RL-Course 2025/26: Final Project Report

WayneGradientzky: Rajinish Aneel Bhatia, Bartol Markovinović, Mohd Khizir Siddiqui

February 26, 2026

1 Introduction

2 Temporal Difference Learning for Model Predictive Control (TD-MPC)

Temporal Difference Learning for Model Predictive Control (TD-MPC) [6] is a continuous action space model-based RL algorithm that combines learning a Task-Oriented Latent Dynamics (TOLD) model with trajectory optimization using a variation of the Model Predictive Path Integral (MPPI) method. The TOLD model consists of the following learned components:

- **Latent Encoder**: $z_t = h_\theta(s_t)$ – Encodes the state into a latent representation which captures relevant features for predicting rewards, dynamics, and value.
- **Latent Dynamics**: $z_{t+1} = d_\theta(z_t, a_t)$ – Predicts the next latent state given the current latent state and agent's action. This allows the model to roll out trajectories during planning. For hockey environment, dynamics model also implicitly predicts opponent's behavior since the opponent's features are part of the state and opponent's actions affect the next state.
- **Reward**: $\hat{r_t} = r_\theta(z_t, a_t)$ – Predicts immediate reward for taking an action in a given latent state.
- **Q-Value Function**: $\hat{Q}_t = Q_\theta(z_t, a_t)$ – Predicts the value of taking an action in a given latent state. Two Q-networks are used to mitigate overestimation bias, similar to TD3.
- **Policy**: $\hat{a_t} \sim \pi_\theta(z_t)$ – Outputs a distribution over actions given the latent state. Policy is used for training the Q-value function and for generating additional trajectories during planning.

Models $h_\theta, d_\theta, r_\theta, Q_\theta$ are jointly trained by minimizing the loss $\mathcal{J}(\theta, \Gamma) = \sum_{i=t}^{t+H} \lambda^{i-t} \mathcal{L}(\theta, \Gamma_i)$ where the objective $\mathcal{L}(\theta, \Gamma_i)$ for each step i consists of reward prediction error, value loss with respect to the TD target, and L2 loss between the predicted next latent state and the encoded actual next state:

$$\mathcal{L}(\theta, \Gamma_i) = c_1 \|r_\theta(z_i, a_i) - r_i\|_2^2 + c_2 \|Q_\theta(z_i, a_i) - (r_i + \gamma Q_{\theta'}(z_{i+1}, \pi_\theta(z_{i+1})))\|^2 + c_3 \|d_\theta(z_i, a_i) - h_{\theta'}(s_{i+1})\|_2^2$$

Symbol Γ denotes a trajectory of states, rewards and agent's actions sampled from the replay buffer, λ is a constant that decreases importance of losses for later steps, γ is the discount factor, and c_1, c_2, c_3 are constants that balance the different loss components. For the Q-value loss and the latent dynamics loss, the target networks θ' are used. Target networks are updated every 2 mini-batch updates with slow-moving averages of the main network parameters to stabilize training. For sampling mini-batches of trajectories, a prioritized experience replay buffer is used, where states are sampled with probability proportional to their TD error. The policy network is trained separately, by minimizing the temporally weighed Q-value objective of the same sampled trajectories: $\mathcal{J}_\pi(\theta) = -\sum_{i=t}^{t+H} \lambda^{i-t} Q_\theta(z_i, \pi_\theta(z_i))$.

To select an action, several iterations of modified MPPI planning are performed. In the first iteration, N random action sequences of length H (horizon) are sampled from the normal distribution $\mathcal{N}(\mu, \sigma)$ and rolled out using the learned dynamics model to obtain future latent states and rewards H steps into the future. Additionally, the policy network is used to generate extra trajectories by rolling out the policy model and adding truncated Gaussian noise. Each trajectory is scored by the discounted sum of predicted rewards and the final Q-value. The top K trajectories are selected and weighed by their exponentiated scores to update μ and σ for the next iteration. After last iteration, a trajectory is sampled from the score weighed multinomial distribution of the final elite set and the first action of that trajectory is executed.

2.1 Network Architecture Changes

Initially, each part of the TOLD model was implemented as an MLP with ELU function [1] as the non-linearity, as described in the original paper. However, this resulted in training instability caused by exploding loss values. To solve this issue, network architectures described in the TD-MPC2 paper [5] were implemented. Activation function was changed to Mish [7], Layer Normalization was added, and latent space was normalized with SimNorm [5]. Furthermore, the Q-value and reward networks were changed to output logits of a softmax distribution over exponentially spaced bins, as described in DreamerV3 [4].

2.2 Adding Ideas from iCEM to the Planning Algorithm

One attempt to improve the planning part of TD-MPC was to implement ideas from the Improved Cross-Entropy Method (iCEM) [8]. These included sampling actions from temporally correlated colored noise instead of independent Gaussian noise, keeping a fraction of the elite set between iterations, including the shifted elite set from previous step into the initial trajectory set of the next iteration, and executing the best action from the final elite set instead of sampling from the score weighted distribution.

2.3 Action Hints – Guiding Planning with Custom Action Sequences

Another idea to improve planning was to replace a part of the initial random action sequences with hand-crafted action sequences that either lead agent to a future position of the puck or to one of the equidistantly spaced positions in front of the goal. These action sequences were generated with PD control, and the idea was used in conjunction with keeping a fraction of the elite set between planning iterations to potentially preserve exact trajectories that lead to the puck or the goal.

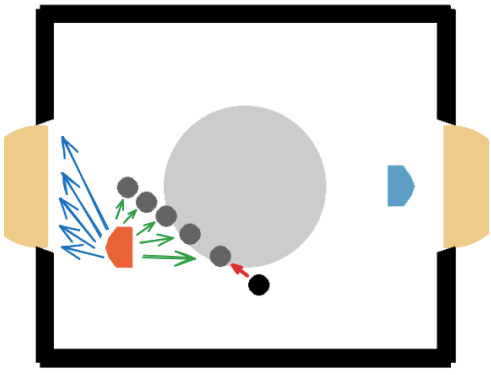


Figure 1: Illustration of action hints

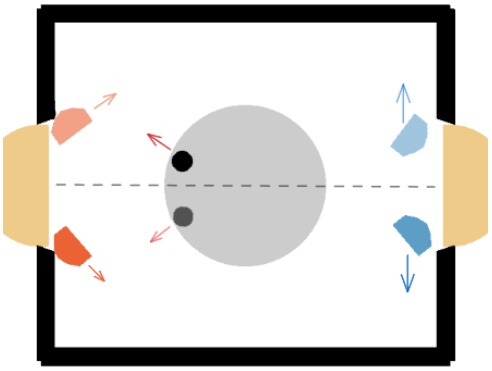


Figure 2: State and action mirroring

2.4 Training Setup and Self-Play

For training, only the terminal (sparse) reward of $+10$ for winning and -10 for losing was used. When adding an episode to the replay buffer, the mirrored episode with respect to the horizontal axis (2) was also added. This doubles the amount of training data and encourages model to learn symmetric strategies.

2.4.1 Training Curriculum

Training is started against only weak and strong bots, and a frozen checkpoint of the model is added to the opponent pool every 100 000 steps, or earlier if win rate against all opponents exceeds 80%. Additionally, 3 TD3 agents with different behaviors are added to the opponent pool after 2 million steps. Maximum opponent pool size is set to 10, and when the pool is full, the oldest frozen checkpoint is removed. This curriculum ensures that the model is exposed to a variety of opponents of appropriate difficulty throughout training. To monitor generalization performance against unseen strong opponents, model was **not trained, but only evaluated** against publicly available SAC agent from last year's competition.

2.4.2 Windowed Thompson Sampling Opponent Selection

Since all opponents in the training pool are beatable due to the curriculum, selecting the most challenging opponent seems like a reasonable strategy to maximize learning signal. To do this, the number of wins, draws, and losses against each opponent in the pool was tracked in a sliding window of the last 100 games (in total, not per opponent). Then, win, loss and draw rates of the TD-MPC agent against each opponent were estimated by sampling from a Dirichlet distribution with parameters $\alpha = [\text{wins} + 1, \text{losses} + 1, \text{draws} + 1]$. The opponent where the $loss\_rate + 0.5 * draw\_rate$ is maximized is selected. The intuition behind this is that the problem of choosing the most challenging opponent can be seen as similar to a multi-armed bandit problem, where each opponent is an arm and the reward is 1 for a loss, 0.5 for a draw, and 0 for a win. Using only last 100 episodes accounts for constant change of policy, and Dirichlet distribution was chosen because it is a conjugate prior for the multinomial distribution, which models the outcomes of games against each opponent.

2.5 Experiment Results

All experiment configurations were first run for 3 million steps (or less if unsuccessful), and the best performing experiments were prolonged to 10 million steps. In Figure 3 we see that during all experiments the model quickly learns to defeat the weak bot. However, with the initial bad model architecture, the win rate against the weak bot plummets after 500 thousand steps, showing the training instability. After fixing the architecture, the model maintains a high win rate against the weak bot. The variations that used iCEM ideas do not generalize well to the validation agent, regardless of the colored noise β parameter. The checkpoint that generalized the best seemed to be the one with action hints trained for 4.3 million steps. This was confirmed with the final tournament results. Additionally, a number of different smaller modifications were tested, such as training with defensive mode, using closeness to goal as an additional reward signal, and using dropout in Q-value networks, but they did not lead to any improvements.

3 Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed Deep Deterministic Policy Gradient (TD3) [3] is a model-free, off-policy algorithm for continuous action spaces, improving over DDPG via clipped double Q-learning, delayed policy updates,

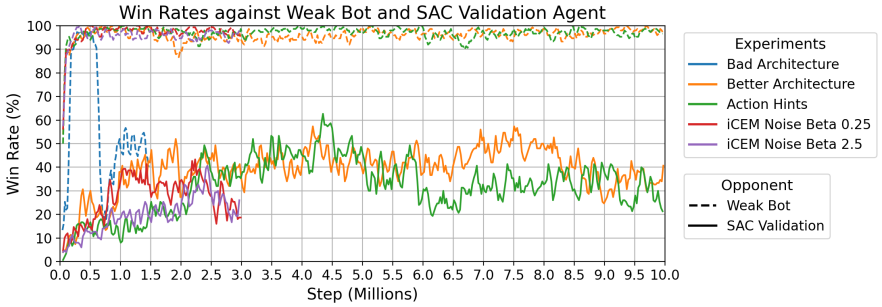


Figure 3: Evaluation win rates against Weak Bot and the SAC validation agent for different experiments

and target policy smoothing. TD3 maintains two Q-networks (Q_{ϕ_1} and Q_{ϕ_2}) and uses their minimum as the TD target:

$$y = r + \gamma \min_{i=1,2} Q_{\phi_i'}(s', \tilde{a}), \quad \tilde{a} = \pi_{\theta'}(s') + \epsilon, \quad \epsilon \sim \texttt{clip}(\mathcal{N}(0, \sigma), -c, c)$$

Each critic minimizes the squared Bellman error $\mathcal{L}(\phi_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}(Q_{\phi_i}(s,a) - y)^2$, while the actor maximizes $J(\theta) = \mathbb{E}_{s\sim\mathcal{D}} Q_{\phi_1}(s, \pi_\theta(s))$ and is updated every 2 critic steps.

3.1 Method

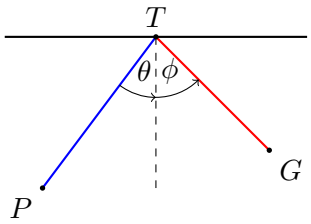
3.1.1 N-Step Returns

Instead of the standard one-step TD target, we use n-step returns to propagate reward signal faster:

$$y = r(s_0) + \gamma r(s_1) + \cdots + \gamma^{n-1} r(s_{n-1}) + \gamma^n q(s_n, \tilde{a})$$

3.1.2 Custom Opponent and Bank-Shot Reward

We noticed early on that the model did not always manage to learn ricochet (bank) shots, so to help it defend against opponents that hit those shots we introdued a hard-coded opponent that always plays them. Given puck position P, goal position G, and T as the target we must shoot at to reach G. Then T_y is known because that is the boundary and we only need to find T_x. We can assume that $T_x \geq P_x$ and $G_x \geq T_x$. If we assume frictionless walls than $\theta = \phi$.



From $\tan\theta = \tan\phi$ we obtain:

$$T_x = \frac{|T_y - P_y|\, G_x + |T_y - G_y|\, P_x}{|T_y - G_y| + |T_y - P_y|}$$

This agent achieves $\approx 94\%$ win rate against the weak opponent and $\approx 80\%$ against the strong opponent. We also shaped the reward by calculating how aligned the agent's direction was to this direction to encourage the learning agent to prefer shots in this direction.

3.1.3 Layer Normalization

Layer Normalization. Layer normalization normalizes activations across the feature dimension for each sample independently. For a given input vector x, it computes:

$$\texttt{LayerNorm}(x) = \frac{x - \mu}{\sigma + \varepsilon}\gamma + \beta$$

where μ and σ are the mean and standard deviation of x, and γ, β are learnable scale and shift parameters. In the RL setting this reduces sensitivity to the scale of inputs and gradients, which can vary dramatically during training. We apply LayerNorm after each linear layer in both actor and critic networks.

3.2 Experiments

3.2.1 Training in Phases (Opponent Scheduling)

Training proceeded in three phases. Phase 1 used only the weak opponent. Phase 2 sampled weak/strong opponents with probabilities 30%/70%. Phase 3 used a mixed pool: 20% weak, 20% strong, 10% custom bank-shot opponent, and 50% from a rolling queue of past checkpoints (self-play). An adaptive opponent selection strategy based on Thompson sampling was evaluated as an alternative but underperformed the manual schedule, as shown in Figure 5.

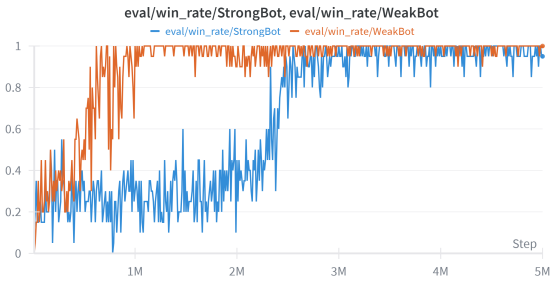


Figure 4: Winrate (trained in phases)

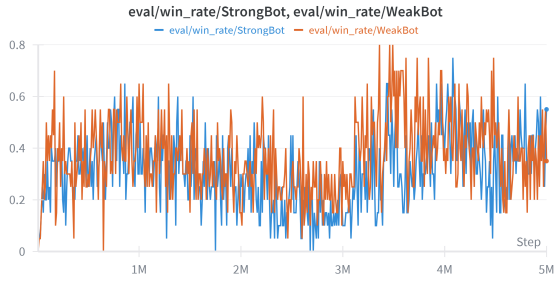


Figure 5: Winrate (trained using Thompson sampling)

3.2.2 Self-Play

Self play was the part of the third phase of training and was implemented by keeping a queue of 50 checkpoints and adding a new checkpoint every 50000 timesteps. This ensured that the model had enough learning experience from each of its previous checkpoints to be able to defeat them.

3.2.3 N-Step Returns, Episode Mirroring and Environment Scheduling

N-step returns and episode mirroring improve the speed of learning as reflected in (Figure 6), compared to one-step TD targets. We chose 3-step for the final training. Environment scheduling (30% SHOOTING_MODE, 70% DEFENSE_MODE) was introduced in Phase 3. Figure 7 shows the win rate when the opponent starts with possession and confirms a clear improvement.

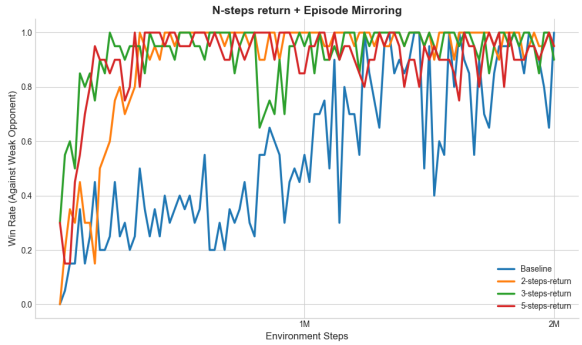


Figure 6: Win rate vs. past checkpoints (n-step returns)

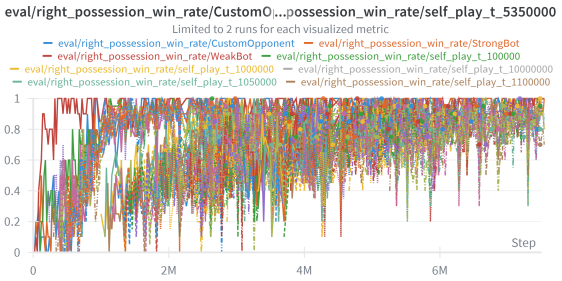


Figure 7: Win rate when opponent starts with possession using environment scheduler

3.2.4 Bank-Shot Reward

The bank-shot preference reward visibly shifts puck density toward the walls (Figures 8, 9), confirming that the agent learned to incorporate ricochet shots into its strategy.

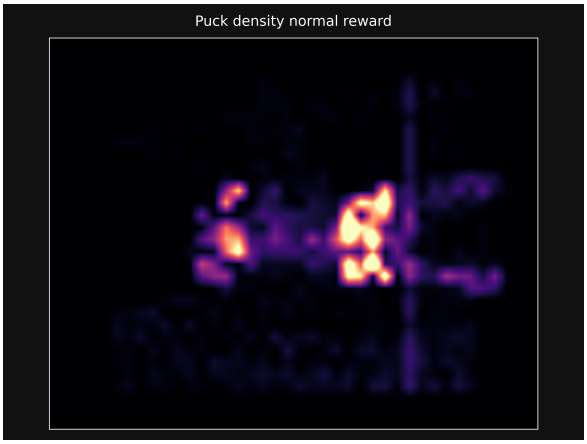


Figure 8: Puck density — standard reward

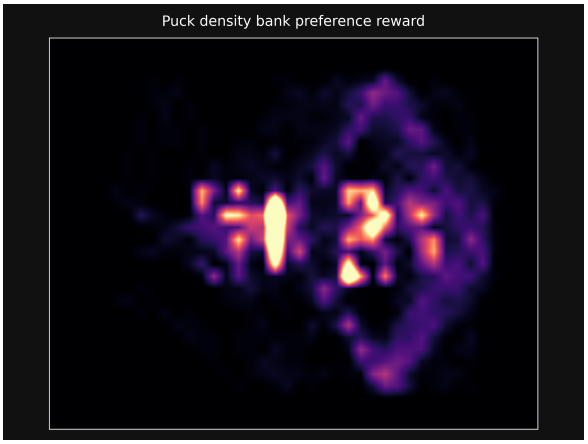


Figure 9: Puck density — bank-shot preference reward

3.2.5 Layer Normalization

LayerNorm converged faster against the fixed bots (Figure 10) but hurt generalization during self-play: the agent progressively drew against all previous checkpoints (Figure 11) rather than maintaining dominance, so it was excluded from the final model.

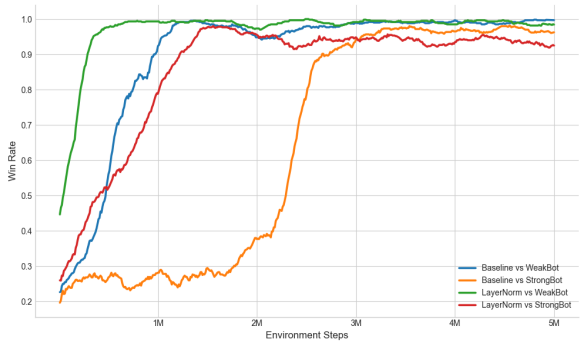


Figure 10: Win rate: baseline vs. LayerNorm

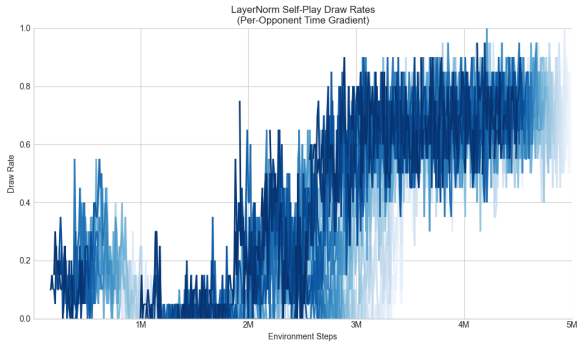


Figure 11: Draw rate of LayerNorm vs. past checkpoints

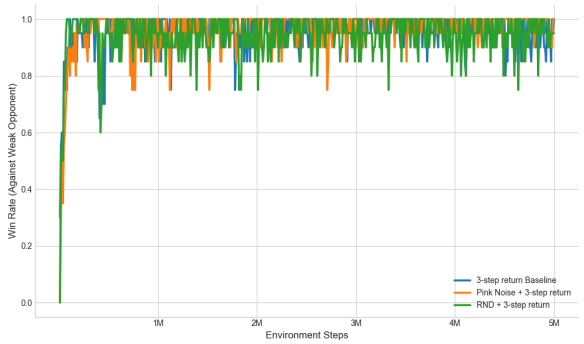


Figure 12: Winrate with Gaussian Noise, RND, and Pink Noise

We experimented with Random Network Distillation (RND) for intrinsic motivation and pink noise[2] for exploration. While both techniques slightly increased exploration diversity, they did not provide significant improvement in win rate or long-term performance. Consequently, they were not included in the final training configuration.

3.2.6 RND and Pink Noise Exploration

4 Discussions and Conclusion

References

[1] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[2] O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning. 2023.

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

[4] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models, 2024.

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

[6] N. Hansen, X. Wang, and H. Su. Temporal difference learning for model predictive control, 2022.

[7] D. Misra. Mish: A self regularized non-monotonic activation function, 2020.

[8] C. Pinneri, S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius. Sample-efficient cross-entropy method for real-time planning, 2020.