Reinforcement Learning WS 2025/2026: Final Project Report

Tournament Team: wann_uRLaub

Karim Wahba (6996482) Philipp Bauer (7232576) Felix Springel (7370560)

February 25, 2026

1 Introduction

This project benchmarks Reinforcement Learning (RL) agents within a physics-based Gymnasium hockey environment [14, 10], characterized by high-velocity dynamics and complex strategic planning. The development process involved implementing and comparing three architectures to handle the environment's high-dimensional control requirements:

- **TD3 (Twin Delayed DDPG):** Implemented by Felix Springel(Sections 3.1 and 4.1), extending the standard algorithm known from [5] with Pink Noise for Action Noise and Linear Epsilon Decay. The algorithm was manually implemented by Felix Springel, while using the `pink` Python library for noise generation.
- **SAC (Soft Actor-Critic):** Implemented by Karim Wahba(Sections 3.2 and 4.2), based on an exercise template and extended with Reward-Based Prioritized Experience Replay, automatic α-tuning [7], custom reward shaping, and curriculum learning. SAC GitHub Repository
- **Rainbow-DQN:** Implemented by Philipp Bauer (Sections 3.3 and 4.3), which was extended with geometric data augmentation, symmetry-aware regularization, and Quantile Regression (Section 3.3). Rainbow-DQN GitHub Repository

Agents were trained against non-learning basic opponents to establish baseline performance before undergoing self-play and evaluation. This report outlines the theoretical background of TD3, SAC, and Rainbow-DQN, details their specific implementations, and concludes with a comparative performance analysis.

All code, experimental results, and text in this report were produced manually by the authors. No AI tools or automated generators were used for coding or writing.

2 Laser Hockey Environment

To understand the learning requirements, the underlying mechanics of the Laser Hockey environment must be defined. The Laser Hockey environment utilizes a terminal reward $R_T \in \{10, -10, 0\}$ for match outcomes, augmented by a continuous proximity reward R_{prox} to address sparse feedback and encourage puck interaction.

Performance is benchmarked against two scripted, non-learning controllers: O_{weak} for initial policy convergence and O_{hard} for evaluating high-velocity dynamics and strategic robustness. These baselines provide consistent metrics for agent progress prior to final evaluation.

3 Methodology

3.1 Twin Delayed Deep Deterministic policy gradient (TD3) & Pink Noise

The Twin Delayed Deep Deterministic policy gradient algorithm, from now on referred to as "TD3", introduced by Fujimoto et al. [5], is a continuous actor-critic method designed to fix the overestimation bias inherent in DDPG. This bias, stemming from the interaction between policy and value updates, often causes divergence or suboptimality. TD3 stabilizes training through three key features:
Clipped Double-Q Learning To prevent over-optimistic value estimates, TD3 uses the minimum of two independent critic estimates, Q_{θ_1} and Q_{θ_2}, when calculating targets. **Target Policy Smoothing** By adding clipped random noise to target actions, TD3 regularizes the value surface, ensuring similar actions yield similar value estimates **Delayed Policy Updates** To reduce oscillations from unstable early estimates, the actor is updated less frequently than the critic, allowing the value function to stabilize before the policy shifts. The success of the agent is governed by the optimization of two primary functions:

The Critic Loss The critics are trained by minimizing the Mean Squared Bellman Error (MSBE) between the current estimate and the clipped double-Q target:

$$L(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[(Q_{\theta_i}(s,a) - y)^2\right]$$

The Actor (Policy) Gradient The actor is updated using the deterministic policy gradient theorem, but only with respect to the first critic (Q_{θ_1}):

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_a Q_{\theta_1}(s,a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)\right]$$

3.1.1 Pink Noise

Continuous control faces a critical exploration-exploitation trade-off. While standard TD3 uses Gaussian noise [5], it lacks the temporal correlation needed for long-horizon trajectories, often causing unrealistic high-frequency oscillations [3]. This project implements Pink Noise ($1/f$ noise) instead, which sits structurally between white noise and Brownian motion. Its temporal "memory" and low-frequency power distribution enable smooth, coordinated movements — essential for intercepting or shooting a puck — while maintaining the microadjustments required for precise aiming.

3.1.2 Reward Shaping und Epsilon Decay

To solve the sparse reward problem (win = $+10$), a custom, shaped reward function was used to provide constant feedback: $R_{total} = R_{base} + (0.5 \cdot R_{touch}) + (0.2 \cdot R_{direction})$. These weights were empirically tuned. Specifically, R_{touch} had to be capped to prevent the agent from merely hovering near the puck without scoring. To transition from exploration to exploitation, a Linear Epsilon Decay was used: ϵ decreases from eps_start during the first half of the training duration, then plateaus at eps_end for the remainder. This ensures the policy stabilizes while retaining enough stochasticity to avoid local minima and worked great in this scenario.

3.1.3 Training Logic

Warm-up & Scaling To prevent premature convergence, the agent initially samples random actions to populate the replay buffer with diverse transitions. Input states (18-dimensions) are normalized via a scaling matrix, as proposed in the environments code. The action space is restricted to four degrees of freedom ([:4]), halving dimensionality to focus on controllable parameters. After the warm-up, a 1:1

Environment Step to Training Step ratio is utilized, as proposed in the original paper. Other ratios were used for testing, but showed to degrade the training quality.

Multi-Stage Training Strategy The agent was trained iteratively, first until reaching a 100% win rate against the weak BasicOpponent, then against the strong BasicOpponent. To generalize its hockey-playing abilities further, Self-Play was introduced: the model was trained against a static version of its previous self until it achieved a consistent 100% win rate there as well.

Adversarial Mixing To prevent the agent from "forgetting" simpler strategies needed to beat the scripted opponents while mastering self-play, an Adversarial Mixing strategy was implemented. During the final stage of training, opponents were randomly selected from a distribution: Weak Bot (20%), Strong Bot (30%), and Self-Play Model (50%). This diversified exposure and forced the development of a more universal winning-strategy, resulting in a single generalized model with a near to 100% win rate against all seen opponents.

3.1.4 Hyperparameters

The agent uses an Actor and Twin Critics, each structured as feedforward networks with two hidden layers (400 and 300 neurons). Training spanned 20,000 episodes (ca. 1.5-2M steps), preceded by a 20,000-step warmup with a batch size of 1024. All parameters were carefully optimized empirically for stability. To ensure a stable value surface, asymmetric learning rates were used (Critic: 0.0003; Actor: 0.0001). Besides that, standard TD3 constants as proposed in the original paper were maintained: $\gamma = 0.99$, $\tau = 0.005$, and a policy update delay of 2.

3.2 SAC: Soft-Actor-Critic

3.2.1 SAC-Framework

Soft Actor–Critic (SAC) [6] was chosen for its sample efficiency and entropy-maximization properties. As an off-policy, model-free algorithm, it effectively reuses experience while learning a stochastic policy. The main concepts of the method are outlined in the following subsections.

3.2.2 Entropy Maximization

Entropy is a measure of how random a random variable is. The more dispersion of probabilities there is, the higher is the entropy. It is used to measure the uncertainty of the policy here, formally defined as

$$\mathcal{H}(\pi(\cdot|s)) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[-\log\pi(a|s)\right] \tag{1}$$

where $\pi(\cdot|s)$ is the policy network $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Higher entropy hence means higher exploration here.

Entropy Regularisation is used to weigh the importance of entropy in SAC using the α parameter. The larger it is the more exploratory, helping policy learning find a better local optimum. Formulated as

$$\mathcal{J}(\pi) = \textstyle\sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right] \tag{2}$$

Automatic Entropy Tuning In the standard SAC, alpha is a hyperparameter that is set to a fixed value, in the implementation auto tuning of alpha was implemented based on a later paper [7] modifying the original SAC. The idea is to have the alpha optimized during training to make it more stable by increasing the gradient wrt. α if current entropy is less than the target and vice versa. Formulation is

$$\mathcal{L}_\alpha = -\mathbb{E}_{s\sim\mathcal{D},a\sim\pi}\left[\log\alpha\cdot(\log\pi(a|s)+\mathcal{H}_{\text{target}})\right] \tag{3}$$

where we are optimizing $\log\alpha$ and $(\log\pi(a|s)+\mathcal{H}_{\text{target}})$ is a detached term (i.e: no gradient flow).

Implementation Details Following the heuristic proposed in the paper, The target entropy $\mathcal{H}_{\text{target}}$ was set to the negative dimension of the action space $-\dim(\mathcal{A})$. For this environment, we set $\mathcal{H}_{\text{target}} = -4.0$

3.2.3 Soft Policy Evaluation - Critic

SAC uses Clipped Double-Q Learning which was explained in section 3.1. In summary, there are two critic Q-networks namely Q_{θ_1} and Q_{θ_2} and to reduce overestimation bias we use the $min(Q_{\theta_1}, Q_{\theta_2})$. Each critic has a corresponding target network used to compute stable targets; Q_{target_1} and Q_{target_2}.

Temporal Difference The target Q-networks are used to calculate the temporal difference error instead of the online critics in order to stabilize training. The target is then formulated as:

$$y = r + \gamma\,\mathbb{E}_{a'\sim\pi(\cdot|s')}\left[\min\nolimits_{i=1,2} Q_{\text{target}_i}(s',a') - \alpha\log\pi(a'|s')\right] \tag{4}$$

The target Q-networks are updated by Polyak averaging [9]. This ensures that the target networks evolve slowly over time, providing a more stable reference. Formula: $\theta_{\text{target}_i} \leftarrow \tau\theta_i + (1-\tau)\theta_{\text{target}_i}, \tau\in(0,1)$.

Critic Loss The Critics are trained as described earlier in section 3.1 with the exception of using $SmoothL1Loss$ instead of $MSBE$. Formulation:

$$\mathcal{L}_{\text{critic}} = \sum\nolimits_{i=1}^{2} \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\text{SmoothL1}\left(Q_{\theta_i}(s,a) - y\right)\right] \tag{5}$$

Implementation details To prevent exploding Q-values in off-policy learning, both the online and target Q-values are clamped to the range before computing the loss or temporal difference targets. Additionally, gradient clipping is applied during backpropagation for both the critic and actor networks ensuring stability during periods of high temporal difference error.

3.2.4 Soft Policy Improvement - Actor

Having explained entropy and critics, next is actor policy. In SAC, the policy is a stochastic policy and the objective is to maximize both the rewards (Q-values) and the entropy so that the agent can be both good and diverse enough. The actor policy is parametrized by a multitask multi layer perceptron network that outputs a mean $\mu(s)$ and a standard deviation $\sigma(s)$ given the input state s. The action a is sampled from a gaussian: $a \sim \mathcal{N}(\mu(s), \sigma(s))$ and the objective is formally defined as:

$$\mathcal{L}_{\text{actor}} = \mathbb{E}_{s\sim\mathcal{D},a\sim\pi}\left[\alpha\log\pi(a|s) - \min\nolimits_{i=1,2} Q_{\theta_i}(s,a)\right] \tag{6}$$

Implementation Details Having the sampled action a is from a Gaussian distribution, a squashing function $(\tanh)$ is applied to map the action to the finite range $[-1, 1]$ and scaled to match the specific action space boundaries of the Laser-Hockey environment $[a_{\text{low}}, a_{\text{high}}]$ using the transformation: $a_{\text{scaled}} = a_{\text{low}} + \frac{\tanh(a)+1}{2}(a_{\text{high}} - a_{\text{low}})$. The reparameterization trick is utilized during this sampling process to ensure gradient flow through this process.

3.2.5 Reward-Based Weighted Experience Replay

A reward weighted prioritized experience replay buffer [12] with the formula: $P(i) = \frac{(|r_i|+\epsilon^{\beta_{\text{reward}}})^{\alpha_{\text{reward}}}}{\sum_k (|r_k|+\epsilon^{\beta_{\text{reward}}})^{\alpha_{\text{reward}}}}$ is implemented and α_{reward}, the reward bias, was set empirically to 2.0.

3.2.6 Reward Shaping and Observation Normalization

State Normalization To prevent varying feature scales in the Laser-Hockey environment from causing unstable gradients, observations are normalized using a running mean μ and variance σ^2 that are updated during training and frozen for evaluation.

Table 1: SAC r_{shape} components at time t.

Term	**Description**
d_t	Agent-puck distance at step t. $(d_{t-1} - d_t)$ rewards moving closer.
$\mathbb{I}_{\text{touch},t}$	1 if touching puck, else 0.
$r_{\text{dir},t}$	The direction of the puck with respect to the target.

Reward Shaping Reward shaping strategies were empirically evaluated and we settled on the dense reward shaping function $r_{\text{shape},t} = w_{\text{dist}}(d_{t-1} - d_t) + w_{\text{touch}}\mathbb{I}_{\text{touch},t} + w_{\text{dir}}r_{\text{dir},t}$. Description in Table 1.

3.3 Rainbow-DQN: Integrated Value-Based Learning

While the agent leverages the Rainbow-DQN architecture [8, 11] to stabilize learning, this section prioritizes the specific implementation details and novel modifications introduced to address environmental instabilities.

3.3.1 Rainbow-DQN framework

Laser Hockey is modeled as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ with state space $\mathcal{S} \subset \mathbb{R}^{18}$ containing the kinematic features of both agents and the puck. Because the environment provides continuous controls, the action space is discretized via $\phi : \mathcal{A}_{disc} \to \mathcal{A}_{cont}$ into 24 predefined control primitives. This enables optimization of the action-value function

$$Q(s, a) \approx \mathbb{E}\big[\textstyle\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a\big]\,. \tag{7}$$

To stabilize learning, a Rainbow-DQN agent, which integrates multiple extensions, is employed to modify the standard Temporal Difference (TD) update. To mitigate overestimation bias inherent in standrad DQN, action selection is decoupled from evaluation using Double Q-Learning (DDQN) [16]. Thereby, the online network θ selects the greedy action $a^* = argmax_a Q(s_{t+1}, a; \theta)$, while a seperate target network θ^-, maintained via Soft Updates [9], evaluates it. This update rule, $\theta^- \leftarrow \tau\theta + (1.0 - \tau)\theta^-$ with $\tau \ll 1$, ensures a stationary value distribution for the TD target: $Y_t^{DDQN} = R_{t+1} + \gamma Q(s_{t+1}, a^*; \theta^-)$. The resulting TD-error, $\delta_t = Y_t^{DDQN} - Q(s_t, a_t; \theta)$, determines the sampling priority $P(i) \propto |\delta_i|^\alpha$ within a Prioritized Experience Replay (PER) [12] buffer. This focusses the gradient descent on high-utility transitions (e.g. goals) while importance sampling weights w_i correct for the induced distribution bias. Architecturally, a Dueling Network Architecture [15] decomposes the action-value function $Q(s, a)$ into a state-value stream $V(s; \theta_V)$ and an action-advantage stream $A(s, a; \theta_A)$, allowing the agent to learn the intrinsic quality of states (e.g. defeisive positioning) independently of specific actions. In addition, Multi-step Returns [13] are employed, which accelerate credit assignment by propagating rewards over n steps before bootstrapping. Finally, exploration is achieved through Noisy Linear Layers [4], which injects learned, parameterized Gaussian noise into the network weights to produce state-dependent exploration. A small ϵ-greedy component is retained to ensure residual uniform exploration.

3.3.2 Symmetry-Aware Rainbow: Incorporating Physical Prior and Distributional Stability

Reward Shaping To mitigate sparse feedback, the agent receives shaped rewards: $R = R_{env} + r_{direction} + 0.2\, r_{touch}$, encouraging puck movement toward the opponent's goal and physical contact to accelerate early training.

Distributional Q-Learning To manage high reward variance, the agent utilizes Quantile Regression (QR-DQN) [2]rather than the standard Categorical DQN (C51) [1] used in the original Rainbow paper. Unlike C51, QR-DQN predicts return values $\hat{Z}_{\tau_i}(s,a)$ through N quantiles corresponding to cumulative probabilities $\tau_i = \frac{2i-1}{2N}$, avoiding pre-define reward bounds (V_{min}, V_{max}). Consistent with the DDQN framework, the target distribution is defined as $Y_j = R_{t+1} + \gamma \hat{Z}_{\tau_j}(s_{t+1}, a^*; \theta^-)$, where the optimal action a^* is selected via the online network θ such that $a^* = \arg\max_a \sum_i \hat{Z}_{\tau_i}(s_{t+1}, a; \theta)$. The network is optimized via the Quantile Huber Loss over pairwise TD-errors $\delta_{ij} = Y_j - \hat{Z}_{\tau_i}(s_t, a_t; \theta)$:

$$\mathcal{L}_{QR} = \mathbb{E}_i \left[\sum_j |\tau_i - \mathbb{I}_{\delta_{ij}<0}| \cdot \text{Huber}(\delta_{ij}) \right] \tag{8}$$

This allows the agent to approximate the full return distribution, capturing both risky maneuvers and certain outcomes. PER priorities are calculated from the mean absolute distributional shift: $\delta_{PER} = \frac{1}{N} \sum_i |Y_i - \hat{Z}_{\tau_i}|$.

Geometric Data Augmentation The laser hockey game is bilaterally symmetric, so the physics and optimal policy are invariant under reflection across the longitudinal axis. To exploit this, each sampled transition (s, a, r, s_{t+1}) is mirrored on-the-fly to produce $(s_{\text{mir}}, a_{\text{mir}}, r, s_{t+1,\text{mir}})$. Both are optimized in a single GPU update, increasing the effective batch size from N to $2N$ without additional environment interactions. This improves sample efficiency and enforces symmetry-consistent feature learning.

Symmetry-Aware Regularization A regularization term $\mathcal{L}_{sym} = \text{MSE}(Q(s,a), Q(s_{mir}, a_{mir}))$ is added to penalize discrepancies between mirrored state-action pairs. The total loss is defined as $\mathcal{L}_{total} = \mathcal{L}_{QR} + \lambda_{sym}\mathcal{L}_{sym}$, where $\mathcal{L}_{QR}$ represents the Quantile Huber Loss (see Section 3.3.2) and $\lambda_{sym} = 0.0001$ scales the constraint. This enforces $Q(s,a) \approx Q(s_{mir}, a_{mir})$, promoting a smoother optimization landscape and consistent value estimates across the hockey environment.

4 Experimental Evaluation and Results

The performance and generalization of TD3, SAC, and Rainbow-DQN are evaluated by monitoring win-rate evolution and reward convergence across the progressive opponent curriculum.

4.1 TD3

4.1.1 Pendulum Environment

Before the Hockey task, the TD3 implementation was benchmarked on the Pendulum-v1 environment to compare Pink and Gaussian noise. As shown in Figure 1, both noise types solved the task reliably. While the Gaussian agent converged faster (ca. 75 episodes), the Pink Noise agent (stabilizing at ca. 125 episodes) ultimately achieved

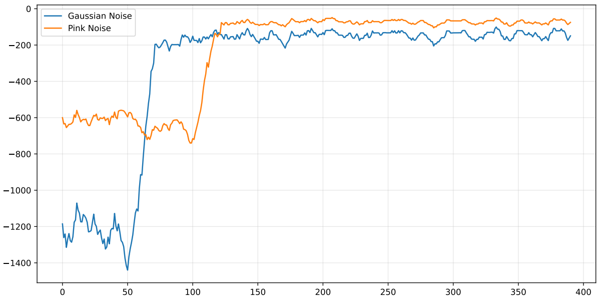


Figure 1: Training Rewards during HalfCheetah Training, Gaussian vs. Pink Noise

higher consistent rewards and a smoother final policy.

4.1.2 HalfCheetah Environment

As seen in Figure 2, testing on the complex HalfCheetah-v4 environment (2M steps) confirmed the superiority of Pink Noise. While both noise types successfully trained the agent to run, Pink Noise lead to a much smoother convergence to higher reward levels. Visual analysis also revealed that Pink Noise produced a highly coordinated, fluid gait, where the Gaussian-trained agent appeared comparatively "clumsy." Given its consistent outperformance across Pendulum and HalfCheetah, I transitioned exclusively to Pink Noise for the final Laser Hockey implementation.

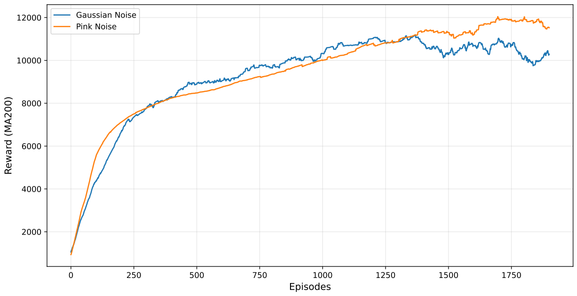
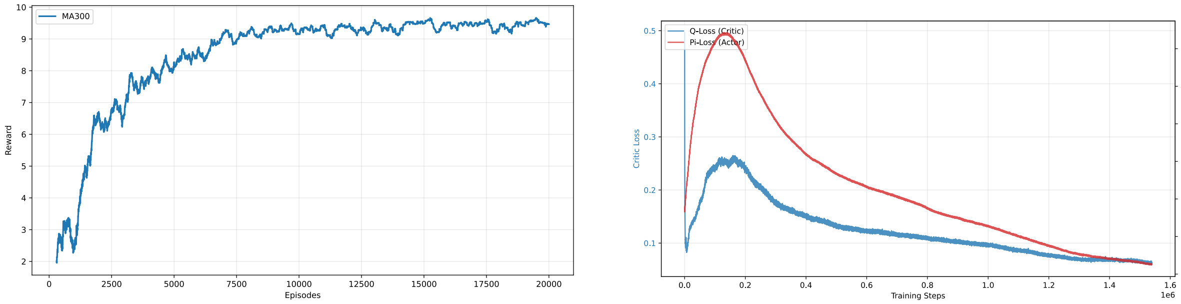


Figure 2: Training Rewards during HalfCheetah Training, Gaussian vs. Pink Noise

4.1.3 Hockey Environment

Training on the way to the Final model Since a lot of different training runs were conducted until the final model, I could add a lot of different plots here - which essentially all showed beautiful convergence. But since the scope of this report is quite limited, I will limit myself to the evaluation of my final model which will compete in the tournament. The plots and statistics of the various training stages are added to the handed-in codebase for the interested reader.

The Final Model As described in the Methology Section in Paragraph 3.1.3, this final Model was initialized with the weight of the self-play-model (trained using a curriculum approach) followed by a final training using mixed adversarials. The adversarial was chosen per-episode based on a random policy, to achieve the extra skill gained through self-play, while not losing the capability to beat the weak and the strong bot. The resulting instability in the reward curve can be seen in Figure 3a, which differed strongly from the very smooth reward curves observed in the scenarios with a non-changing opponent. Nevertheless, strong convergence was achieved over the course of ca. 10.000 Episodes.



(a) Training Rewards

(b) Training Losses

Figure 3: Training performance League (TD3)

The model's performance was benchmarked over 1,000 evaluation episodes without stochastic action noise, the results are detailed in Table 2. This showed that this final Model now learned to generalize

Opponent	**Wins (%)**	**Ties (%)**	**Losses (%)**	**Avg. Reward**
Strong Bot	**994 (99.4%)**	1 (0.1%)	5 (0.5%)	9.09
Weak Bot	**992 (99.2%)**	7 (0.7%)	1 (0.1%)	9.29
Self-Play Model	**998 (99.8%)**	2 (0.2%)	1 (0.1%)	8.94
Self-Play	**Left: 1.0%**	**Tie: 97.7%**	**Right: 1.3%**	–

Table 2: Evaluation results over 1,000 independent games per opponent type.

the hockey problem and is now able to beat all the seen opponents extremely consistent. Visual control against all the opponents showed a very impressive performance as well.

4.2 SAC Evaluation and Results

Architecture and Hyperparameters Different Q-network, policy network architectures and hyperparameters were ablated on small training runs (10K episodes each) against the weak agent of the Laser-Hockey environment. The results concluded the chosen hyperparameters listed in Table 3.

Table 3: SAC Hyperparameters.

Hyperparameter	**Value**
Actor/Q-networks	2L, 256 units with LayerNorms
Batch Size (N)	512
Initial (α)	0.2
Discount Factor (γ)	0.99
Learning Rate (α_{critic})	1×10^{-4}
Learning Rate (α_{actor})	5×10^{-5}
Buffer Size	10^6
Tau (τ)	0.005
Grad Update Step	2

Weak agent and Strong Training Performance The training strategy was performed in a curricular manner where the aforementioned model was trained for 50K episodes on the weak agent and then a saved snapshot was trained on the strong agent. The Win rates over a window of 100 episodes, rewards and loss curves are shown in Figure 4.

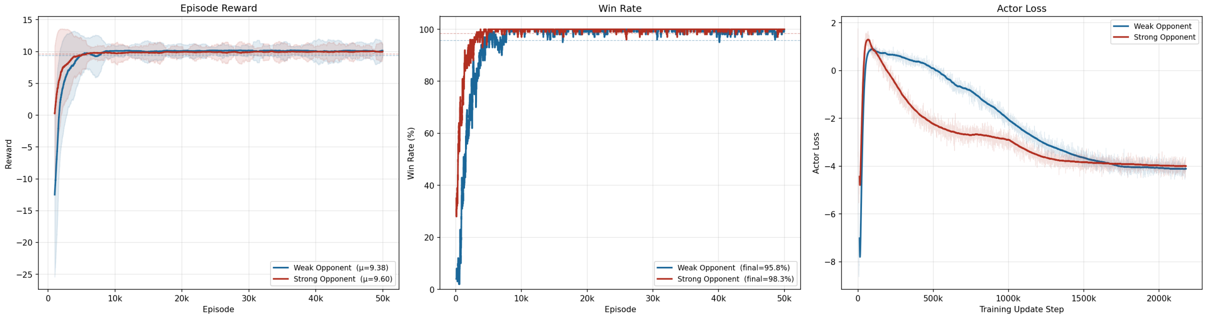


Figure 4: The performance against weak and strong agents. Actor loss converges to $\mathcal{H}_{target} = -\dim(\mathcal{A}) = -4.0$. The rewards and the win rate are calculated over a window of 100 episodes.

Self-play Training Performance Similarly, Self-play training was also curricular, an opponent pool was kept where a saved snapshot of the current agent was added in the pool every 50K episodes along with the weak and strong agents. The opponent was sampled randomly each episode to prevent catastrophic forgetting. With every snapshot added, the convergence rate was slowing as the pool was getting bigger, this can be seen in the Win rate curve in Figure 5.

The Final Model The final model was chosen to be Snapshot 3 based on a round-robin tournament score that was carried out of 1K matches each against the other checkpoints shown in figure Figure 6.

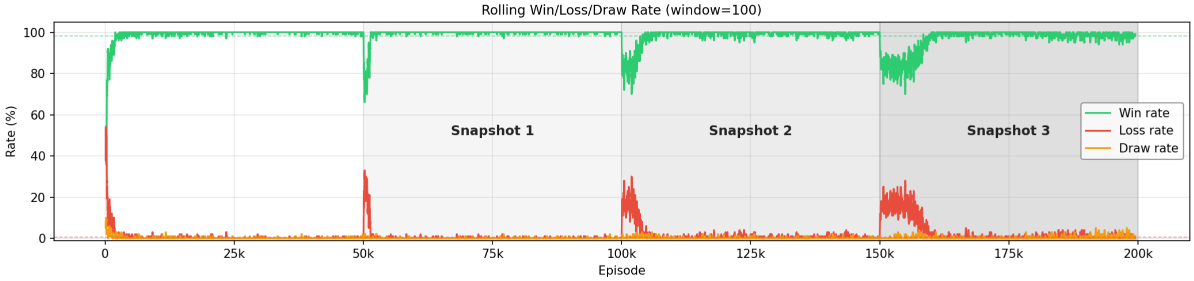


Figure 5: Self-play training where each snapshot was trained for 50K episodes on an opponent pool of [the older snapshots, the weak and the strong agents].

It is worth mentioning that since newer snapshots are trained on a bigger training pool as explained, a slight degradation in the performance against the weak and strong agents was observed. Refer to Table 4.

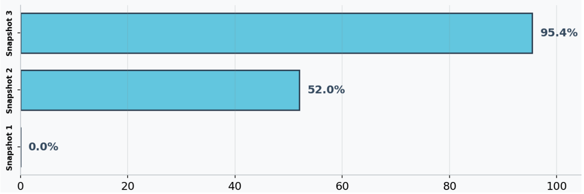


Figure 6: Mean win rate of 1K episodes' Round-Robin tournament for different checkpoints.

	Snapshot 1		Snapshot 2		Snapshot 3	
Opp.	**WR (%)**	**Rwd**	**WR (%)**	**Rwd**	**WR (%)**	**Rwd**
Weak	100.0	9.48	99.1	8.98	96.0	7.42
Strong	99.4	9.22	98.7	8.81	99.0	8.41

Table 4: Performance degradation of self-play trained snapshots on weak and strong agents.

4.3 Rainbow-DQN Evaluation and Results

The agent is trained for 30,000 episodes using a combination of self-play and curriculum learning. The architecture consists of hidden layers [128, 256, 256] with Noisy Linear layers for exploration. While the baseline uses standard Rainbow-DQN, the advanced modules (QR-DQN, Augmentation, and Regularization) are evaluated as optional components in subsequent sections.

Hyperparameter	**Value**	**Description**
Batch Size (N)	32	Transitions sampled per update (augmented to 64)
Learning Rate	2×10^{-4}	Initial LR with decay factor 0.5 at 200k, 350k, 700k steps
Discount (γ)	$0.98 \rightarrow 0.99$	Linear scheduled growth over 1.5M steps
α (PER)	0.6	Determines the strength of prioritization (0 is uniform)
β (PER)	$0.4 \rightarrow 1.0$	Annealed to fully compensate for importance-sampling bias
ϵ (Exploration)	$0.2 \rightarrow 0.05$	Decay schedule for ϵ-greedy policy over 1.5M steps
Buffer Size	2×10^5	Prioritized Experience Replay capacity
n-step Returns	5	Multi-step bootstrapping horizon
Tau (τ)	0.001	Soft update coefficient for target network

Table 5: Core Hyperparameters for Rainbow-DQN Agent Training

Training Stability and Win Rate Under a Progressive Opponent Curriculum Trained over 30,000 episodes using a three-stage curriculum (O_{weak}, O_{hard}, self-play), DQN agents were evaluated on win rate and shaped reward. QR-DQN provided the most substantial gains, enabling models to surpass a 0.8 win rate early in the second phase. Data augmentation further stabilized the learning signal, though its marginal benefit was reduced when paired with distributional methods. As shown in Figure 7, the model integrating QR-DQN, Augmentation, and Regularization achieved the highest terminal performance, maintaining a stable win rate near 0.95 and high average rewards even during the self-play regime.

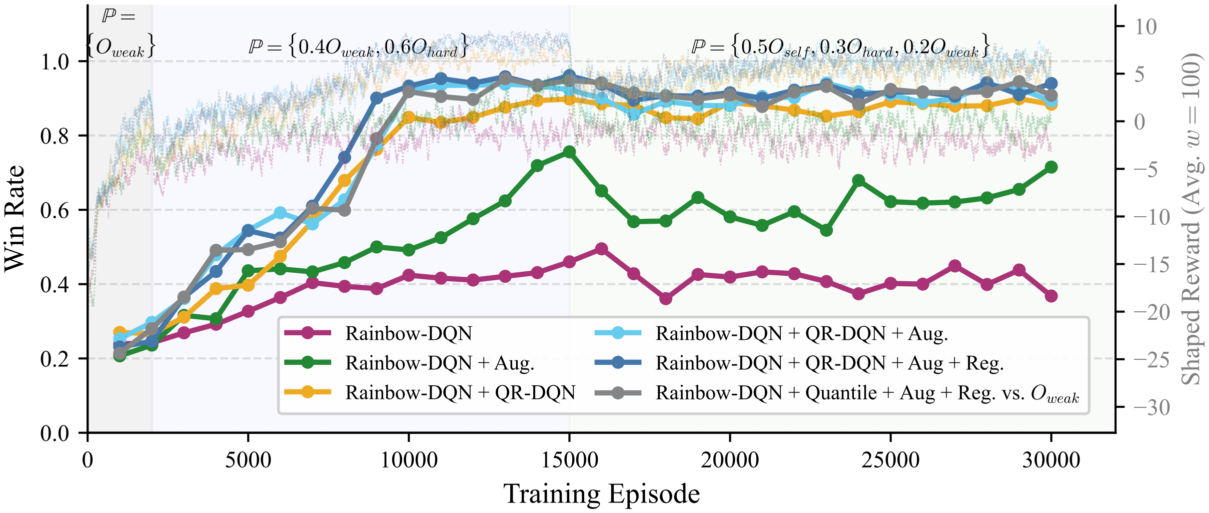
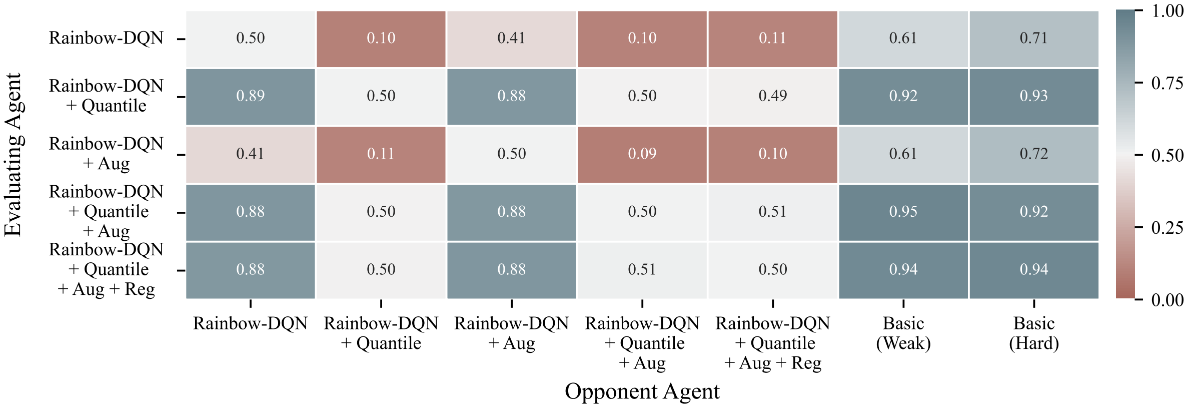


Figure 7: Win rate and shaped reward (Avg. $w = 100$) during three-stage curriculum. Lines denote win rates against O_{hard}, with best-performing agent (grey) also shown against O_{weak}. Background shading indicates the opponent sampling probability $\mathbb{P}$, transitioning from heuristics to self-play at Episode 15k.

Agent Robustness and Head-to-Head Analysis A round-robin tournament, 1,000 games per pair, was conducted to evaluate model generalization. As shown in Figure 8, QR-DQN emerged as the primary driver of performance, significantly outperforming standard Rainbow-DQN and heuristic baselines. While geometric data augmentation and symmetry regularization improved training stability and convergence speed, they resulted in approximately 50% win rates against other advanced variants, indicating they assist more with optimization than with raising the performance ceiling.



	Rainbow-DQN	Rainbow-DQN + Quantile	Rainbow-DQN + Aug	Rainbow-DQN + Quantile + Aug	Rainbow-DQN + Quantile + Aug + Reg	Basic (Weak)	Basic (Hard)
Rainbow-DQN	0.50	0.10	0.41	0.10	0.11	0.61	0.71
Rainbow-DQN + Quantile	0.89	0.50	0.88	0.50	0.49	0.92	0.93
Rainbow-DQN + Aug	0.41	0.11	0.50	0.09	0.10	0.61	0.72
Rainbow-DQN + Quantile + Aug	0.88	0.50	0.88	0.50	0.51	0.95	0.92
Rainbow-DQN + Quantile + Aug + Reg	0.88	0.50	0.88	0.51	0.50	0.94	0.94

Figure 8: Round-Robin Tournament Matrix for 1,000 games per pair. Cell values indicate the win rate of the row agent against the column opponent.

Tournament Rainbow-DQN Based on enhanced Rainbow-DQN (QR-DQN, augmentation, symmetry-aware), following a O_{weak}, O_{hard} stability warmup, the model was fine-tuned for 30k episodes against a O_{weak}, O_{hard}, TD3, SAC and self-play opponent pool. This multi-agent exposure prevents overfitting and ensures an adaptable competitive policy.

5 Final discussion

This section evaluates the training performance and competitive strength of the TD3, SAC, and Rainbow-DQN architectures. The training evaluation, Figure 9, reveals a performance difference between the continuous actor-critic models and the discrete Rainbow-DQN. Both TD3 and SAC exhibit exceptional sample efficiency, converging to a 100% win rate against O_{hard} within just 8,000 episodes. This suggests that continuous action spaces are better suited for the high-precision movements required in hockey.
Conversely, Rainbow-DQN demonstrates a slower learning trajectory, reaching an asymptotic win rate of approximately 95%. While highly stable due to Quantile Regression and symmetry regularization, the inherent discretization of actions likely prevents the agent from achieving the perfect precision necessary to always defeat the O_{hard} baseline.

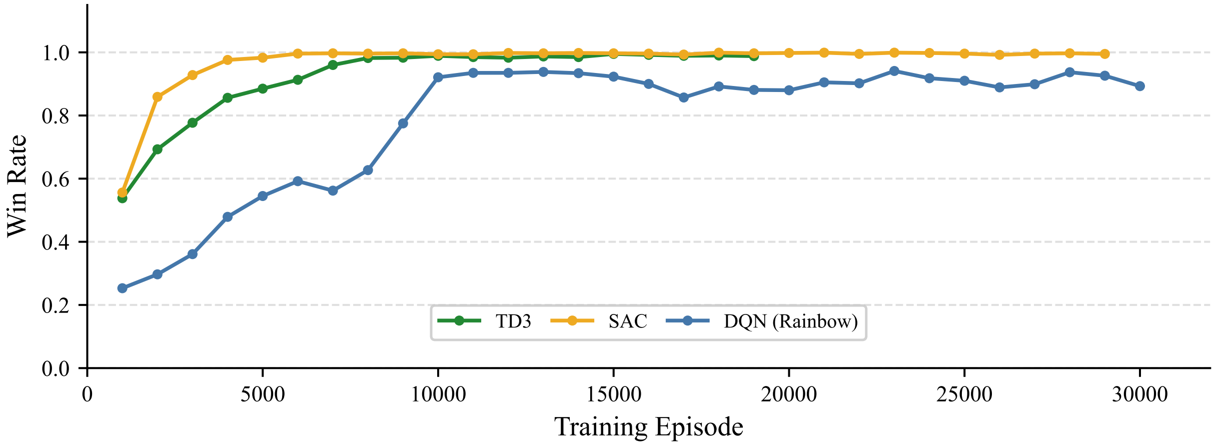


Figure 9: Agents Learning Performance Comparison: Win rate across training episodes for TD3, SAC and Rainbow-DQN algorithms.

In the head-to-head tournament, Figure 10, Rainbow-DQN proved superior to both continuous models, despite its slower training convergence and lower asymptotic win rate (95%) against the fixed bot. This discrepancy suggests that the rapid convergence of TD3 and SAC led to overfitting against the specific movement patterns of O_{hard}. In contrast, the distributional nature of Rainbow-QR-DQN, combined with symmetry regularization, appears to have produced a more robust and adaptable policy capable of handling the varied strategies encountered in self-play and cross-agent matchups.

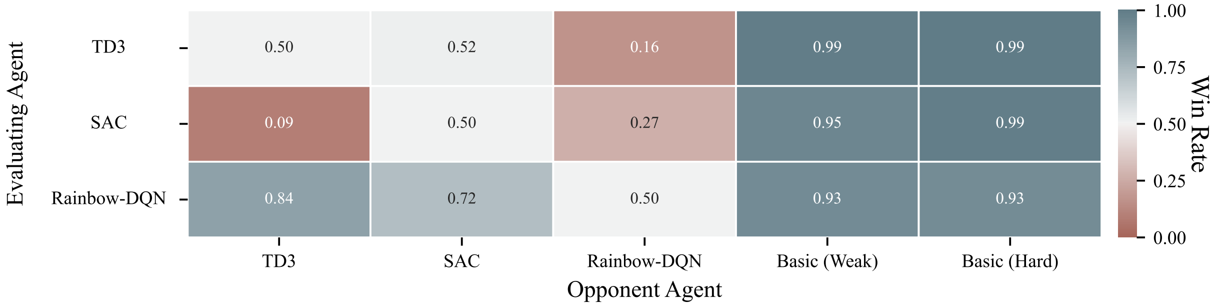


Figure 10: Agents Tournament Win Rate Matrix for 1,000 games per pair. Cell values indicate the win rate of the row agent against the column opponent.

References

[1] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning, 2017.

[2] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression, 2017.

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

[4] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy networks for exploration, 2019.

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

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870. PMLR, 2018.

[7] 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. *arXiv preprint arXiv:1812.05905*, 2018.

[8] 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.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2019.

[10] G. Martius. A simple laser-hockey gym environment for rl agents. `https://github.com/martius-lab/hockey-env`, 2021.

[11] 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.

[12] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, 2016.

[13] R. Sutton and A. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[14] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2025.

[15] Z. Wang, N. Freitas, and M. Lanctot. Dueling network architectures for deep reinforcement learning. 11 2015.

[16] 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.