RL-Course 2025/26 Team: TheRewardWasALie

Ansel Cheung, Jannik Mänzer, Niklas Abraham

February 26, 2026

Abstract

This report presents our application and comparative study of modern Reinforcement Learning (RL) algorithms, specifically Twin Delayed Deep Deterministic Policy Gradient (TD3), Soft Actor-Critic (SAC), and the model based TD-MPC2, within the challenging Laser Hockey environment. We discuss the environment's design, its state and action spaces, and the unique characteristics making it a compelling testbed for RL research. Our work details the methodological approaches taken with these algorithms, summarizes key experimental findings, and reflects on lessons learned in this multi agent, continuous control setting.

1 Introduction

1.1 Environment Overview

The Laser Hockey environment [9] is a custom multi-agent reinforcement learning benchmark built upon the Gymnasium API and the Box2D physics engine. In this environment, two agents control hockey sticks and compete to score goals by hitting a puck into the opponent's net. Both agents receive mirrored observations at each timestep, simplifying the learning process by always letting the agent view itself as "player 1". The observation for each agent is a continuous state vector that describes the positions, velocities, and other relevant properties of the players and puck. Actions are represented as a continuous vector controlling the stick's movement, rotation, and puck release. Rewards are sparse and primarily based on scoring goals, with a small dense component for proximity to the puck to encourage exploration during training. Episodes end with either a goal or timeout. The environment supports both scripted bots and learning agents, and is designed to encourage robust, general strategies in a competitive, continuous control setting.

2 Method

2.1 Self-Play Infrastructure - Jannik Mänzer

Training reinforcement learning agents in multi-agent environments like air hockey presents several challenges, including catastrophic forgetting and the difficulty of evaluating progress once standard baselines are consistently defeated. Relying solely on standard self-play often leads to policies that overfit to recent behaviors, while a constantly changing opponent makes true improvement hard to measure. To address these issues, our approach relies on a diverse archive of opponents, including periodic checkpoints of our training agents, deterministic baselines, and agents trained using different RL algorithms. Sampling opponents from this archive during training forces the agent to adapt to various playstyles while ensuring robustness against older strategies. At the same time, it provides a stable population of opponents to evaluate the agent's overall skill.

2.1.1 TrueSkill Evaluation

To compare different agents in the archive we utilize the TrueSkill rating system [7]. TrueSkill models an agent's skill as a Gaussian distribution with mean μ_i and variance σ_i^2. The probability that agent i defeats agent j is calculated as:

$$P(i \text{ beats } j) = \Phi\left(\frac{\mu_i - \mu_j}{\sqrt{2\beta^2 + \sigma_i^2 + \sigma_j^2}}\right),$$

where β is the environment's inherent variance and Φ is the cumulative distribution function of the standard normal distribution. During training, these ratings are dynamically updated. This running rating serves as both the target for the skill-based matching and a continuous evaluation metric for the active agent. To prevent rating drift, the archive can be periodically recalibrated by executing comprehensive round-robin tournaments among the saved agents.

2.1.2 Opponent Selection

During training, opponents are selected using a mixture of different strategies: Prioritized Fictitious Self-Play (PFSP) [12] and skill-based matching ensure competetive matches, additionally random sampling and deterministic baselines are included to ensure fundamental competencies are maintained and older strategies are not forgotten. While skill-based matching and PFSP both aim to maximize learning signal by selecting opponents at a similar skill level, they differ in how that skill level is determined. Skill-based matching relies on a global skill rating to provide generally competitive games against opponents of similar strength. In contrast, PFSP tracks empirical, pairwise win rates, allowing it to also capture relative dynamics between different agents. We find PFSP to outperform skill-based matching in terms of sample efficiency.

For PFSP we track the empirical win rate $W_{i,j} \in [0, 1]$ of the active training agent i against an archived opponent j using an Exponential Moving Average (EMA). After each match, the win rate is updated as $W_{i,j} \leftarrow (1 - \alpha)W_{i,j} + \alpha r$, where α is a smoothing factor and r is the current result, with $r \in \{0, 0.5, 1\}$ for a loss, draw, or win respectively.

The probability of selecting opponent j from the available archive pool $\mathcal{A}$ is calculated by normalizing a priority function f:

$$P(\text{select } j) = \frac{f(W_{i,j})}{\sum_{k \in \mathcal{A}} f(W_{i,k})}.$$

For our priority function, we utilize $f(x) = x(1 - x)$. This parabola peaks exactly at a 50% win rate and approaches zero as the win rate nears 0% or 100%. Consequently, the matchmaker actively selects the opponents that provide the most informative learning signal. If the active agent completely masters an opponent, its selection priority naturally diminishes. Conversely, if the agent experiences catastrophic forgetting and begins losing to an older opponent, the empirical win rate decays back toward 0.5, automatically pulling that opponent back into the active training curriculum until a counter-strategy is successfully relearned.

2.2 TD-MPC 2 - Niklas Abraham

TD-MPC2 [6] is a model-based RL algorithm that learns a world model and selects actions through planning. The goal is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes expected discounted return in an infinite-horizon MDP.

For planning, TD-MPC2 employs the Model Predictive Control (MPC) framework, optimizing over action sequences of finite horizon H:

$$\pi(s_t) = \arg\max_{a_1,\ldots,a_H} \mathbb{E}_\pi \left[\sum_{\tau=0}^{H} \gamma^\tau r(s_{t+\tau}, a_{t+\tau}) \right]. \tag{1}$$

The return of each trajectory is estimated by simulating action sequences through the learned world model. However, this often leads to locally optimal policies, so TD-MPC2 additionally uses a value function to guide planning toward more globally optimal solutions.
Rather than predicting raw future observation states, TD-MPC2 learns to predict a maximally useful latent representation for accurately estimating the outcomes of action sequences. The algorithm is composed of five distinct neural network components that interact in a coordinated manner, as illustrated in Figure 1 a).

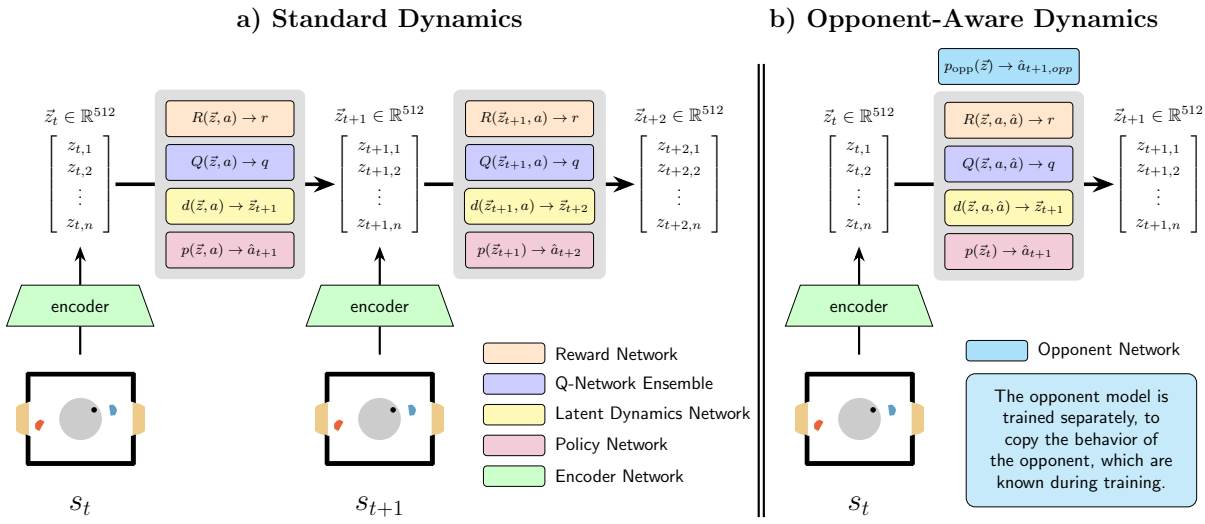

Figure 1: **(a)** TD-MPC2 agent architecture and their individual components. **(b)** The modified architecture with an additional opponent network to model the behavior of the adversarial agent in the Laser Hockey environment.

- **Encoder:** Maps the observed state s to a 512-dimensional latent vector $\vec{z} = h(s)$.
- **Latent Dynamics:** Predicts the next latent $\vec{z}_{t+1}$ from current latent and action: $\vec{z}_{t+1} = d(\vec{z}_t, a)$.
- **Reward Head:** Estimates reward r for a given $(\vec{z}, a)$ pair: $r = R(\vec{z}, a)$.
- **Termination Head:** Predicts early episode end, e.g., when a goal is imminent.
- **Q-Network Ensemble:** An ensemble (5 networks) of Q-functions estimating value $q = Q(\vec{z}, a)$. The minimum of two sampled networks reduces value overestimation.
- **Policy Network:** Guides action selection in planning: $p(\vec{z}, a) \to \hat{a}$.

2.2.1 Architecture and Training

All network components are multi-layer perceptrons (MLPs) with Mish activations. As in [6], the latent representation $\vec{z}$ is projected into L-dimensional simplices via a softmax to stabilize training and enforce sparsity.

Training uses an experience replay buffer $\mathcal{B}$ with full episode trajectories. Model parameters are optimized over sampled subsequences of length $H+1$ from $\mathcal{B}$ by minimizing a joint loss for dynamics, reward, and value prediction:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})_{t=0}^{H}\sim\mathcal{B}}\left[\sum_{t=0}^{H}\lambda^t\left(\|\vec{z}_{t+1} - \text{sg}(h(s_{t+1}))\|_2^2 + \text{CE}(\hat{r}_t, r_t) + \text{CE}(\hat{q}_t, q_t)\right)\right], \quad (2)$$

where $\text{sg}(\cdot)$ is stop-gradient, $\vec{z}_{t+1} = d(\vec{z}_t, a_t)$ is the predicted next latent, $\hat{r}_t = R(\vec{z}_t, a_t)$, $\hat{q}_t = Q(\vec{z}_t, a_t)$, and λ is a temporal discount factor. The Q-value target is $q_t = r_t + \gamma\bar{Q}(\vec{z}_{t+1}, p(\vec{z}_{t+1}, a_{t+1}))$, using an EMA of Q-net parameters ($\bar{Q}$) for stability. Following TD-MPC2, reward and value predictions are regressed in a log-transformed space with cross-entropy loss and soft targets.

The policy p is optimized according to a maximum entropy RL objective:

$$\mathcal{L}_p(\theta) = \mathbb{E}_{(s_t,a_t)_{t=0}^{H}\sim\mathcal{B}}\left[\sum_{t=0}^{H}\lambda^t\left(\alpha Q(\vec{z}_t, p(\vec{z}_t, a_t)) - \beta\mathcal{H}(p(\cdot|\vec{z}_t))\right)\right], \quad (3)$$

where $\vec{z}_{t+1} = d(\vec{z}_t, a_t)$ with $\vec{z}_0 = h(s_0)$, and $\mathcal{H}(p(\cdot|\vec{z}_t))$ is the policy entropy. Hyperparameters α and β balance value maximization and entropy, preventing premature collapse to deterministic policies.

2.2.2 Planning with MPPI

For local planning, TD-MPC2 leverages Model Predictive Path Integral (MPPI) control [13], sampling action sequences with guidance from the policy network. At each step, it estimates $\mu^*, \sigma^* \in \mathbb{R}^{H\times m}$, the mean and standard deviation of a multivariate Gaussian that maximizes expected return:

$$\mu^*, \sigma^* = \arg\max_{\mu,\sigma}\mathbb{E}_{a_{t:t+H}\sim\mathcal{N}(\mu,\sigma^2)}\left[\gamma^H Q(\vec{z}_{t+H}, a_{t+H}) + \sum_{\tau=t}^{H}\gamma^\tau R(\vec{z}_\tau, a_\tau)\right]. \quad (4)$$

This is optimized by iteratively sampling actions from $\mathcal{N}(\mu, \sigma^2)$, evaluating their returns, and updating μ and σ based on weighted top samples. The termination model predicts early ends in sampled rollouts. To speed up convergence, a fraction of samples comes from the policy p, and μ, σ are initialized from the previous step.

2.2.3 Modifying TD-MPC2: Opponent-Aware Dynamics

In classical TD-MPC2, the dynamics model receives only the current latent state and self-action, not the opponent's action. The standard model thus implicitly marginalizes over opponent behavior:

$$P(s_{t+1} \mid s_t, a_t^{\text{self}}) = \mathbb{E}_{a_t^{\text{opp}}\sim\pi_{\text{opp}}(\cdot|s_t)}P(s_{t+1} \mid s_t, a_t^{\text{self}}, a_t^{\text{opp}}), \quad (5)$$

yielding a mean-opponent model with biased long-horizon predictions when the opponent policy is multimodal or strategic. Varying opponents or using self-play alone does not resolve this; the network architecture must be extended.

To model the opponent's behavior, we extend the dynamics model so that it takes the opponent action as an additional input: $\vec{z}_{t+1} = d(\vec{z}_t, a_t^{\text{self}}, a_t^{\text{opp}})$, where a_t^{opp} is the opponent's action. The same is done for the reward model and Q-value network.
With opponent-aware models, evaluating a candidate self-action sequence $a_t^{\text{self}}, \dots, a_{t+H-1}^{\text{self}}$ uses the following planning objective and rollout. The return is

$$\sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} R(\vec{z}_\tau, a_\tau^{\text{self}}, \hat{a}_\tau^{\text{opp}}) + \gamma^H \tilde{V}(\vec{z}_{t+H}), \tag{6}$$

where the latent trajectory and predicted opponent actions are obtained by the recursion

$$\hat{a}_\tau^{\text{opp}} = \pi_{\text{opp}}(\vec{z}_\tau), \qquad \vec{z}_{\tau+1} = d(\vec{z}_\tau, a_\tau^{\text{self}}, \hat{a}_\tau^{\text{opp}}), \tag{7}$$

for $\tau = t, \dots, t+H-1$, with $\vec{z}_t = h(s_t)$. The terminal value $\tilde{V}(\vec{z}_{t+H})$ is given by the Q-ensemble (e.g., $\min_k Q_k(\vec{z}_{t+H}, a)$ at the policy action a). MPPI then maximizes this return over sampled self-action sequences, with opponent actions fixed by the recursion above.
The opponent's action is predicted by a separate network π_{opp}, trained via imitation on actions logged in the replay buffer (Figure 1 b). We train opponent models against the basic weak/strong bots, the TD3 agent (Section 2.4), the SAC agent (Section 2.3), and a standard TD-MPC2 agent. Since both policies are controlled during data collection, a_t^{opp} is recorded exactly. The separate loss is given by

$$\mathcal{L}_{\text{opp}} = \|a_t^{\text{opp}} - \pi_{\text{opp}}(\vec{z}_t)\|^2, \tag{8}$$

where π_{opp} is the opponent network, trained with a frozen encoder in periodic intervals. With this MSE objective, π_{opp} is a deterministic mean predictor: it outputs a single action estimate per latent state.
During inference the opponent action is predicted deterministically as $\hat{a}_t^{\text{opp}} = \pi_{\text{opp}}(\vec{z}_t)$ and fed into the dynamics. When multiple opponent models are available, each MPPI rollout is assigned a fixed model for the full horizon, so diversity comes from varying the model rather than sampling stochastically.

2.3 SAC - Jannik Mänzer

Soft Actor-Critic (SAC) [5] is an off-policy actor-critic algorithm that extends standard reinforcement learning by optimizing a maximum entropy objective. Rather than seeking only the maximum cumulative reward, the agent aims to maximize a weighted objective of reward and the entropy of the policy $\mathcal{H}(\pi(\cdot|\mathbf{s}_t))$ over trajectories $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots)$ generated by the policy π:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{\tau\sim\pi}[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha\mathcal{H}(\pi(\cdot|\mathbf{s}_t))]. \tag{9}$$

This entropy term $\mathcal{H}$ encourages the policy to assign non-zero probability to multiple actions where optimal, preventing premature convergence to deterministic behavior and improving exploration. The temperature parameter α controls the trade-off between the reward and the entropy.
Two soft Q-functions, Q_1 and Q_2, are trained to estimate the expected return plus the future entropy of the policy. To mitigate overestimation, Clipped Double-Q Learning [4] is used, where the target is calculated using the minimum of two target networks $\bar{Q}_{1,2}$, whose weights are obtained via an exponential moving average of the main Q-networks [10]. The target for the Q-function update is given by:

$$y_t = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\mathbb{E}_{\mathbf{a}_{t+1}\sim\pi}\left[\min_{j=1,2} \bar{Q}_j(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \alpha \log \pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})\right]. \tag{10}$$

The parameters θ are updated by minimizing the squared error between the prediction and this entropy-augmented target:

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_i(\mathbf{s}_t,\mathbf{a}_t) - y_t\right)^2\right] \quad \text{for } i \in \{1,2\}. \tag{11}$$

The policy π_ϕ is updated to maximize the value estimate provided by the Q-functions while maintaining high entropy. Instead of directly sampling actions from the stochastic policy $a_t \sim \pi_\phi(\cdot|s_t)$, the reparameterization trick [8] allows to express the action as a deterministic function of an independent noise source $\epsilon_t \sim \mathcal{N}(0, I)$ and the state: $\mathbf{a}_t = f_\phi(\epsilon_t; \mathbf{s}_t) = \mu_\phi(\mathbf{s}_t) + \sigma_\phi(\mathbf{s}_t) \odot \epsilon_t$, effectively moving the stochasticity outside the network, thus allowing for backpropagation. The policy objective is then to maximize:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}\left[\min_{j=1,2} Q_j(\mathbf{s}_t, f_\phi(\epsilon_t;\mathbf{s}_t)) - \alpha \log \pi_\phi(f_\phi(\epsilon_t;\mathbf{s}_t)|\mathbf{s}_t)\right]. \tag{12}$$

2.3.1 Automatic Entropy Tuning

Instead of choosing the temperature hyperparameter α manually, it can be tuned automatically to ensure the policy satisfies a minimum target entropy constraint $\bar{\mathcal{H}}$ (typically chosen as $-\dim(\mathcal{A})$, where $\dim(\mathcal{A})$ is the dimensionality of the action space). This continuously adapts the "exploration pressure" during training:

$$J(\alpha) = \mathbb{E}_{\mathbf{a}_t\sim\pi_t}[-\alpha(\log \pi_t(\mathbf{a}_t|\mathbf{s}_t) + \bar{\mathcal{H}})]. \tag{13}$$

2.3.2 Colored Action Noise

Standard SAC samples the reparameterization noise ϵ_t from an independent Gaussian distribution (white noise). While this uncorrelated sampling, characterized by a flat power spectral density ($S(f) \propto 1$), is standard practice, it may overly restrict the agent to local exploration. Because the noise fluctuates independently at each timestep, the agent rarely commits to a single direction for an extended period, which can hinder comprehensive state-space coverage.
To facilitate broader exploration, temporally correlated noise can replace the independent samples ϵ_t. This noise is defined by a power spectral density inversely proportional to frequency:

$$S(f) \propto \frac{1}{f^\beta} \tag{14}$$

where β dictates the noise color. Eberhard et al. [3] therefore propose using pink noise ($\beta = 1$) as a sensible default to balance local and global exploration.

2.3.3 Experiments

Exploration Strategies: To evaluate exploration, we compared standard white noise against temporally correlated pink and red noise, as shown in Figure 2 (top). The results demonstrate that white noise achieved slightly better cumulative rewards and a higher overall win rate while requiring fewer total steps, indicating superior sample efficiency. Consequently, we retained white noise, relying on SAC's entropy tuning to regulate local exploration.
Value Propagation and Credit Assignment: To analyze credit assignment, Figure 2 (bottom) shows Q-values relative to the temporal distance from the goal during both early and late training stages. The heatmaps reveal that the value signal diminishes rapidly for states temporally distant from the goal. We hypothesized that introducing dense proxy rewards might extend this effective horizon, but they resulted

in an identical propagation depth. We suggest this limited horizon stems from the inherent uncertainty of the multi-agent setting: unpredictable opponent behavior renders long-term value estimation unreliable. Furthermore, during extended runs, the Q-value landscape eventually stabilizes without propagating further backward in time. Yet, overall evaluation performance continues to improve significantly long after this point. This indicates that even after the Critic establishes a stable local state evaluation, the Actor still requires substantial further interaction to fine-tune its policy.

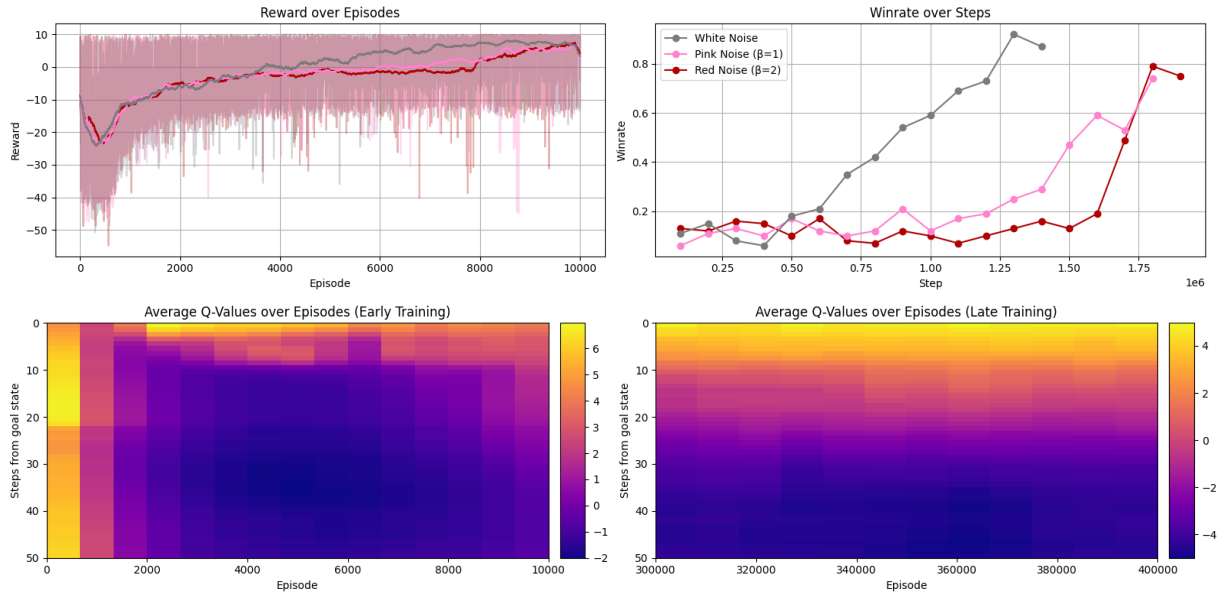

Figure 2: **(top)** Cumulative reward and win rate across different exploration noise types over the first 10,000 episodes against the strong bot (100-episode moving average; raw rewards in the backgound). **(bottom)** Q-value propagation over temporal distance to the goal in early vs. late training stages.

2.4 TD3 and REDQ - Ansel Cheung

2.4.1 The Overestimation Problem

In Standard Actor-Critic methods with continuous action spaces, the critic is responsible for approximating the Q function $Q_\theta(s, a)$ which can overestimate or underestimate the true value [4]. The overestimation of value will cause the Actor to udpate its parameters to select the overestimated action. This in turn causes the critic to use this artificially inflated Q-value from the next state, and ultimately pulls the current Q-value upwards. This cyclical overestimation of actions and Q-value backpropagates and leads to divergent behavior or suboptimal policies in continuous control over a long training timeframe. The predicted Q-values completely deviate from the actual expected returns.

2.4.2 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 [4] aims to fix the overestimation problem in Standard Actor-Critic methods using 3 tricks. The first trick of TD3 is the **Clipped Double-Q Learning**. TD3 maintains two independent critics, Q_1 and Q_2, and uses the minimum value to calculate the Bellman target:

$$y = r + \gamma \min_{i=1,2} Q_i(s', \tilde{a}) \tag{15}$$

These 2 Critics have the exact same architecture but independent sets of weights which have different random initializations. This allows the 2 networks to have different and more varied Q-values. The min operator is used when predicting the Q-value which acts as a pessimistic bound that should disrupt the aforementioned overestimation of Critics. The Actor is then given a much more conservative Q-value estimation.
The second trick of TD3 is **Target Policy Smoothing**. In continuous action spaces, the Critics have a rough Q-value landscape laced with sharp peaks.This causes the Critics to overestimate the Q-value and the Actor exploits these peaks. Hence, TD3 injects small random noise into the Actor's predicted action when updating the Critics.

$$\tilde{a} = \text{clip}(\pi_{\phi_{targ}}(s') + \epsilon, a_{low}, a_{high}), \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c) \tag{16}$$

Clipping is performed to ensure the noisy action does not exceed the envrironment's valid action minimum and maximum. The Actor cannot rely on sharp pinpoints of Q-function overestimation and will learn to maximize actions in broad and stable regions of high Q-values.
The third and final trick of TD3 is **Delayed Policy Updates**. Instead of updating the Actor at every step, the Actor π_ϕ is updated less frequently than the critics (e.g., a $d = 2$ delay). During early parts of training, the Actor is trying to converge towards a target which is unstable. Delaying the updates to the Actor allows the Critics' Q-value landscape to converge and settle, before updating the Actor to a more stable target.

2.4.3 Prioritized Experience Replay (PER)

In a normal buffer, the agent stores its experiences and samples them with a uniform distribution. All samples are equally likely to be sampled. The agent would learn a lot faster if it sees catastrophic failures and surprises/interesting experiences [11]. We can measure this surprise factor with TD error (δ):

$$\delta_i = y_i - Q(s_i, a_i) \tag{17}$$

A low TD error near zero means that the Critic has predicted the Q-value perfectly, and there is nothing much to learn. A high absolute TD error ($|\delta_i|$) means that the Critic has made a wildly wrong prediction. These are the experiences we want to sample more often.
The priority p_i of transition i is defined by its TD error $|\delta_i|$. The sampling probability is given by:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad \text{where } p_i = |\delta_i| + \epsilon \tag{18}$$

ϵ is a small constant to ensure the probabilities do not ever reach 0. α and β the tunable hyperparameter of how much prioritization is used and how it is incremented.
The data distribution that the model sees is being altered by changing the sampling and this introduces bias which might intefere with the gradients. To correct the bias introduced by frequent sampling of high transitions with high TD errors, the loss is weighted by w_i:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{19}$$

where β is annealed from $\beta_0 \rightarrow 1$ over the duration of training.
The implementation for PER utilizes a SumTree data structure to reduce the sampling and updating of priorities in $\mathcal{O}(\log n)$ time.

2.4.4 Randomized Ensembled Double Q-Learning (REDQ)

REDQ [1] can be used to take TD3 to the next level.
Trick 1: **Sample Efficiency via Update-toData (UTD)**. REDQ aims for a high Update-to-Data (UTD) ratio $G \gg 1$, allowing the agent to perform many gradient steps per environment interaction. Using a high UTD ratio on vanilla TD3 make the Actor and Critics learn from the same small batch of transitions aggressively, causing approximation errors to compound and overestimation to occur.
Trick 2: **Ensemble Minimum**. In order to continue to fix the overestimation problem, REDQ uses an even more pessimistic estimation. REDQ maintains an ensemble of N Q-functions (Critics). For each update, using a minimum of all Critics might cause large overestimation bias. Instead, a random subset $\mathcal{M}$ of size M is sampled to compute the target. θ is the policy parameters, $\phi_i, i = 1, \ldots, N$ is Q-function parameters.

$$y = r + \gamma \min_{j \in \mathcal{M}} Q_{\phi_{j,targ}}(s', \tilde{a}') \tag{20}$$

where $\tilde{a}'$ is the same clipped action from equation 16 (**??**). This single target is used to update all N critics. Using N Critics with different initializations allows the approximation error to be even more uncorelated compared to TD3 with 2 Critics.
Trick 3: **Policy Update**. The actor is updated to maximize the mean Q-value across all N Critics. The mean of all N Critics provides a much smoother and reliable signal for the Actor to follow.

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \frac{1}{N} \sum_{i=1}^{N} Q_{\phi_i}(s, \pi_\theta(s)) \tag{21}$$

3 Results

3.1 TD-MPC2 Hyperparameters and Curriculum

We evaluated TD-MPC2 with horizons of 4, 6, 8, 10, and 12, using a learning rate of 0.0003, batch size 512, three 256-unit network layers, latent dim 256, 5 Q-networks, γ=0.99, temperature 0.5, $v_{\text{min}} = -10$, $v_{\text{max}} = 10$, win reward bonus 10, and win reward discount 0.92. Training used 4000 episodes with a basic strong opponent. Results are shown in Figure 3. We also tested "opponent aware dynamics" (with three internal opponent models: TD-MPC2 without opponent-aware dynamics, SAC agent, and a DECOYPOLICY agent trained to mimic the strong opponent), shown in the same figure.

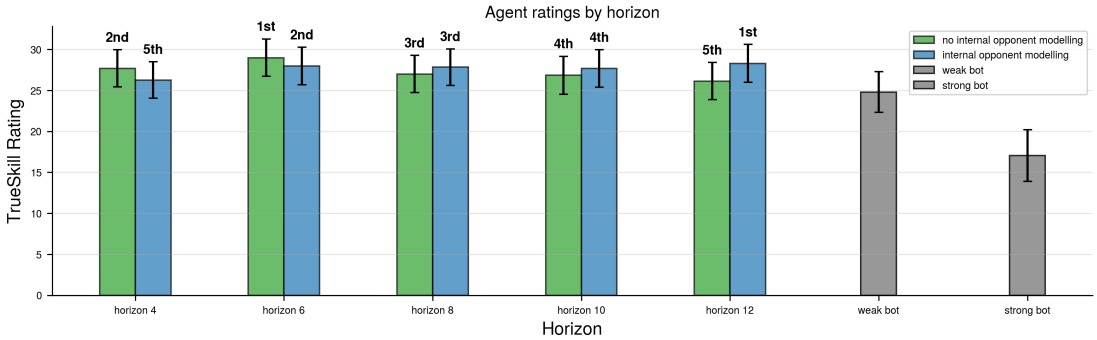

Figure 3: TrueSkill ratings of the TD-MPC2 agent with different horizons and different opponent aware dynamics.

3.2 SAC Hyperparameters & Curriculum

While we evaluated various hyperparameter configurations during the development of the SAC agent, we found that scaling the total number of training steps and relying on a robust curriculum yielded the most significant performance gains.

The training process was divided into distinct phases to maximize sample efficiency. Initially, the agent was pretrained against the deterministic baselines for 10,000 episodes to learn basic movement patterns and puck-handling skills in a stable environment. Following this pretraining, the agent transitioned to the archive-based matchmaking system for an extensive training period of 300,000 episodes.

During this self-play phase, opponent selection relied heavily on the Prioritized Fictitious Self-Play (PFSP) and random sampling strategies discussed in **??**. We found that utilizing PFSP was a critical factor in scaling the agent's performance and achieving robust generalization. While simple skill-based matching based on a global skill rating provides generally competitive games against opponents of similar strength, PFSP's reliance on empirical, pairwise win rates allows it to capture relative dynamics between different agents. This pairwise nature forced the SAC agent to continuously adapt to diverse playstyles within the archive, preventing it from overfitting to any single strategy and significantly improving its generalization capabilities.

Our combined team agent submitted to the course tournament, which placed 7th overall, was a direct continuation of this curriculum. It utilized the exact same architecture and hyperparameters, but was trained for an additional 100,000 episodes within the self-play archive to further stabilize and refine its policy.

A comprehensive table detailing the final SAC hyperparameters, alongside supplementary ablation plots, is provided in Appendix **??**.

3.3 REDQ-TD3, PER Curriculum

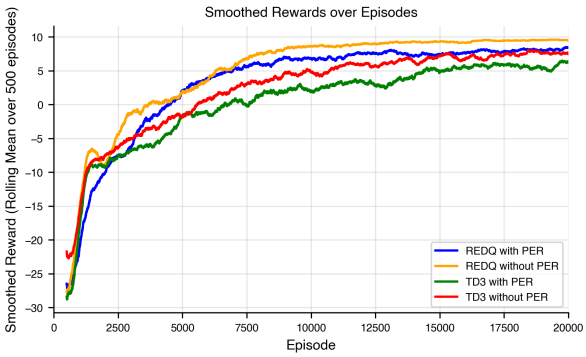

Figure 4: TD3 vs REDQ rewards

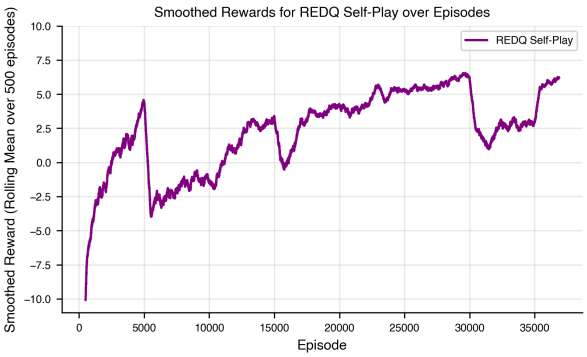

Figure 5: REDQ selfplay rewards trajectory

Figure 4 shows the comparison of REDQ, TD3 with and without PER. PER seems to degrade performance of both REDQ and TD3, while REDQ consistently improves the performance of TD3. In this experiment, we used default hyperparameters of TD3, REDQ and PER. PER was designed for discrete action spaces [11] while air hockey environment is a continuous action space. Retuning of hyperparameters is needed to make PER work. We attempted to tune the hyperparameters but ultimately was unable to achieve a good set of hyperparameters for PER for 2 reasons: (1) PER increases transition sampling and importance weight updating time from $\mathcal{O}(1)$ to $\mathcal{O}(\log n)$ and (2) REDQ increases training time massively due to having 10 Critics and high UTD ratio. Ultimately we decided to continue with REDQ-TD3 without PER. For REDQ-TD3 hyperparameters, an ensemble of $N = 10$ critics and sample of $M = 2$ critics for TD target worked best. Tuning the Polyak hyperparameter from 0.995, 0.9925, 0.990 did not change the performance significantly. The default hyperparameters worked the best. We ended up using

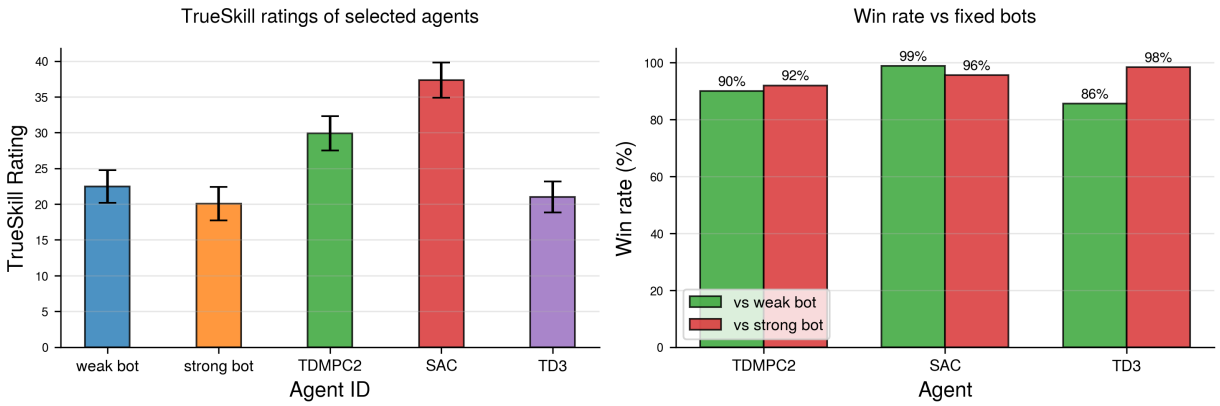

Figure 6: **(left)** TrueSkill ratings ($\mu - 3\sigma$) for a selected set of agents. **(right)** Win rates against the scripted weak and strong bots. Higher is better.

layer norm and gradient clipping for the Critics parameters. We used REDQ-TD3 (no PER) for selfplay (section 2.1) and archive opponents as well. Each drop in rewards shows a change in training phase and opponent sampling.

3.4 Overall Results

To compare all trained agents on a common scale, we evaluated them in a round-robin tournament within the archive matchmaking system using TrueSkill [7], a Bayesian rating algorithm that updates the belief over each agent's skill level after every match outcome, as discussed in **??**. The rating reported is $\mu - 3\sigma$, a conservative lower-bound estimate that accounts for residual uncertainty. Figure 6 shows the TrueSkill ratings and win rates against fixed baselines for the two scripted baselines (weak and strong bot), the best SAC agent, the best TD-MPC2 agent, and the best TD3 agent.

All three learned agents (SAC, TD-MPC2, and TD3) consistently beat both the strong and the weak bot, with win rates against each baseline ranging from 86% to 99%. SAC achieves the highest TrueSkill rating of 37.32 ($\mu = 39.78$, $\sigma = 0.82$, over 391 matches), followed by the best TD-MPC2 agent at 29.90 ($\mu = 32.33$, $\sigma = 0.81$, trained for 16 000 steps). TD3 reaches 21.01 ($\mu = 23.17$, $\sigma = 0.72$), placing it close to the two scripted baselines: the weak bot at 22.46 and the strong bot at 20.06.

Win rates against the fixed baselines confirm these rankings. SAC dominates both scripted opponents with a 99% win rate against the weak bot and 96% against the strong bot. TD-MPC2 achieves 98% against the weak bot and 90% against the strong bot. TD3 obtains 86% against the weak bot but 98% against the strong bot, indicating that it has developed a strategy effective against the strong bot's predictable aggression while struggling to score decisively against the weak bot's passive style. Overall, SAC is the strongest agent in our comparison, with TD-MPC2 (16 000 training steps, $H{=}8$) as the runner-up and a clear margin over TD3 and the scripted baselines.

4 Acknowledgements & Data Availability

We would like to thank the instructors and the staff of the Reinforcement Learning course for their help and support. All of our code can be found in our GitHub repository [2].

4.1 AI Usage Disclosure

During the preparation of this project, we utilized generative AI tools for code autocompletion and the automatic generation of code documentation. Additionally, these tools were used as a linguistic aid for grammar and spell-checking during the drafting of this report, and for plotting. We maintain full responsibility for the technical content, the final implementation of the algorithms, and the results presented herein.

References

[1] X. Chen, C. Wang, Z. Zhou, and K. Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021.

[2] A. Cheung, J. Mänzer, and N. Abraham. Rl-course 2025/26: Final project report. `https://github.com/NiklasAbraham/RL_CheungMaenzerAbraham_Hockey`, 2026.

[3] O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*, May 2023.

[4] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018.

[5] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications, 2019.

[6] N. Hansen, H. Su, and X. Wang. Td-mpc2: Scalable, robust world models for continuous control, 2024.

[7] R. Herbrich, T. Minka, and T. Graepel. TrueSkill™: A bayesian skill rating system. In *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2007.

[8] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022.

[9] G. Martius. Hockey environment. `https://github.com/martius-lab/hockey-env`, 2023.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

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

[12] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. J. Dudzik, J. Chung, D. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350 – 354, 2019.

[13] G. Williams, A. Aldrich, and E. Theodorou. Model predictive path integral control using covariance variable importance sampling, 2015.

A Appendix

A.1 Episode Logs

The episode logs of the TD-MPC2 agent with the horizon 4 without the opponent aware dynamics are shown in Figure 7. These logs were plotted for all runs periodically, and this example is representative for the other runs.

A.2 TD-MPC2 Profiling Summary

Profiling was run on a single NVIDIA GeForce RTX 2080 Ti (11.3 GB) with 50 iterations. Table **??** gives the agent and run configuration. Table **??** summarizes single-step action selection cost. Table **??** reports CPU and CUDA time for each model forward pass (30 calls each).

Table 1: Profiling run setup.

Parameter	**Value**
Device	CUDA (NVIDIA GeForce RTX 2080 Ti, 11.3 GB)
Observation dimension	18
Action dimension	8
Latent dimension	512
Horizon H	16
MPPI samples	512
MPPI iterations	6

Table 2: Single action selection: total CPU and CUDA time per call.

Metric	**Time**
Self CPU total	8.75 s
Self CUDA total	2.08 s

Table 3: Model forward pass: CPU and CUDA time over 30 calls.

Component	**CPU total**	**CUDA total**
Encoder	1.13 s	0.74 ms
Dynamics (single)	1.15 s	0.91 ms
Dynamics (batch)	1.16 s	6.57 ms
Reward (batch)	4.10 s	19.0 ms
Q-ensemble (batch)	72.8 ms	15.5 ms

Errors during profiling. Batch action selection failed with a CUDAGraphs error (tensor overwritten by a subsequent run). The planning step and the training step could not be profiled: planning failed with CUDA out of memory (GPU fully used after single-action profiling); the training step produced an empty error.

A.3 Sparse Reward Problem and Reward Backpropagation

A.3.1 Problem: Reward Stagnation at Zero

The Laser Hockey environment provides an extremely sparse reward signal. An agent receives $+10$ only when it scores a goal, -10 only when the opponent scores, and a very small proximity bonus at every step. In practice, during early training when no learned strategy for scoring exists yet, the vast majority of episode steps yield rewards indistinguishable from zero.

This sparsity posed a critical bootstrapping problem for TD-MPC2. The value function $Q(\vec{z}, a)$ is trained on experience stored in the replay buffer. If nearly every transition has reward ≈ 0, the Q-targets are close to zero throughout, and the gradient signal for both the value function and the policy network vanishes. As a result, training stagnated: the total episode reward remained near zero and the agent never discovered that winning is achievable.

To verify this empirically, we simulated 100 episodes with uniformly random actions for both players and analysed the resulting reward distribution: regardless of whether an episode ends in a player-1 win, player-2 win, or draw, the per-step rewards are overwhelmingly clustered at zero. A thin spike at $+10$ (or -10) appears only for the single terminal step of decided episodes; the rest of each episode contributes essentially nothing to the value-function learning signal.

A.3.2 Solution: Reward Backpropagation

To provide a dense, temporally consistent learning signal without altering the environment's reward function, we implemented reward backpropagation in the replay buffer. Whenever an episode is flushed to the buffer and the outcome is a win (winner $= 1$), a discounted bonus is added to every preceding step before the experience is stored:

$$r_t \;\leftarrow\; r_t + b \cdot \gamma_b^{(T-2)-t}, \qquad t = 0,\, 1,\, \ldots,\, T-2, \tag{22}$$

where b is the win-bonus magnitude, $\gamma_b \in (0, 1]$ is the backpropagation discount, T is the episode length, and $t = T-1$ is the terminal step that already carries the full $+10$ reward and is therefore left unchanged. The exponential decay in Equation (22) is anchored at the *penultimate* step ($t = T-2$, which receives the full bonus b) and fades toward earlier steps, so the credit assignment is greatest just before the decisive action and smallest at the start of the episode.

We set $b = 10$ to match the magnitude of the terminal goal reward, and $\gamma_b = 0.82$ at the beginning of training. These values were chosen so that the injected signal is large enough to make winning episodes clearly distinguishable from non-winning ones in the Q-value landscape, while the discount prevents distant early steps from receiving an unrealistically large credit.

A.3.3 Phase-Out During Training

As training progresses and the agent begins to score goals regularly, the replay buffer naturally accumulates more winning episodes and the true sparse signal becomes a sufficient learning driver on its own. Keeping the synthetic bonus active at this stage would bias the value function away from the true environment signal and could prevent the agent from learning fine-grained credit assignment.

To address this, the bonus is gradually phased out over the course of training by scheduling γ_b to decay from its initial value toward zero as the number of training steps increases. In early training the dense bonus guides the value function; once the agent reliably produces winning episodes, the decay reduces the bonus to negligible levels and the agent transitions to optimising the original sparse reward.

A.3.4 Effect on TD-MPC2 Training

Figure 8 illustrates how reward backpropagation transforms the per-step reward signal of a representative winning episode. The top panel shows the raw reward trace (mostly zero with a single $+10$ spike) alongside the reshaped trace after backpropagation. The middle panel shows the additive bonus at each step, which decays exponentially from the penultimate step toward the start of the episode. The bottom panel confirms that the effect is systematic across all winning episodes: every episode whose total reward was originally concentrated in the single terminal step now receives a significantly higher total, shifting the Q-target distribution into a range where meaningful gradient updates can occur.
Empirically, applying this technique during the initial training phase broke the stagnation: the TD-MPC2 value function was able to associate early-episode states with eventual win outcomes, the policy gradient signal became non-zero, and training loss curves began to decrease. The gradual phase-out then ensured that the agent converged to a policy optimising the true environment reward rather than the synthetic shaped signal.

A.4 SAC Hyperparameters

Table **??** details the final hyperparameter configuration used for both the SAC agent and our combined team agent in the tournament.

Hyperparameter	**Value**
Optimizer	Adam
Learning Rate	10^{-3}
Discount Factor (γ)	0.99
Batch Size	256
Hidden Layers	2
Hidden Units per Layer	256
Activation Function	ReLU
Replay Buffer Size	10^6
Target Smoothing Coefficient (τ)	0.005
Target Entropy	$-\dim(\mathcal{A})$
Temperature Parameter (α)	Automatically Tuned
Exploration Noise	White Noise

Table 4: SAC Final Hyperparameters

As part of our initial evaluation, we conducted experiments on the impact of different learning rates on SAC's sample efficiency. Figure **??** shows win rates against the strong bot across the first 10,000 episodes for learning rates of 10^{-2}, $5 \cdot 10^{-3}$, 10^{-3}, and $3 \cdot 10^{-4}$ (default in the SAC paper). The results indicate that a learning rate of 10^{-3} achieved the best performance.

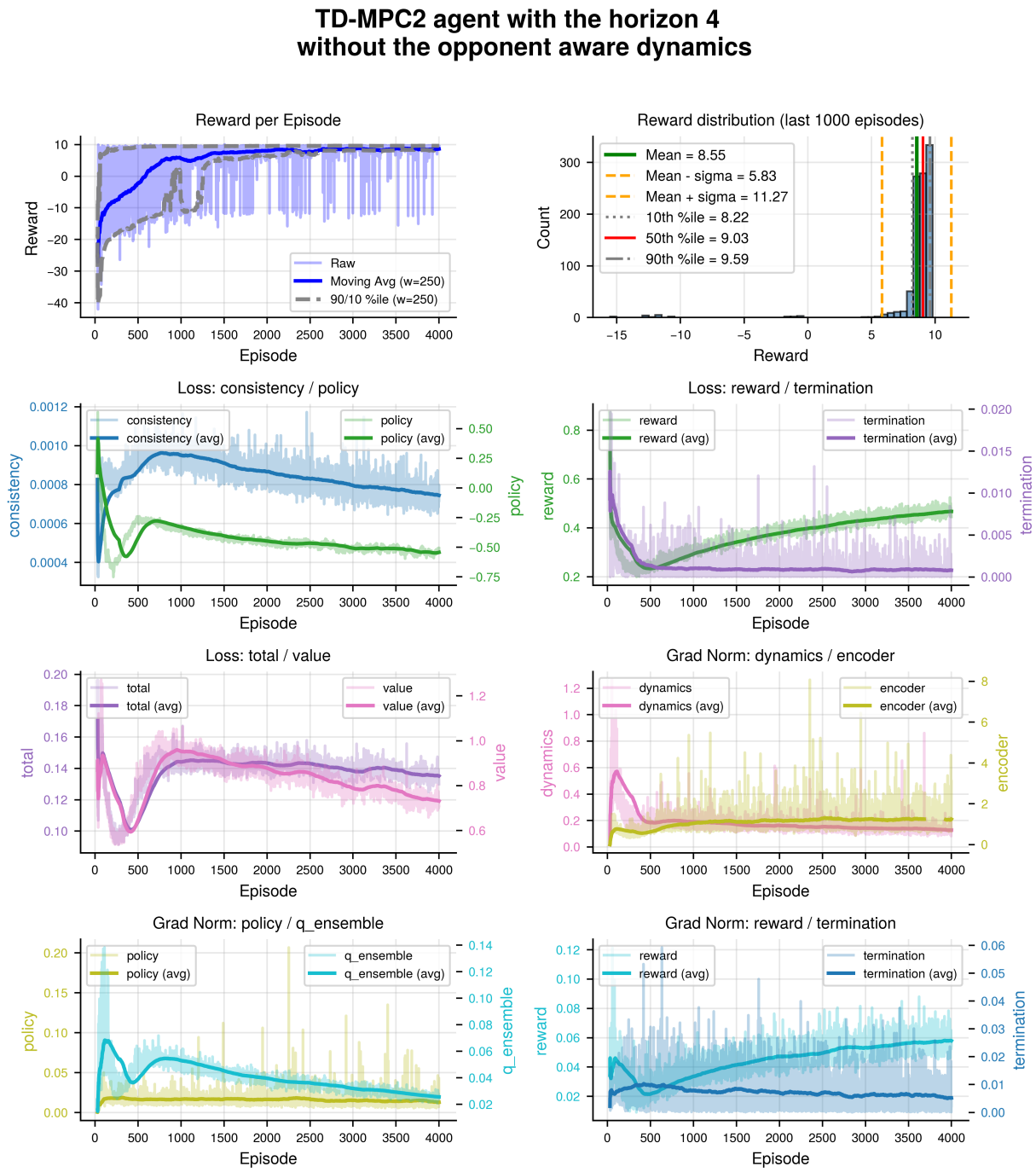

Figure 7: Episode logs of the TD-MPC2 agent with the horizon 4 without the opponent aware dynamics.

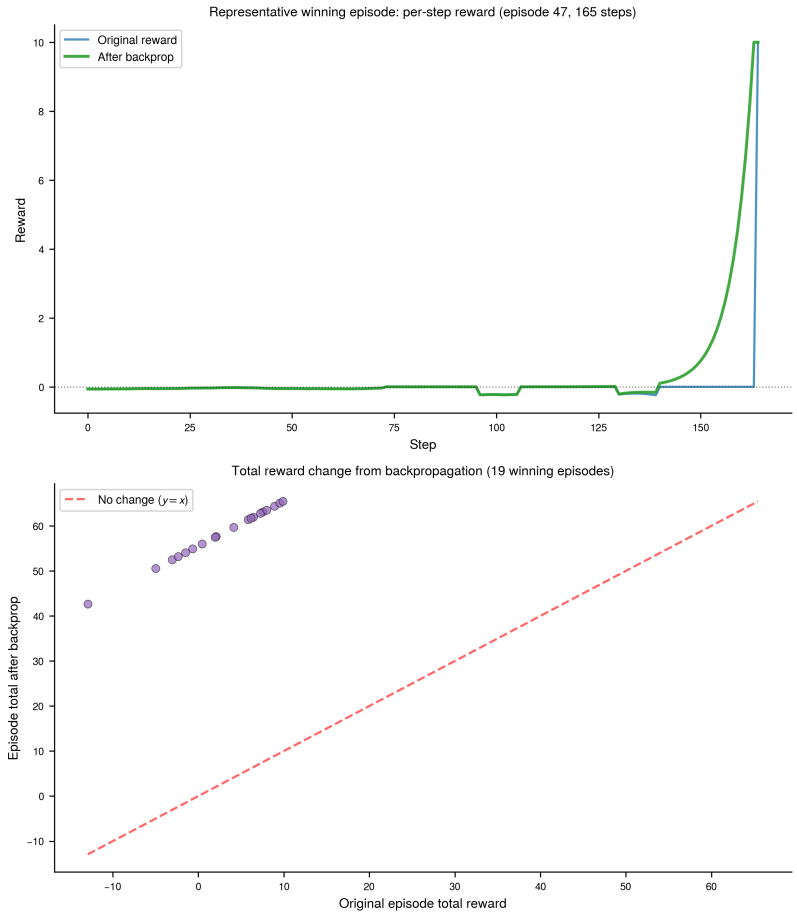

Figure 8: Effect of reward backpropagation on winning episodes (bonus $b = 10$, discount $\gamma_b = 0.82$). **Top:** per-step reward before and after backpropagation for a representative winning episode; the near-zero trace is replaced by a smoothly decaying bonus. **Bottom:** scatter of episode total rewards before versus after backpropagation across all winning episodes; every point lies above the identity line, confirming a consistent upward shift in the training signal.

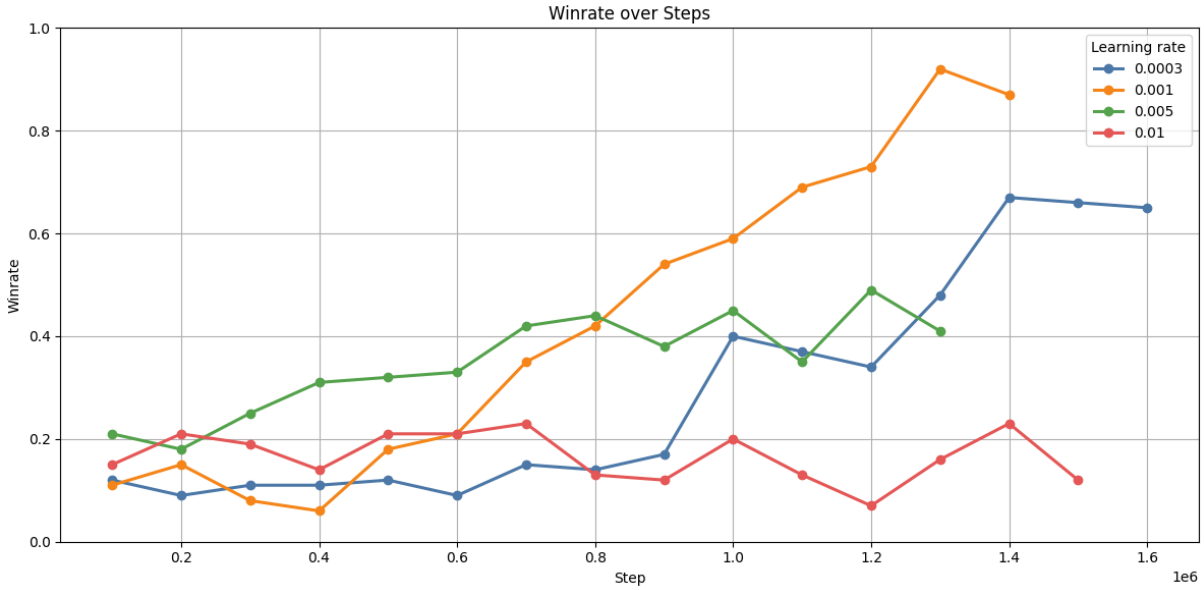

Figure 9: Comparison of SAC performance across different learning rates over the first 10,000 episodes against the strong bot.