Application of Deep Neural Network Imitation Learning to Chaotic Multiplayer Environments

Chris Labbe*
MS Artificial Intelligence
University of Texas at Austin
chris.labbe@utexas.edu

Daniel de Araujo*
MS Computer Science
University of Texas at Austin
dearaujo@utexas.edu

ABSTRACT

In this work, we describe the challenges and approach to apply a Deep Neural Network (DNN) to control players in a 2v2 ice hockey kart game tournament. First, we explore the criteria of success to design an expert algorithm to score goals by exploring the environment, controls, and game mechanics. This automated player was used to generate training data for supervised imitation learning using a DNN model structure. After architecture search, hyperparameter optimization, and training, the DNN agent was evaluated against different tournament agents to measure success. Lastly, we discuss the results and reproducibility challenges encountered.

CCS Concepts
• **CCS → Computing methodologies → Machine learning → Machine learning approaches → Neural networks**.

Keywords
Deep Neural Networks; Imitation Learning; Chaos Theory; Multiplayer Environments; Reproducibility; K-fold Cross-Validation.

1. MOTIVATION

Deep neural networks (DNN) have been applied in a variety of areas such as computer vision, natural language processing, speech recognition, recommender systems, fraud detection and others [1]. From architectural improvements, hardware acceleration, and ample availability of data, tasks deemed impossible just a mere decade ago have been solved with superhuman performance [2].

The challenge this work addresses is to apply DNNs to control a team of kart agents in an ice hockey game tournament against 4 different teams. There are many ways to approach this problem such as using computer vision agent model to identify objects and locations within the image view of the player to decide the course of action. Another approach would be to use the state agent model and apply reinforcement learning techniques such as actor critic [3], Trust Region Policy Optimization [4], or Proximal Policy Optimization [5] to train the agent. We chose not to implement the first approach because agent only has a limited, 'first-person' view of the environment and this task would also require the creation of the game playing logic for the agent based on the image data. The second approach with a state agent provides complete data of the team, puck, and adversarial team, however, the DNN controlling the player must contain all the intelligence of the agent, so no additional logic or custom designed features are allowed. We observed early on that the game is chaotic in nature where game outcomes are not only highly sensitive to initial conditions, but also to small changes in player dynamics. Our approach was to design our state-based agent using imitation learning on an expertly designed controller. The design and tuning of this controller are based on Behavior Cloning with Human Feedback (BCHF), inspired from Reinforcement Learning with Human Feedback (RLHF) techniques used to align large language models.

This work is organized as follows. The motivation and overall approach are outlined in Section 1. In Section 2, we describe in detail the environment, game setup, and agent/game state data available. Section 3 explores the golden agent design and optimization challenges. Section 4 describes the DNN architecture, data collection, training methodology and convergence evaluation. In Section 5, we discuss the results, with conclusions in Section 6. References are included in Section 7.

*Authors listed in alphabetical order, equal contribution.

2. ENVIRONMENT

In this section we describe the application environment for the DNN. An open-source game, SuperTuxKart [7] is used for experimental grounds where a 2 player vs. 2 player ice hockey game provides the environment described in the game area section. In this section, we discuss the game setup, rules and scoring, and telemetry available for agent control.

2.1 Game Arena

The ice hockey kart game with 2 players in each team is shown in Figure 1. The origin is at the center of the arena (0,0) where the positive x and y axis are shown. The red team is located on the bottom of the arena and needs to score in the top goal while the blue team is located on top and needs to score on the bottom goal. The puck starting position and team side assignment changes during the tournament to add uncertainty in the game play. The initial player (x, y) locations are offset for player zero (-6.7, -56) and centered for player one (0, -53). The opposing team karts have the same x coordinates, but at y=56 and 53 respectively.

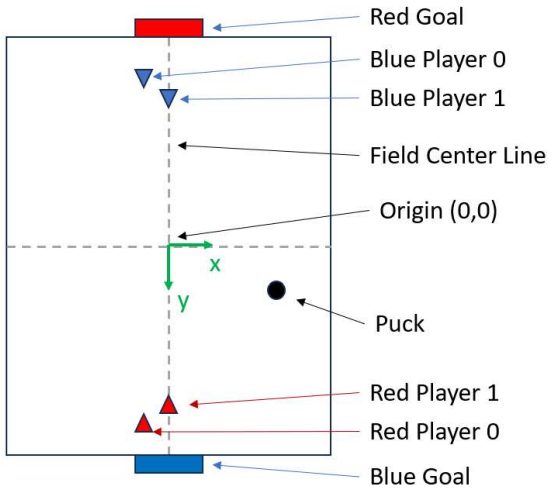

Figure 1. Ice Hockey Area Setup.

2.2 Rules, Scoring, Match

The rules and setup are simple. A team scores a goal when the puck enters the adversary's goal. Each tournament used for evaluation consists of games against 4 agents (`geoffrey`, `jurgen`, `yann`, `yoshua`) with four starting ball positions (0, 1), (0, -1), (1, 0), (-1, 0), as blue team and red team for a total of 32 games. Each game runs for 1200 game frames, each lasting 50ms for a total of up to 60 seconds of playtime or a total of 3 goals, whichever happens first. Each agent is worth 25 points and the grade/figure of merit increases linearly from 0 to 25 depending on the player's ability to score goals in each game. This tournament is provided in the starter code [7].

2.3 Agent Telemetry

At each decision point, the game calls the agent function (JIT script) with the game state and expects in return an action list to proceed for the next state update. The game state contains information regarding the team player's state (`pstate`), the game state (`soccer_state`) and opponent players state (`opponent_state`) [8]. From this information, specific metrics are extracted such as angles differences among kart, puck and goal for navigation decisions. The three features used the most in our agent were derived from the kart location, angle, and puck position. The goal centerline position is also important, but it is fixed at team assignment (red vs. blue) and does not change throughout the match unlike the kart and puck positions/angles.

3. AGENT DESIGN

In this section, we discuss the agent design approaches and tradeoffs.

3.1 Kart Choice

Each of the 2 players on a team can be a kart from the 3 size classes [9]:

- **Light**: Wilber, Hexley, Kiki, Sara the racer
- **Medium**: Tux, Konqi, Suzanne, Adiumy, Xue, Gavroche, Nolok, Gnu, Emule, Sara the wizard
- **Heavy**: Pidgin, Amanda, Beastie, Puffy

Each class and kart offer tradeoffs in weight/acceleration, maximum speed, hitbox size, and turning. After extensive interactive experimentation, the light kart '`Sarah the racer`' and medium kart '`Emule`' were chosen as the kart types for the team. While the light kart '`Sarah the racer`' was very agile, we found that having the same kart]using the same logic yielded worse performance due to within-team interference since both players were trying to do the same action and have the same driving dynamics. By choosing the second kart to be a medium kart '`Emule`', the different driving dynamics would create emergent team play behavior not seen in identical player teams. With a slower turning, or faster accelerating kart, catching rebounds or blocking the other team players became more common with mixed kart teams.

3.2 Initial Position

When the karts in the team are not the same, the order in the initialization will determine the initial position and this can significantly alter the first few seconds of the game.

3.3 Agent Algorithm

Our initial algorithm checked if the kart to puck angle difference (`k2pad`) and the puck to goal angle difference (`p2gad`) were less than a certain threshold (0.1) to set acceleration to 1.0 as both kart and puck were going the same direction or 0.4 to enable tighter turning. An additional threshold check was used for a narrower angle for `k2pad` and `p2gad` of 0.02 to trigger '`nitro`' and '`fire`' as this would speed the kart and puck towards the goal as well as potentially kick the puck with a projectile into the goal, respectively. Additional checks to limit kart velocities dependent on the direction of the kart and puck towards the goal were implemented.

Using fixed values for some of the thresholds, an optimization study was conducted to explore and fine tune kart to puck angle difference limits, turn acceleration derating, and max velocity given the parameter ranges in Table 1. One thousand tournaments were conducted with the grader to optimize the score and the improved scores can be seen in the monotonically increasing blue line in Figure 2 using commercially available SHERPA optimizer in Siemens HyperLynx Design Space Explorer [10].

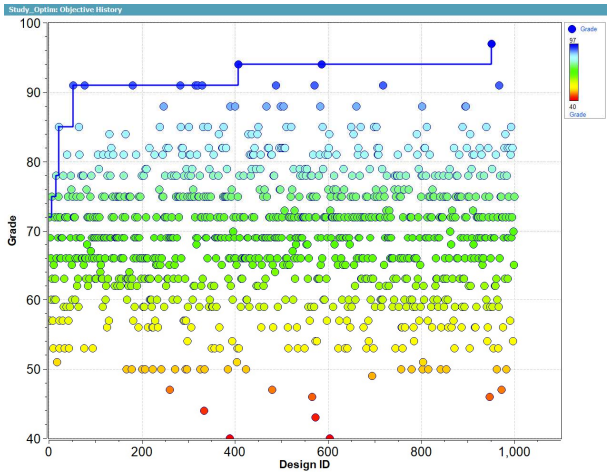

Figure 2. Grade optimization with max velocity, turn acceleration, and kart to puck angle difference parameters for 1,000 tournaments.

Table 1. Parameter Ranges and Optimization Results

Parameter	Min	Max	Best
`k2pad_limit`	0.01	0.50	0.275
`turn_acceleration`	0.01	1.00	0.4
`max_velocity`	5.0	25.0	9.13
Grade	40	97	97

With optimized driving dynamic parameters for the player's algorithm, additional exploration was done to see if further improvements could be made. We then discovered how chaotic the game is.

3.4 Agent Algorithm Challenges

As additional studies were conducted to further optimize and understand the effect and sensitivity of the parameters, the true chaotic nature of the game emerged as sensitive dependence on initial conditions commonly known as the 'butterfly effect' became evident.

With the first simple study, the turn acceleration (`turn_accel`) was fixed at a value of 0.4 and the other two parameters of `k2pad_limit` and `max_velocity` were swept with a uniform fine grid of 31 values each for a total of 961 datapoints. The grader result surface response is shown in Figure 3. Despite achieving good performance at certain points, even small changes could result in drastic outcome differences. For values of `max_velocity` of 12.3, 13.0, 13.6 for example, the scores were 60, 88, 54 with a fixed value of 0.484 for the `k2pad_limit`.

As the max velocity threshold value exceeded 21, we can see somewhat consistent behavior with minor variations. This is primarily due to the condition being triggered much less often, thus having a smaller overall impact on the outcome.

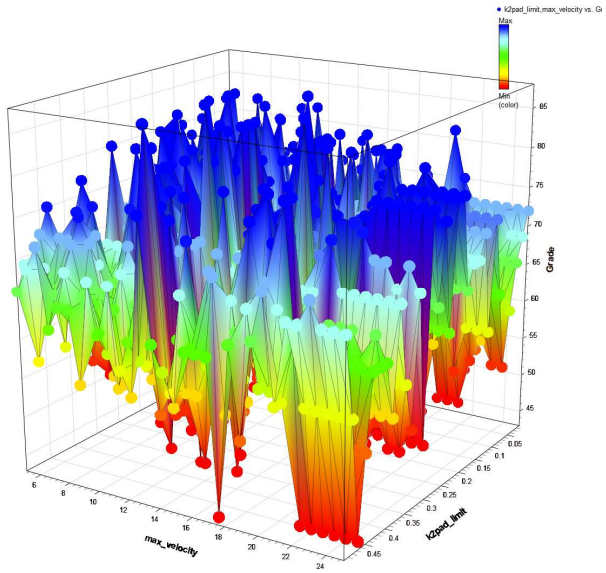

Figure 3. Grader 2D Surface vs. max velocity and kart to puck angle difference parameters.

In another experiment, the `k2pad_limit` was fixed at 0.353 and the `max_velocity` and `turn_acceleration` were swept in 21 points each for a total of 441 observations shown in Figure 4. Like the previous experiment, good parameter setting results were surrounded by poorly performing ones. Determining parameter values is far from trivial.

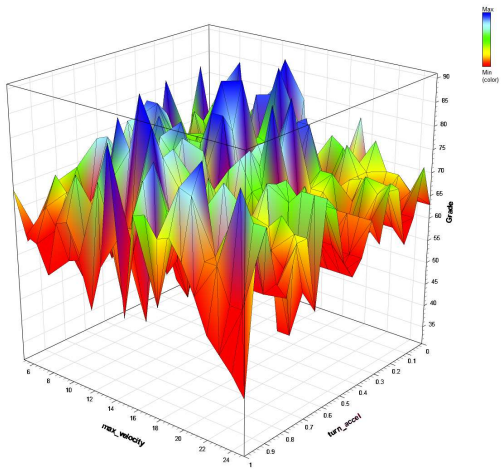

Figure 4. Grader 2D Surface vs. max velocity and turn acceleration parameters.

What we observed was that these small changes in parameters would have minor deviations in kart trajectories, however, whenever there were collisions with the puck or other players or the timing thereof, this could completely change the direction and velocities of the elements. As each frame is dependent on the previous these small changes had a compounding effect that made predictability difficult. Sometimes the changes would trigger 'errors' in the other teams playing making them score on their own goal for a very specific set of circumstances, however, this

behavior was not generalized. A different approach from raw optimization on grader score was needed.

3.5 Agent Design: Behavior Cloning with Human Feedback

Given the initial experience trying to optimize the vehicle driving algorithm, thresholds and dynamics with the grader score, a simpler model with less logic was implemented using Behavior Cloning with Human Feedback (BCHF). There are many aspects of the environment that make optimization difficult such as availability of in-game items, chaotic nature of small changes, player-to-player and player-to-environment interactions. To simplify the learning and evaluation, we removed 3 of the 4 players from the environment by fixing their action to `steer`=0.0, `brake`=True, `acceleration`=0.0. This would make them back-out into the goal and exit the arena to avoid interfering with the player that is being evaluated. The number of input and outputs were also reduced to minimize complexity. The final model only had 2 inputs (`k2pad` and `k2gad`) and one output (`steering`). Through the BCHF process and reviewing videos of the player's behavior, we found it often getting stuck at walls when driving forward. With a much tighter turning radius when driving backwards, it was getting stuck much less in addition to being able to better aim for and control the puck direction to the goal. By fixing the `brake`=True and `acceleration`=0.0 it would always operate in reverse. The maneuverability was much higher at the cost of reduced speed.

A simple observation that with just a single parameter, kart to puck angle difference (`k2pad`) it can be scaled and directly applied to the steering output to make the kart move towards the puck. While effective in guiding the kart to move towards the puck center, the result will be that it will just push the puck in this direction as shown with the red arrow in Figure 5 and scoring will be a matter of luck as kart and puck will bounce off the perimeter until it scores into either goal. However, if we offset this angle aimpoint towards the yellow star as shown with the blue arrow, it will have the effect of approaching the puck in an arc instead of straight line and it will bump the puck towards the goal (dashed blue arrow) instead of pushing it forward. Using the kart to goal angle difference (`k2gad`) as feedback into the steering, this greatly increases the quality of the player.

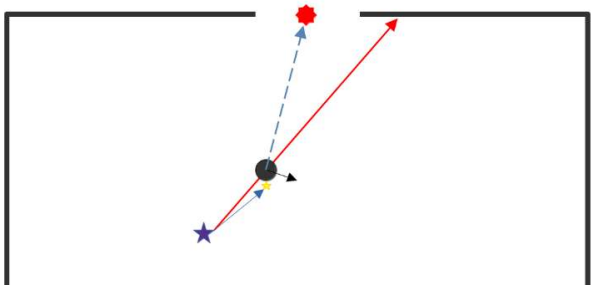

Figure 5. Direct vs. offset steering

This simple agent with just 2 inputs and 1 output was able to achieve a score of 97 on the 32 game, 4 player / 8 settings tournament locally and was used to generate the training data for the DNN agent model.

4. DNN Model

Given input features that are boolean or float and predicting the same, we decided to start with a very simple DNN and then grow model complexity based on performance. The first model created has a structure of:

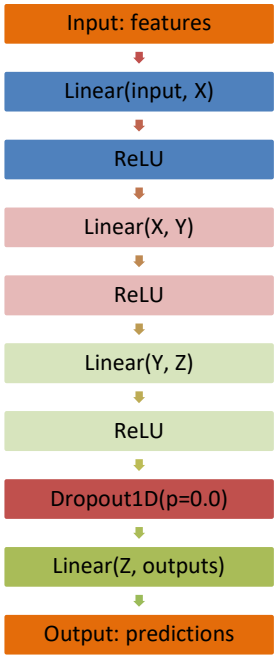

Figure 6: DNN Architecture Used in Development

This proved to be very effective and was not modified during the continued R&D into other aspects of the challenge.

4.1 DNN Architecture

Instead of changing the primary model architecture, we explored the performance and cost against different shapes for the hidden layers, in addition to the benefits of fewer features and predictions. The resulting models that were considered with the scores from local grader testing are shown in Table 2 below.

Features	Layers	Predict Outputs	Params	Local Score
5	64, 128, 256	5	43,013	56
3	64, 128, 256	3	42,371	50
3	64,32,16	3	2,915	69
3	64,32,16	1	2,881	78
2	**64,32,16**	**1**	**2,817**	**97**
2	32, 16, 8	1	769	88
2	16, 8, 4	1	225	82
2	8,4,2	1	73	72
2	4,2,2	1	31	85

Table 2. Model shapes, parameters and resulting scores

4.2 Data Generation

Data was generated using code modifications to the provided functions for running test games against individual agents. This "generator" utility takes an input number of total frames to generate and then collects the target feature data and resulting actions in the following ways:

- 50% - same starting ball locations as grader
  - 4 games per side
  - unlimited scoring
  - 1/16th of total data generated per game
- 50% - run "one-shot" games until all data collected
  - Randomized ball location
  - Randomized ball velocity
  - Randomized side

This generator was run for 50K frames against each agent and using different Karts resulting in ~400K datapoints.

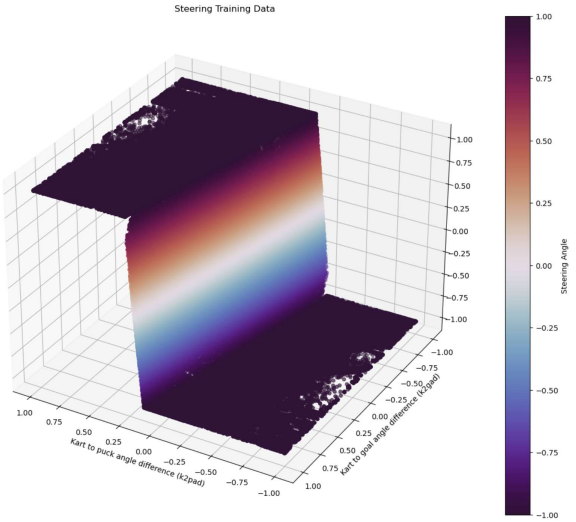

Figure 7: Output steering vs. input angle differences training data point coverage.

Visualization of all ~400,000 training points across the two input parameters can be seen in Figure 7. There are some areas when the steering values are +1 / -1 where data was not captured, but interpolation and generalization of the model should not be affected by its absence. More importantly, in the region where there is a sharp transition between maximum steering values, the training data provides ample quantity of samples.

Data point density for `k2pad` and `k2gad` vs. `steering` is shown in 8. This plot shows the concentration of data points from the training runs where the kart agent collected data. Much of the time, the kart was aligned with the puck with a `k2pad` of small magnitude with the puck directed at the goal with small `k2gad`.

While the coverage of the training points is not perfect as we can see small gaps in the plot where training data is not present, however, we can note that the simple strategy provides a simple function for the imitation agent to learn which makes the task easier to replicate.

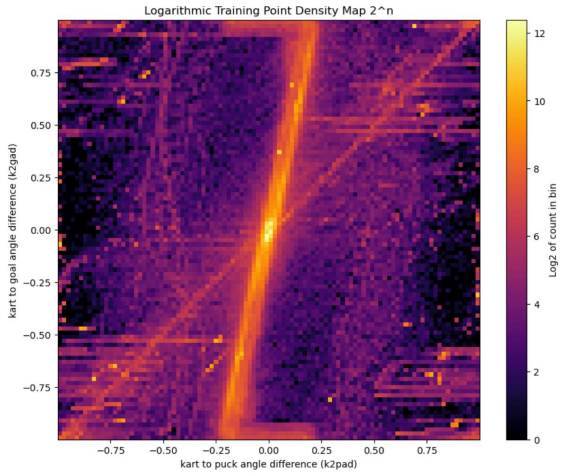

Figure 8: Log2 count of training sample heatmap.

4.3 Training

The training loop was a standard Torch structure that used data for training and validation as an 80%/20% split. The validation data was not part of the training during hyperparameter selection or final training.

4.3.1 Training options

The following training options were explored in our hyperparameter search:

- Optimizer: SGD, Adam, AdamW
- Loss function: L1Loss, MSELoss
- LR Scheduler: Plateau
- Training batch size
- Dropout probability

4.3.2 Training metrics

Running the grader to generate scores to guide training is not feasible. One standard tournament takes almost 2 minutes on a GPU system. And the high variability in the scoring output would likely not create a predictive signal for model convergence.

4.3.3 Training and validation loss metrics

The primary metrics we could observe that would indicate positive learning was measuring how close the training and validation losses were to each other (`loss_delta`). If `loss_delta` starts to grow, this could indicate a stalled learning process or overfitting to the training data.

As an example, when down sampling data to see if we really need 400K data points, we can see that there is a strong divergence between training and validation loss as the dataset gets smaller. Even though the local grades show a decent ability to score, it is very unlikely that the model would generalize correctly across many systems and games (Figure 9).

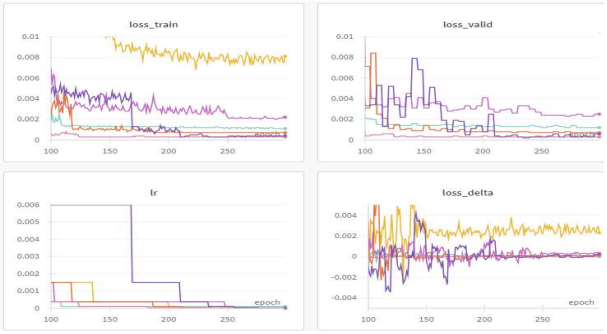

Figure 9. Training Loss.

4.3.4 K-fold cross-validation approach

The wide range of training options results in a very large search space to try and find the optimal set of hyperparameters for training. We started with coarse manual search to see how the proposed model architecture would perform in the games. One surprising outcome was that any amount of dropout would stop all learning once we were down to only a few features focused on a single output.

Once we were confident in the approach, we implemented a full k-fold validation structure to run a series of short epoch cycles using Optuna [11] to perform Bayesian search and help identify our parameter space. Figure 10 shows an example of this approach. In this case the search was across 10 different learning rates and 3 different optimizers focusing on training and validation loss. It shows that AdamW with a starting LR of 0.006 has a healthy combination of low loss with a small difference between training and validation.

SUM of train_loss	lr										
optimizer	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.01	Grand Total
adam	0.00515	0.00065	0.0007	0.0005			0.0004		0.0005		0.0079
adamw			0.0004		0.0005	0.00035	0.0005	0.00075	0.00055	0.0005	0.00355
sgd	0.00045			0.0008	0.001	0.00075	0.0007	0.00045		0.00055	0.0047
Grand Total	**0.0056**	**0.00065**	**0.0011**	**0.0013**	**0.0015**	**0.0011**	**0.0016**	**0.0012**	**0.00105**	**0.00105**	**0.01615**

SUM of valid_loss	lr										
optimizer	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.01	Grand Total
adam	0.0047	0.0006	0.0007	0.0006			0.0005		0.00065		0.00775
adamw			0.00045		0.0004	0.0004	0.0007	0.0009	0.0004	0.00045	0.0037
sgd	0.00045			0.0021	0.00055	0.002	0.0004	0.0004		0.00045	0.00635
Grand Total	**0.00515**	**0.0006**	**0.00115**	**0.0027**	**0.00095**	**0.0024**	**0.0016**	**0.0013**	**0.00105**	**0.0009**	**0.0178**

Figure 10. Example of Optuna based search for hyperparameters.

In addition to optimizer and learning rate, this technique was applied to explore model shapes, loss functions, weight decay and training batch size.

Some of these results suggested that our model architecture was likely much larger than needed since much of the search space would converge to near zero loss in only a couple hundred epochs. The hyperparameters used to train the final model are shown in Table 3.

Table 3. Final Hyperparameters

Parameter	Value
Optimizer	AdamW
`learning_rate`	0.006
`weight_decay`	1e-5
Scheduler	Plateau
`patience`	20
`step`	0.25
Training batch size	128
Hidden layers	16, 8, 4
Loss function	L1Loss

5. Results

Satisfactory results were achieved in the design of the expert controller, capture of data, model training and performance both locally and on the remote grader despite the challenging and inconsistent environment.

5.1 Tournament Results

While the model and training were conducted with the local tournament grader, the final agent evaluation took place in a remote grader that runs a subset of the game variations. Local scores ranged from 66 to 97 depending on the system and the remote grader yielded a score of 82 as shown in Table 4.

Table 4. Tournament Score: 82/100

Agent	Goals Scored	Game 1	2	3	4
`geoffrey`	3	`1:1`	`1:0`	`1:2`	`0:2`
`jurgen`	2	`1:0`	`0:3`	`1:1`	`0:3`
`yann`	4	`2:0`	`1:2`	`1:0`	`0:3`
`yoshua`	12	`3:0`	`3:0`	`3:0`	`3:0`

5.2 Reproducibility Challenge

One of the challenges of this environment, beyond its chaotic simulation nature, is its difficulty in reproducibility of results. We tested this issue within the same system, across different operating systems (Windows, Ubuntu, MacOS), across different CPU architectures (AMD Ryzen 9, AMD Threadripper, Intel Xeon, Apple M2 max).

5.2.1 Within Same System

When the tournament grader is run on the same system, we found the results to be 'mostly deterministic'. In one experiment, out of 100 runs with the same agent, 3 out of the 4 agents had the exact same 8 game scores. The agent that had discrepancies, `jurgen`, had 7 of its 8 scores consistent across all 100 runs and the second game with 0:1 result 62% of the time and 0:3 38% of the time. While this did not alter the overall resulting score, it clearly showed that there are minor variations within the same system for the same run.

5.2.2 Different Operating Systems, CPUs

Different operating systems and CPUs yielded a much higher grader discrepancy in results despite consistency in the environment setup (python 3.9, Torch 1.9, etc…), and code synchronization via Github.

Table 5. Reproducibility Challenge

Operating System	CPU	Goals Scored				Score
		Geo	Jur	Yan	Yos	
Ubuntu 23.10	Intel i7	9	7	9	19	97
Ubuntu 22.04	AMD 5950x	12	3	10	20	84
MacOS 14.4	M2 max	10	3	6	21	78
Windows 11	Intel i7	9	1	9	22	78
Windows 11	Intel i9	9	1	9	22	78
Windows 11	AMD 5950x	5	2	6	24	66
Windows 11	AMD 5975WX	5	2	6	24	66
Remote*		3	2	4	12	82

* Remote system runs 4 games per agent instead of 8 locally

We can see from Table 5 that for a given agent and same environment, the scores can vary widely (66 → 97). While the local tournament uses the 8-game format, the final tournament (remote grader) uses the four fixed ball positions but randomizes the team assignment which essentially makes the final score a randomized variable with a binomial distribution. Even when an agent has symmetric behavior with respect to the team assignment, the agents it is playing against do not, so different results are inevitable.

6. CONCLUSION

In this work, we described the challenges and approach to apply imitation learning for a Deep Neural Network (DNN) to control players in a 2v2 ice hockey kart game tournament. When we explored the environment and criteria of success to design an expert algorithm to score goals by exploring the environment, controls, and game mechanics, we discovered the chaotic nature of the environment and difficulty of using tournament results as the only figure of merit. We created an algorithmic player using Behavior Cloning with Human Feedback (BCHF) to imprint the desired player and team behaviors in a more controlled setting. We used this automated player for imitation learning to generate training data for the supervised DNN model creation. After architecture search, hyperparameter optimization, and training, the DNN agent was evaluated against different tournament agents to measure success achieving a score of 97 locally and 82 on the remote test. Lastly, we discussed the 'mostly deterministic' nature of the experiments and reproducibility difficulty of the environment from the operating systems (Windows / Ubuntu / MacOS) as well as CPU (AMD Ryzen 9, AMD Threadripper, Intel i7, Intel I9, Apple M2 max).

7. REFERENCES

[1] Chai J., Li A., Deep learning in natural language processing: A state-of-the-art survey, Proceedings of the 2019 international conference on machine learning and cybernetics (icmlc) (2019), pp. 1-6, 10.1109 / ICMLC48188.2019.8949185

[2] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961

[3] V. R. Konda and J. N. Tsitsiklis, "On Actor-Critic Algorithms," SIAM Journal on Control and Optimization, vol. 42, no. 4, pp. 1143–1166, 2003.

[4] Schulman, J., Levine, S., Abbeel, P., Jordan, M. Moritz, P.. (2015). Trust Region Policy Optimization. Proceedings of the 32nd International Conference on Machine Learning, in Proceedings of Machine Learning Research 37:1889-1897

[5] J Schulman, F Wolski, P Dhariwal, A Radford, O Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347, 2017

[6] SuperTuxKart Developers. (n.d.). stk-code: The code base of SuperTuxKart [Computer software]. GitHub. Accessed April 20, 2024, from https://github.com/supertuxkart/stk-code

[7] Krähenbühl, P. (2024). Final project - SuperTuxKart ice hockey. Retrieved April 20, 2024, from https://www.philkr.net/dl_class/homework/final/

[8] Pystk documentation, Docs/Race/Game State, https://pystk.readthedocs.io/en/latest/state.html#pystk.Player

[9] "Which Kart is Fastest? [1.0]" SpeedRun.com https://www.speedrun.com/stk/guides/ocm26

[10] SHERPA Optimization, Siemens HyperLynx Design Space Explorer, https://eda.sw.siemens.com/en-US/pcb/hyperlynx/release-highlights-2-11/

[11] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In KDD.