamerV3 (Hafner et al., 2025), each representing different strategies for learning from visual observations.

For SAC, we begin with the standard implementation for discrete action spaces proposed by Christodoulou (2019). We then examine several representation learning variants: RAD (Laskin et al., 2020b), which employs data augmentation; CURL (Laskin et al., 2020a), which uses contrastive learning; SPR (Schwarzer et al., 2020), which incorporates self-supervised prediction; DBC (Zhang et al., 2021), which focuses on state metric learning; SAC-AE and SAC-VAE (Yarats et al., 2021b), which utilize reconstruction-based learning; and Simple Baseline (SB) (Tomar et al., 2023), which implements a simplified approach with reward and transition prediction.

For PPO, we evaluate three encoder configurations: the standard version with random initialization, a variant pretrained on the same image distribution (in-distribution, ID), and a variant pretrained on a different image distribution (out-of-distribution, OOD). These pretrained variants respectively provide an upper bound on expected pretraining performance and help establish the potential benefits

of generally pretrained encoders, though we acknowledge this may make direct sample efficiency comparisons with other methods less fair.

For DreamerV3, we compare the standard version against a variant without decoder gradients to evaluate the impact of the reconstruction objective on performance.

Hyperparameter tuning and configuration. RAD, CURL, and SPR require data augmentation. We apply these augmentations to observations after sampling them from the replay buffer. For each algorithm, we conducted individual augmentation searches with the objective of maximizing sample efficiency (detailed in Section 7.5.2). These experiments consistently converged to a simple two-step pipeline, which we use throughout all evaluations: first converting to grayscale with 20% probability, then randomly shuffling the color channels.

We evaluate out-of-the-box performance of existing approaches in SPGym by adopting neural architectures and hyperparameters from established visual discrete control implementations. Our base architecture uses three-layer CNN encoders with mirrored deconvolutional decoders, while actor, critic, and auxiliary components employ multi-layer perceptrons (MLPs).

For SAC-based agents, we follow Yarats et al. (2021b) and Tomar et al. (2023) by blocking actor gradients through the encoder while allowing critic and auxiliary gradients to prevent representation collapse. PPO and DreamerV3 maintain their original gradient flow patterns. While PPO and DreamerV3 worked robustly with default configurations, SAC-based agents required tuning of the temperature parameter α (Section 7.5.1). We found a fixed value of 0.05 to work best across all SAC variants, as the automatic tuning from Haarnoja et al. (2018b) proved ineffective here.

Where applicable, we preserve uniform hyperparameters while drawing algorithm-specific configurations from their respective source papers.

7.2 Model Architectures

We base our implementations on CleanRL's Atari agents for both PPO and SAC, with minor architectural modifications including additional normalization layers and increased network depth to approximate our SAC implementation to the one used by Tomar et al. (2023). The architectures are detailed below.

For PPO agents:

```
SharedEncoder(
    (encoder): Sequential(
        (0): Conv2d(3, 32, kernel_size=(8, 8), stride=(4, 4))
        (1): ReLU()
        (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, stats=True)
        (3): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
        (4): ReLU()
        (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, stats=True)
        (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
        (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, stats=True)
        (8): Flatten(start_dim=1, end_dim=-1)
    )
    (projection): Sequential(
        (0): Linear(in_features=3136, out_features=512, bias=True)
        (1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (2): Tanh()
    )
)
Actor(
    (encoder): SharedEncoder
    (mlp): Linear(in_features=512, out_features=4, bias=True)
)
```

```
Critic(
    (encoder): SharedEncoder
    (mlp): Linear(in_features=512, out_features=1, bias=True)
)
```

For SAC agents:

```
SharedEncoder(
    (encoder): Sequential(
        (0): Conv2d(3, 32, kernel_size=(8, 8), stride=(4, 4))
        (1): ReLU()
        (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, stats=True)
        (3): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
        (4): ReLU()
        (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, stats=True)
        (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
        (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, stats=True)
        (8): Flatten(start_dim=1, end_dim=-1)
    )
)
Actor(
    (encoder): Encoder(
        (shared_encoder): SharedEncoder
        (projection): Sequential(
            (0): Linear(in_features=3136, out_features=512, bias=True)
            (1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
            (2): Tanh()
        )
    )
    (mlp): Sequential(
        (0): Linear(in_features=512, out_features=512, bias=True)
        (1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (2): ReLU()
        (3): Linear(in_features=512, out_features=4, bias=True)
    )
)
Critic(
    (encoder): Encoder(
        (shared_encoder): SharedEncoder
        (projection): Sequential(
            (0): Linear(in_features=3136, out_features=512, bias=True)
            (1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
            (2): Tanh()
        )
    )
    (mlp): Sequential(
        (0): Linear(in_features=512, out_features=512, bias=True)
        (1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (2): ReLU()
        (3): Linear(in_features=512, out_features=4, bias=True)
    )
    (mlp): Sequential(
        (0): Linear(in_features=512, out_features=512, bias=True)
        (1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (2): ReLU()
        (3): Linear(in_features=512, out_features=4, bias=True)
    )
)
```

For the representation learning methods, we maintain the same base architecture and use a consistent MLP structure (2 layers with ReLU activation) for the projector, predictor, transition and reward models. The decoder architecture is as follows:

```
ImageDecoder(
    (decoder): Sequential(
        (0): Linear(in_features=512, out_features=512, bias=True)
        (1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (2): ReLU()
        (3): Linear(in_features=512, out_features=3136, bias=True)
        (4): Unflatten(dim=1, unflattened_size=(3, 7, 7))
        (5): ConvTranspose2d(3, 64, kernel_size=(3, 3), stride=(1, 1))
        (6): ReLU()
        (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, stats=True)
        (8): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2))
        (9): ReLU()
        (10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, stats=True)
        (11): ConvTranspose2d(32, 3, kernel_size=(8, 8), stride=(4, 4))
        (12): Sigmoid()
    )
)
```

For DreamerV3 agents, we use the same base 12M architecture as the one used by Hafner et al. (2025). Section 7.3 contains the specific hyperparameters used in our experiments.

7.3 Hyperparameters

Table 5, Table 6, and Table 7 list hyperparameters used across all experiments, unless noted otherwise. For DreamerV3, we adopted hyperparameters from (Hafner et al., 2025), modifying only the decoder loss scale (set to 0) for the version without decoder. Table 8 lists hyperparameters for representation learning methods and components. We use a separate optimizer for the representation learning gradient flow, and we adopt a higher learning rate. When using representation learning methods, we update the target network's encoder faster, with an EMA τ of 0.025. When using a crop augmentation, we set the image size to 100 and crop it back to 84.

Table 4: **Benchmark settings**

Parameter	Value
Max steps	10M
Puzzle size	3x3
Action space	discrete
Variation	image
Render size	100x100 (for crop augmentation) 84x84 (otherwise)
Dataset	ImageNet-1k validation split

Table 5: **Hyperparameters for PPO**

PPO Parameter	**Value**
Input image size	84x84
Env instances	64
Optimizer	Adam
Learning Rate (LR)	2.5e-4
LR annealing	yes
Adam ϵ	1e-5
Num. steps	16
Num. epochs	4
Batch size	64
Num. minibatches	4
γ	0.99
GAE λ	0.95
Advantage normalization	yes
Clip coef.	0.1
Clip value loss	yes
Value function coef.	0.5

Table 6: **Hyperparameters for SAC**

SAC Parameter	**Value**
Input image size	84x84
Env instances	64
Optimizer	Adam
Learning rate	3e-4
Replay buffer capacity	3e5
Batch size	4096
Warmup steps	2e4
γ	0.99
Policy update frequency	2
Fixed α temperature	0.05
Target network update frequency	1
Target Q functions EMA τ	0.005
Target encoder EMA τ	0.005 (standard) 0.025 (otherwise)

Table 7: **Hyperparameters for DreamerV3**

DreamerV3 Parameter	Value
Input image size	80x80
Env instances	16
Model size	12M
RSSM deterministic size	2048
RSSM hidden size	256
RSSM classes	16
Network depth	16
Network units	256
Replay buffer capacity	5e5
Replay ratio	32
Action repeat	1
Learning rate	4e-5
Batch size	16
Batch length	64
Imagination horizon	15
Discount horizon	333
Decoder loss scale	1 (standard) 0 (no decoder)

Table 8: **Hyperparameters for Representation Learning Methods and Components**

Method	Hyperparameter	Value
General	Learning rate	1e-3
	Loss coefficient	1.0
Transition and reward models	Min sigma	1e-4
	Max sigma	10
	Probabilistic	no
SAC-AE	Latent space decay weight	1e-6
	Decoder decay weight	1e-7
SAC-VAE	Variational KL weight β	1e-7
CURL	Temperature	0.1
	Positive samples	temporal/augmented
	Augmentations	crop
RAD	Augmentations	crop channel_shuffle color_jitter
SPR	Horizon H	5
	Augmentations	crop

7.4 Augmentation Strategies

We evaluated several image augmentations in our preliminary experiments, as described in Section 7.5.2. These augmentations are illustrated in Figure 5, and are as follows:

- **No augmentation**: The image is fed as is to the agent.
- **Crop**: Randomly crops a portion of the image and resizes it back to the original dimensions. This helps learn translation invariance by forcing the agent to recognize patterns regardless of their position in the frame.

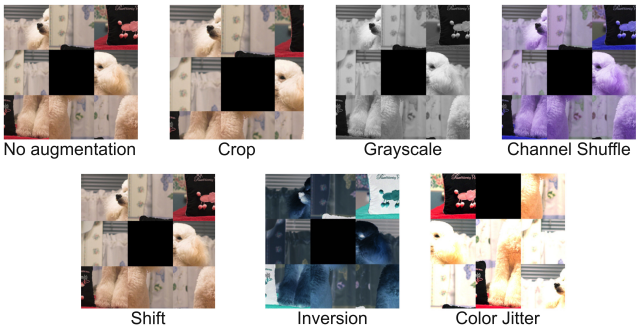

Figure 5: **Example of image augmentations.** The top left image shows the original observation, and the subsequent images show the observation after each augmentation procedure is independently applied.

- **Grayscale**: Converts the RGB image to grayscale by averaging across color channels. This reduces the visual complexity and helps the agent focus on structural features rather than color information.
- **Channel Shuffle**: Randomly permutes the RGB color channels. This encourages the agent to be invariant to color transformations while preserving the image structure.
- **Shift**: Translates the image by a small random amount in both horizontal and vertical directions. Similar to crop, this promotes translation invariance in the learned representations.
- **Inversion**: Inverts the pixel values by subtracting them from the maximum possible value (255 for 8-bit images). This teaches the agent to recognize patterns independent of absolute intensity values.
- **Color Jitter**: Applies random color variations to the image, including brightness, contrast, saturation, and hue. This helps the agent to be invariant to color transformations while preserving the image structure.

After extensive experimentation, we found that the combination of grayscale and channel shuffle consistently produced the best results. This combination effectively reduces visual complexity while maintaining important structural information. We adopted this augmentation pair as the standard for all our agents that use augmentation techniques.

7.5 Preliminary Experiments and Design Rationale

7.5.1 Hyperparameter Selection Process

For SAC agents, the entropy coefficient α significantly impacts performance (Haarnoja et al., 2018a). While Haarnoja et al. (2018b) proposed automatic tuning (autotune) based on policy entropy, we found this approach ineffective for SPGym, even with various learning rates (LRs). Through systematic Hyperband (Li et al., 2018) sweeps over pools of size 1, we identified $\alpha = 0.05$ as optimal (Figure 6). This value provides a good balance between exploration and exploitation, allowing the agent to efficiently learn the puzzle mechanics while maintaining enough randomness to discover new solutions.

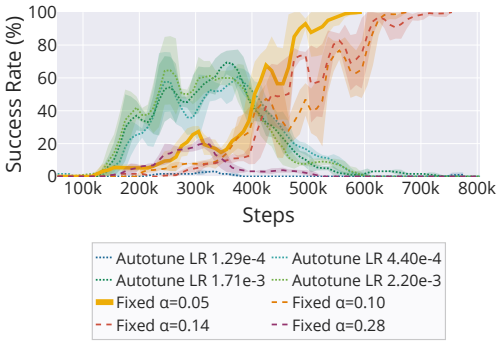

Figure 6: **Performance comparison of different α values for SAC agents on pool size 1.** Fixed $\alpha = 0.05$ outperforms automatic tuning approaches across different learning rates.

PPO and DreamerV3 agents proved more robust to hyperparameter choices, performing well with their default configurations. This robustness is particularly valuable in our benchmark setting, as it suggests these algorithms can adapt to new tasks without extensive tuning. Complete hyperparameter settings are provided in Section 7.3.

7.5.2 Data Augmentation Analysis and Choice

Building on insights from Laskin et al. (2020b) that environment-specific invariances influence optimal augmentation strategies, we systematically evaluated augmentation pipelines for RAD, CURL, and SPR in SPGym. Our analysis focused on sample efficiency to maintain consistency with our core experimental objectives. We tested the five augmentation techniques detailed on Section 7.4 on pools of 5 images, which offered a good balance between task complexity and convergence speed. For SPR, specifically, we also experimented with shift + color jitter, as suggested by Schwarzer et al. (2020). Across all algorithms, our experiments consistently converged to a simple two-stage augmentation process: probabilistic grayscale conversion (with 20% chance) followed by channel shuffling (Figure 7). This pipeline's effectiveness likely stems from its ability to simultaneously reduce visual complexity through grayscale conversion while introducing beneficial stochasticity via channel shuffling, with both transformations preserving critical structural information while preventing overfitting to specific color patterns. We adopted this augmentation combination for all subsequent experiments. While our current evaluation focused on sample efficiency, investigating how different augmentation strategies affect generalization remains an important direction for future work.

7.6 Dataset Analysis and Choice

Our choice of ImageNet-1k as the primary dataset for SPGym was motivated by several key considerations. First, ImageNet provides a diverse set of real-world images that challenge agents to learn generalizable visual representations. We hypothesized that real-world images would provide unique insights related to representation learning that should be applicable to other domains beyond the puzzle proposed in SPGym, as they contain the complex visual patterns and structures that agents encounter in practical applications. However, as shown in Figure 8 and in comparison to Figure 4, the performance scaling patterns we observe on ImageNet closely mirror those on DiffusionDB, suggesting that our findings are not specific to a particular dataset but rather reflect fundamental properties of the algorithms being tested.

The similarity in scaling behavior between ImageNet and DiffusionDB is particularly noteworthy because these datasets differ substantially in their composition and generation process. While ImageNet consists of real photographs, DiffusionDB contains synthetic images generated by text-to-

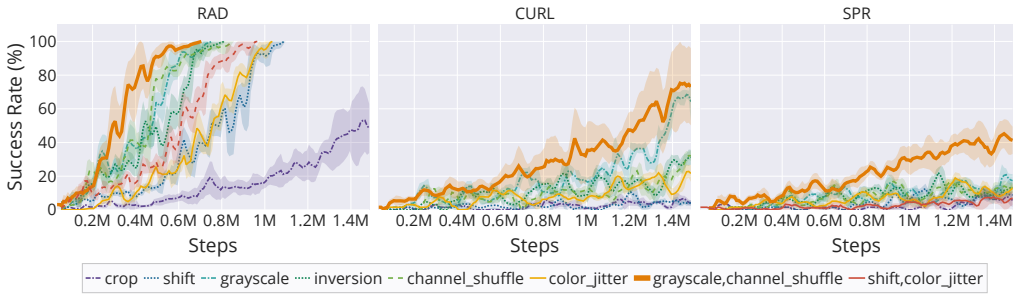

Figure 7: **Comparative analysis of data augmentation strategies.** Results show performance of SAC with RAD, CURL and SPR on 5-image pools. Grayscale conversion and channel shuffling emerge as the most effective combination, significantly outperforming other augmentation strategies.

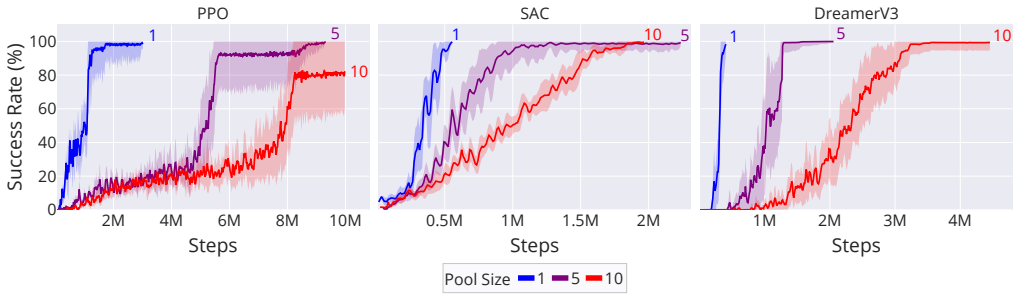

Figure 8: **Performance scaling with DiffusionDB images.** Success rates for PPO, SAC, and DreamerV3 agents across different pool sizes (1, 5, and 10) using DiffusionDB images. The performance patterns closely mirror those seen with ImageNet (Figure 4). Shaded regions represent 95% confidence intervals across 5 independent seeds.

image models. The consistent performance patterns across these datasets suggest that our results capture fundamental algorithmic behaviors rather than dataset-specific artifacts.

Our demonstration with DiffusionDB reveals promising directions for future work with procedurally generated datasets. As shown in our analysis, agents perform similarly on ImageNet and DiffusionDB, suggesting that visual diversity rather than semantic content drives difficulty. Procedurally generated images offer several compelling advantages worth further investigation: they eliminate the need for large image storage by generating unique images on demand for each episode, enable fine-grained control over the generalization challenge by gradually increasing visual differences between generated images, and provide virtually unlimited training data diversity. These capabilities could enable more systematic studies of visual generalization in reinforcement learning.

This cross-dataset consistency strengthens our confidence in the generalizability of our findings. It indicates that the relative performance of different representation learning methods is driven more by their core assumptions and architectural choices than by the specific characteristics of the training data. This is particularly important for our goal of understanding how different approaches handle increasing visual diversity, as it suggests our conclusions may extend to other domains beyond the specific datasets used in our experiments.

7.7 Hardware Setup and Runtime

Our hardware setup consists of an AMD Ryzen 7 3700X CPU, an NVIDIA RTX 3090 GPU, 64GB of RAM, and 128GB of swap space. Using this configuration, DreamerV3 experiments take approximately 20 hours per run, primarily because of the heavy use of swap space for replay buffers, which must store hundreds of thousands of images in memory. SAC takes between 2 (e.g. standard, RAD) to 11 hours (e.g. SPR, SB) depending on the representation learning components used. For SAC, the longer runtimes are due to the number of sequential inference steps required to train the agent and auxiliary networks. PPO experiments are significantly faster, with the longest runs completing in about 1 hour and 30 minutes.

8 Algorithms and Variations

We describe the algorithms and their variations used in this work in detail. Our experiments employ three main algorithms – PPO, SAC, and DreamerV3 – each with different representation learning approaches. We focus particularly on how these methods process and learn from visual observations, as this is crucial for performance in our benchmark.

8.1 Pretraining and PPO

Drawing inspiration from previous work on pretraining methods in RL (Higgins et al., 2017; Stooke et al., 2021; Schwarzer et al., 2021), we implement a pretraining approach for PPO that focuses on learning task-relevant visual representations. The pretraining process involves training a PPO agent to completion on a single environment instance, then extracting its CNN weights. These pretrained weights are then used to initialize new PPO agents, while all other network components (policy and value networks) start from random initialization. We evaluate two scenarios: in-distribution (ID), where new agents are trained on the same pool of images used during pretraining, and out-of-distribution (OOD), where a different image pool is sampled. The ID setting represents an upper bound on what pretraining can achieve with perfect visual alignment, while the OOD setting reflects the more realistic scenario of deploying pretrained encoders on novel visual inputs, similar to how general-purpose pretrained models would be used in practice.

8.2 Data Augmentation and RAD

Reinforcement Learning from Augmented Data (RAD) (Laskin et al., 2020b) represents a simple yet effective approach to visual representation learning in RL. The key insight is that applying data augmentation to observations can improve sample efficiency by exposing the agent to transformed versions of experienced states. This creates an implicit regularization effect that helps learn more robust representations.

In our implementation, we combine RAD with SAC and apply data augmentation consistently during both policy updates and value function learning. These augmentations are applied to observations sampled from the replay buffer before being processed by the encoder network. The augmentation pipeline consists of two key transformations identified through our preliminary experiments (Section 7.5.2): grayscale conversion and channel shuffling. Following the SPR approach (Schwarzer et al., 2020), we apply augmentations independently to each transition after sampling batches from the replay buffer, meaning that samples from the same episode may be augmented differently. We detail how each augmentation procedure is implemented in Section 7.4.

8.3 Contrastive Learning and CURL

Contrastive learning methods learn representations by maximizing similarity between different views of the same observation while minimizing similarity to other observations. In the context of RL, CURL (Laskin et al., 2020a) applies this principle by using data augmentation to create positive pairs, enabling agents to learn invariant representations. The method achieves this by applying random crops to observations and treating differently augmented views of the same observation as positive pairs.

The contrastive loss for a positive pair of observations (x, x^+) and a set of negative examples $\{x_i^-\}$ is formulated as:

$$\mathcal{L}_{CURL} = -\log \frac{\exp(f_\theta(x)^T f_\theta(x^+)/\alpha)}{\exp(f_\theta(x)^T f_\theta(x^+)/\alpha) + \sum_i \exp(f_\theta(x)^T f_\theta(x_i^-)/\alpha)}, \tag{2}$$

where α is a temperature parameter and f_θ is the encoder function.

In our implementation, we combine CURL with SAC and use the same augmentation strategy identified in Section 7.4 (grayscale conversion and channel shuffling) rather than the random crops from the original CURL paper. The encoder is trained jointly with the RL objective, allowing the representations to adapt to both the contrastive learning task and the control problem. Negative examples are drawn from other observations within the same batch, providing a computationally efficient way to obtain contrastive pairs without requiring additional memory storage.

8.4 State Metrics and DBC

Deep Bisimulation for Control (DBC) (Zhang et al., 2021) takes a different approach to representation learning by focusing on behavioral similarity between states rather than visual similarity. The key idea is to learn an encoder that maps states to a representation space where distances reflect how similarly states behave in terms of rewards and transitions, rather than how visually similar they appear.

Given pairs of observations (x_i, x_j), DBC trains an encoder f_θ to minimize:

$$J(\phi) = \left(\|\hat{z}_i - \hat{z}_j\|_1 - |r_i - r_j| - \gamma W_2(\hat{\mathcal{P}}(\cdot|\bar{z}_i, a_i), \hat{\mathcal{P}}(\cdot|\bar{z}_j, a_j)) \right)^2, \tag{3}$$

where $\hat{z}_i = f_\theta(x_i)$ represents the encoded state, $\bar{z}_i = sg(f_\theta(x_i))$ is the stop-gradient version of the encoding, and $\hat{\mathcal{P}}$ is a probabilistic transition model that predicts the next state distribution. The W_2 term represents the 2-Wasserstein distance between predicted transition distributions, which for Gaussian distributions has a closed-form solution (Zhang et al., 2021).

In our implementation, we combine DBC with SAC, jointly training the encoder with both the bisimulation objective and the RL objective. The transition model operates in latent space, predicting Gaussian distributions over next states. This approach helps the agent learn representations that capture behaviorally meaningful features while ignoring visual distractors that don't affect the game dynamics. Unlike methods that rely on data augmentation or reconstruction, DBC's focus on behavioral similarity makes it particularly suited for environments where visually different states might require similar actions.

8.5 Reconstruction-Based Methods and SAC-AE/VAE

Reconstruction-based methods learn representations by training an encoder-decoder architecture to compress and reconstruct observations. We evaluate two variants combined with SAC: SAC-AE using a deterministic autoencoder and SAC-VAE using a variational autoencoder (Yarats et al., 2021b).

For SAC-AE, given an observation x from the replay buffer, we train an encoder f_θ and decoder g_ϕ to minimize:

$$\mathcal{L}_{RAE} = \mathbb{E}_{x\sim\mathcal{D}} \left[\|x - g_\phi(f_\theta(x))\|^2 + \lambda_z \|f_\theta(x)\|_2^2 + \lambda_\phi \|\phi\|_2^2 \right], \tag{4}$$

where λ_z and λ_ϕ are regularization coefficients that help prevent representation collapse and overfitting respectively (Ghosh et al., 2020).

For SAC-VAE, we replace the deterministic encoder with a probabilistic encoder q_ψ that outputs a distribution over latent states. The training objective becomes:

$$\mathcal{L}_{VAE} = \mathbb{E}_{q_\psi(\hat{z}|x)}[\log g_\phi(x|\hat{z})] - \beta D_{KL}(q_\psi(\hat{z}|x)\|\mathcal{N}(0,1)), \tag{5}$$

where β controls the trade-off between reconstruction quality and latent space regularization.

In both variants, we train the encoder jointly with the SAC objective, allowing the representations to adapt to both reconstruction and control tasks. The encoded states are used as inputs to the policy and value networks. Unlike methods that rely on data augmentation or behavioral similarity, these approaches learn representations by explicitly modeling the visual structure of observations through reconstruction.

8.6 World Models and DreamerV3

World models learn to predict future states and outcomes by learning a compact latent representation of the environment. DreamerV3 (Hafner et al., 2025) represents the state-of-the-art in world model-based reinforcement learning, employing an online encoder f_θ that maps observations x_t into latent states $\hat{z}_t$. A recurrent dynamics model h_ω operates in this latent space to predict future states conditioned on actions, while a reward predictor estimates immediate rewards.

The model is trained using multiple objectives that create a multi-task learning pressure. The encoder and a corresponding decoder are trained to reconstruct observations, ensuring the latent space captures relevant visual features. The dynamics model is trained to predict future latent states that lead to accurate reconstructions of future observations. This temporal consistency objective forces the representations to be predictive of future states while supporting reconstruction and control.

DreamerV3 introduces several innovations for stable representation learning, including KL balancing to maintain informative latent states and symmetric cross-entropy loss for better gradients. Unlike methods focused solely on visual similarity, world models must learn representations that serve multiple purposes – capturing visual features, encoding dynamics, and providing a suitable space for policy learning. We refer readers to Hafner et al. (2025) for implementation details.

8.7 Temporal Consistency Methods

Several methods leverage temporal consistency in the environment to learn better representations. These approaches are based on the principle that a good representation should not only capture the current state but also be predictive of future states and outcomes.

8.7.1 Self-Predictive Representations (SPR)

SPR (Schwarzer et al., 2020) represents a non-contrastive approach that learns by predicting future latent states. Given a sequence of states and actions $(x_{t:t+K}, a_{t:t+K})$ from the replay buffer, where K is the prediction horizon, SPR employs:

- An online encoder f_θ that maps observations to latent states: $\hat{z}_t = f_\theta(x_t)$
- A target encoder $f_{\theta'}$ providing stable training targets, updated via exponential moving average
- An action-conditioned transition model h_ω that predicts future latent states: $\hat{z}_{t+k+1} = h_\omega(\hat{z}_{t+k}, a_{t+k})$
- Projection networks p_ξ, $p_{\xi'}$ and prediction head w_ζ that transform representations for the prediction task

The model generates predictions $\hat{y}_{t+k} = w_\zeta(p_\xi(\hat{z}_{t+k}))$ and compares them to target projections $\tilde{y}_{t+k} = p_{\xi'}(\tilde{z}_{t+k})$ using a cosine similarity loss:

$$\mathcal{L}_{\text{SPR}} = -\sum_{k=1}^{K} \left(\frac{\tilde{y}_{t+k}}{\|\tilde{y}_{t+k}\|_2} \right)^T \left(\frac{\hat{y}_{t+k}}{\|\hat{y}_{t+k}\|_2} \right) \tag{6}$$

8.7.2 Simple Baseline Method

Tomar et al. (2023) take a minimalist approach to temporal consistency by combining two predictive objectives: reward prediction and transition prediction. We refer to this approach as the Simple Baseline (SB) method. While originally intended as a baseline, it demonstrates the effectiveness of basic temporal prediction for representation learning.

The method augments standard RL algorithms with two predictive components in latent space:

- A transition model h_ω that predicts the next encoded state
- A reward predictor h_{reward} that estimates immediate rewards

The transition loss $\mathcal{L}_{\text{dyn}}$ measures the mean squared error between predicted and actual next encoded states, while the reward loss $\mathcal{L}_{\text{reward}}$ measures the error in reward predictions. These predictive losses are combined with the standard RL objective:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{RL}} + \mathcal{L}_{\text{reward}} + \mathcal{L}_{\text{dyn}} \tag{7}$$

In our implementations, we combine both SPR and SB with SAC. For SPR, we apply the augmentation strategy identified in Section 7.4 before encoding observations. Both methods operate entirely in latent space, avoiding the computational cost of pixel-space reconstruction while leveraging temporal structure to learn meaningful representations.

9 Supplementary Analyses and Results

9.1 Representation Learning Analysis

We further evaluate SPGym's representation learning assessment capabilities through two analyses: (1) comparing raw pixel vs. ground-truth state learning, and (2) linear probes of learned representations.

9.1.1 State-based vs. Image-based Observations

To establish a baseline and understand the impact of the visual representation challenge, we trained PPO, SAC, and DreamerV3 agents using SPGym's one-hot encoding variation. These one-hot vectors represent the ground-truth puzzle state, identical to the targets used for our linear probes (see Section 9.1.2). We compared their sample efficiency (steps to 80% success, averaged over 5 seeds) against their image-based counterparts. For PPO and SAC agents processing one-hot vectors, CNN encoders were replaced with 2-layer MLPs. DreamerV3 utilized its default non-image encoder, a 3-layer MLP. Hyperparameters were kept consistent with image-based experiments, without specific tuning for the one-hot setting.

Table 9: **Steps to 80% success on one-hot vs. image-based observations.** Lower is better. '-' indicates experiments were not run for that specific configuration due to computational constraints.

Algorithm	Grid Size	One-hot	Image (Pool 1)	Image (Pool 5)
PPO	3x3	661.69k±81.44k	1.75M±444.81k	7.80M±1.08M
	4x4	12.29M±467.84k	24.46M±7.58M	-
SAC	3x3	672.51k±63.10k	334.26k±67.47k	907.21k±116.20k
	4x4	5.09M±463.14k	8.14M±3.64M	-
DreamerV3	3x3	834.86k±61.10k	417.09k±55.03k	1.23M±199.49k
	4x4	3.68M±436.97k	2.26M±287.23k	5.81M ± 2.17M

The results in Table 9 offer several insights. For PPO (both grid sizes) and SAC (4x4 grid), learning directly from ground-truth one-hot states is more sample efficient than learning from images. This is expected, as the one-hot encoding removes the burden of representation learning from pixels. The instances where image-based agents (SAC and DreamerV3 on 3x3 grid, pool 1) converged faster than their one-hot counterparts might be attributed to differences in network architectures (MLP vs. CNN/Transformer backbones) and the absence of specific tuning for the one-hot setting.

Crucially, the one-hot encoding setting presents a fixed, minimal representation learning challenge. In contrast, SPGym's image-based variations allow for a systematic scaling of the visual diversity challenge by increasing the image pool size (e.g., Pool 1 vs. Pool 5 vs. Pool 10, etc., see Section 9.2). Across all agents and grid sizes, increasing visual diversity from pool size 1 to pool size 5 (and beyond) consistently increases sample complexity. This demonstrates SPGym's ability to isolate and stress the visual representation learning component, as the underlying task dynamics remain constant. This controlled evaluation reveals limitations in how effectively different RL agents learn representations under scalable visual diversity, insights not apparent from the one-hot setting alone. While perfect disentanglement of representation learning from policy learning is challenging in end-to-end training, SPGym provides a valuable framework for structured, comparative evaluation of visual representation learning capabilities in RL.

9.1.2 Linear Probe Analysis

To directly assess the quality of learned visual representations, we conducted linear probing experiments. For each trained PPO and SAC agent, we froze its encoder and trained a single-layer MLP classifier on top of the features extracted by it until convergence. The classifier's task was to predict the one-hot encoded ground-truth puzzle state corresponding to the input image. This setup allows us to quantify how much task-relevant spatial information is captured by the agent's encoder. High probe accuracy indicates the encoder has learned features that are linearly separable with respect to the underlying game state.

Our analysis reveals several key insights. First, we find a strong, statistically significant correlation between linear probe accuracy and agent sample efficiency (Pearson r=-0.81, p=1.1e-13), with higher probe accuracy being highly predictive of fewer environment steps required to reach 80% success. This suggests that agents whose encoders capture more task-relevant spatial information tend to learn the task more efficiently.

Second, examining probe accuracies across pool sizes and algorithms further clarifies this relationship. As image pool size increases, both probe accuracy and task performance systematically degrade, isolating the effect of visual diversity on representation learning. For example, standard SAC maintains high probe accuracy (100% at pool size 1, 97.63% at pool size 5), mirroring its strong sample efficiency. In contrast, methods with lower sample efficiency show reduced probe performance: SAC+VAE achieves only 78.21% probe accuracy at pool size 5, while SAC+RAD reaches 98.66%. Other representation learning methods that underperform standard SAC, such as SPR and DBC, also exhibit declining probe accuracy as pool size increases (e.g., SPR drops from 94.31% at pool size 5 to 75.48% at pool size 10).

These trends indicate that different algorithms develop representations with varying alignment to the spatial reasoning demands of the task. The consistent link between probe accuracy and sample efficiency across diverse methods suggests that SPGym can help identify which learning procedures lead to representations that better support task performance.

Full linear probe accuracy data for all agents and pool sizes are presented in Table 10 below. Comprehensive performance metrics, including the sample efficiency data used in the correlation analysis, can be found in Section 9.2.

9.2 Detailed Performance Analysis

This section provides comprehensive analysis of algorithmic performance in SPGym, examining how different representation learning approaches handle increasing visual diversity. We present detailed learning curves showing the training dynamics of each method, along with quantitative performance metrics across varying pool sizes.

The analysis includes detailed algorithmic variant analysis explaining why certain methods succeed or fail, and comprehensive performance curves for PPO (Figure 10), SAC (Figure 11), and DreamerV3 (Figure 12). Tables 11 to 13 provide quantitative metrics per pool size (Pool) including sample efficiency (Steps), and episode length (Length), across algorithms and variants. Tables 11 and 12 also include in-distribution and out-of-distribution success rates (ID Success, OOD Easy, OOD Hard), which were not computed for DreamerV3 due to complexities in the original codebase. The rare non-zero success rates in Hard OOD likely stem from chance encounters with nearly-solved initial states rather than genuine generalization.

9.2.1 Analysis of Algorithmic Variants

Our preliminary analysis of algorithmic variants suggests how method assumptions may influence effectiveness in SPGym. PPO with in-distribution pretraining (PT (ID)) significantly boosts sample efficiency across all pool sizes, though this advantage diminishes at pool size 10. Out-of-distribution pretraining (PT (OOD)) offers more modest gains that also decrease with larger pools, suggest-

Table 10: **Linear probe accuracy (%) for each agent and pool size.** Accuracy of the linear probe indicates linearly separable features capturing task-relevant spatial information. There is a statistically significant correlation between probe accuracy and agent sample efficiency (steps to 80% success).

Agent	Pool Size	Steps (M)	Linear Probe Accuracy (%)
PPO	1	1.75±0.44	99.81±0.14
	5	7.80±1.08	96.42±1.06
	10	9.73±0.36	87.84±1.40
	20	10.00±0.00	66.97±5.28
PPO+PT(ID)	1	0.95±0.21	99.81±0.12
	5	5.55±1.22	96.83±0.61
	10	9.17±1.10	89.54±0.95
PPO+PT(OOD)	1	1.34±0.42	99.59±0.33
	5	7.03±1.07	95.68±0.77
	10	9.70±0.41	88.90±1.11
SAC	1	0.33±0.07	100.00±0.00
	5	0.91±0.12	97.63±0.68
	10	1.65±0.31	93.34±0.48
	20	4.52±1.43	80.74±6.31
	30	9.23±0.96	66.69±8.41
	50	10.00±0.00	55.52±0.09
SAC+RAD	1	0.24±0.03	99.99±0.01
	5	0.42±0.06	98.66±0.20
	10	0.82±0.18	89.74±0.73
SAC+CURL	1	0.46±0.10	99.98±0.03
	5	1.56±0.31	97.14±0.17
	10	5.24±1.92	89.47±1.23
SAC+SPR	1	2.09±0.81	99.99±0.01
	5	3.68±1.68	94.31±0.24
	10	10.00±0.00	75.48±1.82
SAC+DBC	1	0.44±0.04	100.00±0.00
	5	0.99±0.25	94.26±1.19
	10	10.00±0.00	76.59±5.82
SAC+AE	1	0.42±0.09	100.00±0.00
	5	1.04±0.24	95.52±4.56
	10	2.03±0.38	88.66±1.88
SAC+VAE	1	1.13±0.14	99.66±0.06
	5	5.30±0.68	78.21±2.35
	10	10.00±0.00	64.76±0.11
SAC+SB	1	0.98±0.88	99.90±0.03
	5	2.08±0.30	96.69±1.08
	10	10.00±0.00	81.93±6.06

ing limited transfer from general-purpose pretrained encoders. For SAC, data augmentation via RAD consistently improves efficiency across all pool sizes, with particularly pronounced benefits for larger pools. Conversely, many sophisticated auxiliary methods struggle: CURL, SPR, and VAE variants consistently require more samples than standard SAC, with particularly poor performance on larger pools. DBC and AE generally underperform or offer marginal improvements. DreamerV3 demonstrates particularly strong performance, consistently outperforming both PPO and SAC variants across all pool sizes with remarkably stable performance. The variant without decoder gradients shows reduced performance, highlighting the importance of the reconstruction objective and suggesting that learning a predictive environment model provides an effective foundation for handling visual diversity. We now provide more detailed hypotheses for these behaviors.

Pretraining. As shown in Figure 10, pretraining provides clear benefits for PPO, especially with larger pools. While in-distribution pretraining provides strong gains, almost matching the performance of PPO with one-hot-based observations (see Section 9.1.1), this represents an optimistic upper bound since real-world scenarios rarely permit task-specific pretraining. Interestingly, out-of-distribution pretraining also shows benefits compared to random initialization (90% vs 86% success at pool size 5), suggesting some transfer of useful visual features from the pretrained encoder. This indicates that even general-purpose pretrained encoders can provide a helpful initialization for RL tasks, though not matching the performance of task-specific pretraining.

Data augmentation. RAD succeeds by enforcing spatial invariances through grayscale+channel shuffling, preserving structural relationships critical for puzzle solving while adding beneficial stochasticity. Its weak assumptions make it robust across diversity levels, maintaining strong and consistent performance across pool sizes through this simple augmentation-based approach.

Contrastive learning. CURL underperforms as instance discrimination may prioritize whole-image features over tile-level details needed for puzzle solving. This suggests contrastive learning's focus on global image similarity may not align well with the local spatial reasoning required for puzzle solving.

State similarity learning. DBC fails possibly because its core assumption – that states with similar dynamics should have similar representations – breaks down in two ways: identical puzzle states appear radically distinct between episodes with different sampled images, while different states can share visual patterns due to being from the same episode or having the same base image.

Temporal consistency. While the environment's underlying dynamics are deterministic, temporal consistency methods such as SPR and SB face three key challenges: (1) The visual manifestation of state transitions varies dramatically between episodes due to different base images, forcing the encoder to learn position-invariant representations that capture tile relationships rather than visual content – a difficult disentanglement problem. (2) The assumption of smooth latent space transitions is violated by the discrete nature of tile movements, where single actions induce significant changes in both visual appearance and puzzle state. (3) Most crucially, these methods must simultaneously learn two competing objectives: temporal predictability in latent space (for transition modeling) and visual discriminability (for representation learning). This creates a conflict where features useful for predicting latent transitions (tile positions) are obscured by visually salient but dynamically irrelevant image content. SPR's prediction horizon mechanism exacerbates this by compounding representation errors through multiple latent transition steps. Similarly, SB's transition/reward prediction suffers because the latent space conflates visual features with positional information – while rewards depend solely on tile positions, the visual diversity in observations provides no direct positional cues. The same absolute position (e.g., top-left corner) shows completely different visual content each episode, making position-aware latent representations particularly difficult to learn.

Reconstruction-based learning. The success of DreamerV3's decoder highlights the value of the discrete reconstruction loss in learning useful representations for SPGym. In contrast, simple autoencoders (AE) offer little benefit for SAC, and variational autoencoders (VAE) hurt performance possibly because their continuous latent space assumptions conflict with SPGym's discrete state transitions.

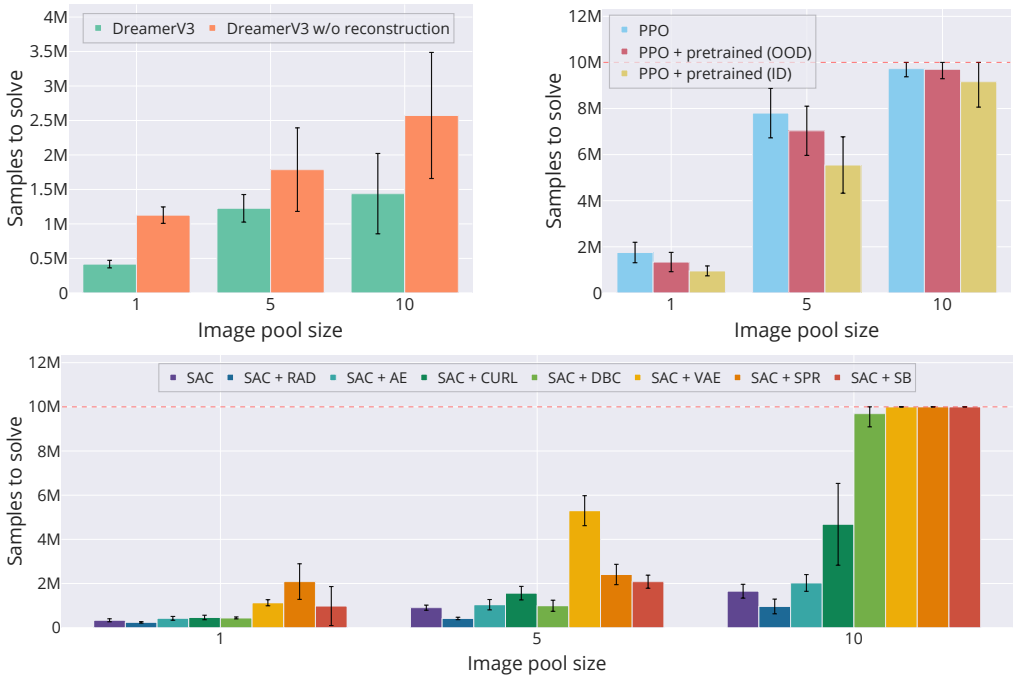

Figure 9: **Sample efficiency of different methods across pool sizes (lower is better).** SPGym differentiates agents based on their representation learning capabilities. Top left: DreamerV3 variants demonstrate the value of reconstruction learning. Top right: PPO results show benefits of pretraining. Bottom: Comprehensive comparison of SAC variants reveals trade-offs between different representation learning approaches.

These findings align with observations from Tomar et al. (2023), who noted that many representation learning methods underperform or fail completely when tested outside their original domain. We note that our evaluation focused on using each algorithm's suggested hyperparameters for visual RL with discrete actions, aiming to assess their out-of-the-box performance. The results may not reflect the best possible performance achievable through extensive hyperparameter tuning.

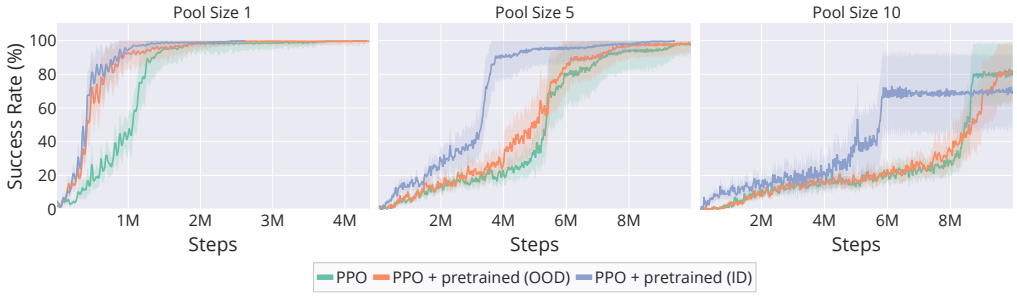

Figure 10: **Learning curves for PPO variants.** Success rate during training for baseline PPO and versions with pretrained encoders across different pool sizes. Shaded regions represent 95% confidence intervals across 5 independent runs.

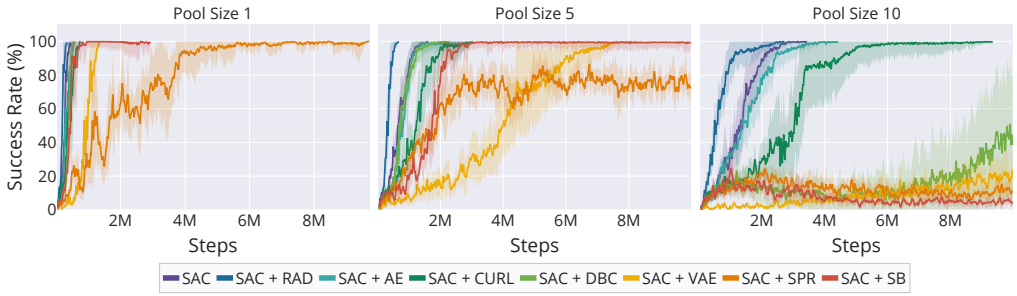

Figure 11: **Learning curves for SAC variants.** Success rate during training for baseline SAC and versions with different representation learning components across different pool sizes. Shaded regions represent 95% confidence intervals across 5 independent runs.

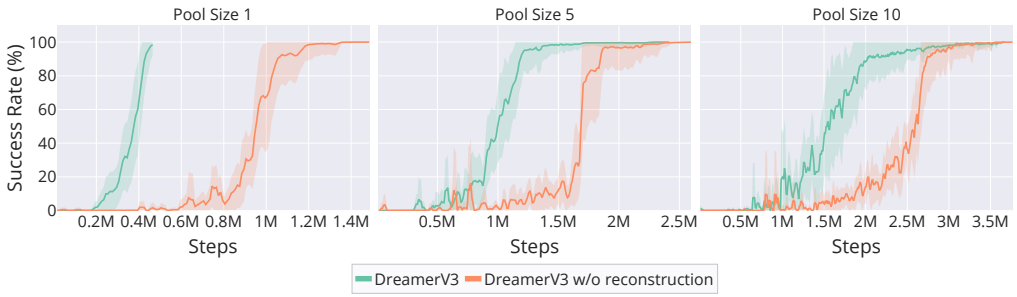

Figure 12: **Learning curves for DreamerV3 variants.** Success rate during training for DreamerV3 with and without decoder across different pool sizes. Shaded regions represent 95% confidence intervals across 5 independent runs.

Table 11: **Performance metrics for PPO agents across image pool sizes.** 'Steps (M)' indicates the number of environment steps (in millions) required to reach 80% success rate during training. 'ID (%)' shows the final success rate at the end of 10M training steps when evaluating on the same pool used during training. 'OOD-E (%)' shows the success rate when evaluated on 100 unseen images from the 'Easy' distribution. 'OOD-H (%)' shows the success rate when evaluated on 100 unseen images from the 'Hard' distribution. 'Length' indicates the average number of steps required to solve an instance of the puzzle at the last 100 episodes.

Algorithm	**Pool**	**Steps (M)**	**ID (%)**	**OOD-E (%)**	**OOD-H (%)**	**Length**
PPO	1	1.75±0.44	100±0	0.49±0.13	0.0±0.0	72.9±4.4
	5	7.80±1.08	86±16	0.53±0.14	0.3±0.3	212.0±19.0
	10	9.73±0.36	47±18	0.34±0.08	0.6±0.4	599.6±25.6
	20	10.00±0.00	12±7	0.12±0.03	0.0±0.0	903.4±23.9
PPO PT (ID)	1	0.95±0.21	100±0	0.33±0.09	0.1±0.1	76.2±4.8
	5	5.55±1.22	100±0	0.53±0.16	0.3±0.4	83.2±7.2
	10	9.17±1.10	38±20	0.27±0.07	0.9±0.7	658.4±25.3
PPO PT (OOD)	1	1.34±0.42	100±0	0.49±0.12	0.2±0.3	72.7±4.0
	5	7.03±1.07	90±16	0.52±0.14	0.3±0.3	156.0±16.0
	10	9.70±0.41	46±19	0.34±0.08	0.8±1.2	572.0±26.0

Table 12: **Performance metrics for SAC agents across image pool sizes.** 'Steps (M)' indicates the number of environment steps (in millions) required to reach 80% success rate during training. 'ID (%)' shows the final success rate at the end of 10M training steps when evaluating on the same pool used during training. 'OOD-E (%)' shows the success rate when evaluated on 100 unseen images from the 'Easy' distribution. 'OOD-H (%)' shows the success rate when evaluated on 100 unseen images from the 'Hard' distribution. 'Length' indicates the average number of steps required to solve an instance of the puzzle at the last 100 episodes.

Algorithm	Pool	Steps (M)	ID (%)	OOD-E (%)	OOD-H (%)	Length
SAC	1	0.33±0.07	100±0	0.45±0.12	0.0±0.0	86.7±19.5
	5	0.91±0.12	100±0	0.58±0.12	0.0±0.0	76.8±16.2
	10	1.65±0.31	100±0	0.46±0.12	0.0±0.0	78.7±16.5
	20	4.52±1.43	98±2	0.35±0.11	0.0±0.0	63.2±18.4
	30	9.23±0.96	47±38	0.19±0.04	0.0±0.0	525.0±63.1
	50	10.00±0.00	7±5	0.06±0.02	0.0±0.0	917.0±35.5
SAC RAD	1	0.24±0.03	100±0	0.62±0.15	0.0±0.0	50.9±6.8
	5	0.42±0.06	100±0	0.42±0.13	0.0±0.0	76.9±12.0
	10	0.82±0.18	100±0	0.30±0.11	0.0±0.0	144.1±25.4
SAC CURL	1	0.46±0.10	100±0	0.76±0.09	0.0±0.0	77.2±13.1
	5	1.56±0.31	100±0	0.44±0.10	0.2±0.4	84.9±17.7
	10	5.24±1.92	100±0	0.37±0.11	0.0±0.0	88.0±20.3
SAC SPR	1	2.09±0.81	100±0	0.65±0.13	0.6±1.2	206.4±23.8
	5	3.68±1.68	69±13	0.21±0.09	0.0±0.0	523.9±55.6
	10	10.00±0.00	9±3	0.07±0.04	0.0±0.0	912.1±36.0
SAC DBC	1	0.44±0.04	100±0	0.44±0.13	0.0±0.0	65.1±13.8
	5	0.99±0.25	100±0	0.34±0.13	0.0±0.0	111.2±22.7
	10	10.00±0.00	2±1	0.13±0.04	0.0±0.0	588.5±63.3
SAC AE	1	0.42±0.09	100±0	0.78±0.11	0.0±0.0	85.4±22.1
	5	1.04±0.24	100±0	0.64±0.16	0.0±0.0	102.6±22.6
	10	2.03±0.38	100±0	0.55±0.12	1.3±2.6	78.0±17.1
SAC VAE	1	1.13±0.14	100±0	0.64±0.15	0.0±0.0	75.1±15.6
	5	5.30±0.68	100±0	0.30±0.08	0.0±0.0	81.2±18.2
	10	10.00±0.00	25±17	0.12±0.03	0.4±0.5	834.3±47.5
SAC SB	1	0.98±0.88	100±0	0.89±0.08	0.0±0.0	130.1±25.0
	5	2.08±0.30	91±17	0.65±0.12	0.0±0.0	117.8±20.5
	10	10.00±0.00	3±2	0.06±0.02	0.2±0.4	980.3±18.0

Table 13: **Performance metrics for DreamerV3 agents across image pool sizes.** 'Steps' indicates the number of environment steps (in millions) required to reach 80% success rate during training. 'ID Success' shows the final success rate at the end of 10M training steps when evaluating on the same pool used during training. 'Length' indicates the average number of steps required to solve an instance of the puzzle at the last 100 episodes.

Algorithm	**Pool**	**Steps (M)**	**ID Success (%)**	**Length**
DreamerV3	1	0.42±0.06	100±0	83.5±11.8
	5	1.23±0.20	100±0	31.0±3.9
	10	1.44±0.58	100±0	32.9±4.0
	20	3.96±0.61	100±0	27.8±3.5
	30	5.84±0.71	99±1	38.0±6.2
	50	6.62±2.67	87±14	177.8±24.8
	100	8.18±2.33	29±14	676.1±32.8
DreamerV3 w/o decoder	1	1.13±0.12	100±0	36.0±2.5
	5	1.79±0.61	100±0	42.1±3.2
	10	2.57±0.91	100±0	46.7±3.9