Leiden Institute of Advanced Computer Science
Leiden University

Thesis submitted for the degree

Computer Science & Artificial Intelligence

Adaptive Reinforcement Learning for Human-AI collaboration

by

Elin Benja Dijkstra

Supervisor: **M. Preuss**
Co-Supervisor: **T. Verhoef and T. Kouwenhoven**

August 2022

Abstract

Adaptive Reinforcement Learning for Human-AI collaboration

Reinforcement learning has shown above human results in competitive settings. However, it can be argued that implementing reinforcement learning in collaborative settings is more valuable to the general public.

While collaborative self-play algorithms achieve high results when paired together, they fail when paired with humans due to mutual lack of understanding of the others intentions and tactics. In this thesis, I build upon a study done by Caroll et al. who argue that training a collaborative algorithm on human data, instead of a self-play algorithm, improves its ability to collaborate with humans. This study is done in the collaborative AI benchmarking environment overcooked-ai, which implements a simplified version of the restaurant game Overcooked.

By defining player style specific features, it is made possible to analyze and categorise player styles. The created features are used as the basis for a k-means clustering. These clusters are used to train behavior cloning models to represent different types of human behavior. Additionally, different skill levels are represented by model checkpoints at different stages during training. These human-like models are alternated between during the training of the reinforcement learning model, specifically a Proximal Policy Optimization (PPO). The hypothesis is that varying the training data in this way improves the performance of the collaborative reinforcement learning model.

Comparing a Proximal Policy optimization with self-play to a PPO using a behavior cloning model as 'training-mate', my results are consistent with the findings of Caroll et al. that the performance of the model improves when using human data. However, the suggested approach for alternating between player styles does not lead to a higher reward in the current setup. This is also the case when the pool of behavior cloning models is supplemented with different skill levels. While these results do not improve upon the baseline, this is potentially due to a small sample size. Additionally, further investigation could be done on the approach of alternating between the behavior cloning 'teammates'.

Acknowledgment

Special thanks to my supervisors Mike Preuss, Tom Kouwenhoven and Tessa Verhoef for supporting me through this thesis. Thank you for giving me the freedom to choose this topic and approaching it as I saw fit, along with giving helpful information and guidance. I would also like to thank Micah Caroll, who gave me the inspiration to go this direction with my research and was able to answer questions related to the environment and models along the way. Lastly, I would like to thank my family and Tim for the mental support, motivation and proofreading when I no longer noticed my own mistakes.

Thank you all, this thesis would not have been the same without you.

Abbreviations

NPC	Non-Player Character
RL	Reinforcement Learning
BHC	Behavior Cloning
PPO	Proximal Policy Optimization
AI	Artificial Intelligence
HPO	Hyper Parameter Optimization

Contents

Introduction

With the rise of artificial intelligence (AI) and reinforcement learning (RL) as a large research area, it has now also caught the attention of the general public. One question that is often asked is how useful these methods are in the day to day life. This question related to RL partially stems from the fact that most machine learning algorithms applied in games are designed with the aim of outperforming or beating a human, for example at games like chess. While this illustrates the potential use of AI models for replacing humans, many situations more useful to the general public will require the input of a human in some way or another. In these cases, it is of great importance that the model and the human can interact and collaborate together.

Collaborative settings are a relatively new and difficult problem to tackle. Many agents are currently trained with self-play[25] of population-based algorithms [12]. While these algorithms perform well when paired with other AI-models, they are often considered as a black box by humans. In practice two problems arise: when paired with a human, the model does not understand the human and is unable to read its intentions. Similarly, the human does not understand the model's behavior. This means that although the models have high performance in an environment when paired with other reinforcement learning agents, they fail when paired with a human and therefore are regarded less useful by the general public.

In this thesis, I will focus on techniques to increase the agent's performance when paired with humans. This is an extension of the work by Caroll et al. who have created an environment based on the collaborative computer game Overcooked [6]. In this environment, multiple players work together in a restaurant setting where the players have to collaborate in order to complete orders and thereby score points. Multiple layouts are provided, each with their own challenges. Caroll et al. show that RL models trained on human-like behavior outperform those that are trained through self-play.

As an extension of this work, the aim is to develop an RL agent able to identify the type of player it is collaborating with in the course of an interaction and adapt its own play to further improve the collaboration and therefore the resulting game score. This will be done by training multiple behavior cloning models on clusters of player-style and using the trajectories generated to train the RL agent. This thesis will first go into current research done in the field of human-AI collaboration, hereafter describing the data and methods used in this project. Finally, we will interpret the results and conclude if the training on a pool of behavior cloning models representing different player styles improves the performance of the reinforcement learning algorithm.

Chapter 1

Problem Statement

Currently, many multi-agent reinforcement learning methods make use of training methods that combine a set of AI agents, often previous versions of the agent itself [25]. While this has resulted in a strong performance on two-player zero-sum games like IBM´s Deep-Blue for chess [5] and AlphaStar for Starcraft [17], these are all games with a competitive nature. Only few real-world problems are characterized by pure conflict or competition. As touched upon by Dafoe et al., the hard problems of co-operations have not been tackled to the same extent [7]. They divide co-operation into three sections: AI-AI, AI-Human, AI for improving Human-Human co-operation. For all these categories, it is important for machine learning algorithms to have social understanding and cooperative intelligence in order to integrate fluently into society. The research done in this thesis focuses on Human-AI collaboration. Although real-world relationships almost always involve a mixture of common and conflicting interests, Dafoe et al. state that games of pure common interest can be a step in the right direction. In games where the AI and the human share the same goal, collaboration is essential to being successful in the task. Suggested research avenues include building AI models that can understand what their teammates are thinking and planning, communicate plans and even cooperate with varying types of teammates (ad-hoc teamwork).

This thesis focuses on the latter. In order to improve the performance of cooperative reinforcement learning, the aim is to train an agent to adapt to different play styles. The approach taken is to expose the reinforcement learning algorithm to varying player styles during training.

Chapter 2

Related Work

In this chapter, I will outline the current work done in the fields that this thesis will cover: multi-agent reinforcement learning, adaptive algorithms, and player style categorization. For the each study, I will identify its contribution to the research field and its relevance to this thesis.

2.1 Competitive Multi-agent Reinforcement Learning

Multi-agent reinforcement learning can be divided into three subcategories: fully competitive, fully cooperative, and mixed cooperative-competitive[4]. Competitive algorithms have a famous success history in being able to solve boards games like chess and Go. A paper by Google Deepmind describes a highly successful *self-play* approach used for the latter named AlphaGo Zero [26]. This algorithm receives only the current board state as input, along with the general rules of the game and learns to play the game by playing many games, without further instruction by a human. While earlier versions of the AlphaGo algorithms used professional players as an opponent, Alpha Zero plays against itself and earlier versions of itself. After playing many iterations of the game, the neural network learns to evaluate the current board state as well as predict the next moves. After three days of self-play training, the algorithm was able to defeat earlier versions of AlphaGo, as well as the worlds best human players.

As the field of reinforcement learning became more advanced, the area of interest began to shift from board games to video games. This type of environment is more closely related to real-life situations but drastically scales up the complexity. Some of the challenges this type of environment entail but are not limited to: processing the frames, setting a more extensive reward scheme, and requiring a shorter reaction time[24]. A study done by Jagerberg et al. found a *population based training* scheme to extend to reinforcement learning algorithms [12]. Here, multiple networks are optimized individually and will learn different policies. The knowledge learned by the separate networks will be shared with the population at a fixed interval. Jagerberg et al. found this technique to be successful in building reliable and robust agents quickly and found them to generalize across reinforcement learning frameworks as Atari and Starcraft II.

One of the most well-known state-of-the-art models in multi-agent reinforcement learning that has achieved above-human performance is Alpha-star [17] applied to the game

Starcraft. Its great success is attributed to the league-like training scheme where a population of self-play agents battle against each other and previous versions of themselves resulting in an agents capable of combining and adapting multiple tactics. Alhpa-star intuitively combines the techniques of the two papers described above.

While these algorithms utilising self-play yield great success on competitive multi-agent settings, they do not extend to collaborative settings with a human. Agents trained through self-play assume their partner to act optimal, and therefore fail to understand sub-optimal human behavior. On the other hand, agents that are allowed to train simultaneously with other agents often converge to opaque strategies and therefore fail to be understood by humans [10]. While these strategies do not directly generalize to collaborative settings, we will still take inspiration from the population-based scheme by training multiple models that learn directly from human behavior instead of self-play. The aim is to train a population that represents a set of players with different skill-level and player style.

2.2 Collaborative Multi-Agent Reinforcement Learning

In previous research conducted on collaboration between multiple agents, the focus has been on AI-AI communication. This has many applications such as in robotics and healthcare. This is illustrated in a paper by Spaan et al. where multiple robots have to coordinate their task in a dynamic environment[14]. They show that using coordination graphs, the robots are able to coordinate their assigned role and, using additional assumptions, to predict the actions of the other robots.

While these techniques are successful in allowing non-humans to coordinate, many real-life applications will require the input of a human. Many of the techniques that work on AI-AI communication fail when the human and AI explicitly have to cooperate due to many deep reinforcement learning algorithms developing an opaque strategy[29]. This results in them being a black box, making it difficult for humans to understand the behavior of these algorithms [10]. Similarly, the agent assumes the human acts according to the same strategy and when it does not, it fails. In order to bridge this gap to real-life applications, research has been experimenting with human-AI collaboration.

A study conducted by DeepMind on the game Capture the Flag focuses on this collaboration with an AI. In this game, two teams consisting of AI-AI or human-AI battle against each other to capture the flag at the other team's camp [13]. The researchers implement a new method they call For The Win (FTW) which makes use of a population of agents. Each of these agents also optimises an internal reward, in contrast to most policy optimization techniques which use only rewards from the environment. Additionally, a crucial part of the algorithms is that it has a cell for long-term and short-term memory, greatly improving their use of memory to adapt to the current state of the game. This study finds that the performance of the human-AI teams is higher than the human-human teams. While the AI-human teams trained with these expanded self-play methods yield great performance, Caroll et al argue that this might be due to the AI's individual capability, rather than it having the ability to coordinate with the humans[6].

The approach taken by Caroll et al. for an agent capable of collaboration is to expose the reinforcement learning model to human behavior at training time[6]. To test

the explicit collaboration between the AI and the human, they created an environment consisting of a simplified version of the collaborative game OverCooked. By running experiments in which humans participate together in the game, human trajectory data is collected. The human data is used to train behavior cloning models (BHC), which is a supervised learning method that learns the policy from expert demonstrations [20]. Behavior cloning models are in turn used to function as the teammate for the reinforcement learning model at training time. The behavior cloning model is embedded in the environment in order to consider it as a single-player environment. Subsequently, a Proximal Policy Optimization (PPO)[22] model and a planning model are trained. When being paired with a human-proxy model at testing time, the performance of the collaborative reinforcement learning algorithms being trained with these human-like models outperform methods trained via self-play or population-based techniques. While this technique showed improved performance, Caroll et al. speculate that this increase is partially due to increased interaction flexibility when using deep reinforcement learning. Interaction flexibility can be described as a measure of how deterministic a models is: if the agents finds it self in the same state multiple times, how often will it choose the same reaction?

2.3 Adaptive algorithms

The research on adaptive algorithms is largely inspired by the way humans can adapt to a situation. Human players often form a good team when they complement each other. For example, a player that is strong at keeping overview over a game might take on a leader role, where a player with good fine motor skills is quick in the field and will take the role of carrying out the instructions. In the case of OverCooked, the players would come to an understanding that the first player keeps an eye on which orders come in and does the final plating while the other moves around, collects and cooks the ingredients. In order for the artificial agent to reach its full potential in a collaborative task, it needs to be able to adapt to the player it is working with. Nelapka et al. explore the interaction flexibility in human-AI teams and study the effect of the artificial agent's design on this measure[19]. In order to quantify the functioning of a team, recurrence quantification analysis as proposed by Marwan et al. is conducted whereby recurrence plots are constructed for each player [18]. The patterns in these plots allow the researchers to gain insight in the stability of the system. Additionally, joint recurrence plots allow for the studying of the interactions between the players. An informative statistic that can be obtained from these plots is the percentage of determinism (%Det), which can be interpreted as the extent to which the interaction between team members follows stable, repeatable patterns. This is a measure to quantify the interaction flexibility mentioned earlier. During their experiments in the OverCooked environment, their findings are consistent with the conclusion by Caroll et al.: higher performance is achieved when the human is paired with a human-aware model, compared to a model trained through self-play. Additionally, the interactions within the team with the human-aware models showed significantly more flexibility than that of the self-player model, 65.98 %Det and 73.28 %Det respectively.

In an attempt to increase the performance of the collaborative model, DeepMind configured a model that is simple and yet effective in representing players of different player styles and levels they introduce as Fictitious Co-Play (FCO)[21]. This technique

combines both the strengths of self-play and population based techniques. The first step is to create a pool of self-play agents. In order to represent different play-styles, these agents are initialised in different ways. In order to represent different skill levels, checkpoints at different stages in the training phase are taken and added to the agent pool, where a model in the early stages of training would represent a low-skill levels players, and a fully trained model would represent a highly-skilled player. During testing, this method significantly outperforms the PPO with BHC model used by Caroll. et al and self-play algorithms. It has the additional advantage of not needing human trajectory data to train the models. While this is a great advantage for easy of training, I believe there is information in human-play that goes beyond substituting this with initialisations. Therefore, while taking inspiration from the pooling of agents, I will include the human data for the training of these agents.

2.4 Player style categorization

The presence of non-player characters (NPC) in video games has been around since the late 1950's. At this time, the research focused on adjusting the NPC in a way that improves the experience for the player. To keep players engaged while playing a game, it is of importance to have an opponent or teammate that matches the skill level of the player: when the opponent is too skilled a player becomes demotivated and gives up, whereas if the opponent is not good at all, the player does not feel challenged, both resulting in loss of motivation and engagement[30]. Therefore, many computer games, think of Mario Kart or Call of Duty, make use of an adaptive algorithm. To cater to a player's specific needs, it is useful to gather information about its play-style and skill level. The earlier 'game-AI' algorithms made less use of modern techniques like machine learning and reinforcement learning, but were more based on rules of heuristics.

To learn more about how a human adapts, Lerer et al. conducted a study on the way humans adapt to a sequential learning task[15]. The participants were asked to take part in a serial reaction time experiment, where player had to click on a block that lights up on a computer screen. The players had to learn the required action through rewards and penalties; a reinforcement learning task. Players could be categorized into low and high-performing players, where the reaction time for the high performing players got lower over time while for the lower performing players it did not improve. In the learning curve of the high-performing players, they develop a mouse trajectory indicating they get better at predicting the next block to light up. While distinguishing players into high and low-performing might not add enough value to our algorithm alone, a combination of skill level categorization and player style does have potential.

Bartle et. al. developed a theory on taxonomy of players-types based of the text-based adventure game Multi-User Dungeon[3]. From the compilation of the game and observations made on the forum discussion, Bartle et al. theorized that players can be divided into four main player-characteristics: killers, achievers, explorers and socializers. Killers are the player who likes to provoke other players and cause drama. Trolls, hackers and cheaters and examples of players that would fit in this category. Achievers are competitive players and play the game with the intention of beating the challenges set by the game. Explorers like to find out all the mechanics, short-cuts and tricks the game offers

and thrive off discovering more of the world as they progress. Lastly, socializers and the players who care less about the game itself and more about the relationship towards the other players and the community surrounding the game. While this taxonomy can aid game designers in creating a game that is interesting for many different type of players, Bartle himself also mentions that these player-types are found on the Multi-User Dungeon and cannot necessarily be extrapolated to ever other game.

Ben Cowley et al. take inspiration from Bartle's taxonomy, mentioning the four player-styles described in the literature they define as Conqueror, Manager, Participant, and Wanderer. Within each of these types, they also make a distinction between hardcore and casual. To categorize players by player-style, Ben Cowley et al make use of the rule-based approach decision trees [15]. Due to a lack of data, they choose to focus on classifying the player into conqueror/non-conqueror. After setting a large number of features, they narrow down the selection to high-level behavior traits: Aggression, speed, caution, planning, decisiveness, thoroughness, control skill, and resource hoarding. This results in a feature vector for each of the players containing the normalized features. They then apply various machine learning techniques such as Feed Forward Neural Network, K-means clustering and decision trees on the feature vector. They found decision trees to have the highest accuracy. A drawback of this approach is the need to predetermine the different possible players. Additionally, when the algorithms encounters a player type not seen before, it might result in unusual behavior.

A more flexible solution is proposed by Aiolli et al. who use the non-collaborative game Ghosts to illustrate how categorizing a player's style can lead to more stimulant and challenging gameplay [2]. In this study, they aim to construct a profile of the player during an interaction in the form of a feature vector. This feature vector is constructed using simple machine learning techniques on eight features.

Taking inspiration from the methods described in this section, I will aim to distinguish different players types and skill levels. However, instead of predetermining the number of player types, I will perform clustering on aggregated data describing the player to distinguish players from one and another. By eliminating the need for predetermining the categories, the experiment pipeline can be translated to other games and environments.

Chapter 3

Data and Environment

In this chapter, the environment and dataset used for the experiments are described. Additionally, a description of the features present in the data is given. Finally, exploratory data analysis is done in order to explore the underlying patterns in the data.

3.1 Environment

The environment used in this thesis is a simplified version of the popular restaurant game Overcooked adjusted for reinforcement learning experiments. This environment is created by the team at Berkeley in order to set a benchmark for the level of cooperation between two agents.

The game can be played on multiple layouts. On all layouts, the players have the common goal of serving onion soup. The players both take on the role of chef and must retrieve the onions and put them in the pot to cook. Once cooked, a bowl must be held to collect the soup, and subsequently delivered to the serving station. In Figure 3.1 the original layouts used in the 2019 experiment are displayed. The layouts all have their own challenges, e.g. the most left layout in Figure 3.1 gives all players access to all stations, but it is so small that the players might collide as a result of inefficient routing. On the other hand, the fourth layout completely separates the players, forcing them explicitly to divide tasks since they both have access to a different set of stations. A more descriptive overview of the characteristics and challenges for each layout is given in Table 3.1.

The environment offers the functionality to combine agents of different algorithms. During the course of a game, the environment keeps track of all features in the layout including but non limited to: time, location of objects and players, collected points.

3.2 Data description

This section describes the features the environment records at each time step. As part of the experiment, additional features will be constructed to supply the clustering model with more information about the player. This will be described in section 4.1.1.

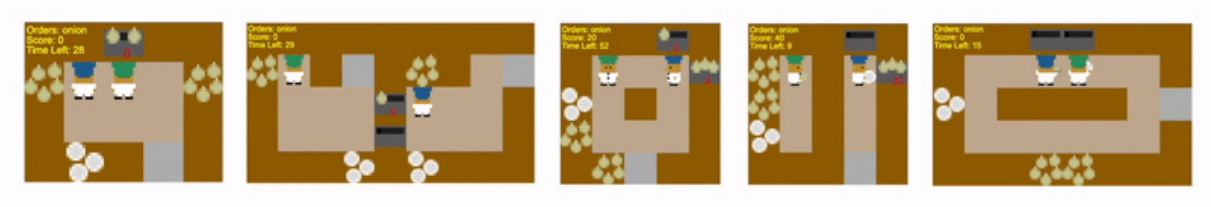


Figure 3.1: The layouts used during the experiments conducted in 2019. For left to right: Cramped room, Asymmetric advantages, Coordination ring, Forced Coordination, Counter circuit

Layout	Characteristics	Challenges
Cramped Room	All players have access to each station	Collisions are probable due to limited walk space
Asymmetric advantages	Players have their own space, access to all stations	
Coordination Ring	The path is only one block wide	Players are forced to coordinate their walking direction
Forced Coordination	The players have access to different stations	The players are forced to pass on the ingredients
Counter Circuit	Path is only one block wide, comparable to Coordination Ring	Players have to coordinate their walking direction but have more time to do so

Table 3.1: Description of the layouts for the 2019 experiment.

3.2.1 Original Dataset

This thesis uses a dataset collected and made public by Caroll et al.[6], available at `https://github.com/HumanCompatibleAI/overcooked_ai`.

The dataset contains the records of two periods: 2019 and 2020. During the experiment in 2019, human-to-human trajectory data was collected on five layouts, all played for 180 time steps. In the experiments conducted in 2020, multiple layouts were added to the experiment that required more high-level actions and planning ahead. These layouts were only played for 60 time steps. In the original paper by Caroll et all., only the 2019 data was used. Similarly, for this thesis, the main focus will be on the data collected in 2019 to supply the model with more time steps to determine the play style of their partner.

In the processed dataset used, we find features supplied in the raw output of the environment and some basic additional features constructed by Caroll et al. during preprocessing. An concise overview of these features can be found in Table 3.2 and Table 3.3.

The first features are used to identify the trial. *Trial ID* is the specific game between the two players *Player 0 ID* and *Player 1 ID* on layout *Layout Name*.

Secondly, the variables relating to time are defined. *Current Gameloop* states the current time step in the integer range [0, 180] whereas *Time Elapsed* tell us the number of seconds passed in real time.

The current state of the game is encoded in *State*. This vector includes information about the location of the players, their view direction, a boolean value indicating whether they are holding an object and the orders in line. Additionally, an encoded state of the grid layout is recorded to show the current location of objects, stations.

To keep track of the actions taken during the game, the *Joint Action* is logged. This includes walking moves across the layout and interacting with the stations. Additionally, the number of button presses within a time step is logged.

When actions are taken, this can result in receiving a *Reward* from the environment. Where taking a step might result in a relatively low reward, interacting with a station or delivering an order will yield a higher reward. In addition to the reward from the environment, the players also receive a *Score* for each order they fulfill.

The numerical features described above are recorded per timestamp and are later supplemented by the total achieved, and average per time step when the game is finished.

A full overview of the data can be found in A.1

Caroll et al. composed several basic features from the underlying data during preprocessing to supply the user with easy access.

Variable	Description	format
state	A JSON serialized version of a OvercookedState instance. Support for converting JSON into an OvercookedState python object is found in the Overcooked-ai repp	JSON
joint_action	A JSON serialized version of a joint overcooked action. Player 0 action is at index 0, similarly for player 1.	JSON
reward	The sparse reward achieved in this particular transition	int
time_left	The wall-clock time remaining in the trial	float
score	Cumulative sparse reward achieved by both players at this point in the game	float
time_elapsed	Wall clock time since begining of the trial	int
cur_gameloop	Number of discrete MDP time steps since beginning of trialcur_gameloop	string
layout	The 'terrain', or all static parts (pots, ingredients, counters, etc) of the layout, serialized in a string encoding	string
layout_name	Human readable name given to the specific layout	string
trial_id	Unique identifier given to the trial (again, note this is a single Overcooked game; a single player pair could experience many trials).	string
player_0_id	Anonymized ID given to this particular Psiturk worker. Note, these were independently generated by us on the backend so there is no relation to Turk ID. If player is AI, the the the player ID is a hardcoded AI_ID constant	string
player_1_id	Symmetric to player_0_id	string
player_0_is_human	Indicates whether player_0 is controlled by human or AI Data	bool
player_1_is_human	Symmetric to player_0_is_human	bool

Table 3.2: The overcooked environment returns a set amount of features per timestep, shown in this table. These features relate to the state of the environment, trial and players.

Variable	Description	format
cur_gameloop_total	Total number of MDP time steps in this trial. Note that this is a constant across all rows with equivalent trial_id	int
score_total	Final score of the trial	int
button_press	Whether a keyboard stroke was performed by a human at this time step. Each non-wait action counts as one button press	int
button_press_total	Total number of (human) button presses performed in entire trial	int
button_presses_per_time step	button_press_total / cur_gameloop_total	float
time steps_since_interact	Number of MDP time steps since the last human-input 'INTERACT' action	int

Table 3.3: Caroll et al. computed several additional columns from the underlying raw data and these are added for convenience. These were added during their pre-processing and will be included in the original dataset for this thesis.

3.3 Exploratory Data Analysis

Now that the features in the raw dataset have been defined, we start our exploratory data analysis. The purpose of this is to get an overview of the amount and quality of the data, along with some insight in the underlying distributions, outliers and errors. While performing this analysis, we also hope to gain insight on the differences between layouts and players.

3.3.1 General findings

First, an overview of the general statistics will be given. This contains information about the amount of data available.

In Table 3.4 the number of trials and unique players per layout is shown. When looking at the number of trials, we see that each layout is played on average 18 times by a different player pair. While most players play all layouts, some only play on a subselection. The order in which players play different layouts is always the same. All trials are played until the total time of 180 seconds has passed. Every 0.150 seconds, a record of the environment is saved, resulting in 1200 rows per trial. The dataset does not contain any missing values.

Layout	#trials	#unique players
Cramped Room	19	38
Assymetric Advantages	19	38
Coordination Ring	18	36
Forced Coordination	18	36
Counter Circuit	12	24

Table 3.4: General statistics of the gathered trajectories: The number of trials completed per layout and the number of players that have played each layout. Note that each player only plays a layout once.

Action	Number of occurances
[0, 0]	123290
[-1, 0]	18685
[1, 0]	16667
[0, -1]	16505
[0, 1]	13745
INTERACT	17390

Table 3.5: Number of times an action is taken. The format of the directions is [step in x direction, step in y direction]. The INTERACT action occurs every time a player interacts with one of the stations being the onion, pot, plate and delivery stations

In Table 3.5 an overview of the possible actions is given: a step to the left/right, a step up/down, an interaction with one of the stations or object, or nothing. This table shows us that during approximately 40 percent of the time steps, an action is taken.

In Figure 3.2 we see a scatter plot of the button presses versus the number of button presses. This leads to the suspicion that more button presses lead to higher rewards.

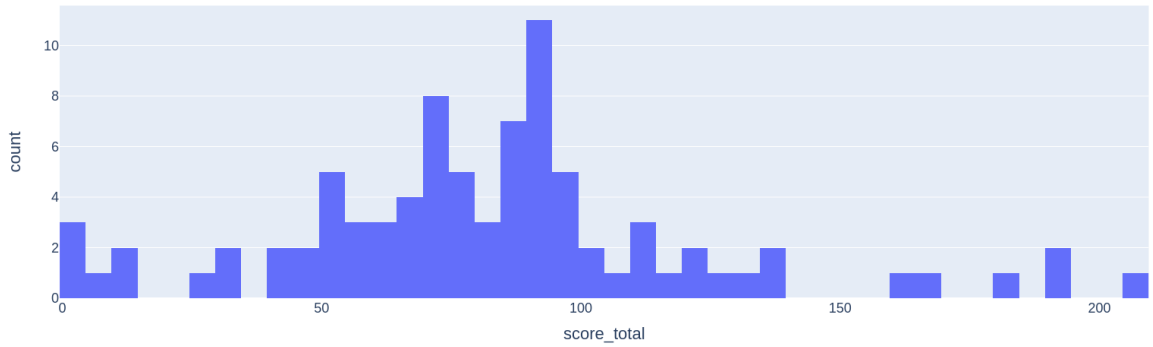


Figure 3.3: Histogram of the scores gained in all layouts. The peak lies between 50 and 100, but goes as high as 200.

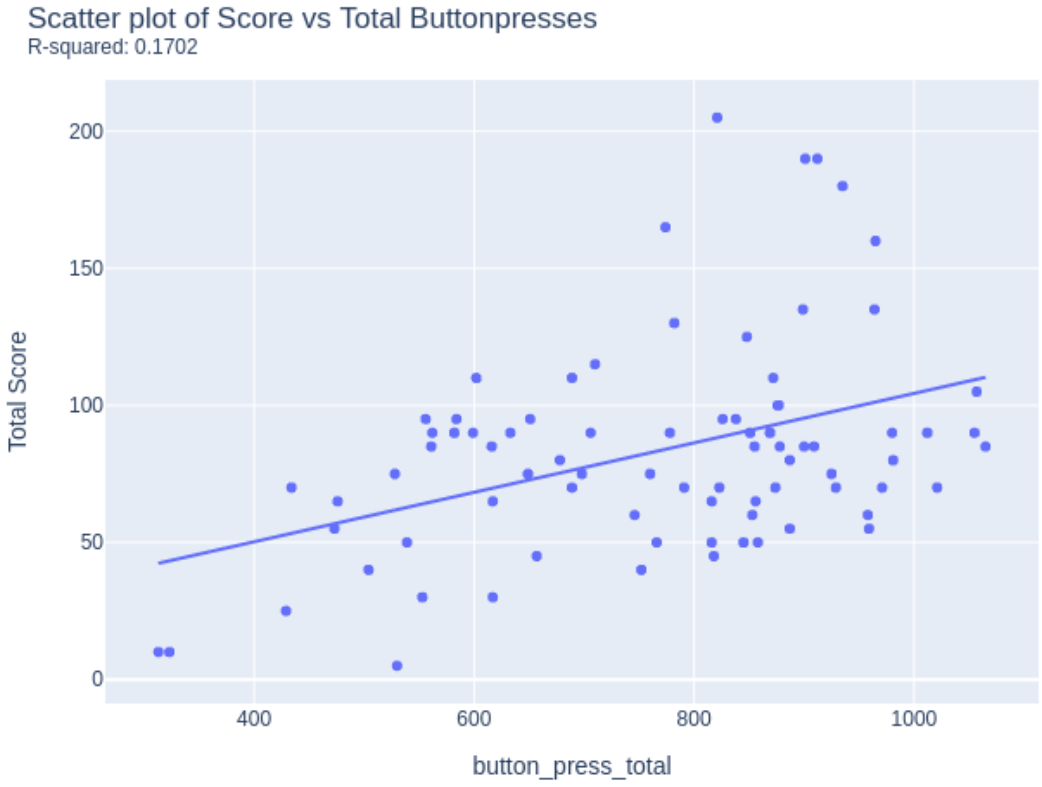


Figure 3.2: Scatterplot of the relation between button presses and gained scores. The regression line indicates a positive relation.

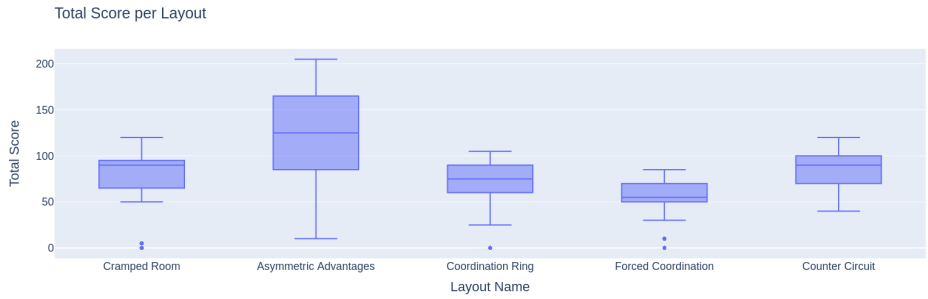
In Fig 3.3 a histogram of the total scores is shown. Scores are gained by handing in soup. A histogram plots the frequency at which a certain value range is found in the data. Here we see that the biggest peak is seen within the range of 50 - 100 points. However, this does not provide all information since it might be easier on particular layouts to gain

a high score than others. For example, a layout with two cooking plots, e.g. asymmetric advantages, makes it easier to produce more soup, and thus has a higher potential score.

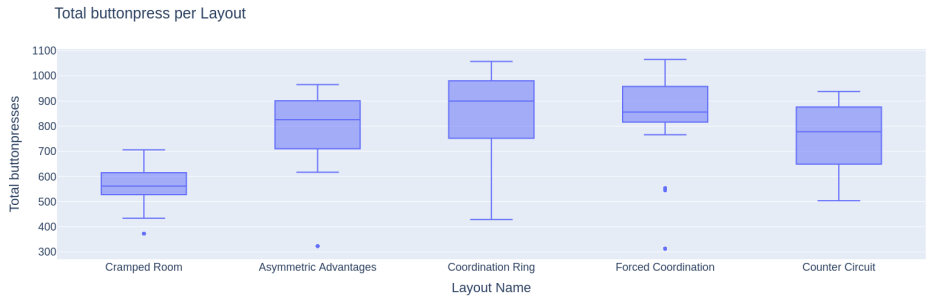
3.3.2 Exploring Layouts

To gain some understanding of the effect of the different layouts on the distributions of features, we visualize the distributions of multiple features per layout. To show multiple metrics simultaneously, we make use of boxplots. A boxplot shows us the range of the values in a variable, the median, the quantiles and the outliers[28].

In Fig 3.4 the number of button presses during a trial and the total score is displayed per layout. We see that the distributions of these variables vary greatly between layouts. In order to create a model that can be applied to all layouts, I argue normalization is needed to make the player styles comparable across different layouts.



(a)



(b)

Figure 3.4: Boxplots visualising the number of button presses and score distribution for all layouts before normalization. The difference in distribution across layout indicates the need for normalization.

From Fig 3.4a we gain the information that while the right most layouts are similar in the score distribution, Asymmetric Advantage seem to have a higher average score and more variation in the scores. It is possible that asymmetric advantage allows for more to apply different player styles.

In Fig 3.5 we see the average score per layout. This plot confirms the Asymmetric Advantage yields a higher average score at the end of a trial. Note that asymmetric advantages is a layout where both players have access to all stations and are therefore

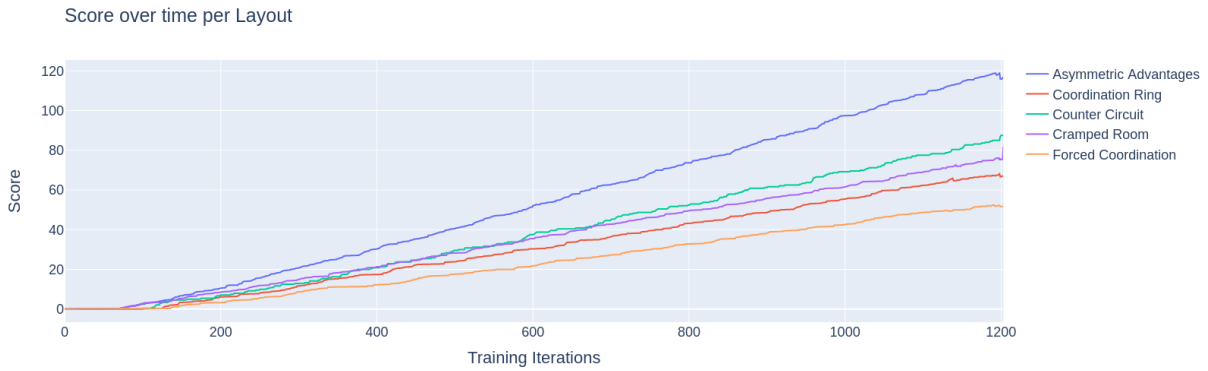


Figure 3.5: Score over time per Layout. This emphasizes the difference in layouts, showing that asymmetric advantages yields the highest rewards, most likely due to the fact that it has two pots and two hand in stations, making it possible to cook to soup simultaneously.

not explicitly forced to divide tasks. We also see that it takes around 100 time steps (15 seconds) for the first points to be gained. This has to do with the cooking time of the soup.

3.3.3 Exploring player-types

Now that the effect of layouts on features are apparent, and the need for normalization is discussed, the potential differentiation between players is visualized.

In Fig 3.6 we see the distributions of the total scores for each player combination, aggregated over the five layouts. This also shows us that each player only plays with the same opponent.

Here we see that while many duos gain an average score between 50 to 100 points on average, as also seen in the score histogram, there are a few outliers. Duos (14, 15) and (20, 21) have a lower average score than the other duos, which indicates a different, lower, skill level. Duo (6, 7) have a broad range with scores going as high as 200 points.

Also in the button presses we see a great variation between players, as shown in Fig 3.7. This feature is aggregated over all layouts. This shows that some players inherently perform more button presses than others, which points towards different player styles. In the experiment in this thesis, more features in order to categorize the type of action to further specify a player style.

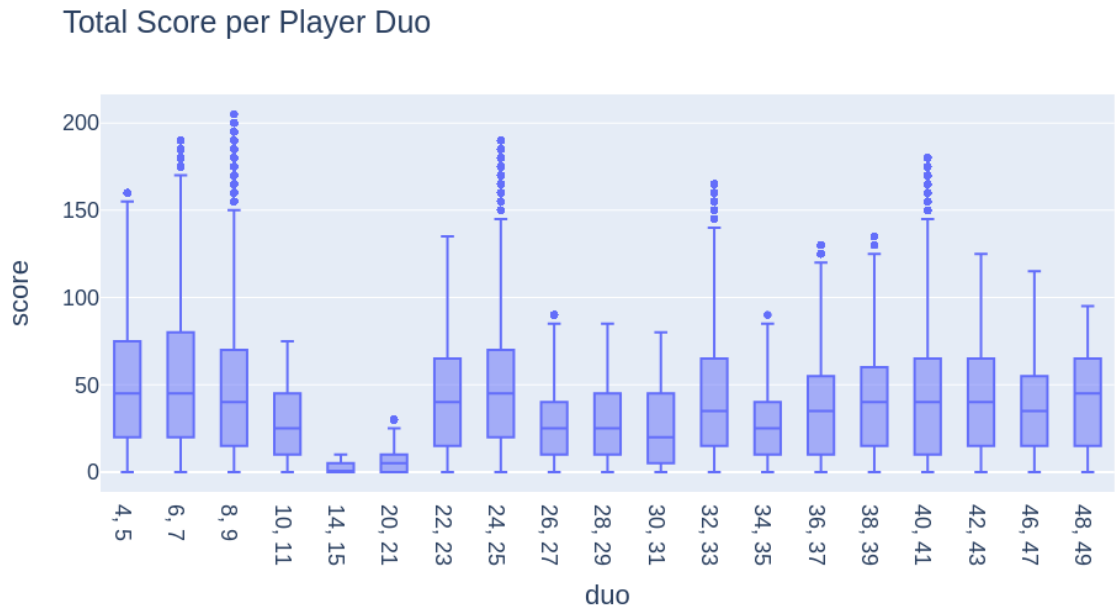


Figure 3.6: Distribution of the total score per Player duo. While many duos have comparable scores, (14, 15) amd (21, 21) perform worse. This is interpreted to show a difference in skill level.

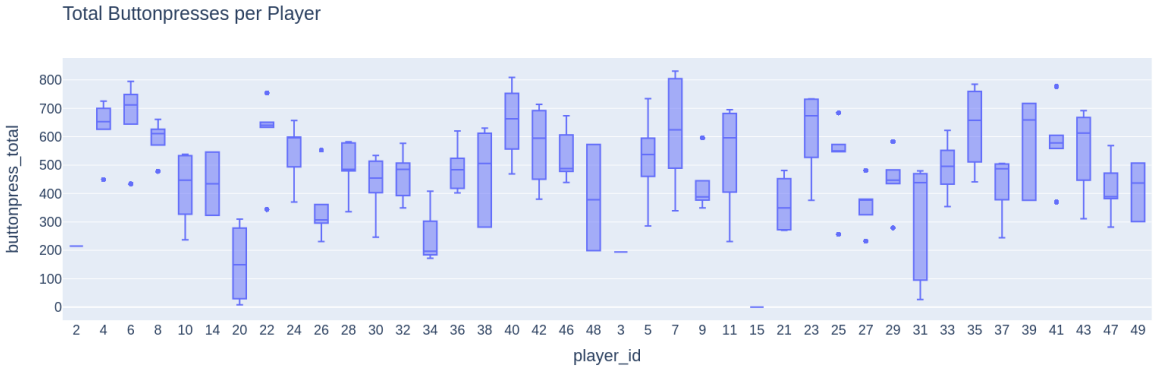


Figure 3.7: Button presses per player, aggregated over all layouts. Similarly to Figure 3.6, it shows the varying in players and the potential in distinguishing them.

Chapter 4

Experiments

In this chapter, the techniques used in this thesis are described. The general approach taken is to use the available data to separate the players into different play style clusters. On each cluster, a behavior cloning is trained in order to learn a specific play style which will consequently be fed to the final reinforcement learning algorithm.

4.1 Player Style Clustering

To prepare for the clustering, we construct additional features describing player styles and identify features that hold most explanatory power. Secondly, the clustering algorithm is explained. In order to let our algorithm work as optimally as possible, a feature and parameter selection will be performed, which will conclude this section.

4.1.1 Feature Construction

The features described in Section 3.2.1 consist of generic high-level features extracted directly from the environment. In order to supply our model with more information about the individual players, we construct additional features. These features are described below.

To indicate the *activity level* of a player, we calculate the ratio of the number of button presses to the total number of time steps. Per time step, only one button press can occur.

$$Activity = \frac{\sum buttonpresses}{\sum timesteps} \tag{4.1}$$

To specify this measure further, we look specifically at interactions with stations. An *interactivity* feature is constructed to indicate the frequency at which the player interacts with the stations for picking up ingredients, cooking them, and delivering them. This feature gives us an understanding of how often the player actually works, instead of e.g. roaming around.

$$Interactivity = \frac{\sum Interactions}{\sum timesteps} \tag{4.2}$$

Combining these features, we construct a *stress* feature. Stress could be indicated by how much work the player tries to get done within a limited time frame. This usually

does not result in efficient work, but pressing buttons in the hope to do something right, so called 'button-smashing'. The number of button presses within a window of a variable amount of time steps gives an quantified indication of how much stress the player is feeling.

$$Stress = \frac{Activity}{Interactivity} = \frac{\sum buttonpresses}{\sum interactions} \tag{4.3}$$

While some players resort to button-smashing, other plays might simply start working faster. To measure this characteristic we construct the *efficiency* feature, defined as

$$Efficiency = \frac{Score}{Activity} \tag{4.4}$$

The features described above are based on the button presses and interactions. These statistics on the activity of the player are supplemented with features related to game state, game play and approach.

Firstly, features based on the players holding object are constructed. For each time step, it is determined if the player is holding an object, and which type of object it is: ingredient, empty dish or soup. This can later be aggregated into a features describing how much time a player spends holding these objects. Additionally, it can now be extracted how many times a player *grabbed* and *put down* these objects.

To aid in constructing more informative features indicating if the human understand the environment, some object states are recorded. This includes the current state of the soup: empty, cooking of ready. Furthermore, the time elapsed and the current total score are included.

The simple features describing player action and station states can now be aggregated in more meaningful features.

Time next to a human counts the consecutive time steps a player is next to the other player. This indicates if the player tries to "push" a player away when it is in its way as well as giving us information whether the model is smart enough to go around a player when it is blocking its path.

Grabbed plate while cooking is a feature that describes if the player gets the plate only when the soup is cooking/ready or also at random moments.

Hand in points is the points allocated specifically to the human handing in the order. Players might divide tasks and one player will be assigned the role to hand in the completed orders.

Put down ingredient in soup measures the amount of times a player puts the ingredients it is holding in the pot versus a random spot in the environment.

Holding ingredients measures the time steps a player keeps holding the ingredient. This gives an indication on whether or not the player knows what to do with it.

Empty pot measures how much of the time the pot is empty. Do the players prioritise keeping ingredients in the pot.

Now that all the features are constructed, they can be analyzed and used for our clustering method.

4.1.2 K-means Clustering

A clustering algorithm is applied to the features to find clusters of players with a similar player-style. The outcome is used in order to get an understanding of the differences between groups of players, and later to train behavior cloning models representing these groups.

For the clustering of player styles, the widely used clustering technique K-means is applied[11]. This is a vector-based clustering technique relying only on one parameter, being the total number of clusters k. The algorithm start by selection k points to represent a cluster center. The algorithm then continues to repeat a two step update until the algorithm has converged.

The first step consists of calculation the euclidean distance for each point to the cluster centers and assigning it to the cluster closest.

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \leq j \leq k \right\} \tag{4.5}$$

When all points are assigned to a cluster, the mean centroids of each cluster are recalculated.

$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j \tag{4.6}$$

When the assignment of the points to the clusters no longer changes, the algorithm is considered converged.

4.1.3 Feature and Parameter Selection

When working with high dimensional data, we must always be aware of the curse of dimensionality. Distance based clustering methods are specifically sensitive due to the euclidean shrinkage phenomenon under high dimensions[9]. This relates to the fact that in high dimensional data, the ratio between the nearest and farthest points approaches 1, meaning all points become uniformly distant from one and another[1]. For this reason, we must perform feature selection and dimensionality reduction. This process consists of multiple steps.

For *feature selection*, we want to drop features with low explanatory power. A first indication can be given by analyzing a correlation plot, which visualizes correlation coefficients between all features. Correlations close to 1 indicate that two features move in coordination with each other. Likewise, correlations close to -1 indicate two features move directly opposite to one another. This entails that including both features in the dataset will not provide the model with more information. Therefore, highly positive or negative features are dropped or aggregated into one feature.

Principle component analysis (PCA) is performed in order to further reduce the dimensionality of our data. PCA transforms the data to a new coordination system where the new uncorrelated components explain the direction of greatest variance in descending order. In other words, the first component shows the greatest variance, the second component the second greatest variance and so forth.

When it comes to choosing which features will be included in the dataset, and the settings for PCA and K-means, an *exhaustive search* is performed. For a range of feature size $n_{feat} \in 3, ..., 10$, all possible feature combinations are used as input to the clustering model. The quality of the clusters will be measured with the Davies Bouldin (DB) index[8]. This index is a ratio of the distances within clusters and separation between the clusters, for which a lower value is considered better. All resulting model outcomes are sorted by this value. For each setting of n_{feat}, the clusters with the best DB-score will be analyzed in further detail. The final setting will be chosen by visualizing the results using the high dimensional visualization technique, as well as analyzing their main components. Below, an overview of the clustering process is given in pseudocode.

Algorithm 1 Exhaustive Search

```
Construct additional features
Inital feature selection by analyzing correlation coefficients
for n_pca_components = 2, 3 do
    for k = 3, 4, ..., 7 do
        for n_features = 4, 5, ..., 10 do
            Compute PCA components
            Run K-means clustering
            Calculate Davies Boulding index on the K-means results
        end for
    end for
end for
Sort all K-means results by Davies Bouldin index
Visualize the best 2 K-means results per value for n_features
```

4.2 Behavior Cloning for Training

In order to supply the reinforcement learning model with different player-styles, our approach is to train behavior cloning models representing different player-styles and skill levels. To gather enough human trajectories on the game is expensive. Instead, we train behaviour cloning models on a smaller amount of trajectories the data, which can then represent human behavior and provide a great amount of data for the RL model. To introduce variety in the behavior cloning models, a model is trained on each player style found during clustering. Variety in skill level is introduced by saving the behavior cloning models at different stages during training.

Behaviour cloning is the simplest form of imitation learning. This is inspired by the way humans learn as well, a elder person performs a task, and the child can reproduce the action without explicit instructions[20]. In computer science, this is framed as a supervised learning problem where the network tries to learn the expert's policy from demonstrations[20]. The input provided to the network is gathered from demonstrative trajectories by the expert and will be processed as the set of independent and identically distributed (i.i.d.) state-action pairs $(\mathbf{s_0}, \mathbf{a_0}), (\mathbf{s_1}, \mathbf{a_1}), .., (\mathbf{s_n}, \mathbf{a_n})$. The state $\mathbf{s}$ contains a grid overview of the current game, where the action $\mathbf{a}$ contains the action taken by the

expert in that state. The goal of the algorithms is to learn the policy π_θ so that the loss function $(\mathbf{a}^*, \pi_\theta(\mathbf{s}))$ is minimized. To stay consistent with the study done by Caroll et al. a multi-layer perceptron is used. Its architecture and parameters will be determined during hyper-parameter optimization.

It should be noted that since this algorithm assumes the state-actions pairs to be i.i.d., it can fail quickly in states that have not been encountered before. However, since the overcooked game does not need a lot of long term planning, and small mistakes are not necessarily catastrophic, the efficiency and simplicity of this algorithm outweighs the risk of failure.

In the setting of this thesis, we train multiple behavior cloning models. For each cluster found in Section 4.1 we train an behavior cloning model. Additionally, in order to improve the generalization further, we take inspiration from Deepmind's fictitious co-play [21] by bootstrapping from the players within these clusters and using the hold-one-out method. Here, one player is excluded from the subset. Using these techniques we create multiple behavior cloning models per cluster, varying slightly in player-style and skill-level.

4.3 Proximal Policy Optimization

Now that we have trained the models that will be used for generating training data, we start developing our collaborative model. For this task we will use Proximal Policy Optimization. This algorithm is developed by OpenAI and is one of the current state-of-the-art algorithms within reinforcement learning[22]. It provides an improvement on other policy gradient methods.

Traditional policy gradient methods have a loss function defined as

$$\mathbf{L^{PG}}(\mathbf{\Theta}) = \hat{\mathbf{E}}[\mathbf{log}\pi_{\mathbf{\Theta}}(\mathbf{a_t}|\mathbf{s_t})\hat{\mathbf{A}}_\mathbf{t}] \tag{4.7}$$

describing the policy loss as the log probabilities of the output of the value network multiplied by the estimated advantage of the action. A problem that arises with this method is that the value estimate made by the network is noisy due to variance. Additionally, the multiplication with the advantage function can lead to very big and detrimental policy updates.

Proposed solutions as Trust Region Policy Optimization (TRPO) work by constricting the size of the policy update. This is done by restricting the difference between the old policy and the new policy. However, this method has shown to be hard to implement and does not extend to all applications.

Proximal Policy Optimisation (PPO) is a recent advancement in the field of Reinforcement Learning, which utilises some of the advantages of TRPO, but is more simpler to implement. PPO uses a clipped loss function instead:

$$\mathbf{L^{CLIP}}(\mathbf{\Theta}) = \hat{\mathbf{E}}[\mathbf{min}(\mathbf{r_t}(\theta)\hat{\mathbf{A}}_\mathbf{t}, \mathbf{clip}(\mathbf{r_t}(\theta), \mathbf{1}-\epsilon, \mathbf{1}+\epsilon)\hat{\mathbf{A}}_\mathbf{t}] \tag{4.8}$$

This clipped loss function imposes a clip interval on the probability ratio term, which is clipped into a range $[1^-\epsilon, 1+\epsilon]$, where ϵ is a hyper-parameter. This function takes the minimum between the original ratio and the clipped ratio.

Another advantage of PPO is that it enables multiple epochs of minibatches using a surrogate objective, whereas standard policy gradient methods perform one gradient update per data sample. As seen in Figure 4.3, the method alternates between the actor, sampling from the environment and thereby collecting data and calculating the advantage estimates, and the critic, running Stochastic Gradient Descent on the clipped loss function.

Algorithm 2 PPO algorithm (Schulman et al. 2017[22])

for $iteration = 1, 2, \ldots$ **do**
  for $actor = 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T time steps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt. θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

The PPO agent in our experiment will be provided the environment layout with the BHC model embedded. During training it will encounter a population of BHC models, hopefully improving the generalization towards different player-styles and therefore yielding a greater amount of points on average.

4.4 Hyper-parameter Optimization

A well-known drawback of many machine learning models, especially reinforcement learning algorithms, is their sensitivity towards hyperparameter settings. For this reason, it is important to explore a range of settings to determine which parameters influence the models performance the most and what a beneficial setting is. When performing an exhaustive search, where all models are trained to completion on a large range of possible parameter settings, this process quickly becomes extremely time consuming. A novel approach suggested by Li et al. is a method called HyperBand [16]. This algorithm focuses on greatly speeding up the hyper parameter optimization by relying heavily on efficient resource allocation and early stopping. Hyperband is an extension of the Successive Halving algorithms. The intuition goes as follows: for a set amount of epochs, train many different settings. Evaluate their performance at a set amount of epochs and discard the worst half. This is repeated until only one setting is left. It is built on the assumption that the first section of a learning curve indicates whether a parameter setting is going to outperform the most optimal setting yet. When the algorithm runs a new setting and after a small amount of epochs it does not outperform the current best setting, the parameter combination is dropped and the exploration of other settings continues. While this seems harsh, it often results in close to optimal results while speeding up the search considerably compared to Bayesian techniques[23][27].

Chapter 5

Results

In this chapter, the results of the experiments described in Chapter 4 are presented.

5.1 Clustering

5.1.1 Correlations and feature construction

In Figure 5.1, the correlation coefficients between all features in our dataset are visualized. Darker colors mean higher positive or negative correlation. To keep the most information in the dataset with the smallest amount of features, features are dropped and aggregated into one.

$$Soup\ Delivery = \frac{\sum Hand\ in\ Soup}{\sum Put\ Down\ Soup} \tag{5.1}$$

$$Put\ Down\ efficiency = \frac{\sum Hand\ in\ Soup + \sum Put\ down\ Ingredient\ in\ Soup}{\sum Put\ Down\ Object} \tag{5.2}$$

$$Multitaskingwhilecooking = 0.3 * \sum Actionswhilecooking + 0.7 * \sum Grabbedplatewhilecooking \tag{5.3}$$

$$Insecure = \sum Next\ to\ opponent + \sum Holding\ Object \tag{5.4}$$

$$Efficient\ Plate\ Grabbing = \frac{\sum Grabbed\ Plate\ while\ cooking}{\sum Grabbed\ Plate\ total} \tag{5.5}$$

The single features used in these aggregated metrics are dropped, along with features describing the state and performance of the team rather than the individual. This includes:

- cur_gameloop_total_resc
- button_press_total_resc
- score_total_resc

Figure 5.1: Correlation matrix of all features before features selection. Dark green and red squares mean high positive of negative reward. Keeping all these features in the dataset will not result in more information.

- sum_opp_holding_bool_resc
- sum_grabbed_object_resc
- avg_timesteps_since_interact_resc
- sum_empty_pot_resc
- sum_grabbed_ingredient_resc
- sum_grabbed_soup_resc
- sum_put_down_ingredient_resc
- sum_pot_ready_resc
- sum_pot_cooking_resc

When dropping and aggregating the highly correlated features and aggregated some features into ratios of averages, it results in the correlation plot in Figure 5.2.

Figure 5.2: Correlation matrix after feature selection. The previously dark red and green values features are dropped of aggregated. The dataset now contains a similar amount of information while having a smaller dimension.

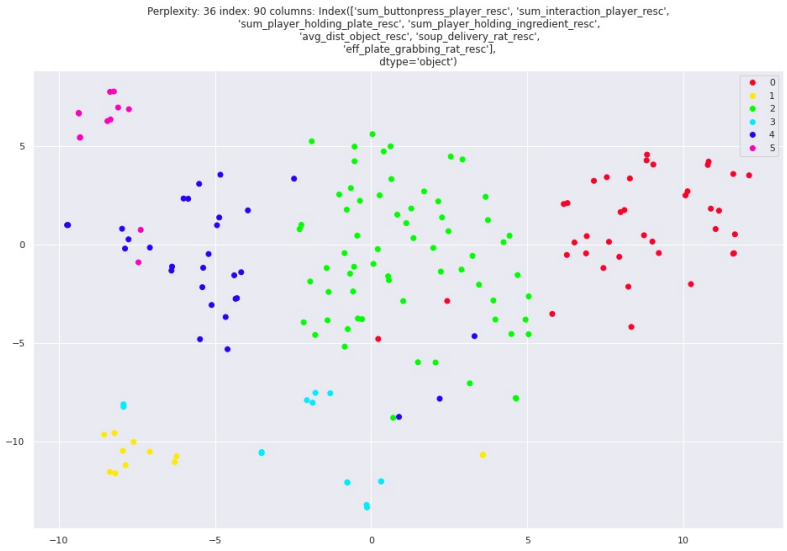
5.1.2 Final clustering model

The exhaustive search resulted in 150 models with a Davies Bouldin value ranging from 0.084 to 0.189. Using visualization of the clusters, the setting chosen for our final model is:

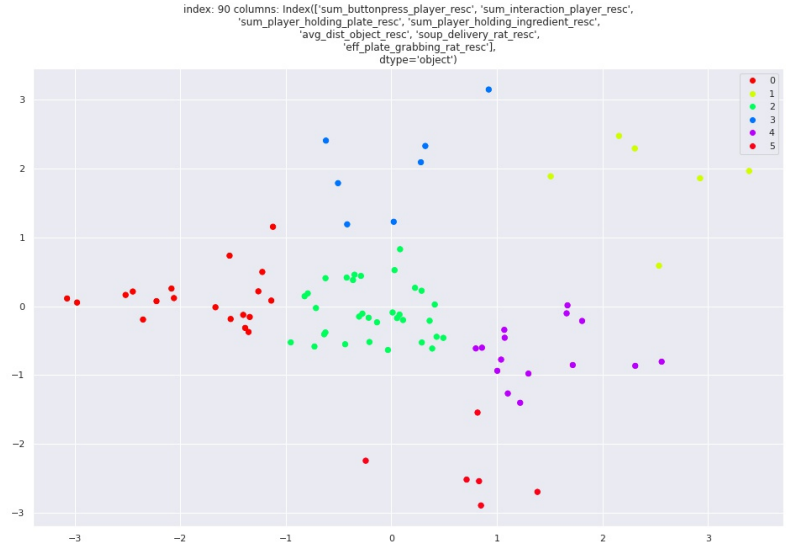
Number of PCA components	Number of Clusters (k)	Number of Features	Combination of Features
2	6	7	'sum_buttonpress', 'sum_interaction', 'sum_player_holding_plate', 'sum_player_holding_ingredient', 'avg_dist_to_object, 'soup_delivery', 'eff_plate_grabbing'

Table 5.1: The final setting for the K-means clustering algorithm

The result of K-means using the setting found during the exhaustive search is shown in Figure 5.3.



(a) Final Clustering visualized using PCA

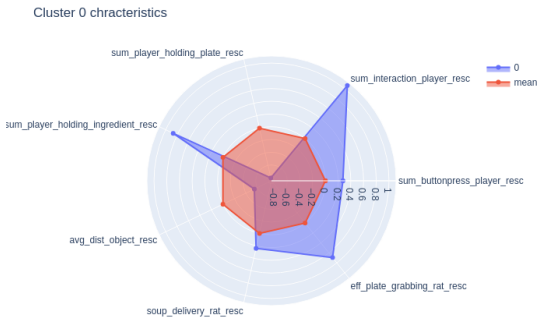


(b) Final Clustering visualized using TSNE

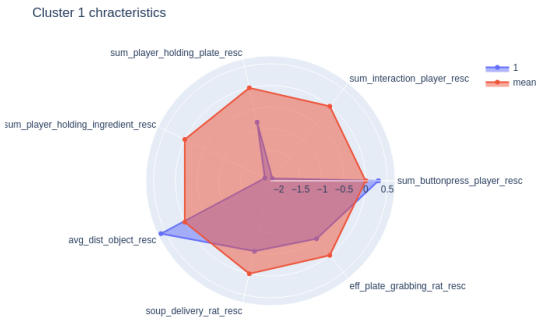
Figure 5.3: The resulting clusters, visualized using dimension reduction techniques PCA and TSNE. Both methods are used in order to confirm that the clusters are visible from different perspective and thereby reducing the risk of interpreting false findings.

5.2 Player Style Analysis

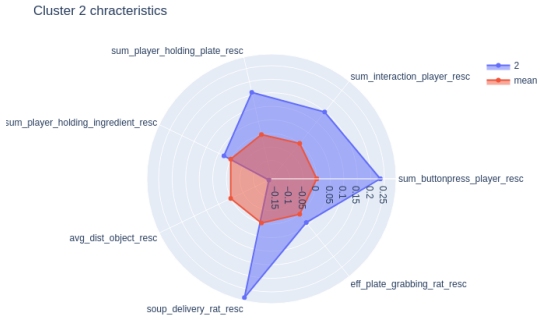
Now that the clusters are formed and visualized through dimensional reduction, the cluster labels can be related back to the original data to get a deeper understanding of what these clusters represent.



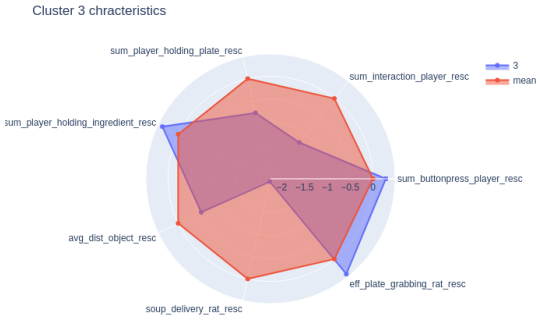
(a) Cluster 0



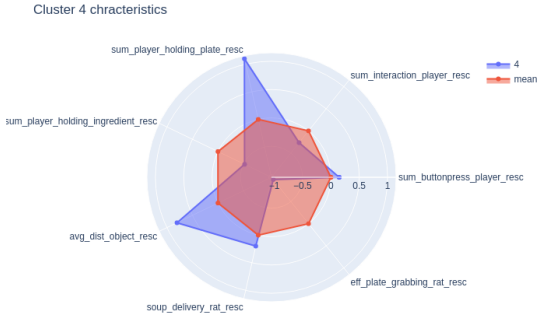
(b) Cluster 1



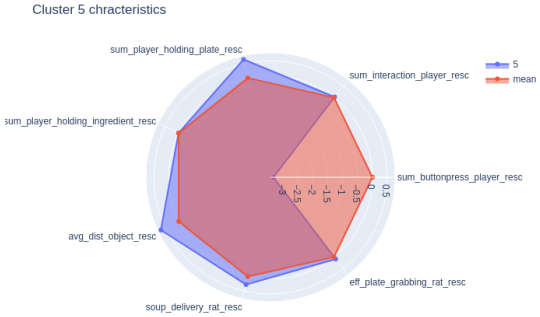
(c) Cluster 2



(d) Cluster 3



(e) Cluster 4



(f) Cluster 5

Figure 5.4: The mean over the chosen characteristics in the k-means exhaustive search per cluster (blue), compared to the mean over all players (red)

For each of the cluster, we analyse the characteristics describing the players. In the starplots in Figure 5.4, the mean values per cluster (blue) are plotted against the mean over the entire dataset (orange). For example, the players in cluster 2 hand in more soups than the average player.

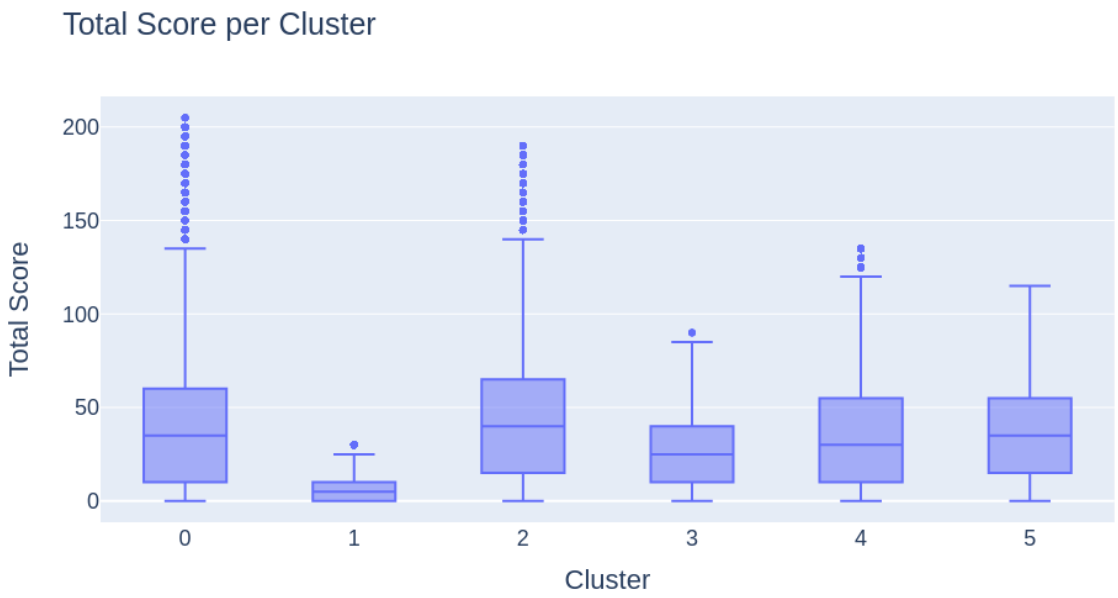


Figure 5.5: The total scores achieved by the different clusters. This shows that cluster 1 has the lowest average reward, where clusters 0 and 2 have the highest maximum score.

In Figure 5.5 the average score obtained for each cluster. Cluster 1 had the lowest average score, where cluster 0 and 2 have the highest average along with the highest maximum score.

5.3 Behavior Cloning Models

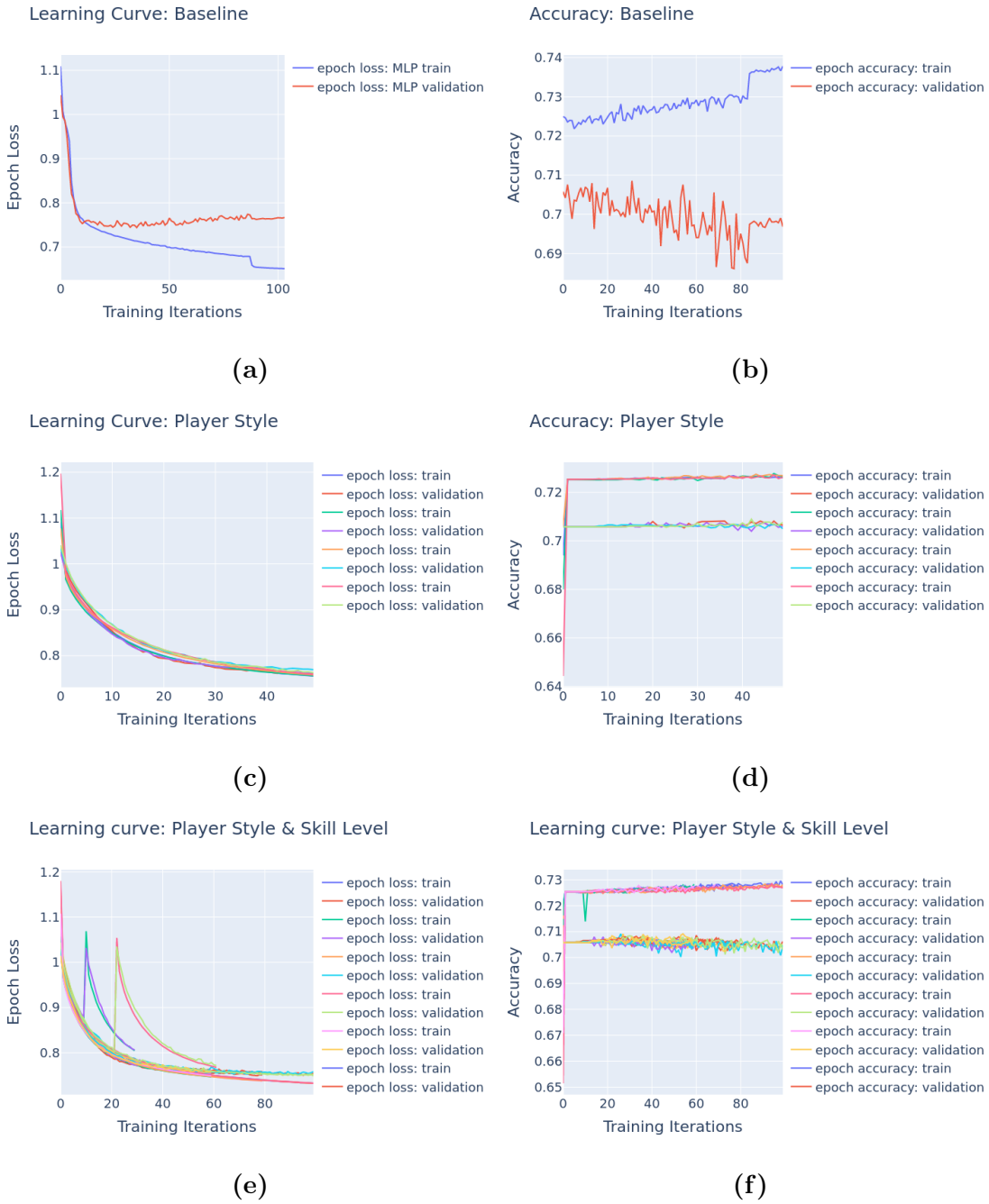


Figure 5.6: The loss (left) and accuracy (right) curves for the different implementations of behavior cloning models.

In Figure 5.6 the learning curves for the different implementations of the behavior cloning models are shown. Starting at the top, this learning curve of the behavior cloning model baseline. This is a Multi Layer Perceptron trained on all available trajectories. Secondly, the learning curve for the behavior cloning models each representing a play style based

on the clusters from the result in 5.1 and lastly, the learning curves for the behavior cloning models representing both different player styles as different skill levels. Note that some curves end in an earlier stage, belonging to the models that are terminated early to represent a lower skill level.

5.3.1 Results Hyperband

The hyperband algorithm explored a total of 250 parameter configurations. The hyperparameter search is complete. The setting leading to the highest validation sparse categorical accuracy for the baseline behavior cloning model with MLP architecure is:

Number of Densely Connected Layers	Units per Layer	Type of Activation Function	Learning rate
4	192	relu	0.001

Table 5.2: Best found hyper parameter setting for BHC

The graphs showing the learning curves of all configurations can be found in Appendix A.3 and A.2.

5.4 Reinforcement Learning

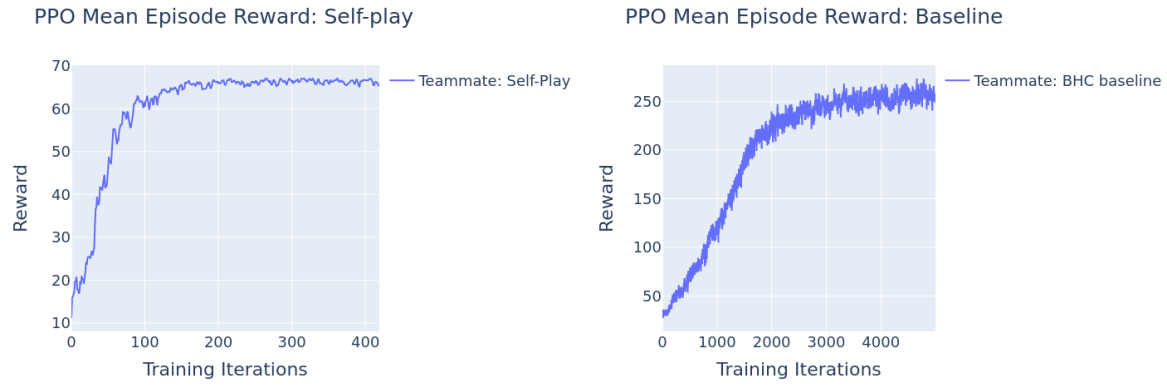


Figure 5.7: The resulting mean episode rewards for the different a PPO using a self-play teammate, compared to a PPO teamed up with an embedded BHC

First, as a sanity check, a PPO using self-play is compared against the PPO with the baseline BHC implementation in Figure 5.7. Here, the mean episode rewars is higher for the implementation using BHC, which is inline with the findings by Caroll et al.

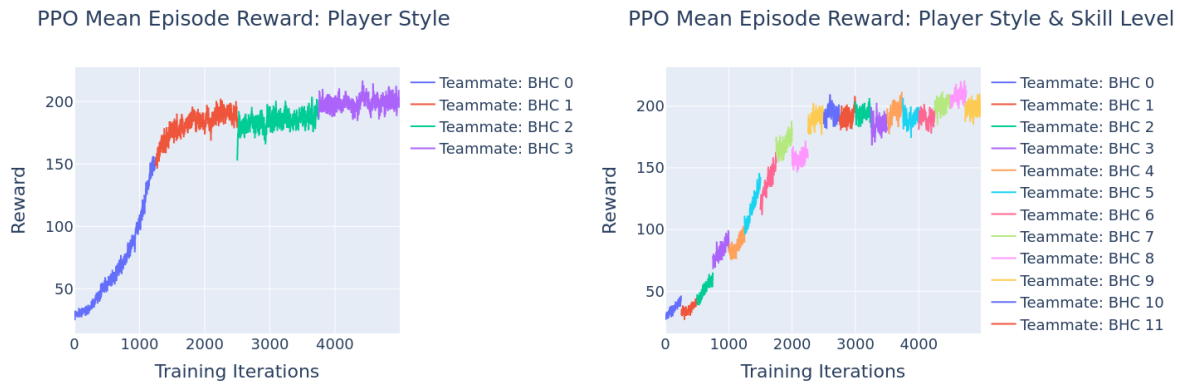


Figure 5.8: The resulting mean episode rewards for the different PPO-BHC implementations

Figure 5.8 show the mean episode reward during training for the PPO implementations using a BHC for each cluster, and the implementation also supplying the agent pool with different model checkpoints representing the skill-level. The reward these curves seem to converge to are similar. However, the curve of with the larger pool of agents is less smooth. Both implementations converge to a lower result than the PPO with the baseline BHC.

Chapter 6

Discussion

In this chapter, the results presented in the Chapter 5 are interpreted.

6.1 Clustering

The exploratory data analysis showed variation in number of button presses per player, indicating a difference in activity between players. To support the idea of training variation using behavior cloning models trained on different player styles, it is important to confirm that there are not only different activity levels, but also multiple player styles present in the data. By analyzing the clusters visualized in Figures 5.3a and 5.3b, the distance between the clusters indicate that there is the possibility to differentiate between players. This belief is confirmed when plotting the feature values using star plots, as shown in Fig 5.4. These plots clearly show that the average feature values between clusters differ. The features that were selected by the exhaustive search are a combination of activity related features, as well as holding and efficiency related features. This confirms the need for features beyond naive activity, established during exploratory data analysis.

The starplots give us the opportunity to interpret the player styles to a higher degree than on the reduced dimensions. For example, Cluster 2 shows higher than average plate holding and hand in rewards. This player possibly takes on the roll of waiter, thereby responsible for picking up the plate and delivering the soup. Some clusters might be considered outliers or bad players. An example of this can be seen in cluster 1, which has a high button press value and distance to any object, but scores below average on the other features. I argue that this is also a player style that will be common to occur in real life. During clustering, all layouts are taken into consideration in order to supply the clustering with as much data as possible, taking into account our limited number of players. Here, for each trial, the player is treated as an individual person. Hereby I also cover the possibility for a player to adapt its player style to the layout and the teammate.

This approach of clustering has a main limitation being that the clusters are defined based on historic data. If the dataset would be supplied with new trajectories, or simulations of the BHC, the centroids of these clusters may shift and a solution should be found for this. This also limits the possible analysis of the PPO player styles after training.

Referring back to Bartles taxonomy, there does not seem to be a direct translation from the player types defined to the player types found in these clusters using these features.

Bartles taxonomy relates to a competitive game, where overcooked is collaborative. Using similar method as applied during this thesis, there is potential for a new taxonomy tailored to the unique challenges provided in identifying player styles in a collaborative setting.

6.2 Behavior Cloning

Using the clusters found, the different implementations of behavior cloning models are trained. While all layouts are included in the clustering, the behavior cloning is only trained on the layout 'cramped room'. This is due to sample size restrictions and to stay consistent with the setup used by Caroll et al.. In order to ensure enough training data for the behavior cloning models, only the clusters with two or more players in 'cramped room' are included. This leaves us with four out of the six clusters. The results of the behavior cloning are shown in Figure 5.6. The top figure shows the result of the baseline model, where all players are included. It can be seen that the best validation score lies around 0.76. When the models train for many iterations, the training loss and validation loss start to deviate, indicating overfitting.

The amount of training data available to each model decreases when only taking the subset of players present in a cluster, resulting in a bigger loss, as seen in Figure 5.6c and 5.6e. This effect would be negligible as the dataset gets bigger. While a higher loss will likely have a negative impact on the performance of the reinforcement learning model, the advantage of variation in player style of the BHC is hypothesised to increase this performance. This is a trade-off between training diversity and sample size.

6.3 PPO

Figure 5.7 shows the baseline considered for this thesis compared to the PPO with a self-play algorithm, where the PPO with the BHC outperforms the former. This is inline with the findings by Caroll et al.

When comparing the three BHC implementations paired with a Proximal Policy Optimization, we use the metric mean episode reward. The reward for the baseline implementation goes to 250 within 4000 iterations. However, for the versions where the PPO is trained with alternating behavior cloning models, this reward converges to a value lower than 200.

One possible explanation of this could be the quality of the behavior cloning models. These models are trained on a much smaller set of trajectories and have a higher loss. Just as a human team with one player that is not skilled will result in a lower reward than a team with two highly skilled players, a team of worse performing BHC model and a PPO is likely to perform worse than a team consisting of a better BHC and a PPO. This indicates that with the current sample size, the loss occurring due to lower quality of the BHC models outweighs the gain by training diversification. Another possible cause could be the method of alternating between the behavior cloning models. Currently, each BHC model is used in a consecutive number of iterations, and will not be revisited again. It is possible the model would benefit from revisiting each model multiple times. This way the model can learn with a certain BHC model using knowledge it has gained along the way. Lastly, some of the models can represent less skilled behavior. Ending the training

with one of these models will register a lower loss than using a high skilled model as the final teammate.

Conclusion and Future Work

This thesis focused on two main goals: categorizing player styles in the game overcooked, and measuring the effect of varying the player styles that the reinforcement learning algorithm encounters during training.

When analyzing the human trajectories using dimensionality reduction, clustering and visualisation, it shows that different types of player styles are present, even in a simple environment as Overcooked. The clusters distinguish between different player styles, as well as different skill levels. I establish a method of characterizing player styles by using starplots. This offers the possibility for a standardized taxonomy for collaborative player styles.

The behavior cloning models trained on a subset of the human trajectories have a higher loss. Supplying these models to the PPO by alternating between BHC agents every x epochs results in a lower reward than the baseline. This is likely to be a result of the small sample size of player trajectories, or the technique used to alternate between these models.

The general experiment pipeline of clustering - behavior cloning - proximal policy optimization can be transferred to other environments.

For future work, there are multiple avenues that can be followed on the basis of this thesis. The avenues relate to the amount of data, definition of a taxonomy, and further adaption of the model.

With regards to the sample size, collecting more human trajectories could help improve the clustering accuracy and thereby the performance of the BHC and PPO models. Making these clusters more precise could also aid the ease of defining standard player types for collaborative play. Since the pipeline of the experiment can be transferred to other environments, it is a possibility to apply this to an environment with a larger dataset.

While this thesis focused on improving the model by supplying more variation in training data, future work could focus on a explicit adaption of the model by appending a categorization network in front of the PPO to distinguish players during an interaction. Additionally, different methods of training with a pool of behavior cloning models can be explored. In the current implementation, each model is only visited once for a certain amount of consecutive iterations. When looking for potential improvements, models could be revisited more often, or a sampling technique could be used to sample from a pool of actions supplied by the various behavior cloning models.

Bibliography

[1] C. Aggarwal, A. Hinneburg, and D. Keim. On the surprising behavior of distance metric in high-dimensional space. *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)*, 02 2002.

[2] F. Aiolli and C. Palazzi. Enhancing artificial intelligence in games by learning the opponent's playing style. *International Federation for Information Processing Digital Library; First IFIP Entertainment Computing Symposium on "New Frontiers for Entertainment Computing" (ECS-2008);*, 279, 01 2008. doi: 10.1007/978-0-387-09701-5_1.

[3] R. Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. *Journal of MUD research*, 1(1):19, 1996.

[4] L. Buşoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, Mar. 2008. doi: 10.1109/TSMCC.2007.913919.

[5] M. Campbell, A. Hoane, and F. hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57–83, 2002. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(01)00129-1. URL `https://www.sciencedirect.com/science/article/pii/S0004370201001291`.

[6] M. Carroll, R. Shah, M. K. Ho, T. L. Griffiths, S. A. Seshia, P. Abbeel, and A. D. Dragan. On the utility of learning about humans for human-ai coordination. *CoRR*, abs/1910.05789, 2019. URL `http://arxiv.org/abs/1910.05789`.

[7] A. Dafoe, Y. Bachrach, G. Hadfield, E. Horvitz, K. Larson, and T. Graepel. Cooperative AI: machines must learn to find common ground. *Nature*, 593(7857):33–36, May 2021. doi: 10.1038/d41586-021-01170-. URL `https://ideas.repec.org/a/nat/nature/v593y2021i7857d10.1038_d41586-021-01170-0.html`.

[8] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979. doi: 10.1109/TPAMI.1979.4766909.

[9] P. Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55:78–87, 10 2012. doi: 10.1145/2347736.2347755.

[10] J. N. Foerster, H. F. Song, E. Hughes, N. Burch, I. Dunning, S. Whiteson, M. M. Botvinick, and M. Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. *CoRR*, abs/1811.01458, 2018. URL `http://arxiv.org/abs/1811.01458`.

[11] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[12] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017. URL `http://arxiv.org/abs/1711.09846`.

[13] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019. doi: 10.1126/science.aau6249. URL `https://www.science.org/doi/abs/10.1126/science.aau6249`.

[14] J. R. Kok, M. T. J. Spaan, and N. Vlassis. Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems*, 50(2-3):99–114, Feb. 2005.

[15] A. Lerer, H. Hu, J. Foerster, and N. Brown. Improving policies via search in cooperative partially observable games, 2019.

[16] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL `http://jmlr.org/papers/v18/16-558.html`.

[17] R.-Z. Liu, W. Wang, Y. Shen, Z. Li, Y. Yu, and T. Lu. An introduction of mini-alphastar, 2021.

[18] N. Marwan, M. Carmen Romano, M. Thiel, and J. Kurths. Recurrence plots for the analysis of complex systems. *Physics Reports*, 438(5):237–329, 2007. ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2006.11.001. URL `https://www.sciencedirect.com/science/article/pii/S0370157306004066`.

[19] P. Nalepka, J. Gregory-Dunsmore, J. Simpson, G. Patil, and M. Richardson. Interaction flexibility in artificial agents teaming with humans. In *CogSci 2021: program for the 43rd Annual Meeting of the Cognitive Science Society*, pages 112–118. Cognitive Science Society, 2021. Annual Meeting of the Cognitive Science Society (43rd : 2021), CogSci 2021 ; Conference date: 26-07-2021 Through 29-07-2021.

[20] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*,

7(1-2):46–117, 2018. doi: 10.1561/2300000053. URL https://doi.org/10.1561%2F2300000053.

[21] S. Perrin, J. Perolat, M. Laurière, M. Geist, R. Elie, and O. Pietquin. Fictitious play for mean field games: Continuous time analysis and applications, 2020.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[23] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016. doi: 10.1109/JPROC.2015.2494218.

[24] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao. A survey of deep reinforcement learning in video games. *CoRR*, abs/1912.10944, 2019. URL http://arxiv.org/abs/1912.10944.

[25] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL http://arxiv.org/abs/1712.01815.

[26] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017. ISSN 1476-4687. doi: 10.1038/nature24270. URL https://doi.org/10.1038/nature24270.

[27] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams. Scalable bayesian optimization using deep neural networks, 2015. URL https://arxiv.org/abs/1502.05700.

[28] D. F. Williamson, R. A. Parker, and J. S. Kendrick. The box plot: a simple visual method to interpret data. *Annals of internal medicine*, 110(11):916–921, 1989.

[29] H. J. Wilson and P. R. Daugherty. Collaborative intelligence: Humans and ai are joining forces. *Harvard Business Review*, 96(4):114–123, 2018.

[30] R. Wilson, A. Shenhav, M. Straccia, and J. Cohen. The eighty five percent rule for optimal learning. *Nature Communications*, 10:4646, 11 2019. doi: 10.1038/s41467-019-12552-4.

Appendix A

Appendices

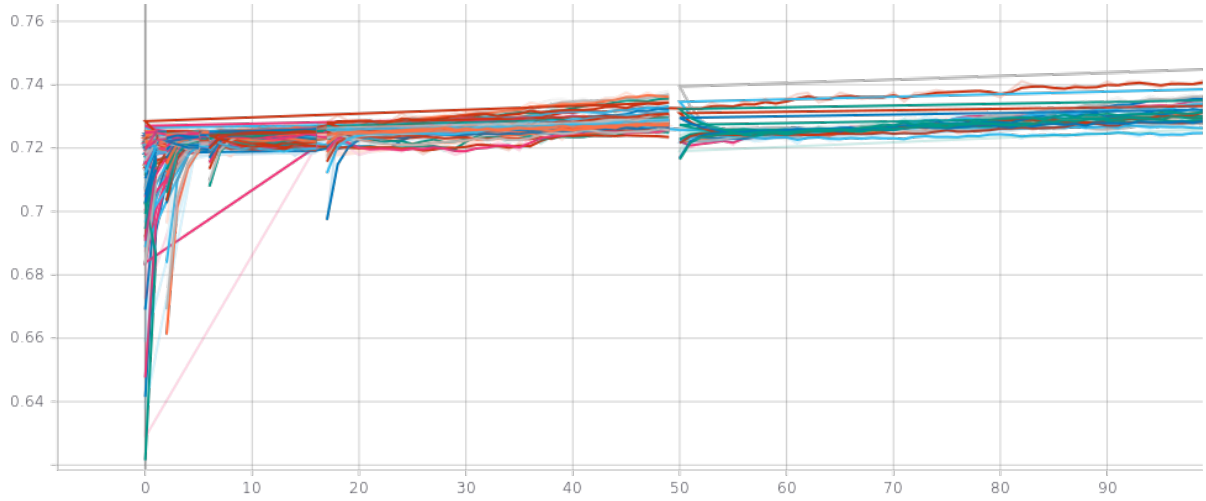


Figure A.2: Sparse categorical accuracy per parameter initialisation

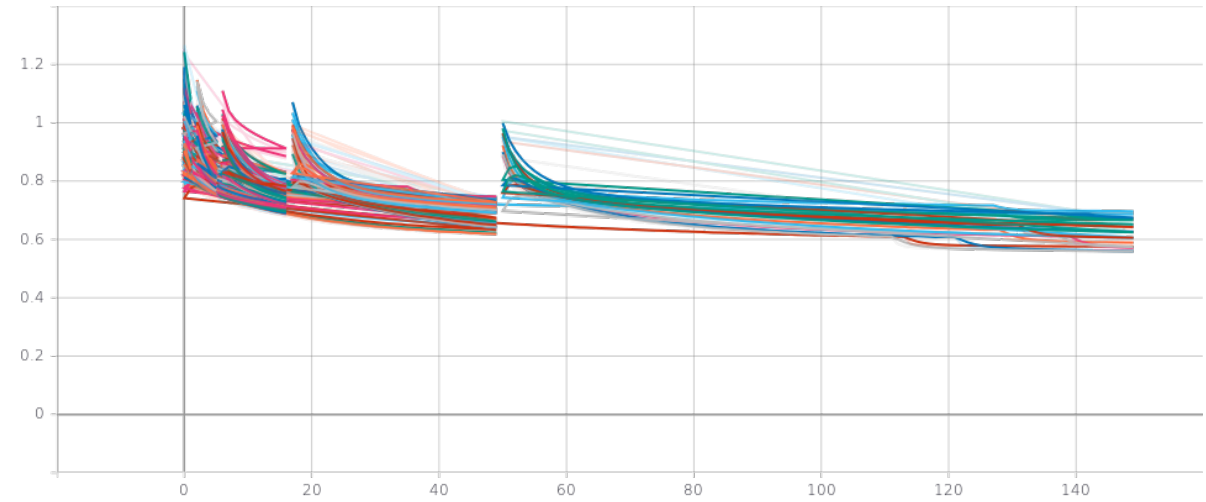


Figure A.3: Sparse categorical loss per parameter initialisation

Features	Description
state (JSON)	A JSON serialized version of a OvercookedState instance. Support for converting JSON into an OvercookedState python object is found in the Overcooked-ai repo
joint action (JSON)	A JSON serialized version of a joint overcooked action. Player 0 action is at index 0, similarly for player 1.
reward	The sparse reward achieved in this particular transition
time left	The wall-clock time remaining in the trial
score	Cumulative sparse reward achieved by both players at this point in the game
time elapsed	Wall clock time since begining of the trial
current gameloop	Number of discrete MDP (markov decision process) timesteps since beginning of trial
layout	The 'terrain', or all static parts (pots, ingredients, counters, etc) of the layout, serialized in a string encoding
layout name	Human readable name given to the specific layout
trial id	Unique identifier given to the trial (again, note this is a single Overcooked game; a single player pair could experience many trials). hline
player 0 id	Anonymized ID given to this particular Psiturk worker. Note, these were independently generated by us on the backend so there is no relation to Turk ID. If player is AI, the the player ID is a hardcoded AI_ID hline
player 1 id	Symmetric to player_0_id
player 0 is human	Indicates whether player_0 is controlled by human or AI
player 1 is human	Symmetric to player_0_is_human
cur gameloop total	Total number of MDP timesteps in this trial. Note that this is a constant across all rows with equivalent trial_id hline
score total	Final score of the trial
button press	Whether a keyboard stroke was performed by a human at this timestep. Each non-wait action counts as one button press
button press total	Total number of (human) button presses performed in entire trial
button presses per timestep	button_press_total / cur_gameloop_total
timesteps since interact	Number of MDP timesteps since the last human-input 'INTERACT' action

Table A.1: Full overview of the features available in the dataset by Caroll et al.

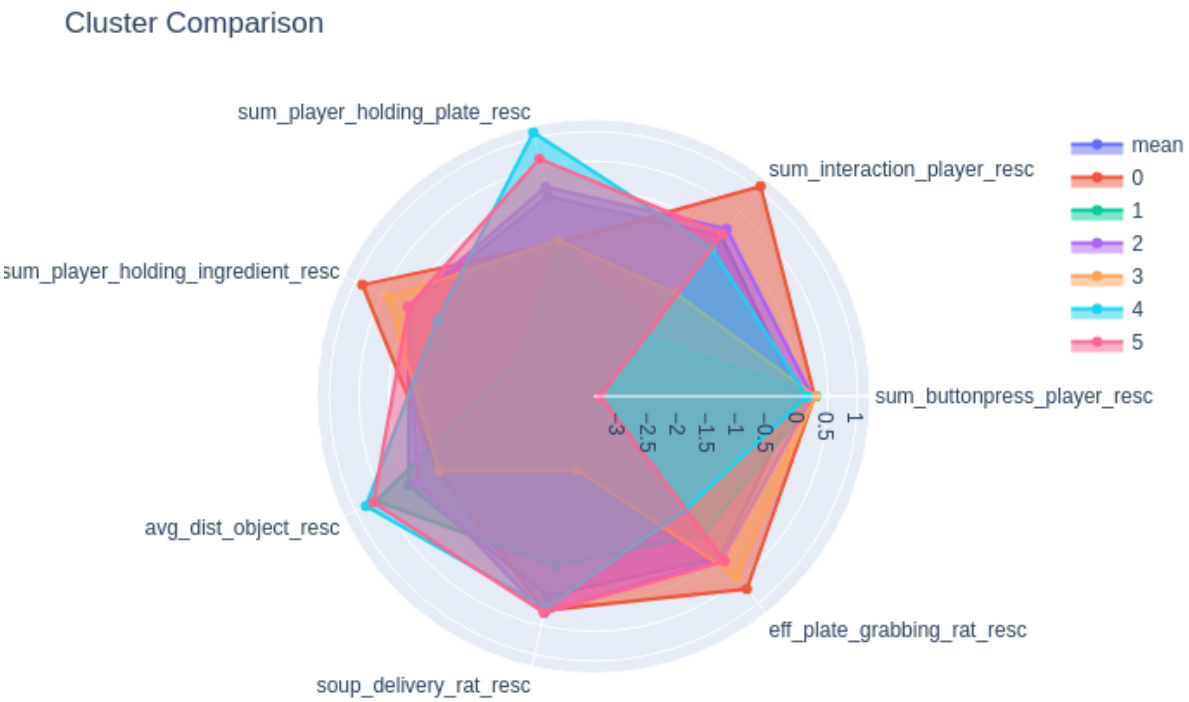


Figure A.1: The comparison of feature values over all clusters