Analyzing the Behavior of an Imitation-Based Agent in a Simulated Setting

Ali Aboughaly, Carlos Barbosa, Hossameldin Rabah, Diego Sariol

Abstract

A comprehensive investigation of imitation learning (**hussein2017imitation**) within the context of simulated ice hockey matches using the Python game engine SuperTuxKart (**supertuxkart**) is presented. By employing a state-based model, this research analyzes dynamic game conditions to imitate an exemplary agent, Jurgen. A selected dataset was used to train a feed-forward neural network, replicating Jurgen's decision-making and actions.

Keywords: SuperTuxKart, imitation learning, Python, deep learning, artificial intelligence, machine learning

Keywords: SuperTuxKart, Imitation Learning, Python, Deep Learning, Artificial Intelligence, Machine Learning.

1 Introduction

In the evolving scenario of artificial intelligence, the adoption of imitation learning techniques have expanded from traditional applications into more dynamic and complex environments. Imitation learning involves training algorithms to mimic expert behaviors, offering a pathway to sophisticated AI development without the necessity for intricate programming of every possible scenario. This study explores the application of an imitation learning framework

within the context of an ice hockey match simulated in the SuperTuxKart game mode.

The methodology centers on a state-based model – a framework that continually captures and analyzes the current conditions of the game environment. These conditions include players positions, puck location and velocity, and other game status, which are encapsulated in a dataset store in a `pickle (.pkl)`. This file format allows data manipulation to extract the desired information and use it for training the neural networks, providing a comprehensive snapshot of gameplay dynamics at any given moment. For this study, several pre-defined agents were analyzed to identify the most effective strategies and behaviors. The Jurgen agent was selected based on its superior performance metrics, including a higher number of matches won and goals scored compared to its counterparts. This agent's behavior serves as the baseline model expert for our imitation learning algorithm.

To evaluate the effectiveness of the imitation learning model, instead of employing a broad array of simulations, our study focused on a meticulously selected dataset derived from scenarios where the Jurgen agent demonstrated optimal performance. This approach involves analyzing a single match data file, representing a game where initial puck positions and velocities were randomly set but ultimately led to exemplary gameplay by the Jurgen agent. This targeted analysis allows us to deeply explore the intricacies of game dynamics and AI decision-making under near-optimal conditions. The Jurgen player agent generates 10 distinct features from raw game data, incorporating several game-specific metrics. Notably, it computes the `kart_to_puck_angle_difference` by evaluating the relative angles between the kart's orientation and the puck's position, which is crucial for strategic maneuvering. Additionally, the player's actions—acceleration, steering, and braking—are extracted and fed into a convolutional neural network (CNN) [3]. This neural model is tasked with predicting the player's next move based on the current game state, aiming to replicate the expert behavior exhibited by the Jurgen agent. This report details the design, implementation, and preliminary outcomes of employing a state-based imitation learning model in a highly controlled environment. By focusing on a single, high-performance instance, we aim to dissect

the effectiveness of specific AI strategies and their impact on game outcomes. This method highlights the potential of advanced machine learning techniques [4] to enhance not only gaming experiences but also complex decision-making processes in simulated environments.

2 Design

2.1 Network Design

Artificial neural networks (ANNs) [5] form the structure of modern machine learning and deep learning systems [6]. Inspired by the structure of biological brains, they consist of interconnected "neurons" organized into layers. Each neuron receives inputs, processes them using a mathematical function, and passes the result as output to subsequent layers. The strength of the connections between neurons is represented by weights, which are continuously adjusted during the training process. In deep networks, the "deep" refers to the presence of multiple hidden layers between the input and output layers. These hidden layers allow neural networks to learn increasingly complex representations of the data. Deep learning is a sub-field of machine learning focused on the use of these deep neural networks for tasks like image classification, natural language processing, and decision-making. Traditional CNNs [7] are primarily designed to process visual data. However, the core principles of deep neural networks make them well-suited for imitation learning with structured, non-visual state data, as well.

- **Feature Encoding:** Transform raw game state inputs into a numerical framework that facilitates neural processing, enabling the network to comprehend and evaluate the current scenario within the simulated environment.
- **Action Prediction:** Learn to anticipate an expert player's tactical responses, including acceleration, steering, and braking maneuvers, based on the encoded features of the game state, thereby mirroring strategic decision-making processes

The neural network is designed to predict player actions based on the given state of the game. The network takes in not only the features of the game state but also the expert actions during the training phase. This allows the network to learn the complex mappings between the game states and the expert's responses. During training, each input feature vector is paired with the corresponding expert actions. These actions are:

- **Acceleration:** A continuous value, typically between 0 (no acceleration) and 1 (full acceleration).
- **Steering:** A categorical value that could be encoded as -1 for left, 0 for straight, and 1 for right.
- **Braking:** A binary value, often 0 for no brake and 1 for brake.

The network's output layer is designed to predict these actions. For acceleration and braking, which are continuous and binary respectively, the network may use a single neuron with a sigmoid activation function to output a value between 0 and 1. For steering, since it is categorical, the network uses a layer that outputs a probability distribution over the three possible actions. The model then uses the highest probability to decide the direction of steering.

2.2 Game Result Analysis

A thorough investigation was conducted to comprehend the complexities of the game dynamics and identify the key parameters influencing match outcomes. The primary objective was to highlight which factors influenced the performance of each agent, thereby guiding the selection process for imitation. Initially, a comprehensive match result data was generated by simulating matches involving all four agents + AI playing against each other in a standard setting. Yann appeared to be the best agent. The second approach is to randomize the initial match setting. This scenario captured the variability in match outcomes, revealing notable disparities based on several factors:

Table 1: Match Outcomes between Agents in a Standard Settings

Team 1Team 2	Jurgen	Yann	Yoshua	Geoffrey	AI
Jurgen	—	Jurgen (2)	Jurgen (1)	Draw	Jurgen (1)
Yann	Yann (2)	—	Yann (1)	Yann (2)	Tie
Yoshua	Jurgen (3)	Yoshua (2)	—	Draw	AI (3)
Geoffrey	Draw	Geoffrey (1)	Draw	—	Tie
AI	Jurgen (3)	AI (1)	AI (1)	AI (1)	—

1. **Agent Location:** (Home vs Guest): The location of the agent within the game environment significantly impacted its performance, suggesting a potential bias in certain game scenarios.

2. **Start Puck Location:** The initial position of the ball exerted a considerable influence on match dynamics, affecting the strategies employed by the agents and ultimately determining the outcome.

3. **Start Puck Velocity:** Variations in the velocity of the puck at the beginning of each match introduced unpredictability, contributing to the diversity of match outcomes.

4. **Agents:** The files included five Agents (Yann, Jurgan, Yoshua, Geoffery and AI).

Table 2: 10 Match Outcomes between Agents in a Randomize Settings

	Jurgen	Yann	Yoshua	Geoffrey	AI
Jurgen	—	Jurgen (6,11)	Jurgen (7,17)	Jurgen (7,15)	Jurgen (7,16)
Yann	Jurgen (7,13)	—	Yann (7,9)	Yann (6,7)	AI (8,13)
Yoshua	Jurgen (9,21)	Yann (6,9)	—	Yoshua (2,2)	AI (8,12)
Geoffrey	Jurgen (3,12)	Yann (6,7)	Yoshua (5,7)	—	AI (6,10)
AI	Jurgen (8,18)	AI (5,10)	AI (6,12)	AI (5,10)	—

Table 2 summarizes the performance of each agent over 10 randomized matches, outlining the victor, number of matches won, and total goals scored. While the preliminary data suggested Yann as the leading agent, further scrutiny of match recordings indicated a need for a more nuanced understanding of gameplay dynamics. It became evident that Jurgen's

strategic execution outperformed others in randomize environment, marking it as the prime candidate for our imitation learning model.

3 Approach

The goal is to mimic only one agent, Jurgen. To do this, we aimed to gather as much data as possible by varying the key parameters identified in the data analysis section. We ended up with 3 working models, in each of which we aimed to enhance the older model by varying some hyper parameters mentioned in the below table

3.1 Data Generation

The analysis of our data guided us in identifying the essential hyper-parameters required for data generation. Typically, prioritizing quality over quantity is crucial, but in our case, we required an extensive dataset to ensure our model could observe a wide range of feature values and avoid exhibiting abnormal behavior when encountering unseen data. To amass the substantial amount of data needed, we focused on varying four key parameters outlined in our data analysis:

- **Agent Location:** Home or Guest
- **Start Puck Location:** Ranging from [-15, +15]
- **Puck Velocity:** Spanning from [-1, +1]
- **Agent to Imitate:** Other agents + AI.

To effectively train an Imitation Agent model to properly mimic the “expert” agent’s actions, it is beneficial to have enough good quality data to train on, as a model trained on a surplus of information can lead to the Imitation model learning incorrect decisions when presented with certain scenarios. This is why it is extremely important to essentially fine

tune / prune the data pool to "feed" in `SuperTuxKart` hockey matches where the player agent we are trying to imitate perform exceptionally well.

The initial provided code base allows for the generation of tournaments matches between any two agents. A total of 5 agents: Yoshua, Yann, Jurgen, Geoffrey, and the AI itself, means that we can generate individual matches with the Jurgen agent and one other agent, even itself. Striving to purely imitate the Jurgen agent, it was extremely crucial to ensure Jurgen was a kart player in each, and every tournament run for data collection. Each match allows for the recording and saving of all states throughout the tournament for both teams and all their corresponding players. This collection of states throughout a single match contains all the crucial information regarding the specific teams and their agents that the model will imitate.

The automation of this entire process can be done by constructing a Python script which generates a list of commands that can be run through the Terminal to execute tournament matches with randomized uniform distributions for both the puck's starting location and its velocity. Additionally, these command combinations can be automated as well to execute each tournament match and save the resulting match score information to a single file whilst saving the different states of each match. This allows for filtering out the generated Pickle files in which Jurgen agent did not perform well as our model strives to imitate the correct and valid game actions taken by Jurgen.

3.2 Data Preparation

To prepare the dataset for training our imitation learning model, we followed a systematic approach for processing data encapsulated in Python pickle files.

We implemented a Python function, `collect_training_dataV2`, to parse through each pickle file within a specified directory. This function was developed in two iterations to refine the feature set utilized for training:

1. **Initial Version (All Features):** Initially, all features extracted from the gameplay

states were included in the training set. This broad approach was aimed at providing the model with maximum contextual information.

2. **Refined Version (Selected Feature):** After conducting multiple training sessions, it became clear that not all features contributed equally to the model's learning efficacy. We pivoted to a strategy that focused on a key feature: `kart_to_goal_line_angle_difference`. This specific feature, which encapsulates the angular alignment of the kart relative to the goal line, proved instrumental in enhancing model performance.

3.3 Training

To train the neural network model on the large quantities of input Pickle files, data handling functions were necessary to efficiently read and load in all the state information from each Pickle file which contained crucial information about both teams, their respective kart players, actions taken in each state of the game, and information about the hockey puck itself. Once all the desired match files were properly read in, the next stage was to sift through and filter out unnecessary information that did not pertain to the team in which Jurgen agent represented.

This process involved utilizing modified functions from Jurgen agent in the given code base. As our model strived to purely imitate Jurgen in its decision-making capabilities and resulting game state actions, the analysis of input game data must also mirror that of the selected imitated agent. This meant extracting the same subset of game features such as kart location, puck location, and other crucial state information from each player on the intended target team. Allowing for the same information to be derived and analyzed in the imitation learning phase of learning from Jurgen's playstyle and decision-making.

Our training function utilizes the Adam optimizer [8], a widely adopted algorithm known

for its efficiency in optimizing deep learning models. Employing backpropagation, the fundamental technique for updating neural network parameters,. Moreover, we utilize two distinct loss functions tailored to different aspects of our model's output. Specifically, the mean squared error (MSE) loss function is employed for the acceleration predictions, used specifically for continuous values.. For the brake and steering predictions, we utilize the cross-entropy loss function, which is well-suited for categorical prediction tasks. This loss function quantifies the difference between the predicted probability distributions and the true class labels for both braking and steering actions.

4 Results

4.1 Model's performance

This study commenced with the development of a baseline model, referred to as Model A. This model was trained exclusively on data from matches involving the agent "Jurgen" against the other opponents. Despite its simplicity and the limited dataset it was trained on, Model A demonstrated promising initial results, suggesting potential for further refinement and improvement.

4.1.1 Model A

- **Average Goals per Match:** 8.4
- **Total Points:** 26

These outcomes are visually represented in the accompanying bar graph, which displays the performance metrics obtained from ten trials using the grader. On the x-axis, we have the trial number, representing each iteration of running the local grader, while the y-axis denotes the number of goals scored in each respective trial.

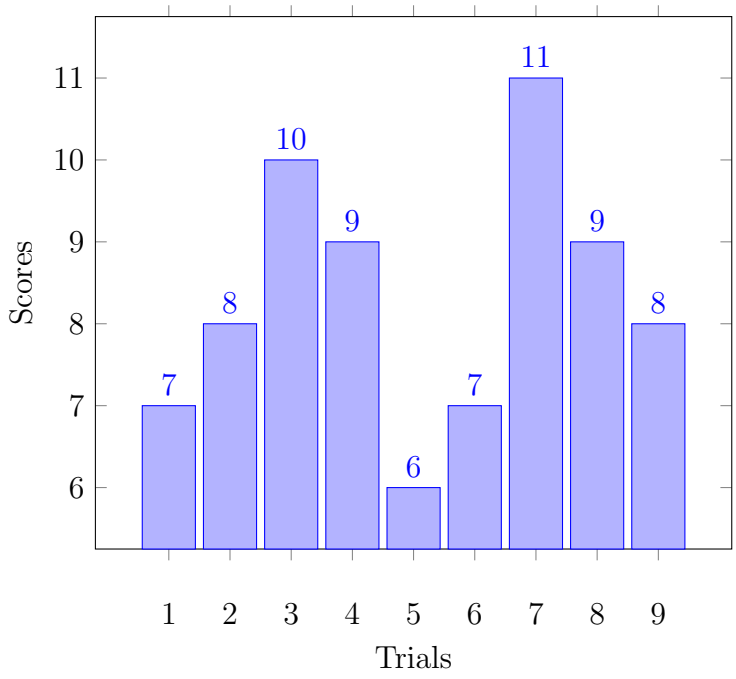
Algorithm 1 ImitationModel Class Pseudocode

```
class ImitationModel:
    Initialization:
        If training is True (default): set training mode
        Create a sequence of linear layers with ReLU activation:
            - Input layer: 1 element
            - Hidden layer 1: 32 elements
            - Hidden layer 2: 64 elements
            - Hidden layer 3: 128 elements
            - Output layer: 256 elements
        Create separate output layers for:
            - acceleration (1 element)
            - steer (3 elements)
            - brake (2 elements)
        Set training flag

    Set Training Mode:
        Update training flag

    Forward Pass:
        If input has only 1 dimension (single element):
            Reshape input to add a batch dimension (1, element)
        Get the last element of the input
        Pass the element through the linear layer sequence (hidden state)
        During evaluation (not training):
            Get predicted actions from hidden state:
                - Steer: Choose element with highest value in steer output (argmax) and subtract 1
                - Brake: Choose element with highest value in brake output (argmax)
                - Acceleration: Clamp value between 0 and 1
            If brake is applied (argmax = 1): set acceleration to 0

    Return:
        - Acceleration, steer, and brake values
```

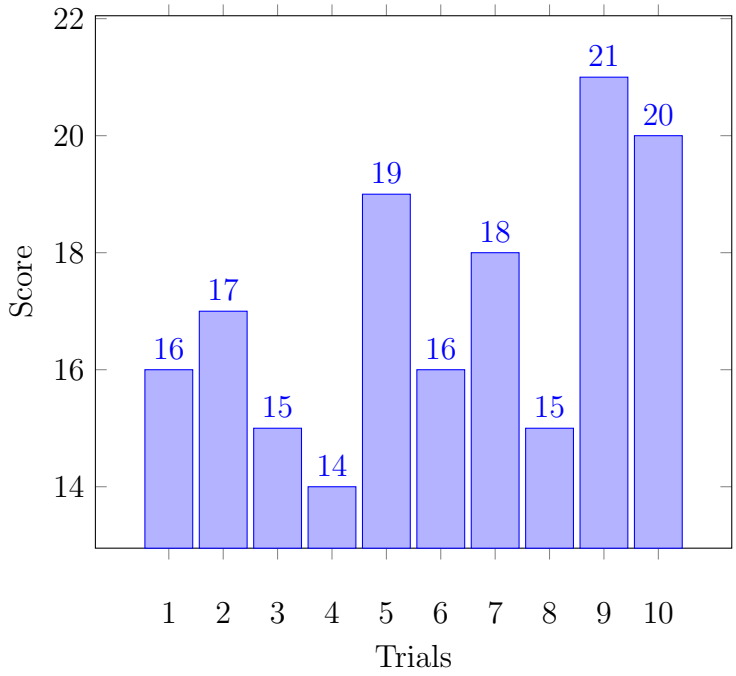


4.1.2 Model B

Subsequently, we developed Model B, which continued to utilize the same architectural framework as Model A but was trained solely on data involving matches between Jurgen and AI-controlled agents. This modification aimed to refine the learning process by focusing on a more consistent opponent type.

- **Average Goals per Match:** 17.7
- **Total Points:** 55

The performance improvements of Model B are illustrated in the subsequent figure, highlighting the enhanced outcomes from this targeted training approach.

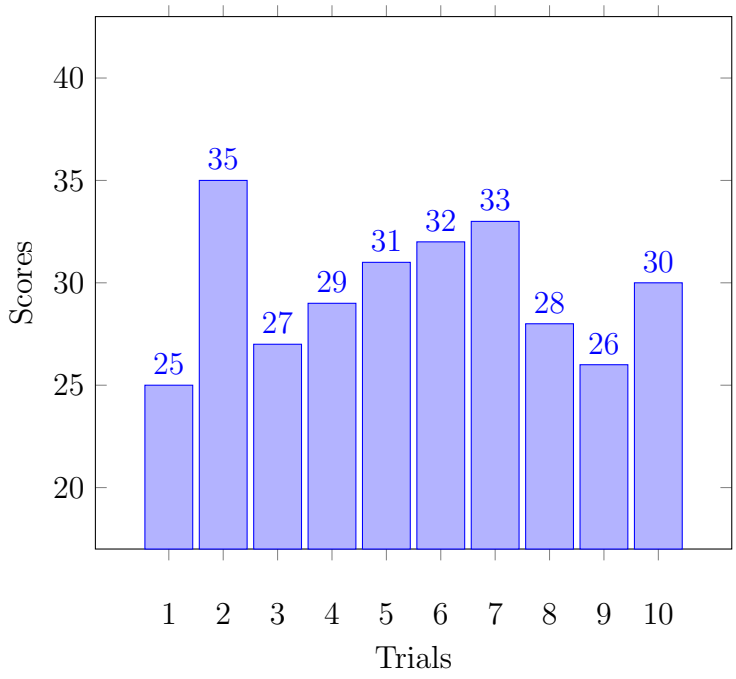


4.1.3 Model C

Building on the success of Model B, we introduced Model C, which featured a more complex architecture with additional layers and parameters. Model C was also trained exclusively on data from matches against AI opponents, like Model B. This model was designed to maximize learning efficiency and performance by leveraging a larger and more diverse dataset.

- **Average Goals per Match:** 29
- **Total Points:** 89

A comparison between the mean and median scores of all models revealed minimal discrepancies, indicating a consistent performance across trials. This consistency further validates the effectiveness of the progressive refinements made from Model A through Model C.



The substantial improvements achieved with Model C are depicted in another figure, which presents the model's performance across ten different trials.

Table 3: Performance Metrics Across Models

Model	Mean Goals	Median Goals	Mean Total Points	Median Total Points
A	8.4	8.5	26	27
B	17.7	17.5	55	55
C	29	29	89	91

4.1.4 Model C: Addressing Inconsistencies in Performance

Initial Inconsistencies

Model C initially exhibited variability in performance when evaluated using our local grading system. This prompted a series of adjustments to enhance model stability and improve predictive accuracy.

Fine-tuning Strategies

1. **Adjusting the Learning Rate (LR):**

   Our fine-tuning journey commenced by modifying the learning rate values. We embarked on a comprehensive exploration, varying the LR from 1e-1 to 1e-6, to discern its impact on model training dynamics. The adjustment in the learning rate had a positive impact, as evidenced by the improved performance in our loss metrics. The total loss—which combines the losses for acceleration, brake, and steering outputs—showed significant reduction, indicating better model learning and prediction accuracy.

   - **Starting Point:** We began fine-tuning our model by tweaking something called the "learning rate" (LR). This LR affects how quickly our model learns from data. We tried different LR values, ranging from 1e-1 to 1e-6, to see which one works best.
   - **Learning from Mistakes:** When we set LR to 1e-1, our model struggled a lot. It behaved unpredictably, shooting way past the mark and giving us very high loss values. This made it hard for the model to learn anything useful.
   - **A Step Forward:** Trying LR 1e-2 was a bit better. The model's behavior became steadier, and its loss values slowly decreased over time. We started seeing signs of improvement and learning as we went through more training rounds.
   - **Getting Closer:** As we moved to LR 1e-4, things got even better. The model started to converge faster, reaching its lowest loss values in less time. This was a big win for us, showing that adjusting LR could really boost our model's performance.
   - **The Small Print:** We also tried LR 1e-6, but it taught us an important lesson. While the model did improve over time, it took much longer to reach good results. This reminded us that setting LR too low can slow down progress, even if it eventually gets there.
   - **Visualization and Validation:** The figures, generated using the tensorboard

tool, provided comprehensive insights into the trajectory of the loss function across 50 epochs for each LR setting. We observed that lower learning rates (LR), particularly at 1e-4, significantly improved model performance, leading to quicker convergence and better outcomes compared to higher LR settings like 1e-1. The figures clearly demonstrate that increasing LR values resulted in decreased model performance, with LR 1e-4 showing the most optimal balance between speed and effectiveness, while LR 1e-1 exhibited erratic behavior and poor learning dynamics, highlighting the critical role of LR adjustments in enhancing model training.

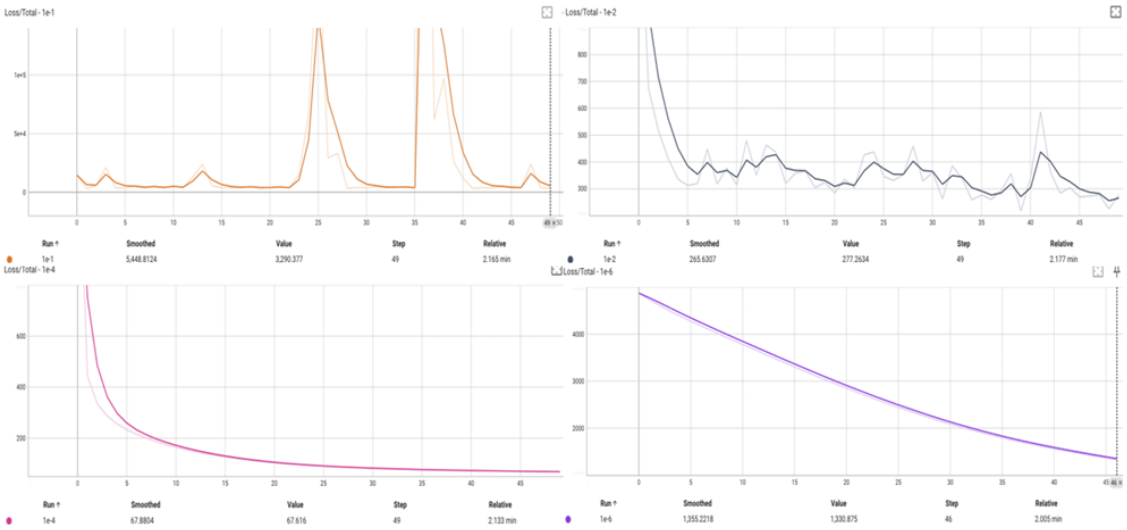


Figure 1: Learning Rates

2. **Increasing Training Data:**

   - To further stabilize the model's performance and ensure robustness, we increased the quantity and quality of training data. This expansion provided the model with a more comprehensive set of scenarios, thereby enhancing its generalization capabilities across different driving conditions.

3. **Exploring Kart Type Variations:**

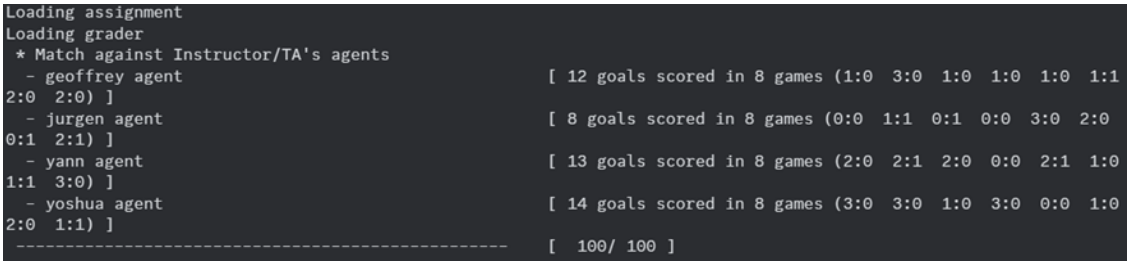


Figure 2: Local Grade Result

One significant aspect we investigated was the choice of kart type, which proved to have a substantial influence on the model's predictive accuracy and consistency.

- **Default Kart Selection:** Initially, our model training utilized the "Tux" kart as the default choice. However, we discovered that the performance of our model could be significantly enhanced by leveraging different kart types.
- **Performance Boost with "Sara_the_racer" Kart:** Among the available kart options, "Sara_the_racer" emerged as the most promising candidate for optimizing our model's performance. Utilizing this kart type consistently resulted in remarkable improvements, with the model consistently achieving high scores and goal counts. The following figure displays one of the results obtained utilizing this kart.
- **Promising Potential of "Sara_the_wizard" Kart:** While "Sara_the_racer" exhibited exceptional performance, our exploration also revealed promising potential in another kart type, namely "Sara_the_wizard." Although not as high-performing as "Sara_the_racer," this kart type demonstrated notable capabilities in enhancing the model's accuracy.
- To provide a comprehensive overview of the performance disparities across different kart types, we conducted a comparative analysis based on 10 iterations of running the local grader. The results, summarized in the table below, highlight the varying performance metrics associated with each kart type, further emphasizing the significance of kart selection in optimizing Model C's efficacy.

Table 4: Kart Performance Metrics

Kart	Mean Goals Scored	Max. goals scored	Total Points Mean
Beastie	28.6	34	86.8
Tux	29	33	89
Wilber	35.3	42	95.4
Sara the Wizard	36	42	97
Sara the Racer	42.7	49	98.6

5 Conclusion

The study commenced with the development of three progressively refined models: Model A, Model B, and Model C, each trained on different datasets and configurations. Model A, trained on Jurgen's data against various opponents, showed promising results despite its simplicity, averaging 8.4 goals per match and totaling 26 points. Building upon this, Model B focused on matches between Jurgen and AI-controlled agents, achieving higher performance with an average of 17.7 goals per match and 55 total points. Finally, Model C, featuring a more complex architecture and trained exclusively on AI opponent data, exhibited significant improvements, averaging 29 goals per match and 89 total points. Fine-tuning strategies, such as adjusting the learning rate, were instrumental in enhancing Model C's performance. Initially, Model C faced inconsistencies, which were addressed through LR adjustments from 1e-1 to 1e-6. Lower LR values, particularly at 1e-4, significantly improved convergence speed and overall performance, as depicted in visualizations generated. Additionally, exploring variations in kart types revealed "Sara_the_racer" as the optimal choice, aligning with Jurgen's kart selection and significantly boosting Model C's predictive accuracy and consistency. "Sara_the_wizard" also showed promise, albeit to a lesser extent. These findings underscore the importance of dataset selection, model refinement, and kart type exploration in optimizing model performance and predictive accuracy.

References

[1] Hussein, Ahmed, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. "Imitation learning: A survey of learning methods." ACM Computing Surveys (CSUR) 50, no. 2 (2017): 1-35.

[2] SuperTuxKart. [Online]. Available: https://supertuxkart.net/Main_Page.

[3] Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3431-3440. 2015.

[4] Yogesh, Singh & Bhatia, Pradeep & Sangwan, Om. "A REVIEW OF STUDIES ON MACHINE LEARNING TECHNIQUES." International Journal of Computer Science and Security. 1. 2007.

[5] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[6] Olaoye, Favour & Potter, Kaledio. "Deep Learning Algorithms and Applications." Dissolution Technologies. 2024.

[7] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[8] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).