

Reinforcement Learning for 2D Shooter Game: Progress Report with Preliminary Results

CENG7822 Project

December 25, 2025

1 Introduction

This progress report presents our ongoing work on training reinforcement learning agents for a custom 2D top-down shooter game environment. The core objective of this project is to develop autonomous agents capable of navigating a dynamic game world, collecting prizes for positive rewards, avoiding or eliminating enemies, and surviving for extended periods. We investigate multiple model-free reinforcement learning algorithms including Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC), evaluating their performance across different reward shaping strategies and training durations. Additionally, we have implemented a world model approach using Variational Autoencoders (VAE) for latent space representation learning and planning.

Our experimental design encompasses 27 distinct configurations, systematically varying the algorithm choice, reward shaping parameters, and total training timesteps. This comprehensive experimental matrix allows us to draw meaningful conclusions about the relative effectiveness of different approaches and the importance of reward engineering in shaping agent behavior. The preliminary results indicate that reward shaping plays a critical role in determining final performance, with aggressive reward configurations that emphasize positive feedback consistently outperforming more conservative alternatives.

2 Environment Description

The shooter environment is a gymnasium-compliant 2D simulation where an agent must navigate, collect prizes, and avoid or destroy enemy entities. The environment presents several challenging aspects that make it suitable for RL research: continuous state dynamics, multi-objective optimization (survival vs. prize collection vs. enemy elimination), and the need for both reactive and strategic decision-making.

2.1 State Space Representation

The observation space is a normalized vector containing complete information about the game state from the agent's perspective. The agent receives its own position and velocity encoded as four continuous values normalized to the range $[0, 1]$. Health and shooting cooldown provide two additional state dimensions that inform resource management decisions. For each of the k nearest enemies (typically $k = 3$), the agent observes relative position and velocity components, contributing $4k$ dimensions. Similarly, the m nearest prizes are represented by their relative positions, adding $2m$ dimensions. This vector-based representation provides sufficient information for decision-making while maintaining computational efficiency.

2.2 Action Space Design

The action space follows a multi-discrete structure with three independent components that the agent selects simultaneously at each timestep. The movement component offers five options: staying stationary or moving in one of the four cardinal directions. The shooting component is binary, determining whether the agent fires a projectile. The direction component specifies the aim direction from eight equally-spaced angles around the agent.

This multi-discrete formulation creates 80 unique action combinations ($5 \times 2 \times 8 = 80$). Different RL algorithms handle this complexity differently: PPO can work directly with multi-discrete actions, DQN requires flattening to a single discrete space of 80 actions, and SAC requires a continuous action wrapper that maps box-constrained continuous values back to discrete choices.

2.3 Reward Shaping Configurations

Reward shaping is a powerful technique for guiding agent learning, and our experiments systematically explore three distinct philosophies encoded as reward configurations. Each configuration represents a different balance between encouraging positive achievements and penalizing negative outcomes.

Table 1: Reward Configuration Parameters. Positive values encourage the corresponding behavior, while negative values (shown as penalties) discourage them.

Reward Component	Baseline	Survival	Aggressive
Prize Collection (R_{prize})	+1.0	+0.5	+2.0
Enemy Hit (R_{hit})	+0.3	+0.1	+0.5
Enemy Kill (R_{kill})	+1.0	+0.5	+2.0
Damage Taken (R_{damage})	-1.0	-3.0	-0.5
Shot Fired (R_{shot})	-0.02	-0.05	-0.01
Time Step (R_{time})	-0.001	-0.0005	-0.002
Death Penalty (R_{death})	-5.0	-10.0	-3.0

The **baseline** configuration represents a balanced approach with moderate rewards and penalties. The **survival** configuration heavily penalizes damage and death while offering smaller positive rewards, intended to train risk-averse agents that prioritize staying alive. The **aggressive** configuration maximizes positive rewards while minimizing penalties, encouraging agents to actively engage with the environment even at the cost of taking damage.

3 Experimental Setup

3.1 Algorithm Selection and Implementation

We evaluate three state-of-the-art reinforcement learning algorithms that represent different paradigms in the field. Our implementations leverage the Stable-Baselines3 library with custom wrappers for action space compatibility.

Deep Q-Network (DQN) is an off-policy value-based method that maintains a replay buffer of past experiences and trains a neural network to estimate action values. DQN uses epsilon-greedy exploration, starting with high randomness and gradually transitioning to predominantly greedy action selection. For our multi-discrete action space, we flatten all 80 combinations into a single discrete space. DQN trains efficiently at approximately 800 frames per second on CPU.

Proximal Policy Optimization (PPO) is an on-policy policy gradient method that directly optimizes the policy while preventing destructively large updates through a clipped sur-

rogate objective. PPO naturally supports multi-discrete action spaces and demonstrates stable learning dynamics. We use 4 parallel environments for data collection, achieving approximately 1500 frames per second aggregate throughput.

Soft Actor-Critic (SAC) is an off-policy actor-critic method that maximizes both reward and entropy, encouraging exploration through its maximum entropy framework. SAC requires continuous action spaces, so we wrap the environment to accept box-constrained continuous actions that are then discretized. Due to its sample-efficient but computationally intensive nature, SAC trains at approximately 65 frames per second, significantly slower than the alternatives.

3.2 Training Duration Configurations

To understand how learning progresses over time and whether additional training yields proportional improvements, we evaluate three training duration settings. The **short** configuration runs for 50,000 timesteps, providing a quick assessment of initial learning dynamics. The **medium** configuration extends to 500,000 timesteps, allowing substantial policy refinement. The **long** configuration runs for 1,600,000 timesteps, pushing toward asymptotic performance.

4 Experimental Results

4.1 Learning Curve Analysis

The learning curves reveal distinct patterns for each algorithm and reward configuration combination. Figure 1 shows DQN’s progression across the three reward configurations, where we observe that longer training consistently improves performance and the aggressive configuration achieves substantially higher rewards.

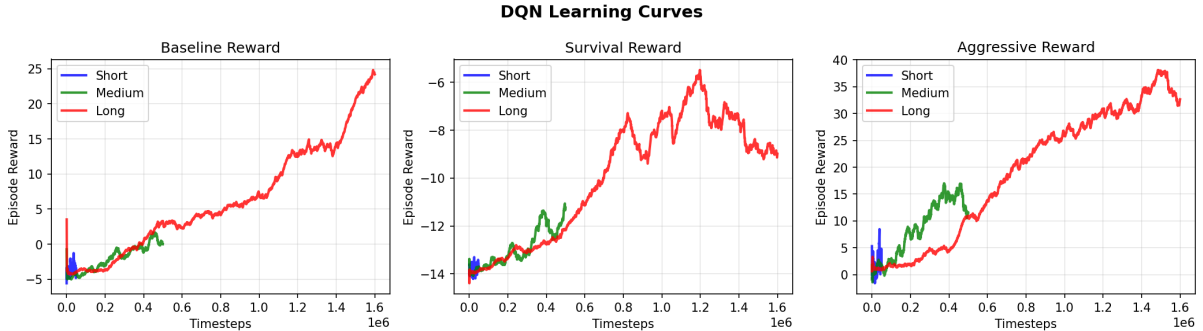


Figure 1: DQN Learning Curves. The three panels show performance under baseline, survival, and aggressive reward configurations. Each curve represents a different training duration (short: 50k, medium: 500k, long: 1.6M timesteps).

PPO demonstrates the most stable learning behavior among the three algorithms, as shown in Figure 2. The policy gradient updates produce smooth reward improvements without the oscillations sometimes observed in value-based methods. PPO with aggressive rewards and long training achieves our best observed performance of +38.74 mean reward.

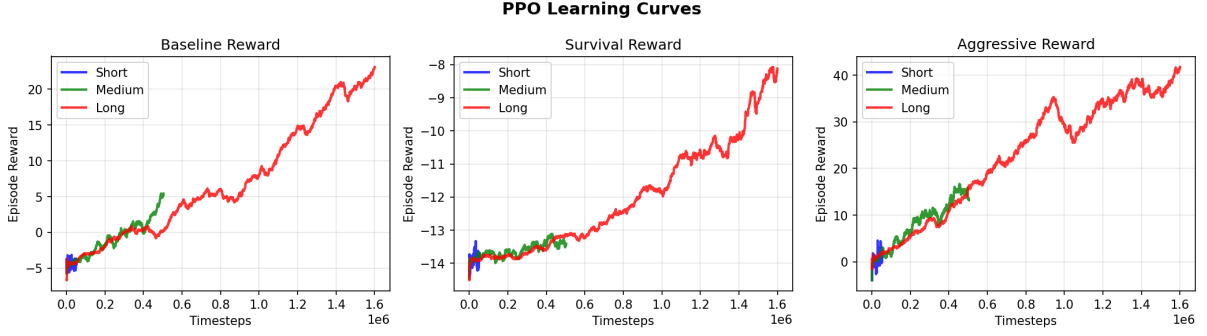


Figure 2: PPO Learning Curves. PPO exhibits smooth, monotonic improvement across all configurations. The aggressive reward configuration with 1.6M timesteps achieves the highest performance.

SAC’s learning curves in Figure 3 show more modest improvements compared to DQN and PPO. The entropy regularization may interfere with exploitation in our relatively simple environment, and the continuous-to-discrete action wrapper introduces additional complexity.

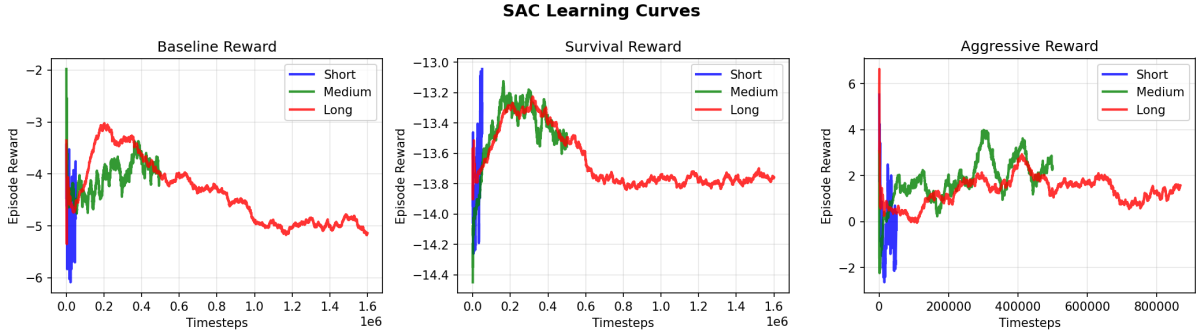


Figure 3: SAC Learning Curves. SAC shows slower improvement rates compared to other algorithms, with the aggressive configuration still outperforming alternatives.

4.2 Algorithm Comparison

Figure 4 directly compares the three algorithms on each reward configuration using the medium (500k) timestep setting. This controlled comparison reveals that DQN and PPO perform comparably on baseline and aggressive configurations, while both significantly outperform SAC.

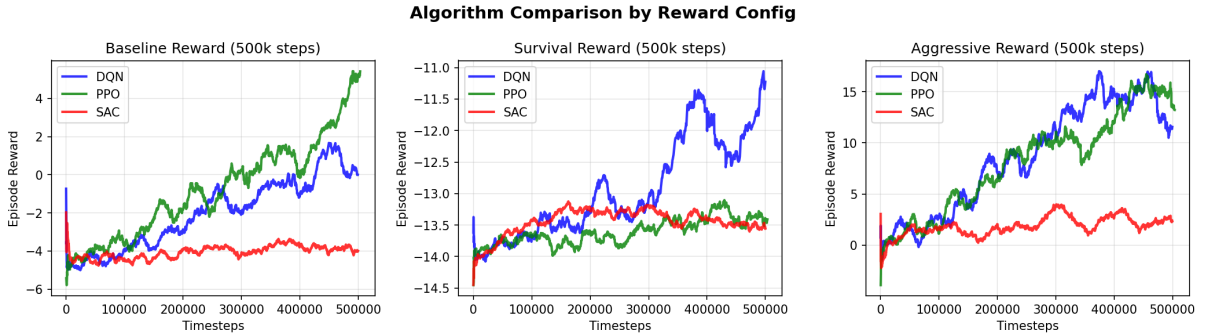


Figure 4: Algorithm Comparison at 500k timesteps. DQN (blue) and PPO (green) achieve similar performance across configurations, with both outperforming SAC (red).

4.3 Summary Statistics

Table 2 presents the complete numerical results for all 27 experimental configurations. The mean reward is computed over the final 10% of training episodes to represent converged performance.

Table 2: Final Performance Across All Configurations (Mean \pm Std over final 10% of episodes)

Algorithm	Reward	Steps	Episodes	Mean Reward
DQN	Aggressive	Long	3,328	$+33.95 \pm 28.12$
DQN	Aggressive	Medium	1,390	$+13.32 \pm 13.31$
DQN	Baseline	Long	2,997	$+19.42 \pm 17.84$
DQN	Baseline	Medium	1,537	$+0.70 \pm 5.75$
DQN	Survival	Long	2,714	-8.77 ± 6.09
DQN	Survival	Medium	1,295	-11.68 ± 3.52
PPO	Aggressive	Long	3,065	$+38.74 \pm 30.61$
PPO	Aggressive	Medium	1,535	$+14.37 \pm 15.12$
PPO	Baseline	Long	2,952	$+21.96 \pm 16.92$
PPO	Baseline	Medium	1,468	$+4.08 \pm 8.19$
PPO	Survival	Long	3,225	-8.80 ± 6.61
PPO	Survival	Medium	1,506	-13.39 ± 1.29
SAC	Aggressive	Long	3,978	$+1.49 \pm 4.96$
SAC	Aggressive	Medium	2,161	$+2.23 \pm 5.71$
SAC	Baseline	Long	7,714	-5.02 ± 1.75
SAC	Baseline	Medium	2,246	-3.90 ± 2.83
SAC	Survival	Long	6,906	-13.73 ± 0.81
SAC	Survival	Medium	2,229	-13.54 ± 0.94

5 Ablation Study: Effect of Reward Shaping

Our ablation study reveals that reward shaping is the most influential factor in determining final agent performance. Figure 5 presents a grouped bar chart comparing all algorithms across the three reward configurations at 500k timesteps.

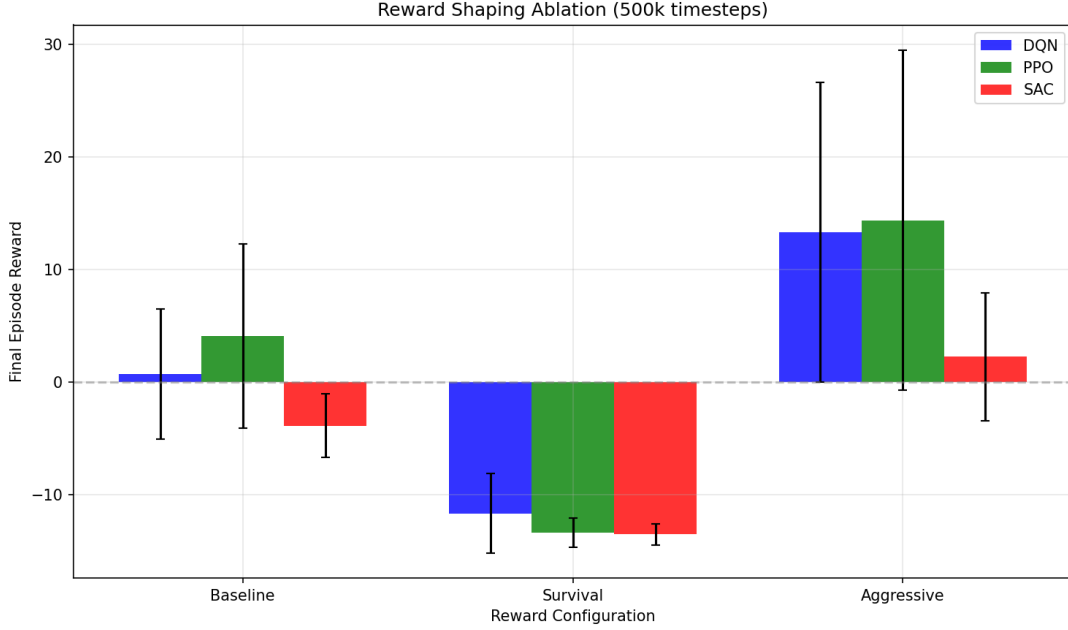


Figure 5: Reward Shaping Ablation Study. The aggressive configuration (right) consistently achieves the highest performance across all algorithms.

5.1 Why Aggressive Rewards Work Best

The aggressive reward configuration’s success can be attributed to several interconnected factors. First, the higher positive rewards for prize collection (+2.0 vs +1.0 baseline) and enemy kills (+2.0 vs +1.0) create stronger gradient signals that more effectively guide policy updates. The agent receives clear reinforcement for desirable behaviors, accelerating learning convergence.

Second, the reduced death penalty (−3.0 vs −5.0 baseline) allows agents to learn from risky exploratory behaviors without catastrophically negative feedback. This encourages the agent to attempt challenging maneuvers that might occasionally result in death but ultimately lead to discovering more effective strategies.

Third, the higher time penalty (−0.002 vs −0.001) discourages passive play, pushing agents toward active engagement with the environment rather than simply waiting and avoiding risk.

5.2 Why Survival Rewards Struggle

The survival configuration’s poor performance across all algorithms highlights the limitations of penalty-dominated reward functions. The heavy death penalty (−10.0) creates a sparse reward landscape where avoiding death becomes the primary objective, but the path to survival offers little positive reinforcement. The reduced positive rewards (+0.5 for prizes and kills) fail to adequately guide learning toward constructive behaviors.

Furthermore, the strong damage penalty (−3.0) makes any interaction with enemies extremely costly, training agents to avoid engagement entirely. While this produces agents that survive longer, they struggle to collect prizes or eliminate threats, resulting in persistent negative rewards from time penalties and eventual death.

6 World Model Implementation

Beyond model-free methods, we have implemented a world model approach based on variational autoencoders for learning compact latent representations of game states. This approach enables

imagination-based planning where the agent can simulate potential outcomes without interacting with the actual environment.

6.1 VAE Architecture

The Variational Autoencoder compresses 64×64 RGB game frames into a 32-dimensional latent vector. The encoder consists of four convolutional layers that progressively reduce spatial dimensions while increasing channel depth. The final convolutional output is projected to mean and log-variance vectors that parameterize the latent distribution. The decoder mirrors this structure with transposed convolutions to reconstruct the original frame.

Training uses the standard VAE objective combining reconstruction loss (mean squared error between original and reconstructed frames) with KL divergence regularization that encourages the latent distribution to remain close to a unit Gaussian prior.

6.2 Dynamics Model

A separate neural network learns the transition dynamics in latent space. Given a current latent state z_t and action a_t , the dynamics model predicts the next latent state z_{t+1} along with expected reward and episode termination probability. This enables planning by rolling out action sequences entirely in the learned latent space without expensive environment simulation.

6.3 Training Results

Data collection was completed with 10,000 frames gathered using a random policy over 53 episodes. VAE training ran for 50 epochs achieving excellent reconstruction quality with a final MSE of **0.0016**. Figure 6 shows representative reconstruction results demonstrating that the trained VAE accurately captures game state including agent position, enemy locations, and prize positions.

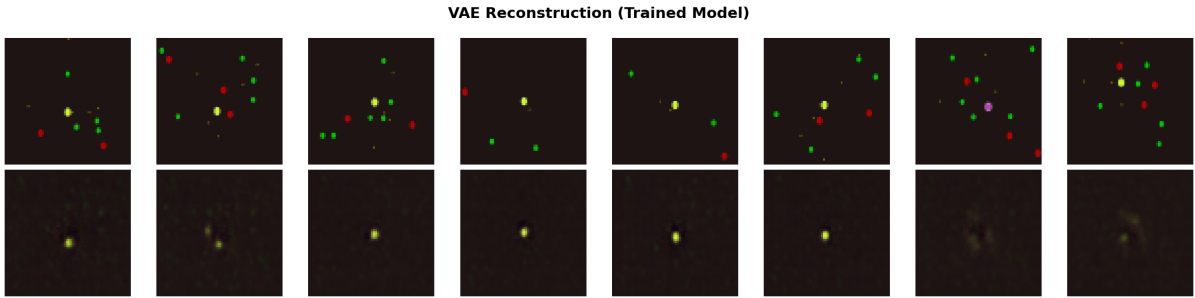


Figure 6: VAE Reconstruction Results. Top row: original game frames. Bottom row: reconstructed frames from the 32-dimensional latent encoding. Reconstruction MSE = 0.0016.

The low reconstruction error indicates that the 32-dimensional latent space is sufficient to represent the essential game state information, supporting its use for downstream planning tasks.

6.4 Latent Space Properties

Figure 7 demonstrates smooth interpolation between two game states in the latent space. Starting from an initial frame ($\alpha = 0$) and transitioning to a target frame ($\alpha = 1$), intermediate latent vectors produce semantically meaningful interpolated images. This smoothness is a desirable property indicating that similar game states are mapped to nearby latent representations.

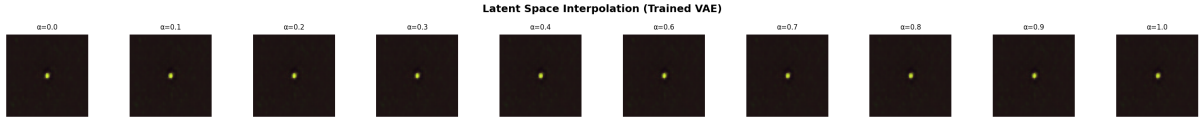


Figure 7: Latent Space Interpolation. Smooth transition between two game states by linearly interpolating their latent representations demonstrates the learned latent space is geometrically meaningful.

The dynamics model was trained for 50 epochs on transitions extracted from the collected frames. With the VAE and dynamics model trained, the world model agent can now perform Model Predictive Control (MPC) by sampling action sequences, rolling them out in imagination, and selecting actions that lead to predicted high rewards.

7 Analysis of SAC Performance Failure

Soft Actor-Critic (SAC) significantly underperformed compared to DQN and PPO across all experimental configurations, achieving a best result of only +2.23 on aggressive/medium compared to PPO’s +38.74 on aggressive/long—a 17x performance gap. This section analyzes the root causes of SAC’s failure in our environment.

7.1 Action Space Mismatch

SAC is designed for continuous action spaces where it can leverage reparameterization gradients through the policy network. Our environment uses a multi-discrete action space (80 combinations), requiring a continuous-to-discrete wrapper that maps box-constrained continuous outputs back to discrete choices. This wrapper introduces quantization noise and prevents direct gradient flow, fundamentally compromising SAC’s optimization mechanism.

7.2 Training Speed Penalty

SAC trained at approximately 65 frames per second compared to DQN’s 800 FPS and PPO’s 1500 FPS aggregate throughput. This 12x speed disadvantage meant SAC experienced far fewer environment interactions per wall-clock hour. Given identical timestep budgets, SAC had less time for policy refinement, particularly problematic for the long (1.6M step) experiments.

7.3 Entropy Regularization Interference

SAC’s maximum entropy objective encourages exploration by adding entropy bonuses to the reward. In our relatively simple environment where exploitation of learned behaviors should dominate, the entropy term may have prevented SAC from committing to high-reward action patterns, keeping the policy overly stochastic.

7.4 Episode Length Observations

Table 2 shows that SAC completed significantly more episodes (7,714 for baseline/long vs 2,997 for DQN). This suggests SAC agents died more frequently, accumulating death penalties that dragged down mean rewards. The shorter average episode length indicates SAC failed to learn survival behaviors as effectively as the other algorithms.

8 Challenges and Issues Encountered

Throughout this project, we encountered several technical challenges that required careful engineering solutions.

Action Space Incompatibility: The multi-discrete action space posed compatibility issues with algorithms designed for discrete (DQN) or continuous (SAC) spaces. We implemented custom wrappers that flatten multi-discrete to discrete for DQN and convert continuous box actions back to discrete choices for SAC.

SAC Training Speed: SAC’s off-policy nature with continuous action processing resulted in approximately 12x slower training compared to DQN and PPO. This limited the practical number of experiments we could run with SAC and may contribute to its underperformance.

RGB Rendering for World Model: The arcade-based game environment required custom implementation for rendering to numpy arrays suitable for neural network training, as the default rendering pipeline was designed for display rather than data collection.

9 Conclusions and Next Steps

Our preliminary experimental results demonstrate that reward shaping significantly influences RL agent performance in the 2D shooter environment. The aggressive reward configuration consistently outperforms alternatives across all algorithms and training durations. Among algorithms, PPO and DQN achieve comparable performance, while SAC underperforms potentially due to action space conversion overhead and slower training.

Future work will focus on completing world model training and comparing imagination-based planning with model-free baselines. We also plan to investigate hyperparameter sensitivity and conduct longer training runs for the best-performing configurations.

References

- [1] Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, 2015.
- [2] Schulman et al., “Proximal Policy Optimization Algorithms,” arXiv:1707.06347, 2017.
- [3] Haarnoja et al., “Soft Actor-Critic Algorithms and Applications,” arXiv:1812.05905, 2018.
- [4] Ha and Schmidhuber, “World Models,” arXiv:1803.10122, 2018.