RL-Course 2025/26: Final Project Report

SwabianOlympics: Simon Rappenecker, Niklas Ehrenfried, Fabian Holzwarth

February 26, 2026

1 Introduction

This is the story on how we became the greatest hockey players in the whole course

us
(before the tournament)

Reinforcement learning is a branch of machine learning in which an agent learns to take actions to maximize its cumulative reward in an environment. One of the most well-known examples of reinforcement learning is AlphaGo [14], groundbreaking research by Google DeepMind that was the first to defeat a professional Go player.

This project investigates reinforcement learning in a simple, physics-based two player environment called Hockey-Game. The task models a rudimentary competitive game in which two players face each other in a rectangular arena. Each of them defending their own goal while attempting to score against the opponent by shooting the puck. The players are confined to their own half of the field and cannot cross the vertical center line. At every time step, a player can apply a movement and a rotation force to their agent, as well as kicking the puck if it currently is held. If the puck is not kicked after 15 steps, this action is forced automatically. Whoever scores the first goal is declared the winner of the game. If no goal was scored after 250 steps, the game will end in a draw.

In this continuous and dynamic environment, the agent receives a complete observation of the game and selects actions for its player every step. By default, rewards are granted mostly on goals. Some smaller metrics are already rewarded during the gameplay and encourage puck possession and motion towards the opponent's goal. Due to the fully observable environment, this can be extended for custom requirements.

On the following pages, we will present our agent implementations to learn a beneficial policy to navigate this game and try to achieve mastery over the opponent strategies:

- **Quantile Soft Actor-Critic (Q-SAC)** - Simon
- **N-PACT** (**N**-step, **P**rioritized, **A**uxiliary, **C**EM, **T**QC) - Niklas
- **Model Based Soft Actor-Critic (MBPO + SAC)** - Fabian

2 Q-SAC

The Soft Actor-Critic, proposed by Haarnoja et al. [4, 5], is a popular off-policy reinforcement learning algorithm, which attempted to overcome the limitations of DDPG [10]. It extends the actor-critic architecture to a stochastic, entropy-based policy, while relying on two critics to mitigate critic overestimation. However, this still proves unstable in a self-play setting and is therefore replaced by a quantile critic, leading to Q-SAC.

2.1 Soft Actor-Critic

The core principle of every SAC is entropy regularization, where the optimal policy is given by [4, 1]:

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi} \sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t) + \alpha H(\pi(\cdot|s_t))\right) \tag{1}$$

R is the reward for states and actions in the current trajectory, α is the temperature term [5] and $H\left(\pi(\cdot|s_t)\right)$ is the entropy of the stochastic policy at the current state. Entropy regularization is effectively a tradeoff between exploration and exploitation. Large temperature terms allow high randomness in the sampled action, while the opposite does not.

The final form of the SAC is then similar to DDPG. It also relies on function approximations using deep neural networks for both actor $\pi_\phi(a|s)$ and critic $Q_\theta(s, a)$. Since we use a stochastic policy, the actor outputs both mean and diagonal covariance of a Gaussian. The critic loss is calculated as follows:

$$L(\theta_i, \mathcal{D}) = \mathop{\mathbb{E}}_{(s_t,a_t,r_t,s_{t+1},d)\sim\mathcal{D}} \left(Q_{\theta_i}(s_t, a_t) - y(r_t, s_{t+1}, d)\right)^2 \tag{2}$$

with $\mathcal{D}$ being the replay buffer. Expectations are approximated with Monte Carlo. y denotes the target:

$$y(r_t, s_{t+1}, d) = r_t + \gamma(1-d)\left(\min_{i=1,2} Q_{\theta_{\text{target},i}}(s_{t+1}, \tilde{a}) - \alpha \log \pi_\phi(\tilde{a}|s_{t+1})\right) \quad \tilde{a} \sim \pi_\phi(\cdot|s_{t+1}) \tag{3}$$

At this point, another major difference to DDPG is obvious: SAC utilizes two critics and chooses the smaller one to avoid overestimation. $Q_{\theta_{\text{target},i}}$ refers to a slightly older version of the critic, to circumvent the semi-gradient problem, which introduces instability. These target networks are updated by Polyak averaging using both Q_{θ_i} and $Q_{\theta_{\text{target},i}}$. Next, the policy is optimized by maximizing the objective:

$$J(\phi, \mathcal{D}) = \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left(\min_{i=1,2} Q_{\theta_i}(s, \tilde{a}) - \alpha \log \pi_\phi(\tilde{a}|s)\right) \quad \tilde{a} \sim \pi_\phi(\cdot|s) \tag{4}$$

Note that the reparameterization trick is required to propagate gradients through the stochastic policy.

An important aspect of SAC is the dynamic tuning of the temperature parameter, as constant values often result in instability. Typically, one aims for high exploration in the beginning and a steady shift towards high exploitation towards the end. To tune the temperature parameter, Haarnoja et al. [5] suggest minimizing

$$J(\alpha) = \mathop{\mathbb{E}}_{a_t\sim\pi_\phi} \left(-\alpha \cdot [\log \pi_\phi(a_t|s_t) + \hat{H}]\right) \tag{5}$$

where $\hat{H}$ represents the target entropy. A common strategy is to set $\hat{H}$ to the negative dimension of the action space ($-\dim(\mathcal{A})$), which corresponds to -4 in the Hockey environment.

2.2 Quantile Critic

The Soft Actor-Critic although relying on the double Q-trick could still suffer from critic overestimation. A way to mitigate this is learning multiple critics, each outputting multiple quantiles. All quantiles are pooled together, and the highest ones are removed, effectively dropping the too optimistic values. Subsequently, all critics are optimized using a Quantile Huber Loss [2]. When updating the policy, it uses the mean of all predicted quantiles to output a scalar. In literature, this technique is also described as Truncated Quantile Critic [8].

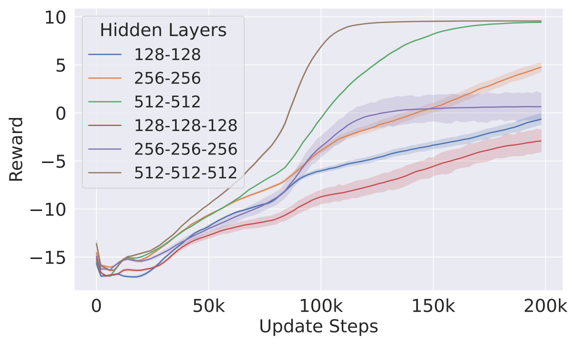
2.3 Self-Play Strategy

Self-play is motivated by the work of Vinyals et al. [15] and is based on sampling opponents weighted by their win rate. The strategy consists of two pools: **base opponents**, consisting of command line checkpoints, and a **self-play pool** that dynamically expands during training. Self-play begins with the warm-up phase, during which only agents from the base pool are sampled. Once the agent reaches a win rate of 80%, self-play is activated. During the self-play phase, a new agent is added to the pool if it consistently wins against all base opponents (80% win rate) and reaches a win rate of 60% against the newest self-play opponent. This enforces that only useful agents are added to the pool. The probability of sampling an opponent from the self-play pool is 60%. Since the newest agent has the lowest win rate, it is sampled most often. When training is almost complete, the fine-tuning phase starts. At this point, the self-play pool is frozen, and no new agents are added. During this phase, the agent tries to optimize its policy on all agents from both pools to become a robust generalist.

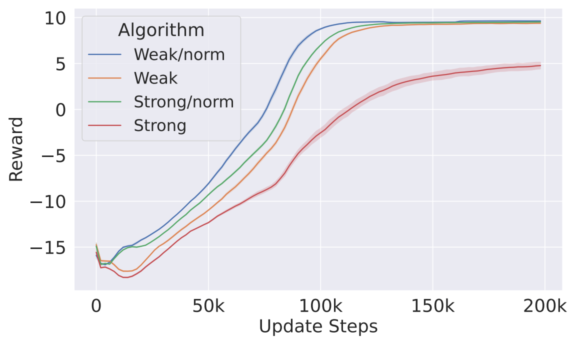
2.4 Experiments

2.4.1 Network Architecture and Normalization

Different hidden layer configurations for both actor and critic were explored, resulting in the final configuration 512-512-512. Note that higher capacity networks increase convergence speed and reduce the variance across random seeds (Fig. 1a). However, when training against the strong opponent, observation normalization is necessary to stabilize training (Fig. 1b).
Training against the weak or the strong opponent takes 20 minutes and results in a win rate of $> 99\%$.



(a)



(b)

Figure 1: **Network Architecture and Observation Normalization.** **(a)** Training against strong with varying critic and actor architectures. **(b)** Training against weak and strong with and without observation normalization. Both plots show the mean and 95% CI over three runs.

2.4.2 Pink Noise

By default, the Soft Actor-Critic uses Gaussian noise to sample actions in the environment. This noise can be replaced by other noise processes, for example with Pink Noise [3]. However, this consistently resulted in slightly reduced convergence speed during training and was therefore not considered further.

2.4.3 Parallel Environments

To improve sample efficiency, 8 parallel environments with different seeds were utilized. During training, a different opponent was sampled for each environment. Q-SAC always performed one update per 8 environment steps.

2.4.4 Quantile Critic and Self-Play

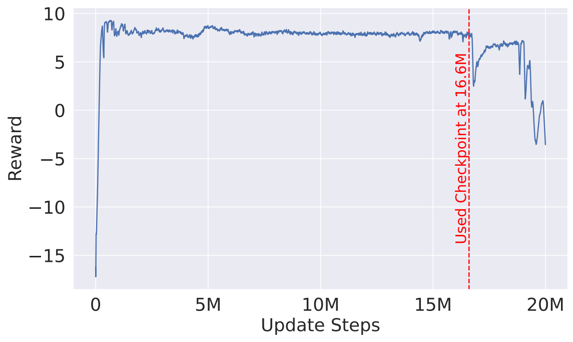
The Quantile-SAC converges slightly slower than the default SAC, but only when training against a single opponent with a fixed strategy (e.g., weak or strong). However, in a self-play setting involving a pool of opponents, the default SAC struggles to find a robust strategy. Q-SAC on the other hand, with a critic aware of the distribution, proves beneficial against overestimation and stabilizes the training. Notably, the Q-SAC finds a counter-strategy considerably faster. This allows it to reach the performance threshold required to add a new agent to the self-play pool sooner, resulting in more experience and therefore better performance in the same amount of update steps (Tab. 2a).

Win Rates	Weak*	Strong*	SAC**
Q-SAC	100%	98.3%	75.33%
SAC	97.6%	80.6%	–

* Mean over 3 trained models, 100 games per model

** Mean over 900 games (100 games for each model pair)

(a)



(b)

Figure 2: **Self-Play Ablation and Training Run.** **(a)** Win rates after 1M update steps of self play comparing Q-SAC and default SAC. **(b)** Self-play training over 20M steps with Q-SAC.

The final agent used in the competition was trained over 20M steps. To ensure a challenging environment, the base opponent pool contained checkpoints from team members and previous runs. However, as shown in Fig. 2b, the self-play training collapsed shortly after 16.6M update steps. This is most likely due to a drastic shift in strategy, resulting in frequent losses against many previous checkpoints, which subsequently increased their sampling probability. Since self-play is an incremental process by design, it is unable to find a strategy against many opponents at once. Consequently, the checkpoint at 16.6M update steps was selected for the final tournament. This agent demonstrates robustness, achieving a win rate of over 95% against both weak and strong opponents, 96% against the base pool of 11 agents, and over 90% against all 87 agents in the self-play pool.

In the competition, the Q-SAC reached place 1 of 149 agents, with an overall win rate of 73%.

3 N-PACT (N-step, Prioritized, Auxiliary, CEM, TQC)

3.1 Motivation and Architecture

The N-PACT Agent explores the synergy of combining several proven features: N-step returns, Prioritized Experience Replay, Auxiliary prediction, CEM planning, and Truncated Quantile Critics. While research generally favors simplicity, my intuition has always been that good features should compound. This combination resulted in an agent that was very robust to hyperparameters and even favored a lower UTD rate. The trade-off for this performance boost is that clean, isolated ablation comparisons become difficult, which is exactly why research usually prefers to focus on single features. To ensure the agent is able to learn a robust policy, N-PACT uses a 512×512 MLP with Mish activations [12], this gives it an advantage over standard 256×256 networks and Mish prevents dead neurons over long trainings. To prevent the small 4-dimensional action vector from being overwhelmed by the 18-dimensional state features, actions are initially mapped to a higher dimension before being processed. This architecture placed 26th in the competition with 43% win rate even with the opponent pool problems.3.3

3.2 Algorithmic Features

3.2.1 Truncated Quantile Critics (TQC) and Quantile Cycling

The critic ensemble for the N-PACT agent uses two TQC critics [9], each outputting 20 quantiles. By default, the top 10% of quantiles are dropped to mitigate overestimation bias. Modeling the full distribution of the state-action return, rather than just its expected value, allows the agent to better assess risk and handle the high variance in rewards, which is exactly what sparse rewards cause. The critic networks are updated by minimizing the Quantile Huber Loss:

$$\rho_\tau(u) = |\tau - \mathbb{I}u < 0| \cdot \mathcal{L}Huber(u) \tag{6}$$

$$L_{TQC}(\theta) = \mathbb{E}_{s,a,r,s'}\left[\frac{1}{N}\sum_{i=1}^{N} \rho_{\tau_i}\left(y - Q_\theta(s,a)_{\tau_i}\right)\right] \tag{7}$$

Instead of a fixed 10% quantile drop, N-PACT uses **Quantile Cycling**. The percentage of dropped upper quantiles cycles over a fixed interval of 400k steps. This serves as an alternative exploration and consolidation mechanism, allowing for optimistic and pessimistic game play which has the potential for breaking local optima where normal noise wouldn't as well as creating diverse self play checkpoints. This approach shares similarities with strategies like Thompson sampling [11], where specific optimistic quantiles are chosen over an episode to drive directed exploration.

3.2.2 Prioritized Experience Replay (PER)

To maximize sample efficiency with a low UTD rate, N-PACT uses a PER buffer [13]. Transitions are sampled based on the size of their TD error δ_i. The sampling probability for a transition i is defined as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \quad p_i = |\delta_i| + \epsilon \tag{8}$$

where ϵ is a small constant guaranteeing non-zero sampling probabilities, and α controls the degree of prioritization. To correct for the induced sampling bias, importance-sampling weights are applied and scaled by β. For the N-PACT Agent, the standard parameters $\alpha = 0.6$ and $\beta = 0.4$ were used from the paper [13].

3.2.3 Auxiliary State Prediction

To speed up early learning and improve the agent's understanding of hockey physics, an auxiliary prediction head regularizes the shared encoder by predicting the change in state Δs. This scales the overall network objective by a decaying coefficient λ:

$$L_{total} = L_{TQC}(\theta) + \lambda \mathbb{E}_{s,a,s'} \left[\| f_{aux}(s,a) - (s' - s) \|^2 \right] \tag{9}$$

3.2.4 Update Ratios and N-Step Returns

To accelerate the backpropagation of delayed sparse rewards (such as goals), n-step returns are used within the PER buffer. Specifically, the agent utilizes a 3-step return, where the target is calculated as:

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n Q_{TQC}(s_{t+n}, \mu(s_{t+n})) \tag{10}$$

where Q_{TQC} represents the truncated quantile target estimate.
However, n-step returns can destabilize off-policy learning if the actor updates too aggressively. By tuning the Update-To-Data (UTD) ratio (Figure 3), it became clear that performing only 2 critic updates and 1 actor update per 8 environment steps (UTD of 0.25) was actually overall better for convergence and wall clock time.
Surprisingly, even with the actor learning rate of 7×10^{-4} and higher UTD rates the agent stayed stable likely due to its robust architecture and Quantile Huber loss

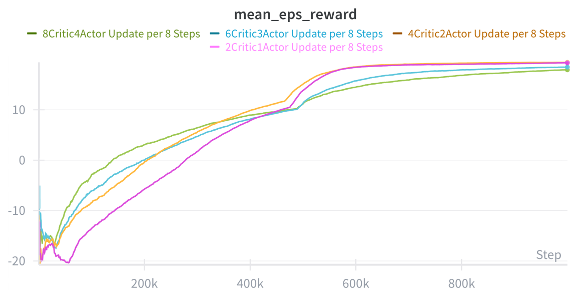


Figure 3: The effect of different actor and critic update frequencies on learning speed. Reducing the update steps reduces wall clock time and maintains policy stability. (ActorLR = 7×10^{-4}, CriticLR = 3×10^{-4})

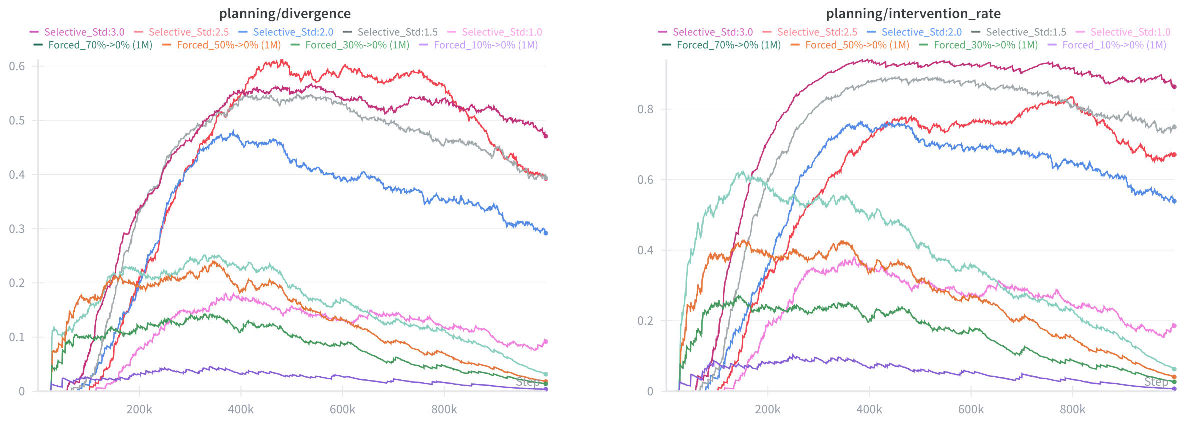
3.3 Reward Shaping, Self-Play, and Opponent Pool

Early iterations of the agent suffered from over-aggression and strategic instability where the N-PACT Agent would chase the puck, abandoning its own goal, even when the puck was on the opponent's side. During training, the agent would often achieve an 80% to 90% win rate with self-play, but standard evaluation against the exact same opponents plateaued at a $\sim 50\%$ win ratio (resulting in an excess of draws, though barely any losses). This happened because of multiple problems: the optimistic exploration induced by Quantile Cycling, my own asymmetric custom reward (+20 for a win instead of +10, a bad idea that carried itself through the whole development) that mathematically incentivized the agent to accept two losses just to secure one win, erratic movement, and a badly balanced self-play opponent pool.
To fix these issues, two penalty terms were added to the reward function:
Action Smoothness Penalty: An L_1 loss applied to rapid consecutive action changes. This stabilized the agent's movement while still permitting sharp, intentional turns.
Goal Anchor: A negative reward that triggers when the agent drifts from its defensive zone while the puck is far away. This drastically improved the agent's defensive posture and prevented reckless chasing.
However, the self-play evaluation gap and overall instability were fundamentally tied to the unnecessarily complex opponent pool, where opponents were sampled based on their win rate against the current

agent, on top of that was a slow probability "creep" to revisit older policies, and a "spiking" mechanism that forcefully injected a sequentially chosen historical opponent every 50k steps to stop catastrophic forgetting. While this seemed to work initially, the training environment it created was simply too aggressive and chaotic, causing the pool to ultimately collapse to uniform sampling after 4 million steps. In retrospect, this complex pool was the primary cause of the final tournament N-PACT agent's skill ceiling. Although spiking and weighted sampling are theoretically sound ideas. A simpler pool, more in line with my teammate Simon's approach, which avoided forcing advanced opponents onto an unprepared agent, would have been better and would have likely created a stabler and higher-performing policy.

3.3.1 Cross-Entropy Method (CEM) with Selective Planning or Forced Annealed Planning

Normal action noise often struggles to discover complex strategies and gets stuck in local optima. To improve this, N-PACT uses directed exploration via CEM combined with an Upper Confidence Bound (UCB) bonus. This allows for either a reward-seeking planner or a high-variance planner, accelerating "critic mapping" and overall learning. The agent uses 32 trajectory samples, 3 iterations, and 10 elites per planning step, along with an exploration/UCB bonus of 0.5. This provides a safe mix between reward exploitation and high-variance state seeking.
To prevent policy collapse caused by over-reliance on narrow learned planner paths, two methods were compared: **Forced Annealed Planning** (blind decay schedule) and **Selective Planning** (adaptive schedule).



(a) Action Divergence (b) Intervention Rate

Figure 4: A comparison between Forced Annealed Planning (n% forced intervention) and Selective Planning (std. threshold intervention) over time. The charts illustrate action divergence between planner and actor (a) and intervention frequency (b).

As shown in Figure 4, Selective Planning provides a much more robust transition. Instead of a fixed decay, the CEM planner only overrides the actor when the critic ensemble has high epistemic uncertainty (Q-value std. is above the threshold). This allows the planner to automatically deactivate as the critics agree and the actor masters the policy. It dynamically reactivates in new, uncertain states, providing help exactly when needed throughout the entire training. Figure 4 suggests a Q-value std. threshold above 3.0 is good, closely mimicking the intervention curve of the proven Forced Annealing method, which was ultimately used in the final N-PACT agent to guarantee stability. The ideal hyperparameter for Selective planning should eventually let it converge to a very low planning rate which only spikes when needed due to time constraints, this was not found but might yield a very robust agent.

4 MBPO + SAC

Even with the very simple environment of the hockey game, we can try to improve sample efficiency further with a combination of a Soft-Actor-Critic (SAC) agent with model based policy optimization (MBPO). The SAC consists of some multi-layer-perceptrons (MLP):

1. Policy Network (Actor) $\pi_\theta(a \mid s)$
   For learning the actual policy, we use a neural network with Gaussian policy and tanh squashing. Given the current observation, the action is calculated.
   $$\mathcal{L}_\pi = \mathbb{E}\left[\alpha \cdot log(\pi(a \mid s)) - min_i(Q_i(s,a))\right] \tag{11}$$

2. Q-Networks (Two Critics) Q_1 and Q_2
   Two independent MLPs are used for estimating action-value function $Q(s,a)$. By using the minimum of both estimates, overestimation bias is reduced and the gradients to train the policy are smoothed out. Together with the soft-updates, this slows down the learning process but improves overall stability.
   $$y = r + \gamma\left(min_i(Q_{target,i})(s', a') - \alpha \cdot log(\pi(a' \mid s'))\right) \tag{12}$$
   $$\mathcal{L}_{Q_i} = \mathbb{E}\left[(Q_i(s,a) - y)^2\right] \tag{13}$$

3. Entropy temperature α
   To balance between maximizing reward and encouraging exploration, a separate optimizer is used to adapt a scalable parameter α for the desired policy entropy $\mathcal{H}_{target}$ [6]).
   $$\mathcal{L}_\alpha = \mathbb{E}\left[-log(\alpha \cdot (log(\pi(a \mid s) + \mathcal{H}_{target})))\right] \tag{14}$$

This already learns the behavior of the game and training opponents quite well. To improve the learning process and also introduce some stochastic variety, this SAC Agent is wrapped by the MBPO construct using

1. Dynamic world-model
   An MLP that is trying to learn how the world steps based on an observation and an action. It gives an approximation of the expected next state for any given combination of state and action $(s,a) \to \hat{s}'$. Using this, we can generate more transitions without stepping the actual environment and exploring actions safely and more effectively. By chaining the generated states, we can also estimate more into the future with increasing uncertainty.
   $$\mathcal{L}_{world} = \mathbb{E}\left[mse(\hat{s}', s')\right] \tag{15}$$

2. Dynamic reward-model
   Analog to the world model, the expected reward system from any given state $s \to \hat{r}$. In combination with the estimated next state, we can estimate the reward without invoking the environment itself.
   $$\mathcal{L}_{reward} = \mathbb{E}\left[mse(\hat{r}, r)\right] \tag{16}$$
   $$\tag{17}$$

To train the world model continuously and keep using some real world data for the SAC training, the experienced transitions (s, a, r, s', d) are stored in a buffer. Training data can now be generated by using random samples from this buffer and extending them with imagined next states and rewards. To prevent throwing off the SAC with random states at the start, the rollout chain of the world-model is increased slowly over time, starting with zero for train until two steps ahead.

4.1 Behavioral analysis

Some development milestones of the agent are clearly reflecting in the rewards 5 and losses 6. The initial behavior of moving toward the puck marks the first relevant point of the learning process, learning the necessary control to decrease immediate punishment. Basic behaviors like moving close to the puck and shooting towards the opponent's side are established. This most likely causes the rapid reward increase at the start, where the agent first distances from random actions and first steady incline.

The first part of the training steps show a steady increase of the actor-policy loss, while the q-critics loss stays very low 6a. The policy seems to be pushed towards high-value actions and discouraged from noisy and weak actions, which in turn creates the higher loss. This aligns with the process of learning consistent and strong movement, instead of cancelling itself out with random value distributions. The introduction of not fully precise one-step predictions after the world loss 6b falls below the threshold can be expected to be one cause of the increasing roughness in policy loss, as the actions no longer consistently match the targeted reality.

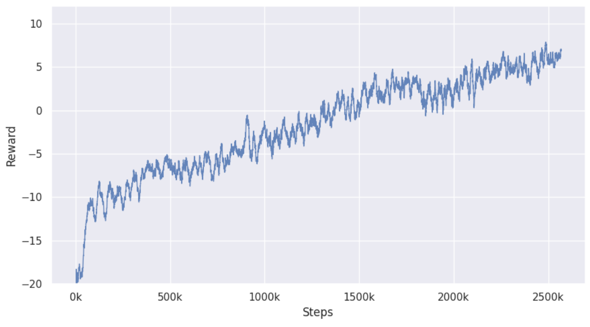
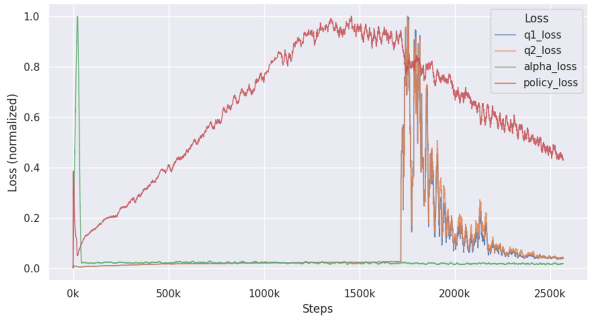


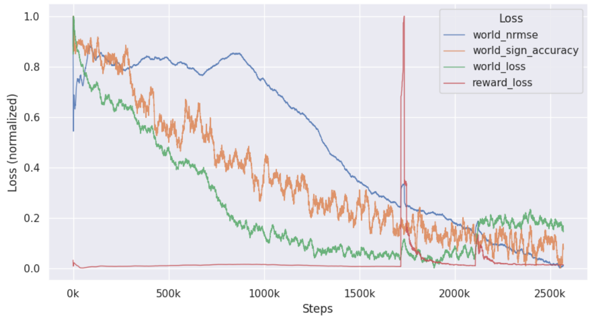
Figure 5: MBPO+SAC rewards over time

An important tipping point is reached in the time between 1300k and 1700k where the policy loss 6a declines and the reward itself stops increasing for a while. This is a sign of a learned behavior that is able to enact a strategy that mostly avoids the consistent immediate punishment. This can be seen in the reward as the tipping point from negative to positive in the mean reward. The agent can no longer increase the reward by improving simple movement and instead has to develop long ranged strategy for higher rewards, which leads to the most significant event.

After around 1700k steps, a sudden and huge spike appears in the q-critics 6a and the reward model 6b, while the policy loss starts decreasing. Only a small spike can be seen in the world model accuracy. This hints at a strong development in the behavioral strategy of the policy, that does significantly change the reward. The previous estimates are thrown off, while the actual world change is still somewhat accurate. The world model seems to have learned the simulated physics well enough to only be thrown off a little by previously unseen events. The q-critics suddenly have a strong change in their target, which requires them to first adapt to this new gameplay style. Observing the gameplay gives reason to the assumption, that the agent learned to estimate puck behavior, as the agent now moves to where the puck will go instead of directly at it and have higher accuracy in shooting indirect goals.



(a) SAC metrics



(b) MBPO metrics

Figure 6: Training metrics

5 Comparison and Conclusion

As visualized in Tab. 1 all team agents are able to defeat weak and strong with a win rate over 92%, fulfilling the course requirements. The performance comparison between team agents requires a big disclaimer: while SAC and N-PACT agents only trained on older and weaker checkpoints of each other, this is not the case for the MBPO agent. It was used by SAC and N-PACT in training, making the MBPO scores below nothing more than a proof, for being able to beat opponent pool agents reliably.

Agent	Q-SAC	N-PACT	MBPO-SAC	Strong	Weak
Q-SAC	46.0 / 8.0 / 46.0	65.3 / 5.5 / 29.2	78.3 / 0.1 / 21.6	97.5 / 0.8 / 1.8	96.8 / 1.1 / 2.2
N-PACT	29.2 / 5.5 / 65.3	41.3 / 17.4 / 41.3	98.2 / 0.0 / 1.8	98.4 / 0.8 / 0.8	97.9 / 1.2 / 0.8
MBPO-SAC	21.6 / 0.1 / 78.3	1.8 / 0.0 / 98.2	49.9 / 0.1 / 49.9	92.2 / 1.4 / 6.4	99.1 / 0.2 / 0.6

Table 1: Internal Tournament with overall win, draw, and loss rates of team and base agents, after 20,000 games per pair. SAC and N-PACT were trained on the tournament checkpoint of MBPO-SAC.

We also report the tournament results:

Agent	Score $\mu - 3\sigma$	Rank
Q-SAC	46.67	1
N-PACT	32.74	26
MBPO-SAC	25.26	52

Table 2: Final course competition results showing the TrueSkill [7] ratings and overall leaderboard placements out of 149 submissions.

We conclude that: First, quantile critics are highly effective for avoiding critic overestimation. Second, while architectural complexity can speed up early learning and create robust agents, a stable opponent pool with self play seems to yield better results.

Q-SAC's reaching first-place demonstrates that a simpler, stable self-play pool leads to superior generalization, whereas N-PACT and MBPO-SAC were ultimately limited by pool collapse or long-term world-model inaccuracies and overfitting. Future work could involve unifying these approaches leveraging MBPO for early sample efficiency and transition to a Q-SAC architecture with N-PACT's selective planning to navigate a highly controlled, stable self-play curriculum.

The code and checkpoints are available on GitHub.[1]

[1]`https://github.com/DerSimi/swabian-olympics`

References

[1] J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[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 *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*, May 2023.

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

[7] R. Herbrich, T. Minka, and T. Graepel. Trueskill™: A bayesian skill rating system. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. The MIT Press, 09 2007.

[8] A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics, 2020.

[9] A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*, pages 5556–5566. PMLR, 2020.

[10] 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.

[11] B. Mavrin, H. Shang, C. Babes-Vroman, A. Balram, D. Jiao, and A. Roberts. Distributional reinforcement learning for efficient exploration. In *International Conference on Machine Learning*, pages 4424–4434. PMLR, 2019.

[12] D. Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[14] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[15] O. Vinyals, I. Babuschkin, W. Czarnecki, M. Mathieu, A. 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. Agapiou, and M. Jaderberg. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350–354, 10 2019.

A AI Usage

A.1 General

The python framework, shared by all agents, offering abstractions, training, self play and checkpoint functionality, is human written, but for small details like nice printouts with rich or a colorful log, we used AI.

visualize_compr.py is a script for visualizing game plays downloaded from the competition server, is completely written by AI. As a prompt, we uploaded the hockey environment and the .pkl file from the website, and asked it to generate a small script.

visualize_gameplay.py visualizes the game play between two checkpoints, completely AI written. Copilot did this, and it used the framework as input. The code was not optimal, and needed a bit of tweaking afterward.

Most of the setup scripts, for installing the environment or setting up sbatch on the cluster, are completely AI written.

A.2 Simon

AI was mainly used for finding functions in APIs, for example in Gymnasium, NumPy or PyTorch. Navigating the wandb API and downloading my run data from wandb was completely handled by AI. I also used AI for plot generation, but always by specifying exactly what I needed. For example, I asked it to generate a plot with a 95% confidence interval. It then suggested seaborn.

A.3 Niklas

In addition to finding functions in libraries as described above, AI was used to find papers and to break down the concepts in them, in a back and forth conversation to see if I understood the contents correctly so that I could implement them. It was also used in the beginning to brainstorm which features to use in the agent and to guess which features might work well together and what issues might arise.

A.4 Fabian

AI was used to create parts of the plotting scripts to get the desired styling and formatting.

B Hyperparameters

B.1 Q-SAC

Hyperparameter	Value
Learning Rates	
Policy learning rate	3×10^{-4}
Critic learning rate	3×10^{-4}
α learning rate	3×10^{-4}
Network Architectures	
Policy hidden layers	512-512-512
Critic hidden layers	512-512-512
General	
Parallel Environments	8
Updates	1 for 8 env. steps
Learning starts	5,000
Buffer size	1,000,000
Batch size	256
γ (Discount)	0.99
Target entropy	-4
τ (Polyak factor)	0.005
Normalize observations	Yes
Quantile Critic	
Number of critics	5
Number of quantiles	25
Top drop	2
Self-Play	
Check start	100,000
Check interval	100,000
Threshold	0.80
Continue threshold	0.60
Max agents	200
Fine-tuning	0.95

B.2 N-PACT

Hyperparameter	Value
Learning Rates	
Actor learning rate	7×10^{-4}
Actor updates per 8 env. steps	1
Critic learning rate	3×10^{-4}
Critic updates per 8 env. steps	2
Network Architectures	
Shared Encoder	512
Actor hidden layers	512-512
Critic hidden layers	512-512
Activation function	Mish
General	
Critic learning starts	25,000 steps
Buffer size	1,000,000
Batch size	256
Mirrored Data	Yes
True Batch size	512
γ (Discount)	0.99
n-step returns	3
τ (Polyak factor)	0.005
Normalize observations	Yes
Action noise	Pink (0.1 scale)
Prioritized Experience Replay	
α (Prioritization)	0.6
β (Importance-sampling)	0.4
Truncated Quantile Critic (TQC)	
Number of critics	2
Number of quantiles	20
Quantile drop range	0% – 10%
Quantile cycling interval	400,000 steps

Hyperparameter	Value
Auxiliary Prediction Unit	
λ (Start $\rightarrow$ End)	5.0 $\rightarrow$ 0.5
Decay steps	250,000 steps
CEM Planning	
CEM Samples	32
CEM Iterations	3
CEM Elites	10
Intervention threshold (Start $\rightarrow$ End)	0.5 $\rightarrow$ 0.0
Intervention decay steps	500,000 steps
Self-Play Curriculum	
Snapshot interval	500,000 steps
Opponent pool type	Safe Weighted
Max pool size	100

B.3 MBPO-SAC

Hyperparameter	Value
Learning Rates	
Policy learning rate	3×10^{-4}
Critics learning rate	3×10^{-4}
α learning rate	3×10^{-3}
World model learning rate	2×10^{-5}
Network Architectures	
Policy hidden layers	256-256
Critics hidden layers	256-256
World model hidden layers	256-256
General	
Prediction rollout horizon	0-2
Update batch size	64
Learning starts	5,000
Buffer size	500,000
Sample batch size	256
γ	0.99
Target entropy	-4
τ	0.005