RL-Course 2024/25: Final Project Report

Jisujo: Joseph Li Wan Wang, Sumitrra Bala, Jiha Kim

February 26, 2025

1 Introduction

We explore Reinforcement Learning (RL) in a 2D hockey environment adapted from the `hockey-env` repository [10] built on the Gymnasium API [2] and Box2D physics [4]. Each episode starts with a puck near center ice and ends when a goal is scored or a time limit is reached.

1.1 Hockey Environment Overview

State Space. At each time step t, the agent observes an 18-dimensional vector:

$$\mathbf{s}_t = \big(\underbrace{x_1, y_1, \theta_1, \dot{x}_1, \dot{y}_1, \dot{\theta}_1}_{\text{Player 1 state}}, \underbrace{x_2, y_2, \theta_2, \dot{x}_2, \dot{y}_2, \dot{\theta}_2}_{\text{Opponent state}}, \underbrace{x_p, y_p, \dot{x}_p, \dot{y}_p}_{\text{Puck}}, \underbrace{t_{\text{puck1}}, t_{\text{puck2}}}_{\text{Puck possession}}\big),$$

where (x_i, y_i) and $(\dot{x}_i, \dot{y}_i)$ represent player positions and velocities, θ_i denotes angular orientation, and (x_p, y_p), $(\dot{x}_p, \dot{y}_p)$ track the puck's position and velocity. The last two entries count how many steps each player can "hold" the puck under `keep_mode` rules.

Action Space. The agent issues continuous commands $\mathbf{a}_t \in \mathbb{R}^4$ of the form:

$$\mathbf{a}_t = (F_x,\, F_y,\, \tau,\, \text{shoot}),$$

where (F_x, F_y) control movement via applied forces, τ adjusts the player's rotation, and shoot triggers a short-range puck push if the agent is in possession. The opponent's action can be either a scripted policy (`BasicOpponent`) or user input.

Reward Function. By default, a goal for Player 1 yields $+10$ reward, and a goal for the opponent yields -10. Episodes terminate upon scoring or after a maximum number of time steps (e.g., 250). Additional shaping terms may reward puck proximity or penalize letting the puck idle in the agent's defensive zone.
We evaluate three **state-of-the-art RL algorithms** on this hockey environment:

- **Double Dueling Deep Q-Network (DDDQN)** [16, 19] : An extension of DQN with double Q-learning and dueling architecture for a discretized action space. *(Joseph Li Wan Wang)*
- **Soft Actor-Critic (SAC)** [8]: A maximum-entropy off-policy algorithm balancing robust exploration and reward maximization. *(Sumitrra Bala)*
- **Twin Delayed Deep Deterministic Policy Gradient (TD3)** [7] An off-policy method refining DDPG via multiple critics and delayed policy updates. *(Jiha Kim)*

2 Dueling Double-DQN (DDDQN): Joseph Li Wan Wang

Since Q-learning in high-dimensional observation spaces became feasible with neural networks as function approximators [12], some DQN-based methods have achieved state-of-the-art performance [9]. In the following section, we address the hockey environment using an improved version of DQN. This implementation builds upon base code for DQN provided in the "Reinforcement Learning (ML-4350)" course at the University of Tübingen, instructed by Prof. Georg Martius during the Winter Semester 2024-25 (Homework assignment 7) [11].

2.1 Method

Basic DQN: "Deep Q-Networks (DQN)" [12] is a model-free, off-policy algorithm that approximates the optimal action-value function $Q^*(s, a)$ using a deep neural network. This implies that:

$$Q^*(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a\right] \tag{1}$$

is estimated by minimizing the temporal difference (TD) loss function, defined as:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(y_i^{DQN} - Q(s, a; \theta_i)\right)^2\right] \tag{2}$$

where y^{DQN} is the target Q-value, computed as:

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{3}$$

Where θ represents the parameters of the main network and θ^- the parameters of the target network that is periodically updated. Since the target network is initialized as a copy of the main network, there is no significant added computational cost. This decoupling reduces the correlation between predictions and targets, improving training stability during bootstrapping. Furthermore, by incorporating an experience replay buffer D, DQN enables learning from past behavior and breaks the correlation between consecutive experiences by random sampling.

Double DQN: However, the inclusion of a max operator in (3) leads to overestimation of the action values. To solve this, Double DQN modifies the target Q-value by decoupling action selection from action evaluation [16]:

$$y_i^{DDQN} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta^-) \tag{4}$$

This means that the target network θ^- which in standard DQN is used for both action selection and evaluation, is now only responsible only for action evaluation, while action selection is done by the main network θ.

Dueling DQN: While Double DQN handles overestimation bias, Dueling DQN [19] improves value estimation efficiency by decomposing the Q function into (1) a state value function $V(s; \theta, \beta)$ independent of the action and (2) an advantage function $A(s, a'; \theta, \alpha)$ that measures the relative importance (advantage) of each action a in a given state. The Q-function is then computed as:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|A|}\sum_{a'} A(s, a'; \theta, \alpha)\right), \tag{5}$$

where subtracting the average advantage ensures identifiability of $Q(s, a; \theta, \alpha, \beta)$. This helps the network to learn states with low action dependence more efficiently and increase robustness to small Q-value differences [19]. The key intuition behind Dueling DQN is that in many states, the choice of action has little effect on the outcome. In those states the network can rely mainly on the state value, thus reducing noise from action-specific value estimates.

Extension 1: Prioritized Experience Replay
The vanilla experience replay buffer performs uniform batch sampling from all transitions (s, a, r, s'). This can be improved by weighting experiences by the magnitude of their TD-errors. By doing so, transitions with larger prediction errors are sampled more frequently, allowing the agent to focus on learning from the most informative experiences. This method is called Prioritized Experience Replay (PER) [15], and has been shown to significantly improve DQN performance [9]. One way of performing PER is by assigning sampling probabilities proportional to the TD errors. The probability of sampling an experience i is given by:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{6}$$

where $p_i = |\delta_i| + \epsilon$ is the priority of transition i, defined by its absolute TD-error δ_i and a small constant ϵ for numerical stability. α regulates the degree of prioritization. Since PER alters the data distribution, it introduces bias in the Q-value updates. To correct this bias, importance-sampling (IS) weights are used:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{7}$$

where N is the number of experiences in the buffer and $\beta \in [0, 1]$ regulates the amount of correction.

Extension 2: ICM
The second extension consists of an intrinsic motivation method, commonly known as the "Intrinsic Curiosity Module (ICM)", which encourages the agent to explore new states [14]. This method has been shown to improve the learning of dynamics in physical environments [5], such as air hockey.
Its architecture consists of an inverse model that predicts the action $\hat{a}$ taken between two consecutive states s_t and s_{t+1}, and a forward model that predicts the next state representation $\hat{\phi}(s_{t+1})$ given the current one $\phi(s_t)$ and the current action a_t. The prediction error between $\hat{\phi}(s_{t+1})$ and the actual $\phi(s_{t+1})$ serves as the intrinsic reward signal, driving the agent to explore less familiar states. For a more detailed explanation about the ICM architecture, see Figure 11 in A.1.1 and the mathematical formulation detailed in A.1.2.

Extension 3: Curriculum learning
Inspired by how humans and animals learn complex tasks through gradual exposure to increasingly challenging scenarios, curriculum learning [1] has been applied to reinforcement learning algorithms to improve training [13]. It consists of implementing a step-wise training schedule, starting with simpler tasks and progressively moving to more demanding ones. This allows the agent to first master basic skills before integrating them into more complex, high-dimensional tasks, facilitating better generalization and faster convergence. For the DDDQN agent used in the tournament, the following training program was implemented, aptly named "Coach":

- **Phase 1 (Specialization):** The agent first trains using the `TRAIN_SHOOTING` mode, followed by `TRAIN_DEFENSE` against the weak opponent to improve specialized skills. *(20% of all episodes)*

- **Phase 2 (Integration):** The agent trains against a mix of weak and strong opponents in normal game mode. The probability of facing a weak opponent starts at 100% and decays linearly until only strong opponents remain, annealing the task difficulty from easy to hard. *(40% of all episodes)*
- **Phase 3 (Full Competency):** The agent is trained in normal mode against a strong opponent. *(40% of all episodes)*

2.2 Experiments

This section presents an experimental evaluation on the implemented methods and extensions. Model performance was assesed using the average win-rate and reward. The neural network was trained using the "ReLU" activation function, "ADAM" optimizer and "Smooth L1" loss. During training, epsilon decay (for sub-linear total regret [11]) and beta annealing (for unbiasedness in PER sampling [15]) were applied.

Hyperparameter Search
To identify the optimal hyperparameters, a sensitivity study was conducted during training against the weak opponent in normal game mode. Given the large combinatorial hyperparameter space and computational time limitations, hyperparameters were tuned independently, assuming minimal interaction effects. Below, a subset of the results is presented to illustrate the effect of parameter variations on training performance. However, a more detailed report is available in appendix A.2 and on Github [18]:

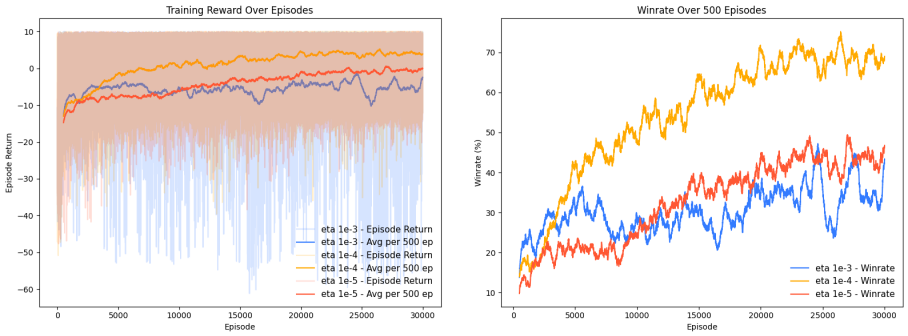

Figure 1: Difference in performance according to learning rate

Figure 1 shows that a learning rate of 1e-4 achieves the highest win-rate and reward by 30.000 steps. In contrast, 1e-5 makes learning too slow, while 1e-3 shows instability due to (potential) overshooting.

Extension Assessments
The following section evaluates the the model extensions through an ablation study and other analyses.

Prioritized Experience Replay (PER): Figure 2 shows that PER improves performance, particularly in late training, compared to uniform sampling. A possible explanation is that with PER, after the replay buffer reaches capacity, high TD-error transitions that often correspond to significant and challenging situations (e.g. near goal states, blocks) are retained and sampled more frequently [15]. In contrast, uniform sampling revisits critical states less often, limiting the agent's ability to correct mistakes, which might explain the performance plateau.
Figure 20 in Appendix A.2.2 (left) shows that most transitions in the replay buffer have relatively small TD-errors (representing non critical transitions), while a minority exhibit significantly larger errors. The

right plot mirrors this distribution in terms of sampling weights, assigning proportionally higher probabilities to those high-error transitions.

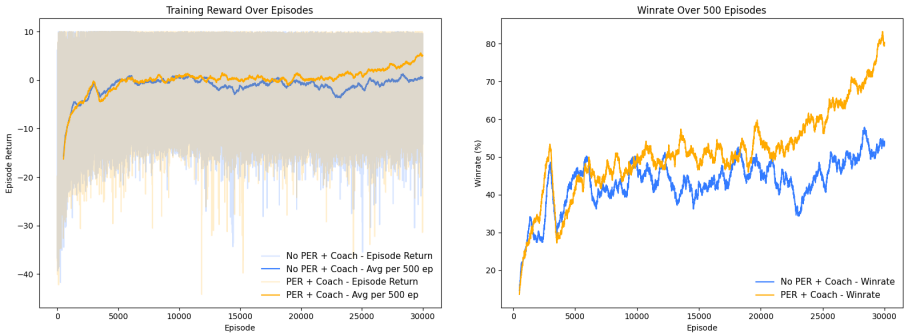

Figure 2: Comparative performance (PER vs. no PER)

Intrinsic Curiosity Module (ICM): Figure 3 illustrates how ICM increases state-space coverage (left) compared to a baseline with no ICM (right) during training. In the "no ICM" plot, the agent's visits are more concentrated around the field center line, whereas the "with ICM" plot shows a more even distribution and lower maximum densities, indicating more movement around the field. Although the density map confirms improved exploration under ICM, its overall performance gain combined with PER and "Coach" remained negligible (see Figure 21 in appendix A.2.2). This is likely because PER already enhances exploration in high-surprise states, and the 'Coach' training program exposes the agent to a progressively wider range of scenarios, reducing the need for additional intrinsic motivation. As a result, ICM was excluded from the final tournament model.

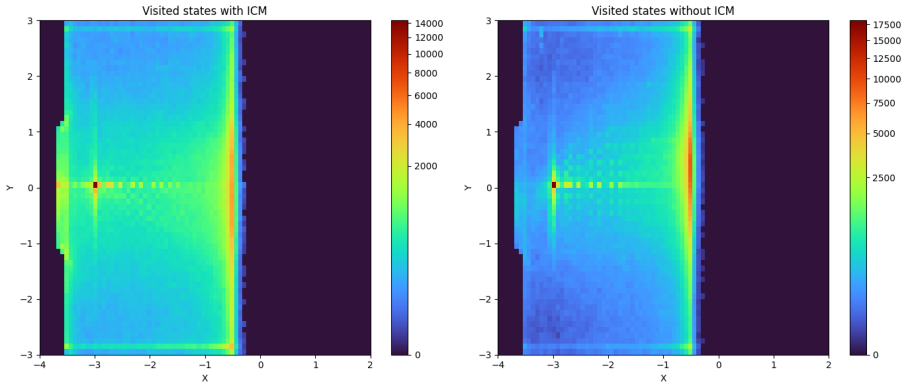

Figure 3: Amount of visited states (density map)

Curriculum Learning: Figure 4 compares an agent trained with the "Coach" curriculum learning program (blue) to one trained directly against a strong opponent (orange). The curriculum-based agent achieves faster learning and higher win rates in all stages. Gradually introducing more complex challenges rather than facing the strong opponent directly, allows for a significant improvement in training stability and overall performance as described in [13].

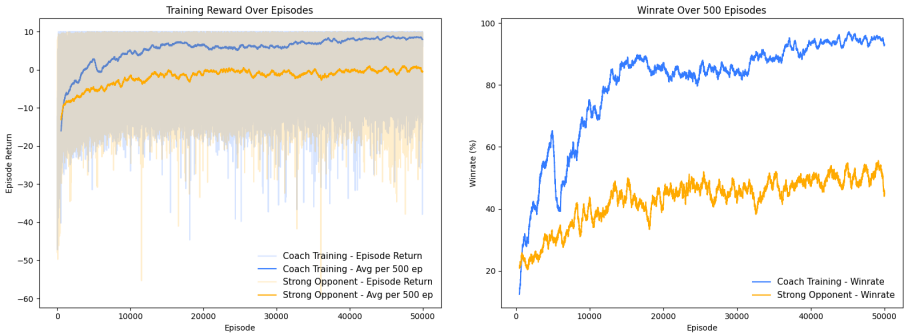

Figure 4: Comparative performance (Curriculum Learning "Coach" vs. normal training against strong agent)

3 Soft-Actor-Critic (SAC): Sumitrra Bala

3.1 Method

Soft Actor-Critic (SAC) [8] is an off-policy actor-critic method that maximizes both the expected return and the policy's entropy. Let $\pi_\phi(a \mid s)$ be the stochastic policy (actor) with parameters ϕ, and $Q_\theta(s,a)$ be a soft Q-function with parameters θ. At each update step, SAC minimizes the following soft Bellman residual for the critic:

$$J_Q(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Big[\Big(Q_\theta(s,a) - r - \gamma \mathbb{E}_{a'\sim\pi_\phi}\big[\min_{i=1,2} Q_{\bar{\theta}_i}(s',a') - \alpha \log \pi_\phi(a' \mid s')\big]\Big)^2\Big], \tag{8}$$

where $\bar{\theta}_i$ are target critic parameters, α is the temperature coefficient that balances exploration and exploitation, and $\mathcal{D}$ is the replay buffer. The actor is then updated by maximizing the expected Q-value under the current policy plus the entropy term:

$$J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D},\, a\sim\pi_\phi}\Big[\alpha \log \pi_\phi(a \mid s) - Q_\theta(s,a)\Big]. \tag{9}$$

Extensions and Modifications. In our implementation for the Hockey environment, we introduce the following key improvements to the vanilla SAC framework:

- **Attention-Based Feature Extractor:** We replace the default MLP with a multi-head attention network (`HockeyAttentionNetwork`) to better capture interactions between the puck, our agent, and the opponent [17]. The input observation $\mathbf{x} \in \mathbb{R}^n$ is first transformed via a linear layer and activation:

  $$\mathbf{f} = \text{GELU}(\text{LayerNorm}(W\mathbf{x} + b)) \in \mathbb{R}^{256}.$$

  These features are then processed by a multi-head attention module, where attention is computed as:

  $$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

  Here, Q, K, V are obtained are the query, key, and value matrices, obtained by projecting $\mathbf{f}$ with learned mappings, and d_k is the key dimension. Softmax normalization ensures focus on relevant input features, helping the agent track puck dynamics, opponent movement, and spatial positioning.

- **Random Network Distillation (RND):** To encourage exploration, we include an intrinsic reward based on RND [3]. Two networks (a fixed "target" and a learned "predictor") process the same observation s, and the mean-squared error between their outputs defines a curiosity bonus:

$$r_{\text{int}}(s) = \beta \, \|\phi_{\text{pred}}(s) - \phi_{\text{target}}(s)\|^2,$$

  which is added to the extrinsic reward from the environment.

- **Curriculum Learning with Mixed Training:** Following the principles of curriculum learning [6], we train the agent in stages using a specialized reward wrapper to guide behavior.

3.2 Experiment Setup

3.2.1 Training Procedure

Curriculum Stages: The agent is trained across four progressively harder stages:

1. **Shooting (Attack Mode):** The agent learns to shoot the puck into an empty goal using `AttackRewardWrapper`, which gives additional rewards for scoring and penalizes distance from the goal.
2. **Defense Mode:** The agent learns to defend against an incoming puck by minimizing its proximity to its own goal using `DefenseRewardWrapper`.
3. **Weak Opponent:** The agent plays against a baseline opponent, which follows a predefined weak strategy.
4. **Strong Opponent:** The agent competes against a more challenging opponent with stronger defensive and offensive behavior.

Mixed Training Between Stages: Instead of transitioning abruptly between stages, we introduce the `MixedTrainingWrapper`, which, with probability p_{mix}, samples an episode from the **previous** stage instead of the current stage. This gradual transition stabilizes learning by preventing catastrophic forgetting of earlier-learned behaviors.

3.2.2 Hyperparameter Scheduling

We further optimize the learning process by dynamically adjusting the following hyperparameters:

- **Learning Rate:** The learning rate η is scheduled to decay across curriculum stages. Earlier stages (e.g., *shooting*) use a higher learning rate (e.g., 3×10^{-4}), while later stages (e.g., *strong opponent*) use a smaller rate (e.g., 1×10^{-4}) to ensure stability.
- **Batch Size:** Small batch sizes (1024) are used for early-stage training, while larger batch sizes (2048) improve stability in later stages.
- **Exploration Parameter (α):** The SAC entropy coefficient α is fine-tuned per stage to balance exploration and exploitation.

3.2.3 Logging and Evaluation

During training, we log key performance metrics using TensorBoard and Stable-Baselines3's built-in monitoring system. We measure the agent's individual episode rewards and tracked the win rates against opponents. Checkpoints are saved every 500 training steps, allowing models to be resumed or analyzed at intermediate stages.

3.3 Results and Discussion

In this section, we compare the performance of our Soft Actor-Critic (SAC) agent across five training setups:

3.3.1 Overall Performance Metrics

Figure 5 shows the *total win rate* over the course of training for each setting. The Curriculum approach (purple curve) achieves rapid improvement in the early stages (around 300k steps) and converges to a win rate over 95%, surpassing the individually trained agents weak and strong opponents in both speed of convergence and final performance.

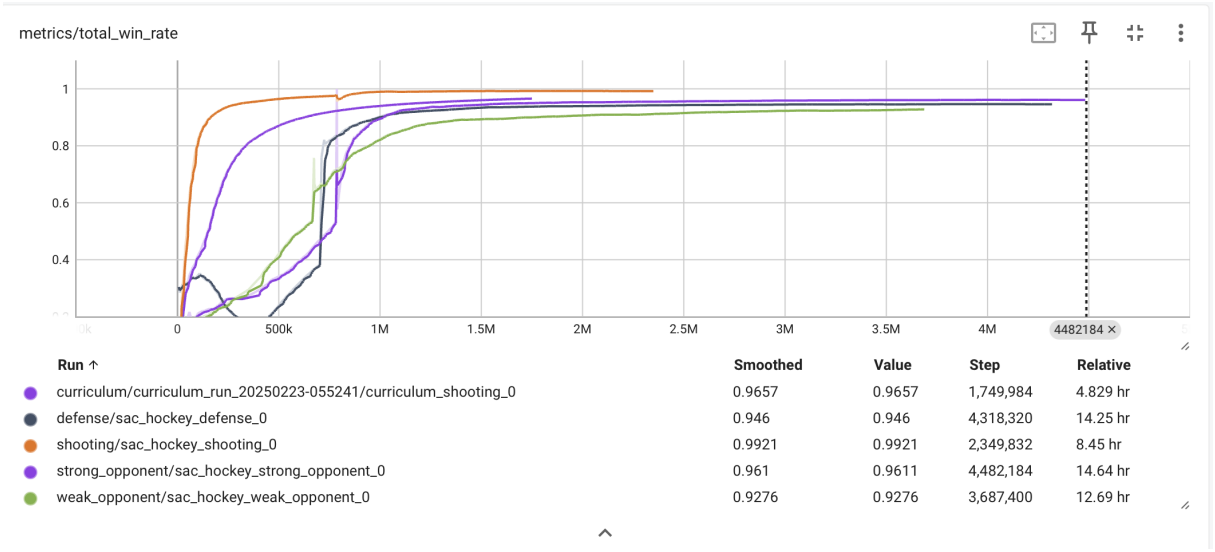

Figure 5: Comparison of total win rate for individual training modes versus the Curriculum Learning approach. The Curriculum agent shows faster convergence and higher asymptotic performance.

3.3.2 Self-Play Comparison: Strong Opponent vs. Curriculum Agent

To further validate performance, we conducted a *self-play* matchup between the agent trained against just the Strong Opponent and the agent trained by curriculum for 500 episodes.

Agent	Win Rate	Avg. Return
SAC (Strong Opp. Only)	23.98%	-5.1
SAC (Curriculum)	**27.53%**	**2.7**

Table 1: Self-play results between the Strong Opponent-trained SAC agent and the Curriculum-trained SAC agent.

Table 1 summarizes the win rate and average returns. The results suggest that the Curriculum agent develops more **balanced offensive and defensive strategies**, likely due to exposure to simpler tasks (shooting, defense, weak opponent) before facing a strong opponent.

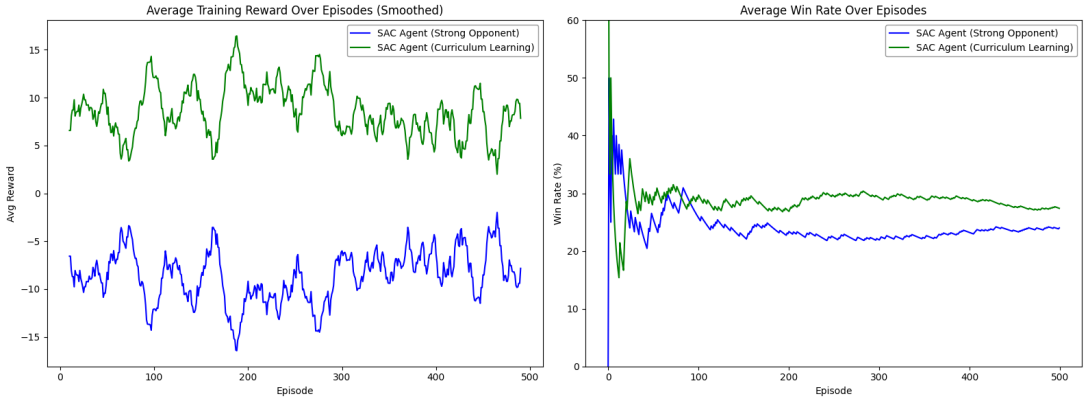

Figure 6: Comparison of training reward (left) and win rate (right) between SAC agents. The Curriculum agent achieves more stable and superior performance.

Statistical Significance Test To determine whether the observed difference is statistically significant, we conducted a **two-proportion Z-test** for the win rates.
The Z-score is computed as:

$$Z = \frac{p_1 - p_2}{\sqrt{p(1-p)\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} \tag{10}$$

where $p_1 = 0.2753$ and $p_2 = 0.2398$ are the win rates of the Curriculum and Strong Opponent agents, respectively. With 500 episodes per agent, the pooled proportion is:

$$p = \frac{x_1 + x_2}{n_1 + n_2} = 0.25755 \tag{11}$$

which gives a Z-score of **1.29** and a corresponding **p-value of 0.0985** which is only marginally significant at a 10% level ($\alpha = 0.10$). Further investigation of the training efficiency of both agents is presented in the Appendix.

4 Twin Delayed Deep Deterministic Policy Gradient(TD3) : Jiha Kim

4.1 Method

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an improved version of Deep Deterministic Policy Gradient (DDPG) with three key modifications:

- **Clipped Double Q-learning:** TD3 addresses Q-value overestimation by using two independent critic networks and taking the minimum value.

$$y = r + \gamma Q(s', \pi(s')) \quad \text{(DDPG Target Value)} \tag{12}$$

$$y = r + \gamma \min(Q_1(s', a'), Q_2(s', a')) \quad \text{(TD3 Target Value)} \tag{13}$$

- **Delayed Policy Updates (Policy Delay):** The policy (actor network) is updated less frequently than the critic network to improve stability.

$$J(\theta_\pi) = \mathbb{E}_{s\sim\rho^\beta}\left[Q(s, \pi(s|\theta_\pi))\right] \tag{14}$$

$$\nabla_{\theta_\pi} J(\theta_\pi) = \mathbb{E}_{s\sim\rho^\beta}\left[\nabla_a Q(s,a)|_{a=\pi(s)} \nabla_{\theta_\pi}\pi(s|\theta_\pi)\right] \tag{15}$$

The actor update occurs only once every d steps (*e.g.*, $d = 2$). Hence, when the step index t satisfies $t \mod d = 0$, we update the actor:

$$J\phi \leftarrow \phi + \alpha_\pi \mathbb{E}_{s\in\mathcal{D}}\Big[\nabla_a Q_{\theta^1}(s,a)\big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)\Big] \tag{16}$$

where α_π is the actor's learning rate. TD3 ensures the actor sees a more reliable estimate of Q, thus improving training stability.

- **Target Policy Smoothing:** Noise is added to the target policy action to prevent overfitting to sharp Q-function estimates.

$$a' = \pi(s') + \epsilon, \quad \epsilon \sim \mathcal{N}(0,\sigma) \tag{17}$$

$$\epsilon = \text{clip}(\epsilon, -c, c) \tag{18}$$

TD3 is sensitive to hyperparameters, as shown by Fujimoto et al. (2018) [7]. Thus, we conduct two main experiments: first, we examine the effect of *Policy Delay* on performance; second, we perform a random search to identify optimal hyperparameter configurations.

4.2 Policy Delay Experiments

In our experiments, we tested Policy Delay values of 1, 2, and 3 in two different environments: **LunarLanderContinuous-v2** and **Hockey Environment**.

Hyperparameter	Value
Discount factor (γ)	0.99
Soft update rate (τ)	0.005
Actor learning rate (lr_{actor})	0.001
Critic learning rate (lr_{critic})	0.001
Policy noise (σ)	0.2
Noise clip (c)	0.5

Table 2: TD3 Default Hyperparameters

Policy Delay of 1 means that the actor is updated at the same frequency as the critic (similar to DDPG). In contrast, Policy Delay values of 2 and 3 update the actor less frequently.
Table 3 presents the average rewards over 2000 episodes in the LunarLander environment.

Policy Delay	Average Reward
1	149.16
2	130.13
3	**184.52**

Table 3: Performance of TD3 with different Policy Delay values in LunarLander

Policy Delay	Average Reward
1	**4.17**
2	3.01
3	-0.54

Table 4: Performance of TD3 with different Policy Delay values in Hockey

4.3 Random Search for Hyperparameters

Hyperparameters interact with one another, making it challenging to pinpoint an optimal configuration. Consequently, we randomly sampled each hyperparameter and experimented with multiple hyperparameter sets to evaluate their performance.

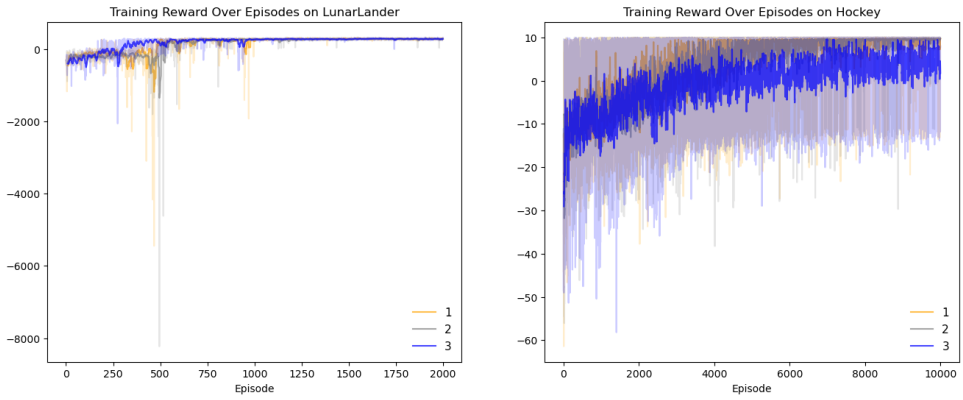

Figure 7: TD3 Policy Delay Performance Comparison

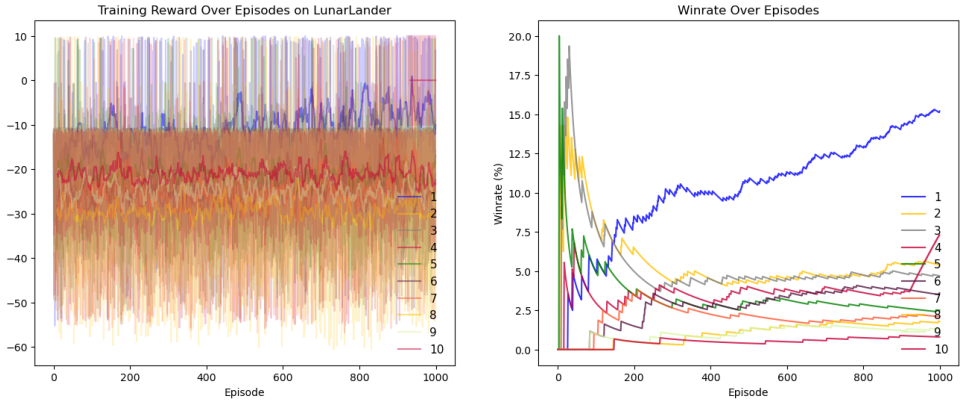

Figure 8: TD3 Policy Delay Performance Comparison

4.4 Results and Discussion

Table 3 shows that setting Policy Delay to 3 yields the highest average reward in the **LunarLanderContinuous-v2** environment, indicating that this configuration provides more stable performance compared to delays of 1 or 2. In contrast, Table 4 shows that in the **Hockey Environment**, a policy delay of 1 achieves the highest average reward (4.17), while a delay of 3 results in a negative average reward (0.54). These contrasting trends suggest that the optimal delay may vary across different tasks or environments. Furthermore, as depicted in Figure 7, using a policy delay of 2 on LunarLanderContinuous-v2 results in instability and lower returns, sometimes even performing worse than vanilla DDPG. This finding highlights that simply introducing a delay does not necessarily guarantee better performance; rather, it necessitates careful tuning to avoid detrimental outcomes. This observation aligns with Fujimoto et al. (2018) [7], who emphasize the challenges introduced by function approximation errors and hyperparameter tuning in actor-critic methods. Therefore, finding optimal hyperparameters, including policy delay, remains crucial for achieving stable and optimal performance when using TD3.

5 Discussion

We present two plots in Figures 9 and 10, showing the relative performance of our algorithms by self-playing.

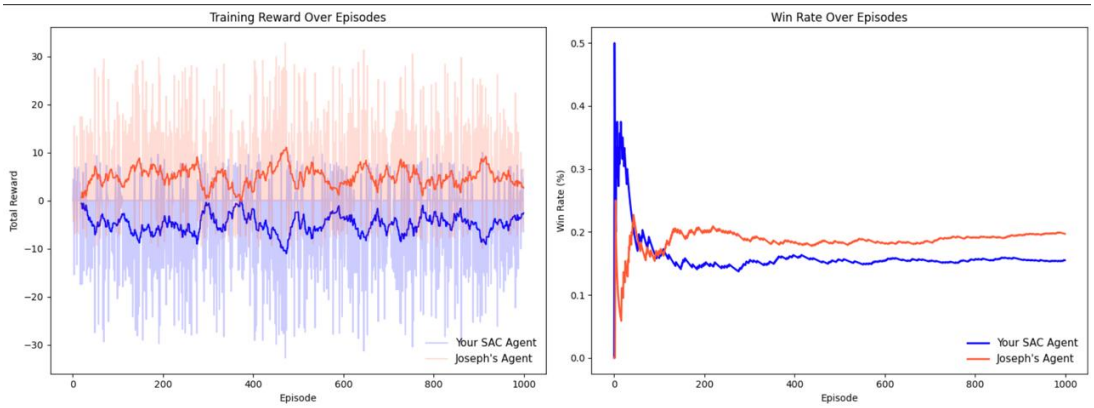

Figure 9: Comparison of training reward (left) and win rate (right) between SAC and DDDQN.

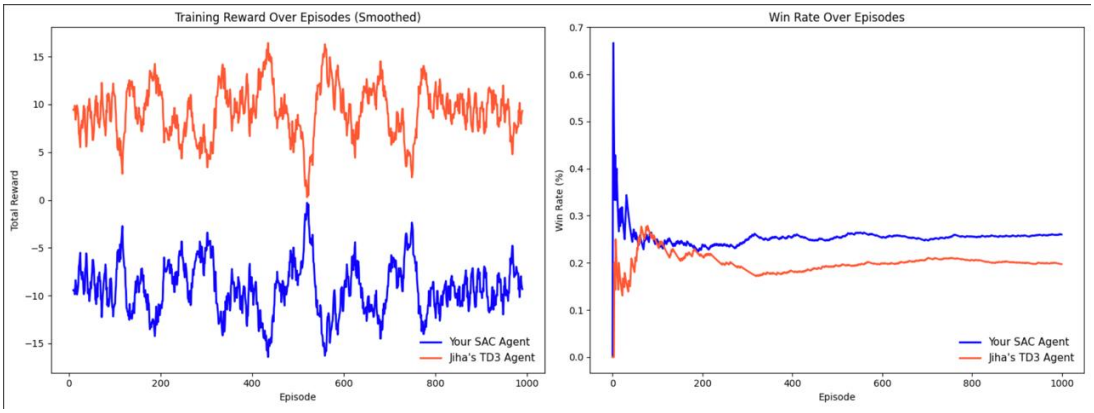

Figure 10: Comparison of training reward (left) and win rate (right) between SAC and TD3.

DDDQN consistently yields the best performance, both in terms of reward and win rate. This result builds on the algorithmic advantages described in Section 2.1, namely *Double Q-learning* and the *Dueling* architecture, which appear well-suited for the Hockey Environment's discretized action space. Furthermore, the additional techniques such as *Prioritized Experience Replay* and *Curriculum Learning* (Section 2.1) effectively boost sample efficiency, helping the agent converge to a robust policy.

SAC displays intermediate performance. Although its maximum-entropy formulation can expedite exploration and provide stable updates, the continuous control setting of the Hockey Environment imposes intricate challenges (e.g., puck possession mechanics, collisions, etc.). SAC's attention-based encoder and Random Network Distillation (RND) indeed improved exploration, yet the agent sometimes struggled with the more discrete, event-driven aspects of hockey (e.g., precisely timing a shot). These nuances may explain why SAC did not surpass DDDQN's final performance.

TD3 exhibits the lowest overall performance among the three. As noted in Section 4, TD3 is sensitive to hyperparameters like policy delay, and in many runs it achieved suboptimal convergence. Unlike DDDQN, which can discretize action selections directly, TD3 must learn accurate Q-values in a fully continuous space, complicating function approximation. The random hyperparameter search (Section 8) occasionally approached good results, but the final average performance remained less competitive com-

pared to DDDQN or SAC.

Discrete vs. Continuous Action Spaces: DDDQN naturally suits environments with discrete or discretized action dimensions. In contrast, SAC and TD3 are designed for continuous action spaces. While the Hockey environment is continuous, it encourages discrete-like behavior, and that may benefit DDDQN, especially when combined with additional improvements and a well-designed curriculum.
Overall, these findings demonstrate that **DDDQN** outperforms **SAC** and **TD3** in our 2D Hockey Environment. We demonstrate distinctive strengths and weaknesses of each algorithm, highlighting the importance of careful hyperparameter tuning, curriculum design, and opponent modeling to leverage each method's potential in hockey environments.

References

[1] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gymnasium: A toolkit for developing and comparing reinforcement learning algorithms. `https://gymnasium.farama.org/`, 2022. Accessed: 26-Feb-2025.

[3] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[4] E. Catto. Box2d: A 2d physics engine for games. `https://box2d.org/`, 2010. Accessed: 26-Feb-2025.

[5] J. Choi and S. eui Yoon. Intrinsic motivation driven intuitive physics learning using deep reinforcement learning with intrinsic reward normalization. *arXiv preprint arXiv:1907.03116*, 2019.

[6] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel. Reverse curriculum generation for reinforcement learning, 2018.

[7] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2017.

[9] M. Hessel, J. Modayil, H. V. Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

[10] M. Lab. Hockey environment. `https://github.com/martius-lab/hockey-env/tree/master/hockey`, 2025. Accessed: 26-Feb-2025.

[11] G. Martius. Reinforcement learning (ml-4350) - homework materials. `https://ilias.uni-tuebingen.de` (restricted access), 2024. Course material in ILIAS, University of Tübingen, Winter Semester 2024-25.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop 2013*, 2013.

[13] S. Narvekar, J. Sinapov, M. Leonetti, and P. Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21:1–50, 2020.

[14] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2016.

[16] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[18] J. Wan. Github repository: Josephlww, 2024. Accessed: February 24, 2025.

[19] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.

A Dueling Double DQN (DDDQN)

A.1 ICM

A.1.1 Architecture

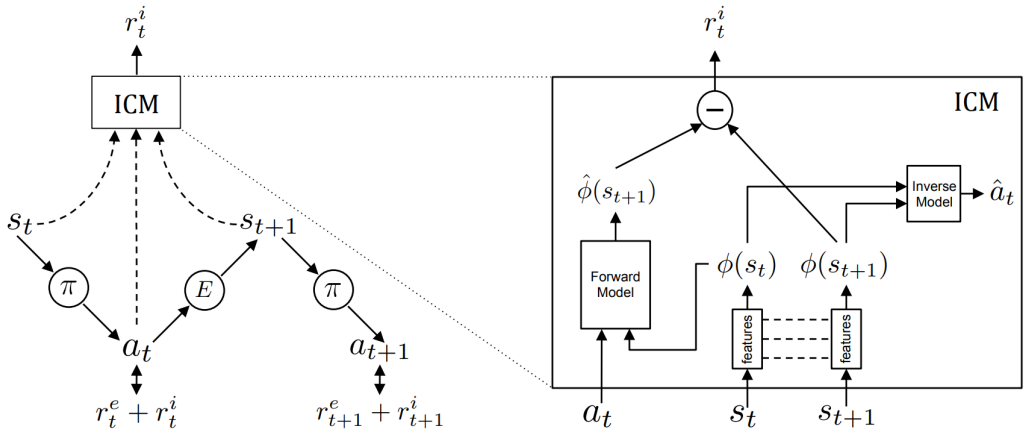

Figure 11: Illustration of the ICM architecture. The agent in state s_t interacts with the environment by executing an action a_t following its policy π, transitioning to state s_{t+1}. The Intrinsic Curiosity Module (ICM) consists of an inverse model, which predicts the action $\hat{a}_t$ given (s_t, s_{t+1}), and a forward model, which predicts the next state representation $\hat{\phi}(s_{t+1})$ based on $\phi(s_t)$ and a_t. The intrinsic reward r_t^i, derived from the prediction error of the forward model, is added to the extrinsic reward r_t^e to guide policy optimization. Since $\phi(s_t)$ only encodes state features that are relevant for the agent's actions, ICM promotes exploration while ignoring uncontrollable environmental factors [14].

A.1.2 Mathematical formulation

ICM [14] defines an intrinsic reward derived from the prediction error of the forward model:

$$r_t^{\text{ICM}} = \eta \cdot L_{\text{F}} = \frac{\eta}{2} \|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2 \tag{19}$$

Where L_{F} is the forward model loss, and η a reward scaling factor. ICM is optimized using the following objective function:

$$\min_{\theta_P, \theta_I, \theta_F} \left[-\lambda \mathbb{E}_{\pi(s_t; \theta_P)} \left[\Sigma_t r_t \right] + (1 - \beta) L_I + \beta L_F \right] \tag{20}$$

where:

- L_I is the inverse model loss
- λ balances extrinsic rewards
- β nudges the weighting between forward and inverse losses

A.2 Additional Experiments

In this section, additional results from the experiments regarding the DDDQN implementation are provided. These results were used for parameter tuning and performance evaluation.

A.2.1 Hyperparameters

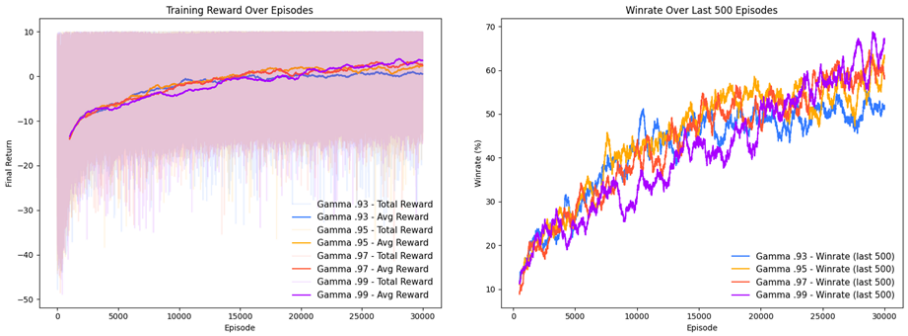

Figure 12: Performance according to discount rate training against the weak agent

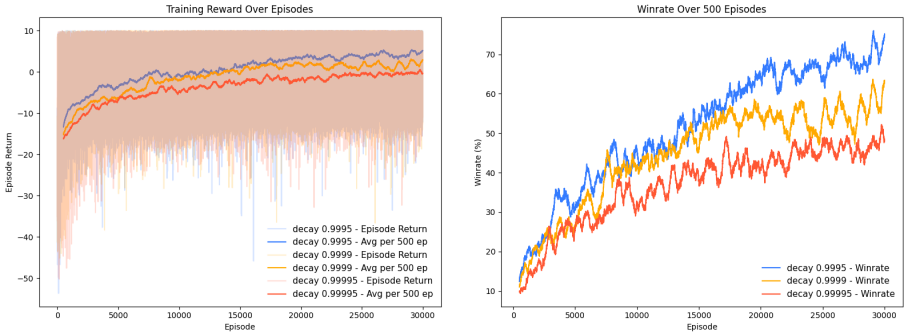

Figure 13: Performance according to epsilon decay training against the weak agent

Figure 13 shows how a epsilon-decay of .9995 suffices to achieve reasonable winrates by the end of 30.000 episodes, while slower decays still allow for exploration by the end of training.

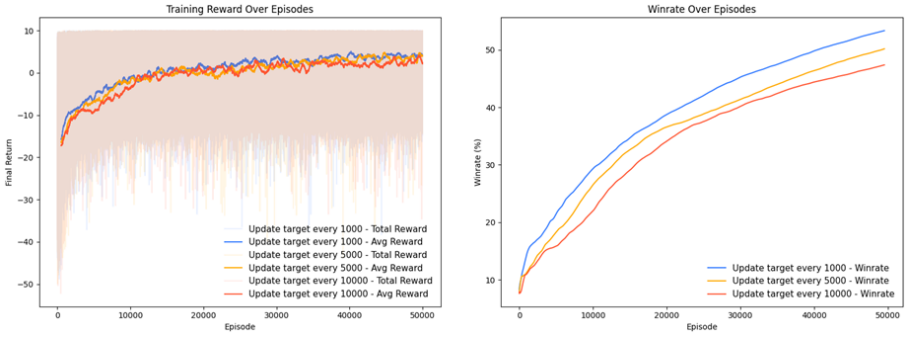

Figure 14: Performance according to target network update frequency (Cumulative win-rate used used for better visual readability)

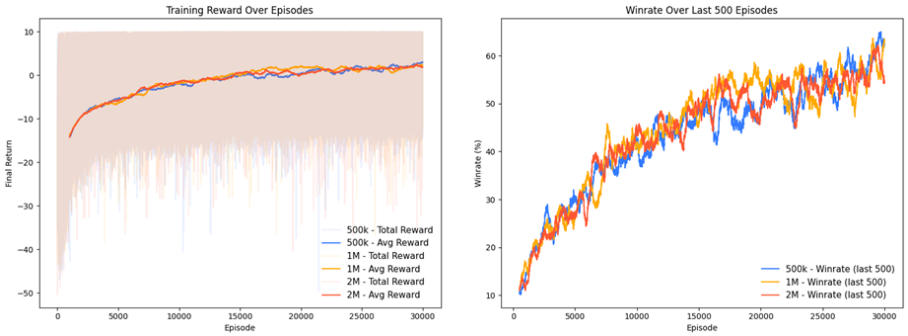

Figure 15: Performance according to buffer size

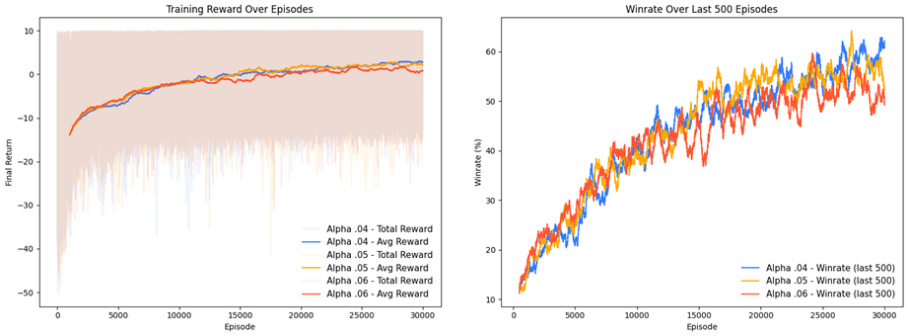

Figure 16: Performance according to alpha parameter (PER)

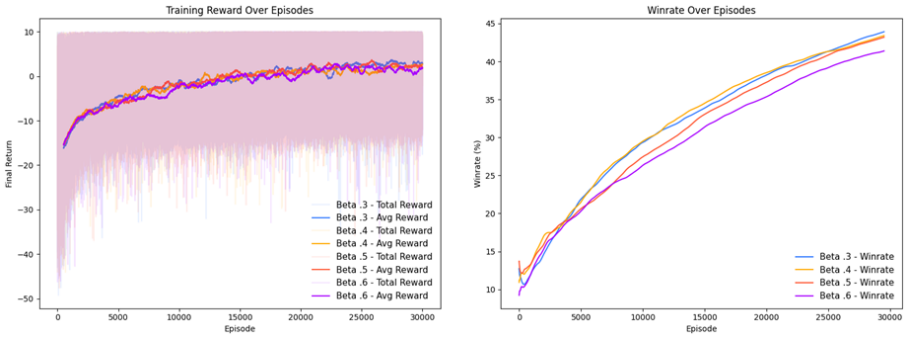

Figure 17: Performance according to initial beta parameter (PER). Cumulative win-rate used for better visual readability

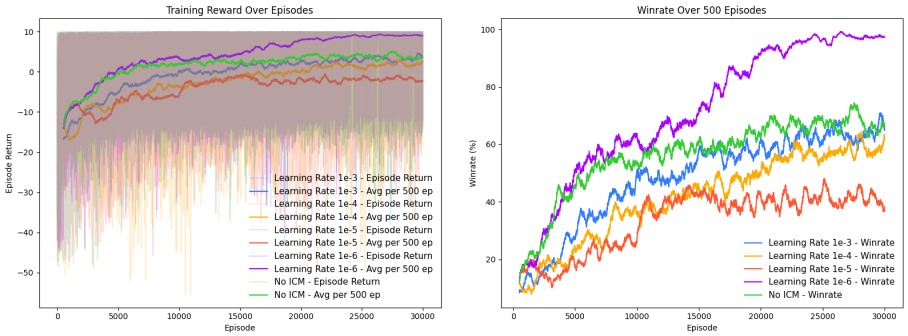

Figure 18: Performance according to learning rate training against the weak agent (ICM)

Table 5: Hyperparameter settings for the Dueling Double DQN used in the tournament.

Hyperparameter	**Value**
General Parameters	
Learning Rate (η)	1e-4
Discount Factor (γ)	0.95
Target Update Frequency	500 steps
Initial Epsilon (ϵ)	1.0
Minimum Epsilon (ϵ_{min})	0.01
Epsilon Decay (ϵ_{decay})	0.9995
PER Parameters	
Replay Buffer Size	1,000,000
Batch Size	32
Alpha (α)	0.5
Beta (β)	0.4
Beta Increment per Sampling	1e-7
Maximum Beta (β_{max})	0.9
ICM Parameters (Deactivated)	
ICM Learning Rate	1e-6
ICM Beta	0.6
ICM Feature Dimension	32
ICM Scale Factor	1e-4

A.2.2 Extensions

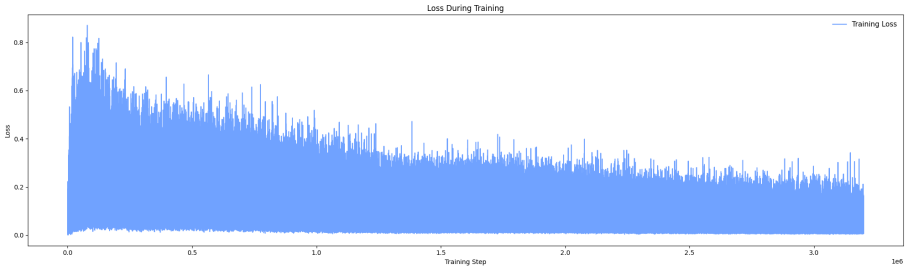

Figure 19: Loss function during curriculum learning

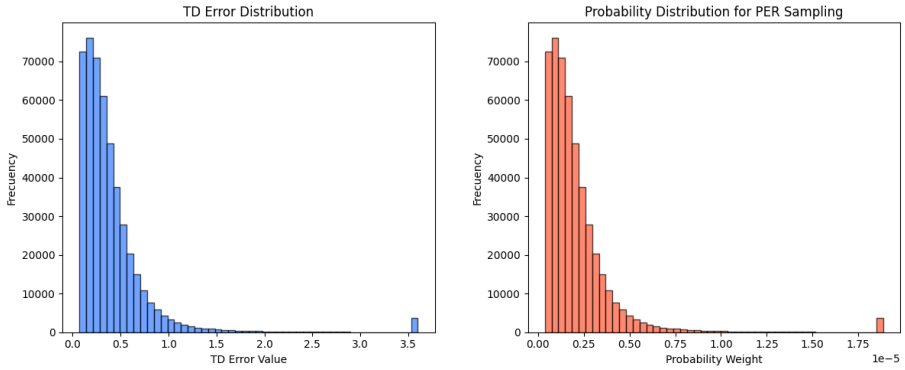

Figure 20: Weight and TD-Error Distribution in PER

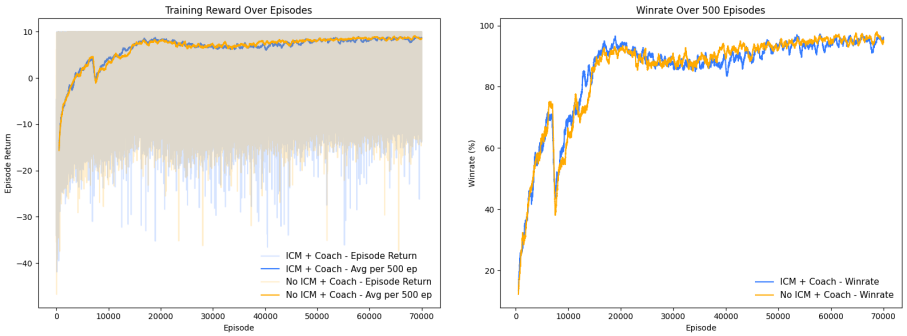

Figure 21: Comparative performance using the "Coach" training program (ICM vs. no ICM)

B Soft Actor-Critic (SAC) Extensions

In this section, we compare the performance of the SAC agent trained against a strong opponent (“SAC Strong”) versus the SAC agent trained via curriculum learning (“SAC Curriculum”). We focus on four metrics: *actor loss*, *critic loss*, *mean reward per episode*, and *entropy coefficient*. Each subsection presents two figures for direct visual comparison.

B.1 Actor Loss

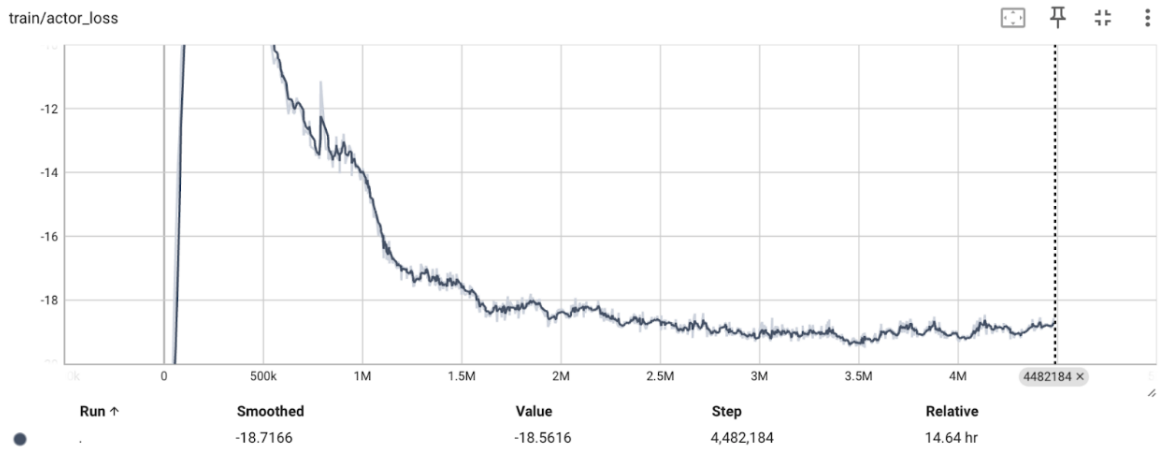

Figure 22: SAC Strong Opponent Actor Loss

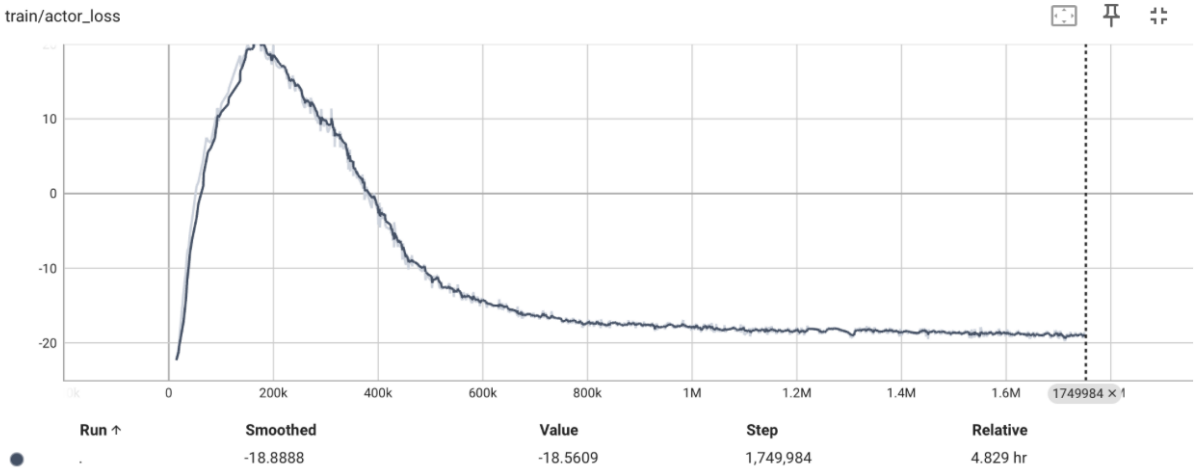

Figure 23: SAC Curriculum Actor Loss

Figure 24: Comparison of actor loss for SAC Strong (top) vs. SAC Curriculum (bottom). Training directly against a strong opponent may force the agent to explore aggressively and can lead to higher variance in the early stages of learning.

B.2 Critic Loss

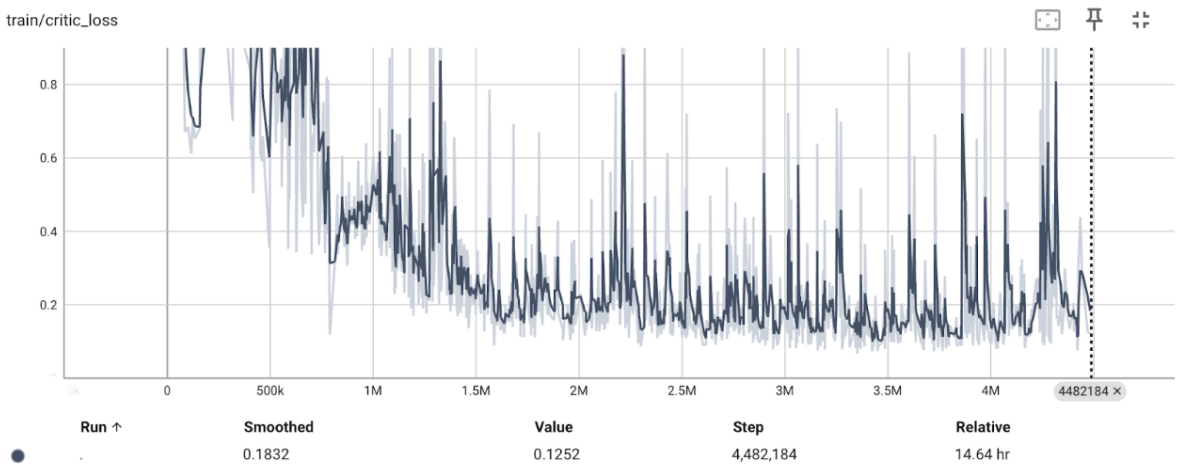

Figure 25: SAC Strong Opponent Critic Loss

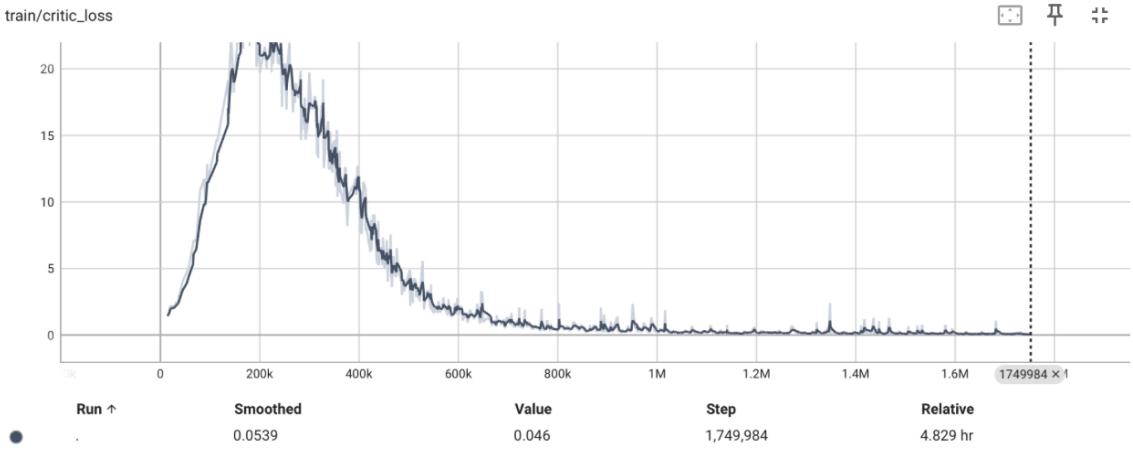

Figure 26: SAC Curriculum Critic Loss

Figure 27: Comparison of critic loss for SAC Strong (top) vs. SAC Curriculum (bottom). The curriculum agent's critic loss converges faster, reflecting smoother value function estimation.

B.3 Mean Reward per Episode

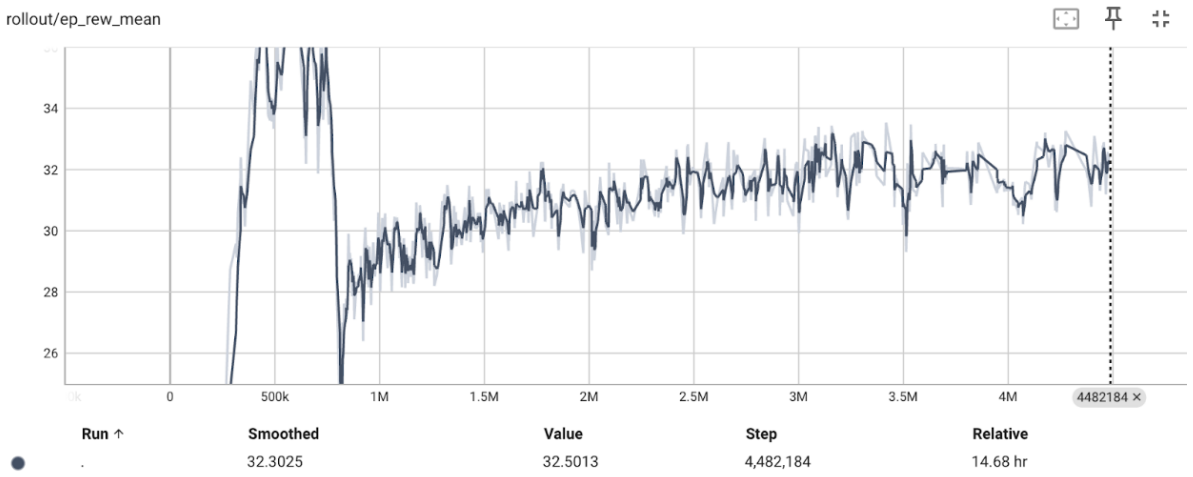

Figure 28: SAC Strong Opponent Mean Reward

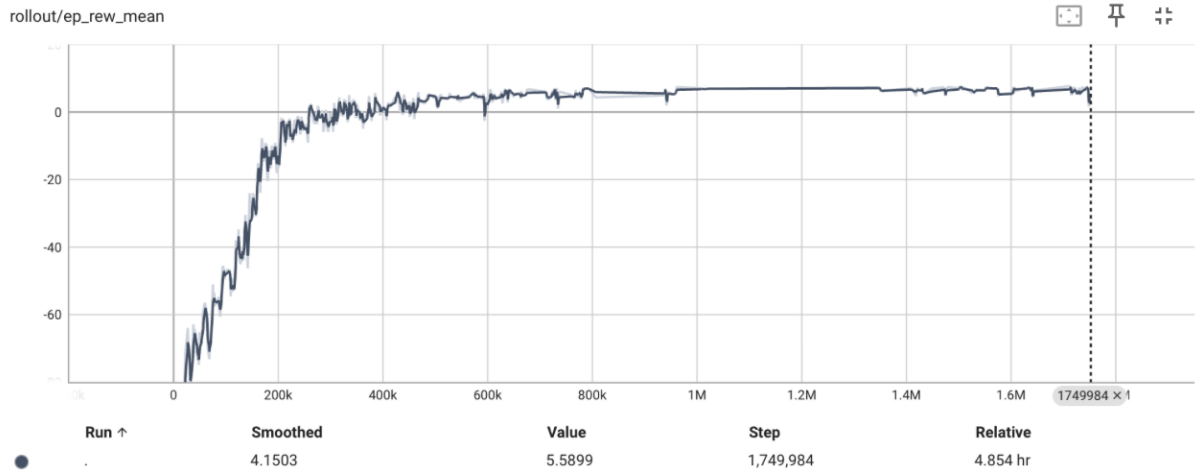

Figure 29: SAC Curriculum Mean Reward per Episode

Figure 30: Comparison of mean reward per episode for SAC Strong (top) vs. SAC Curriculum (bottom). The agent initially struggles to find an optimal strategy against a strong opponent but the curriculum-trained agent's rewards start very low but improves quickly and stabilizes smoothly.

B.4 Entropy Coefficient

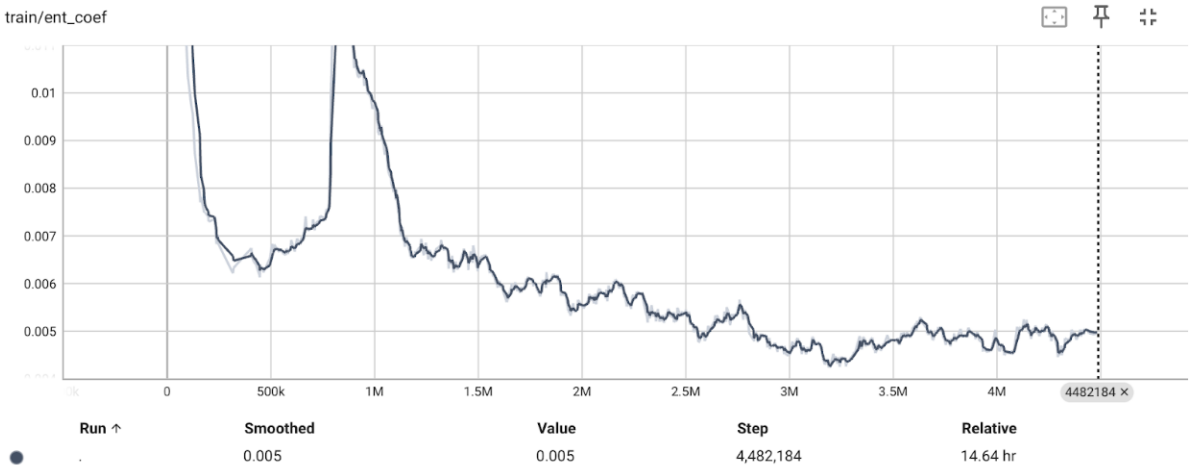

Figure 31: SAC Strong Opponent Entropy Coefficient

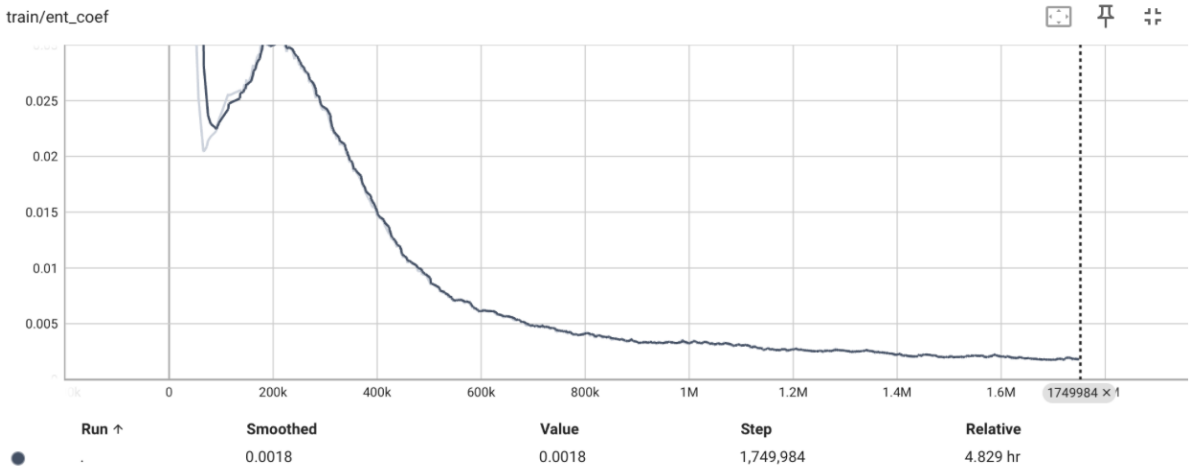

Figure 32: SAC Curriculum Entropy Coefficient

Figure 33: Comparison of entropy coefficient for SAC Strong (top) vs. SAC Curriculum (bottom). The curriculum-trained agent explores for a longer time, potentially leading to a more robust and adaptable policy. In contrast, the strong-opponent-trained agent reduces exploration too quickly, which may limit its ability to find optimal strategies in the long run. This suggests that curriculum learning encourages better long-term policy improvement by delaying premature exploitation.

Summary. Overall, these side-by-side comparisons indicate that the curriculum-trained SAC agent benefits from a more stable training trajectory across all four metrics. In contrast, the agent trained solely against a strong opponent faces higher variance and a slower convergence, particularly evident in the early training phase.