RL-Course 2025/26: Final Project Report

Ansel Cheung, Jannik Mänzer, Niklas Abraham

February 23, 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 [4] is a custom reinforcement learning benchmark built on the Gymnasium Python API and powered by the Box2D physics engine. In this multi agent setting, two agents each control a hockey stick and compete to score goals by striking a puck into the opponent's net. The environment is designed such that both agents receive identical but mirrored observations at each timestep, which eliminates the need for side specific learning strategies.

At each timestep, the agent receives an 18 dimensional continuous state vector that captures the complete game state. This observation includes the position x_1, y_1 and orientation θ_1 of both players relative to the center of the field, along with their linear $v_{x,1}, v_{y,1}$ and angular ω_1 velocities. The state also contains the puck's position x_p, y_p and velocity $v_{x,p}, v_{y,p}$, as well as time remaining for each player's puck possession $t_{\text{puck},1}, t_{\text{puck},2}$ in keep mode, which ranges from 0 to 15 steps. Player 1 refers to the agent currently being controlled, while Player 2 represents the opponent. The observation structure ensures that when the viewpoint switches during training, the indices are automatically mirrored so that each agent always perceives itself as Player 1, maintaining a consistent learning perspective.

The action space consists of a 4 dimensional continuous vector that controls the agent's stick. The first two components specify forces applied for stick movement in the x and y directions, while the third component controls the torque applied to adjust the stick's angle. The fourth component is a thresholded scalar that determines whether to release the puck when the agent is in possession. Reward feedback in this environment is sparse and goal oriented. An agent receives a reward of +10 for scoring a goaland 0 for a draw. Additionally, there is a small reward signal for maintaining proximity to the puck, which helps guide exploration during early learning stages. Each episode begins with all entities positioned at the center of the field and terminates when a goal is scored or a timeout occurs. The environment supports both scripted opponents, such as the BasicOpponent, and learning agents as adversaries, making it a versatile and challenging benchmark for continuous control and multi agent reinforcement learning research.

2 Method

2.1 TD-MPC 2 - Niklas Abraham

TD-MPC2 [2] is a model-based reinforcement learning algorithm that learns a world model to predict future states and rewards, and uses this model to select actions through planning. The core idea is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ in a Markov Decision Process with an infinite horizon. The policy is constructed to maximize the expected discounted return. In TD-MPC2, this is achieved by learning a world model and selecting actions by planning with the learned models.
For planning, TD-MPC2 employs the Model Predictive Control (MPC) framework, in which actions are optimized based on planning over action sequences of a 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 approach often leads to only locally optimal policies. To address this limitation, TD-MPC2 additionally utilizes a value function to guide the planning process and improve the policy toward a more globally optimal solution.
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).

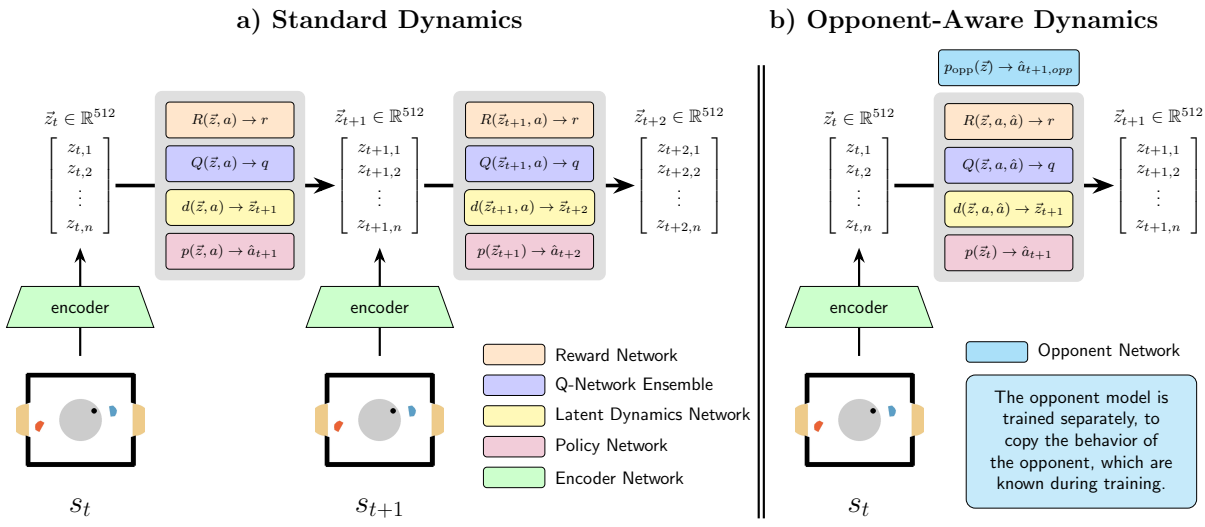

Figure 1: On the left in a) TD-MPC2 agent architecture and their individual components. b) shows the modified architecture with an additional opponent network to model the behavior of the adversarial agent in the Laser Hockey environment.

- **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: $p(\vec{z}, a) \to \hat{a}$.

2.1.1 Architecture and Training

All network components are multi-layer perceptrons (MLPs) with Mish activations. As in [2], 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.

2.1.2 Planning with MPPI

For local planning, TD-MPC2 leverages Model Predictive Path Integral (MPPI) control [5], 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.

2.1.3 Modifying TD-MPC2: Opponent-Aware Dynamics

In the classical TD-MPC2, the dynamics model is trained to predict the next latent state given the current latent state and action. However, in the multi agent setting, with an adversarial opponent, the dynamics

model will only receive the current latent state and action, but not receive the action of the opponent. Thus the standard TD-MPC2 implicitly models

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

which corresponds to marginalizing over opponent behavior. This leads to a mean-opponent model, producing biased long-horizon predictions when the opponent policy is multimodal or strategic. To address this, it is not enough to vary the opponents during training or to use self-play, the network architecture needs to be changed.

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 $a_t^{\text{self}}, \ldots, a_{t+H-1}^{\text{self}}$ 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 $\tau = t, \ldots, t+H-1$, 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 action of the opponent needs to be predicted with a separate network, which receives as input the current latent state and outputs the action of the opponent. This is illustrated in Figure 1 b). This requires the opponent network to be trained separately to imitate the opponent's behavior; the opponent's actions are available in the replay buffer from collected episodes. In the setting in this project, we could choose between different opponents: basic weak, basic strong, the TD3 agent [**?**] or the SAC agent [**?**] as well as the TD-MPC2 agent without the opponent-aware dynamics. During data collection we control both policies (self and opponent), therefore a_t^{opp} is logged exactly in the replay buffer. 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 training, opponent actions are taken from the replay buffer; during inference they must be predicted by the opponent network. We use the opponent action deterministically: $\hat{a}_t^{\text{opp}} = \pi_{\text{opp}}(\vec{z}_t)$. This predicted action is fed into the dynamics to obtain the next latent state (Figure 1 b). When multiple opponent models are used (e.g., different cloned policies), each rollout can be assigned a fixed opponent model for the full horizon; diversity across rollouts then comes from varying which opponent model is used, not from sampling different actions from a stochastic π_{opp}.

2.2 TD3 and REDQ - Ansel Cheung

2.2.1 The Overestimation Problem

In Standard Actor-Critic methods with continuous action spaces, the critic is responsible for approximating the Q function $Q_\theta(s, a)$. This means that they can overestimate or underestimate the true value.

Pairing with the actor, who is responsible for maximizing the critic's expected return, is sensitive to the critic's over/underestimation. However, 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.

2.2.2 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 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_{θ_1} and Q_{θ_2}, and uses the minimum value to calculate the Bellman target:

$$y = r + \gamma \min_{i=1,2} Q_{\theta_{i,targ}}(s', \tilde{a}) \tag{9}$$

These 2 Critics have the exact same architecture, but 2 independent sets of weights which have different (random) initializations. They start at different points of the loss landscape and are updated by stochastic gradient descent. This allows the 2 networks to have different and more varied Q-values. The min operator is used when predicting the Q-value. This 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. Underestimation might occur but this is much better than overestimation, since the Actor is already maximizing the expected critic's return, and will avoid picking an underestimated action. This means that underestimation errors do not propagate as dangerously as overestimation errors.
The second trick of TD3 is **Target Policy Smoothing**. In continuous action spaces, the Critics do not have a smooth Q-value landscape, it is usually laced with random sharp peaks. These peaks 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.

$$\tilde{a} = \text{clip}(\pi_{\phi_{targ}}(s') + \epsilon, a_{low}, a_{high}), \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c) \tag{10}$$

Clipping is performed to ensure the noisy action does not exceed the envrironment's valid action minimum and maximum. Noise allows the Critics to evaluate the possibly exploited Actor's action off the peak, which in the case where it is a sharp peak, means that a little noise will bring the Critic to a much flatter part of the landscape. The Actor, similarly cannot rely on sharp pinpoints of 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 training, especially the early parts, the Critics are highly inaccurate. Updating the Actor at every step drags down both the Critic and the Actors since the Actor updates its policy based on an unstable Critic, which updates its policy using the half-baked Actor. 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.

2.2.3 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. However, as the training progresses, the agent sees more and more of the same kind of experiences due to uniform sampling. The agent would learn a lot faster if it

sees catastrophic failures and surprises/interesting experiences. We can easure this surprise factor with TD error. The TD error (δ) is the difference between the target value (y) and the Critic's current Q-value prediction:

$$\delta_i = y_i - Q(s_i, a_i) \tag{11}$$

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:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad \text{where } p_i = |\delta_i| + \epsilon \tag{12}$$

ϵ is just a small constant to ensure the probabilities do not ever reach 0. α is the tunable hyperparameter of how much prioritization is used, $\alpha = 0$ for uniform sampling, $\alpha = 1$ for full prioritization of TD error. The data distribution that the model sees is being altered by changing the sampling and this introduces bias. These bias affect the expected values calculated by neural networks and 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:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{13}$$

where β is annealed from $\beta_0 \rightarrow 1$ over the duration of training. This scaling with importance weights scales down the gradients for transitions with high probability of being sampled.
The implementation for PER utilizes a SumTree data structure to reduce the sampling and updating of priorities in $\mathcal{O}(\log n)$ time.

2.2.4 Randomized Ensembled Double Q-Learning (REDQ)

REDQ can be used to take TD3 to the next level. If we can use 2 Critics in TD3, why not 10? REDQ also has a few tricks.
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. In the paper, a UTD ratio of 20 was used. However, using. high UTD ratio on vanilla TD3 will cause the overestimation problem to return. The Actor and Critics will be learning from the same small batch of transitions aggressively, causing approximation errors to compound.
Trick 2: **Ensemble Minimum**. In order to continue to fix the overestimation problem since a high UTD is used, 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:

$$y = r + \gamma \left(\min_{j \in \mathcal{M}} Q_{\theta_{j,targ}}(s', a') - \alpha \log \pi_\phi(a'|s') \right) \tag{14}$$

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. The pessimisim is further intensified, and is more dynamic, by using the minimum of a random subset of M 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.

$$\nabla_\phi J(\phi) = \nabla_\phi \mathbb{E}\left[\frac{1}{N} \sum_{i=1}^{N} Q_{\theta_i}(s, \pi_\phi(s)) - \alpha \log \pi_\phi(a|s) \right] \tag{15}$$

3 Results

3.1 TD-MPC2 Hyperparameters and Curriculum

To determine the optimal horizon for TD-MPC2, we trained a TD-MPC2 agent with different horizons and evaluated the performance. The horizons tested were 4, 6, 8, 10, and 12. The runs were done with the following hyperparameters: learning rate 0.0003, batch size 512, network size of three layers with 256 units each, a latent dimension of 256, 5 Q-networks, a gamma of 0.99, a temperature of 0.5, a vmin of -10, a vmax of 10, a win reward bonus of 10, and a win reward discount of 0.92. The runs were done with the following curriculum: 4000 episodes of full competency, with a basic strong opponent. The results are shown in Figure 2. Addtionally with the same hyperparameters, the modified opponent aware dynamics was tested, for this three internal opponent models were used: the TD-MPC2 agent without the opponent-aware dynamics, the SAC agent and the DECOYPOLICY agent, which was trained to mimick the basic strong opponent. The results are shown in the same figure, but with the label "opponent aware dynamics".

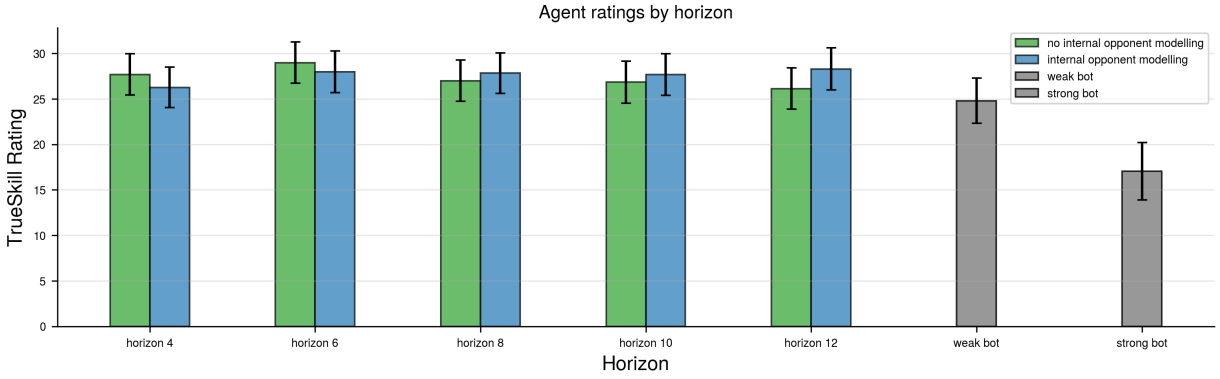

Figure 2: TrueSkill ratings of the TD-MPC2 agent with different horizons and different opponent aware dynamics.

3.2 REDQ-TD3 Hyperparameters and Curriculum

We followed the default hyperparameters from the REDQ paper: UTD Ratio $G = 20$, Ensemble Size $N = 10$, Subset Size $M = 2$, and Learning Rate 3×10^{-4}.

3.2.1 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 [3], a Bayesian rating algorithm that updates the belief over each agent's skill level after every match outcome. The rating reported is $\mu - 3\sigma$, a conservative lower-bound estimate that accounts for residual uncertainty. Figure 3 shows the TrueSkill ratings for a representative selection of agents: the two scripted baselines (weak and strong bot), the SAC agent, and the two best-performing TD-MPC2 variants – one trained with internal opponent modelling at planning horizon H=8 and one at H=6.

All learned agents clearly surpass both scripted baselines. The basic strong bot achieves the lowest rating (15.76), which is counterintuitive given its name but reflects the fact that its deterministic, aggressive strategy is highly exploitable by learned policies; it was not designed as a competitive opponent against

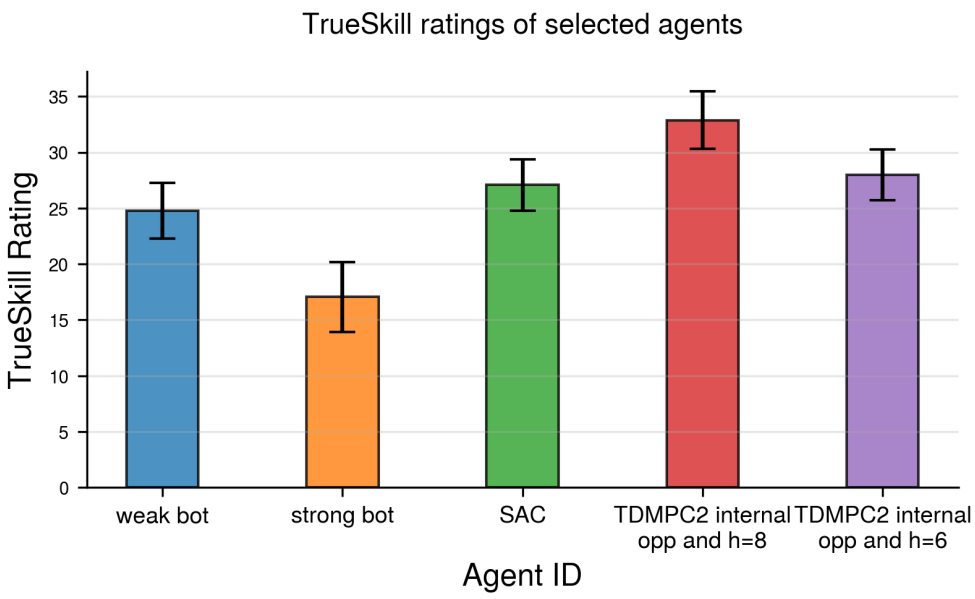

Figure 3: TrueSkill ratings (mean $\pm$ 3σ) for a selected set of agents. Higher is better.

gradient-based agents. The basic weak bot achieves a higher rating (25.24) because its more conservative behavior generates fewer opportunities for the opponents to score, making it harder to accumulate decisive wins against.

Among the learned agents, SAC reaches a rating of 28.34, demonstrating that a well-tuned model-free actor-critic is already a strong baseline in this environment. The TD-MPC2 agent trained with internal opponent modelling at $H{=}6$ achieves a comparable rating of 28.53, while the variant at $H{=}8$ reaches 31.41, the highest rating overall. The gain from $H{=}6$ to $H{=}8$ suggests that a longer planning horizon provides a meaningful advantage, allowing the agent to anticipate multi-step game dynamics more accurately. The overlap in confidence intervals between SAC and TD-MPC2 at $H{=}6$ indicates that, at this horizon, the model-based approach does not yet offer a statistically significant benefit over the model-free baseline. The TD-MPC2 variant at $H{=}8$ with more training steps (16 000 vs. 4 000) does, however, emerge as the clearly strongest agent.

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 on our GitHub repository [1].

References

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

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

[3] 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.

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

[5] 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 4. These logs were plotted fro all runs periodically, and this example is representative for the other runs.

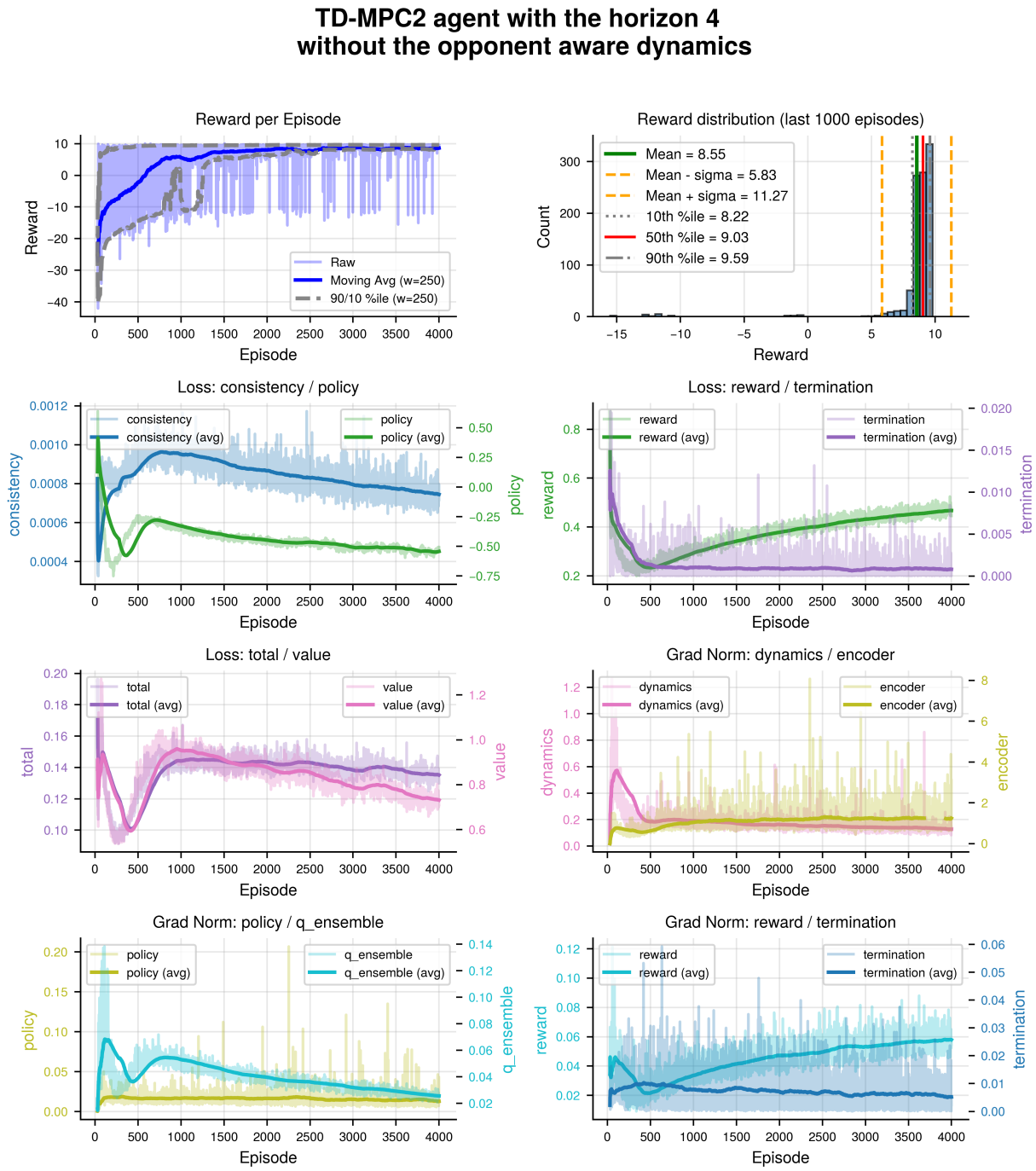

Figure 4: Episode logs of the TD-MPC2 agent with the horizon 4 without the opponent aware dynamics.