MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Poker game with AI opponents

Bachelor's Thesis

RICHARD PLESNÍK

Brno, Fall 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Poker game with AI opponents

Bachelor's Thesis

RICHARD PLESNÍK

Advisor: Ing. Lukáš Grolig

Brno, Fall 2022


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Richard Plesník

Advisor: Ing. Lukáš Grolig

Acknowledgements

I would like to thank my advisor Ing. Lukáš Grolig, for all the valuable guidance, consultations and feedback. I would also like to thank my family and friends for all the support during my study and work on this thesis.

Abstract

This thesis aims to create a client-server poker application for playing against computer-controlled opponents. All of the necessary game logic is implemented on the server side. The client application works as a thin client rendering the server status. As a part of the server, artificial intelligence based on a machine learning (ML) algorithm was implemented. Multiple ML algorithms have been tested and compared to find a suitable model for the player's artificial intelligence (AI).

Keywords

machine learning, artificial intelligence, poker, Python, React, TypeScript

Contents

List of Tables

List of Figures

Introduction

In the last decade, machine learning (ML) has been one of the fastest-growing areas of information technology. It is even becoming more popular in commercial sectors such as banking [1], online marketing and others.

Nowadays, machine learning is also widely used in various games, where it performs significantly better than older specialised algorithms. It is often used to build computer-controlled players not only in classical computer games but also in traditional games such as chess [2].

This thesis focuses on building a functional application for playing poker games against computer-controlled opponents and creating an artificial intelligence player based on machine-learning methods. The goal was to explore the possible usage of different ML models to play poker and to select and train the model which would make a reliable impression of a real human player.

1 Background

1.1 History of AI in games

During the history of artificial intelligence, computers often competed and tried to beat the best human players in many games. At the beginning of artificial intelligence, computers have not been very successful against human players in most games. Despite that, as the performance of modern hardware kept growing and the research drove the AI field forward, computers slowly outperformed humans in the first less complicated board games.

1.1.1 AI in games in the years 1980-2000

In 1980, AI started surpassing humans in the first board games. The program IAGO learned to play the board game Othello on a world-championship level [3].

Fourteen years after, another big success in the field of AI vs human professionals was achieved in the game of checkers. The computer program Chinook has beaten the world champion, Marion Tinsley. To achieve this victory, the AI used a database of all the possible moves in situations with eight or fewer stones [4].

In the year 1997, a computer is winning the first game against professional players in a game of chess. The IBM chess supercomputer Deep Blue beat the world champion, Garry Kasparov, 3½–2½ [5]. Later after the loss, Kasparov asked IBM for a rematch, but his request was refused [6].

Despite that, the computer victory became one of the biggest milestones in the history of artificial intelligence; there was still room for improvement since Deep Blue could not win all of the games.

1.1.2 AI in games in the 21st century

In the 21st century, computers achieved even more success in games against human players. One of the most significant factors contributing to this success was progress in the research of neural networks and

deep learning, as well as the recent improvement in the performance of modern hardware.

Due to a major improvement in machine learning, most of today's chess game engines are beating human professionals with 100 per cent success, even on slower hardware [7]. On the other hand, professional players of the board game Go were unbeaten by AI even a few years ago.

The game Go is so complex that many professional players believed it could not be played well by computers. Even in 2010, computers were too bad to win against human players without a larger handicap [8]. It changed in 2016 when AI called AlphaGo was created by Alphabet's daughter company DeepMind Technologies [9]. With the help of deep learning algorithms, the computer program AlphaGo won four of the five games against a Go grandmaster Lee Sedola in 2016 [10]. After that, the new version of this AI, the AlphaGo-Zero, started to learn just by playing games with itself. After a few days, this self-taught version learned to play the game better than any human professional and became unbeatable [11].

1.2 AI in poker

The poker game was a big challenge for artificial intelligence because it significantly differs from other games such as chess, checkers, or go. There are many factors making poker particularly difficult to learn for computers. In poker, AI must deal with so-called "imperfect information" because the other player's cards are hidden most of the time. Besides that, it needs to operate with a lot of randomnesses since cards are random, and other players can also bluff. In the most popular poker variant, no-limit Texas hold 'em, up to six players can be playing in a single game, and there are no limits for bids, making it even more challenging for computers [12].

1.2.1 Libratus

The first successful attempt to beat human pro players in poker was achieved in 2017 by a poker bot Libratus. Libratus is an artificial intel-

ligence for playing the heads-up no-limit Texas hold 'em poker. It was developed at Carnegie Mellon University in Pittsburgh.

To beat the professional players, the AI used a specialised blueprint strategy, advanced nested subgame solving, and a self-improving algorithm. The self-improving algorithm was used to fix potential weaknesses identified by opponents during the game [13].

1.2.2 Pluribus

In 2019, the same team which developed the Lirbatus bot created a new poker AI Pluribus in Facebook's (now Meta) laboratory at Carnegie Mellon University. This time, the computer learned to play six-player poker games, which is even more complicated than the previous bot for heads-up poker with only two players.

In this case, the program learned primarily by playing against five copies of itself using the Monte Carlo counterfactual regret minimisation. During the actual games against professional players, this strategy was further enhanced by a self-improving algorithm [14].

After the training, Pluribus accomplished to beat all professional players in the game format of one AI against five humans and the format of five humans against one AI. Besides that, the program was also highly effective compared to previous AIs. It was using only a computer with two CPU units. In comparison, in the year 2016, AlphaGo was using a supercomputer with 1,920 CPUs and 280 GPUs to beat the professional player [15].

1.2.3 Pluribus game logs

When playing against five professional human players, all games were recorded and saved to log files. Pluribus has played a total of 10,000 hands of poker. These logs contain all the detailed information. It also includes information on the hands of all participating players.

Because of these logs' good quality and quantity, they were chosen to be used as a training dataset for this thesis's ML model. Only records of professional human players' actions were used for the training to prevent the ML player from copying the actions of the Pluribus AI.

This thesis aims to create a computer player that will play similarly to human players. Due to that, the model trained in this thesis

will be different from AIs from previously mentioned studies since they were focused on achieving the best performance against human professionals.

2 Design

For the creation of the application for playing poker, client-server architecture has been chosen. The game consists of two separate services, the backend and the frontend. The backend service provides a server with the main game logic and computer player model. Frontend service is a React web application which is rendering all the game data and communicating with the server. For the run and synchronisation of application services, containers were used.

2.1 Client-server model

Client-server is a model where the application is divided into two separate parts: client and server. The server works as a provider of the service which is used by one or multiple clients. The server usually provides an application programming interface (API) for communication with the client. The communication is handled by requests from the client and responses from the server.

Nowadays, the client-server model is popular for the development of applications due to the following benefits. Since the client-server model specifies how data are shared between services via its API, communication is safer and usually contains fewer vulnerabilities. Besides that, splitting the application into two services with clearly specified requirements can speed up the entire development process. This way, the work can be distributed between multiple teams who need to share only knowledge of the API schema and product requirements [16].

2.1.1 Client types

In the client-server model, there are two possible types of clients: thin and thick. The thin client is a lightweight client used primarily to communicate with the server and render its content. It is highly dependent on the server, which provides most of the application logic. Without a connection to the server, a standalone thin client cannot work properly.

On the other hand, a thick client is the exact opposite of a thin client. The thick client contains most of the application logic and can usually

work even if the server is not responding. In this architecture, the server is used primarily for the communication and synchronisation of clients [17].

For this thesis's application, thin client architecture has been chosen. Since the client is a web application and the application is running locally on the same computer, it is much easier to implement the machine learning and poker game logic on the server.

2.2 Containers

"Containers are executable units of software in which application code is packaged, along with its libraries and dependencies, in common ways, so that it can be run anywhere, whether it be on desktop, traditional IT, or the cloud" [18].

It is a modern method for the virtualisation of software without a need for a hypervisor. Apart from older virtualisation methods like virtual machines, containers are much more lightweight and versatile. Nowadays, containerisation is one of the fastest-growing information technology trends. Many significant IT companies are investing in it, participating in its development and using it for their own infrastructure [19, 20].

2.2.1 Benefits and use cases

Container technologies are usually used in two main situations; for the local development of the projects or for the deployment of production versions of applications. In each of the use cases, containers bring slightly different benefits.

In the development phase, containers can speed up and improve the development process by ensuring all the dependencies and by simulating the production environment with virtual networks and volumes. As Google states: "Containerization helps [their] development teams move fast, deploy software efficiently, and operate at an unprecedented scale" [21].

In production workloads, container technologies provide reliability, security and scalability. Software is much more reliable as it runs in a previously tested isolated environment. Running each part of the

service in a separate container limits potential damage if any security breach occurs. Some container images are tested and certified to work as intended on a given OS platform [22]. The container orchestration platforms can dynamically scale container resources based on various factors to ensure services are still running sufficiently.

2.2.2 Choice of the container platform

When choosing the right solution for the deployment of the thesis application, the following types of container platforms were considered; Linux containers, application container services, and container orchestration platforms.

Linux containers are predecessors of today's modern containers. It uses Linux namespaces and control groups to limit and isolate containers. Since Linux containers are relatively difficult to maintain and are not working on platforms other than Linux, they are not very used today and got replaced by more suitable solutions such as Docker.

Other options are application container services such as Docker or Podman. They provide a relatively easy-to-learn set of instructions for the management of containers and can run under every operating system. Besides that, there are many pre-created container images in public container catalogues. These images often come from verified sources and can rapidly speed up the whole project setup process.

The last and most advanced options are container management systems such as Google Kubernetes or Red Hat Opneshift. This kind of platform creates an additional layer over Docker/Podman containers and provides a system for "automating deployment, scaling, and management of containerised applications" [23]. Although this type of container orchestration might seem overly complicated for local application development, it becomes a useful tool once the application is launched on production servers.

Since this thesis application was created only to be played on a local computer, the Docker platform has been chosen as the most suitable container service.

3 Machine learning

Machine learning is a type of artificial intelligence in which the program is learned based on its experience. It is capable of predicting the outcome without explicit programming [24]. Due to that, it can be used to solve problems where the exact algorithm for the solution is still undiscovered or too complicated for practical usage. ML can also be described as data-driven programming [25].

3.1 Classification

Machine learning is usually divided into four basic categories: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. In supervised learning, models are trained on "labelled" data. In labelled data, each training vector is associated with a matching answer. Due to that, the results of the models can be easily measured by accuracy and other metrics.

In unsupervised learning, data are "unlabeled", and the goal is usually to analyse and categorise the data. It is usually used for methods like clustering or the detection of outliers. Clustering can be used to divide data into categories based on various elements. Outlier detection can be used, for example, for detecting malicious activity in computer networks or suspicious credit card usage.

Semi-supervised learning is a combination of both. Data used for semi-supervised learning are both labelled and unlabeled. Models are usually trained on supervised data, but they can still analyse unlabeled data afterwards.

The last type of learning is so-called reinforcement learning. In reinforcement learning, models are rewarded or penalised based on the success of their actions. The model aims to find a solution that maximises the cumulative reward [26]. For example, this kind of learning was used in the previously mentioned AlphaGo-Zero [11].

3.2 Supervised learning

In this section, supervised learning will be described since it was the type of ML used in this thesis. Supervised learning can be further divided into two categories: regression and classification.

The regression algorithms are used to predict continuous values. The classification is used for the prediction of discrete values [27].

3.2.1 Decision tree

A decision tree is a classification ML algorithm aiming to create a specialised tree for data classification. The tree is constructed from decision nodes and leaf nodes. The decision node compares the values of specific columns to determine the right subtree for the next classifications. Leaf nodes contain the final classification values. **Figure 3.1** shows an example of a simple decision tree for a given dataset.

	x0	x1	x2	x3	x4	y
0	-2	-5	0	-8	3	0
1	-3	-5	0	-10	-4	1
2	-3	-3	1	-2	-8	0
3	-9	2	-1	7	7	1
4	0	2	-4	0	2	0

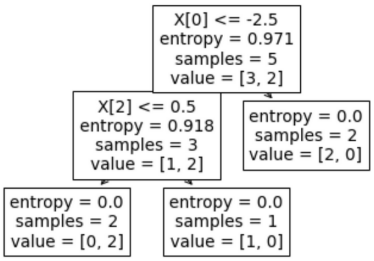

Figure 3.1: Example of a decision tree created on a given dataset.

To achieve the best results, the parameters used in decision nodes must be chosen well. Decision nodes are created based on a calculated information gain. Information gain is a metric calculated as a weighted average of entropies of all possible splits by values of a given column. The formula for calculating the entropy of a given attribute (column) is shown on **Formula** (2) [28].

$$\text{Gain (A)} = \text{H(Set)-(w}_1 \times H(a_1) + w_2 \times H(a_2) + \ldots + w_m \times H(a_m))$$

where a_1, a_2, ..., a_m *are the different values of attribute A,*
and w_1, w_2, ..., w_m *are the weights of the subsets split by*
using the values of attribute A.,
(3.1)

Entropy is a metric used to measure the amount of uncertainty of the information [29]. It is defined by **Formula** (3.2), where the *Pi* is a probability of class *i* in a given state. The highest possible value of entropy is 1 (all the classes are evenly distributed). To find the best dataset split for a decision tree, we aim to minimalise the entropy in a state created after the split.

$$H(p_1, p_2, ..., p_n) = \sum_{k=1}^{n} p^k \log_2 \frac{1}{p_k}, \tag{3.2}$$

3.2.2 Random forest

Random forest is a machine-learning algorithm based on decision trees. The main disadvantages of decision trees are that they are highly dependent on the first splits of branches and can be easily overtrained. These issues are partially resolved in the random forest algorithm.

The random forest randomly chooses multiple data subsets and builds a decision tree for each of them. Besides that, it also randomly chooses attributes to be trained on. This process is called bootstrapping (**Figure 3.2**). As a result of the bootstrapping, multiple smaller decision trees are created instead of one big decision tree. This algorithm makes the whole process more dataset independent.

	x0	x1	x2	x3	x4	y
0	-2	-5	0	-8	3	0
1	-3	-5	0	-10	-4	1
2	-3	-3	1	-2	-8	0
3	-9	2	-1	7	7	1
4	0	2	-4	0	2	0

Original dataset

	x0	x4	y
0	-8	1	0
1	-10	-5	1
2	0	2	0
3	2	2	0
4	0	2	0

Subset 1

	x0	x3	y
0	-8	-2	0
1	-8	1	0
2	2	7	1
3	-1	7	1
4	-9	7	1

Subset 2

	x2	x3	y
0	-3	1	0
1	-9	7	1
2	-9	7	1
3	-8	-2	0
4	-1	7	1

Subset 3

Figure 3.2: Example of bootstraping.

When a random forest model classifies data vectors, they are tested across all individual decision trees, and the result with the highest frequency is chosen. This model can also be used for regression tasks by using the average of classification results.

3.2.3 Support vector machine

Support vector machine (SVM) is an ML model used to solve classification problems. SVM uses hyperplanes to divide data into the right categories in a multidimensional space. The model is learned based on its accuracy, and the distance from the current boundary called the margin. The goal of the training is to maximise the margin between so-called support vectors defined by the nearest data points to the current hyperplane.

Multiple kernel types, such as linear, polynomial or sigmoid, can be used to find the best hyperplane. The most basic kernel version is a linear kernel which can be visualised as a simple line in a two-dimensional space. During the learning process, this line is incrementally moved and rotated to maximise the margin and find the best results.

3.2.4 Neural networks

Neural networks is a type of machine learning when the decisions are determined by a network of units called neurons. It is often used to solve problems such as image recognition, sound analysis and

others. These networks can be created by a large number of individual neurons, which are connected and trained to output the best results.

A neuron is a basic unit of the entire network. Each neuron takes a vector of numerical values as an input and returns a value of 1 or 0 as an output. When the output value is 1, the vector is so-called "activated". The vector of weight is assigned to the neuron's input vector. To calculate the output value, each vector uses three steps. At first, a weighted sum of the input vector is calculated. Afterwards, a number called "bias" is added up to the calculated sum. In the end, the result value is given to the activation function, which determines whether the final output is 1 or 0. As an activation function, the sigmoid (3.3) function is used most frequently, but it can be replaced by other functions such as signum, relu, etc. Because of these mechanisms, each neuron works as a linear function.

$$S(x) = \frac{1}{1 + e^{-}x} \tag{3.3}$$

By a combination of multiple vectors into layers, also more complicated than linear problems can be solved. One of the most popular models for these networks is a Multilayer perceptron.

3.2.5 Multilayer perceptron

Multilayer perceptron is a type of neural network which consists of the input, output and at least one hidden layer of neurons. The first layer takes the input and transfers it to hidden layers, the hidden layers do most of the network thinking, and the final output layer finalises the process and outputs solution. Each layer in the network is connected to the neurons in the next layer.

At the start of the neural network learning, all weights and biases are set to random values. After the initial setting, these parameters are repeatedly adjusted to find the best setting to solve a given problem. To find the right adjustment for the network parameters, an algorithm called Gradient descent is used to minimalise the cost function based on the actual learning results.

The gradient descent calculates the direction in which the cost function increases the most. Reverted value is used for the minimali-

sation. The process of adjusting the parameters based on the gradient descent results is called backpropagation.

4 Used technologies

4.1 Backend

For creating a backend service, Python has been chosen as the main programming language. To build the application programming interface, the Python framework FastAPI was used.

4.1.1 Python

"Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming [30]". It is an interpreted programming language with the broad support of numerous libraries and packages.

Python was chosen as a suitable programming language for the backend service for its simplicity and programming efficiency. With Python, the application's backend can be written faster than with more complex programming languages due to its simplicity and availability of packages for API development.

Python was also used for the machine learning part of this thesis since it can be easily used to train different ML models thanks to the available open-source Scikit-learn library [31].

4.1.2 Fast API

FastAPI is a modern, fast web framework for building APIs with Python. It is based on standard Python-type hints [32]. Because of its simplicity, the whole application API can be created really quickly. Only the FastAPI object and the necessary endpoint functions need to be initialised to build the fully functional API.

4.2 Frontend

To build the frontend service, multiple technologies were needed. TypeScript programming language and React javascript library was chosen as the main frontend application technologies.

4.2.1 TypeScript

TypeScript is an open-source programming language for web development based on Javascript [33, 34]. It is a modified Javascript with added strong typing. Because of the strong typing, the code written in TypeScript is more safe and readable. TypeScript is actively developed and maintained by Microsoft. For this thesis, it was chosen as the main frontend programming language for its readability and capability to prevent most typing-related errors that may be missed if Javascript was used.

4.2.2 React

React is an open-source JavaScript library for building user interfaces maintained by Meta and the community [35].It is a powerful tool for building modern web applications fast. Nowadays, it is also used for the development of desktop or mobile applications. React's most significant benefits are the fast development and the fact that created applications are runnable on all platforms. Nowadays, many modern web and mobile applications, such as Netflix, Discord, Uber and Airbnb, are built in React.

4.2.3 Material UI

Material UI is a modern open-source React library for creating UI components with Google's Material Design [36]. It contains a set of default components that can be further customised. Material UI also allows advanced theming across the entire application. All the styles, colours and fonts can be defined in a single file and impact all the application components. In this thesis, Material UI and CSS were used to design the frontend application UI.

4.2.4 CSS

CSS is a programing language for the styling of HTML components [37]. It can be used to describe parameters such as colour, size, positioning and others.

4.3 Machine learning

Besides the previously mentioned Python programming language, for the machine learning part of the thesis, some additional libraries were required. For the learning of ML models, libraries Skicit-learn and Pandas were used.

4.3.1 Scikit-learn

Scikit learn is an open-source Python machine-learning library [50]. It is built on NumPy, SciPy, and Matplotlib [31]. It contains support for most machine learning algorithms and provides a simple interface for data preprocessing, model training and evaluation of results. It was chosen as the machine learning library for this thesis because it is a powerful and easy-to-use library with detailed documentation.

4.3.2 Pandas

Pandas is an open-source, BSD-licensed library providing high-per formance, easy-to-use data structures and data analysis tools for the Python programming language [38]. In this thesis, it was used for loading and storing all the training and testing dataset records.

4.4 Deployment

To deploy all the application services, Docker was chosen as a container platform. Uvicorn and Nginx were used to start the backend and frontend servers. For the running of the frontend service, the Nginx web server was used. Vite was chosen as a tool for the final build of the fronend application.

4.4.1 Docker

Docker is the most popular container service used for local application development. Due to its high popularity and large user base, most of the popular services have their own prebuilt container images, which can be easily downloaded from the docker hub container catalogue

[39, 40]. For this thesis, it was used since it significantly simplifies the entire deployment process of both application services.

4.4.2 Uvicorn

Uvicorn is an Asynchronous Server Gateway Interface (ASGI) web server implementation for Python [41]. In this thesis, Uvicorn is used to start up the API listening for requests from the frontend service.

4.4.3 Nginx

Nginx is a high-performance open-source HTTP web server [42]. Even though the Nginx server is really lightweight, it can still deliver all the advanced functionality as other popular web servers. It was chosen for its easy configuration and lightweight.

4.4.4 Vite

Vite is a frontend development build tool that provides functionality for easier application development [43]. Besides the building of an application, it can also be used as a development server, which automatically shows application changes without needing a rebuild. In this application, Vite was used for building frontend service before the Nginx server could host it.

5 Backend service

The backend part of the applications creates a server that takes care of the realisation of the poker game. Backend service is divided into three main parts: the game's core, machine learning integration, and API. Each of these parts is stored in a separate package **5.1**. Besides that, the backend also contains a separate "constants" folder, which provides definitions of all necessary constants and type annotations.

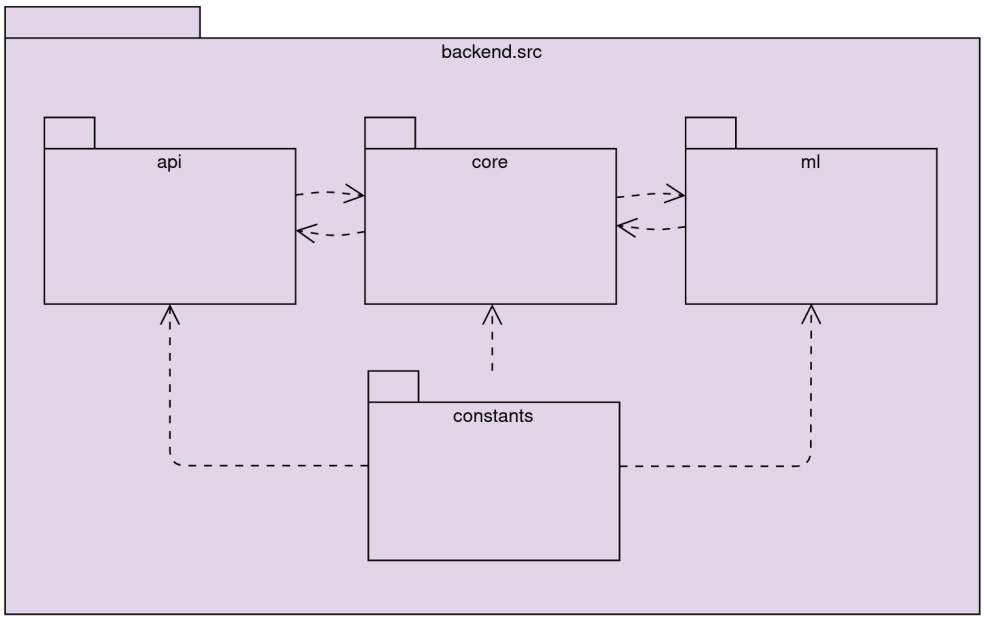

Figure 5.1: Backend package diagram.

5.1 Core package

The core package contains all of the files providing the main game logic. It consists of four modules *game.py*, *player.py* and *card_generator.py* and *winner_checker.py*. Each module consists of exactly one class with a matching name.

5.1.1 Game

The *Game* class has two main goals. To manage and synchronise all core package objects and provide the necessary functionality required to run a poker game.

When a *Game* object is created, basic game parameters and objects are defined. After the creation of the *Game* object, the start_game method is called. This method initialises table values and creates objects representing each of the players. After the start of the game, the *Game* object remains idle and waits for the beginning of the round. After the end of each round, the game returns back to the idle state.

For the start of the round, the game awaits the call of the *start_round* method from the *synchroniser* module. As the round starts, initial big blind and small blind bids are taken from players. After that, the *next_bidding* method is called by a frontend service. This method is used to synchronise phases of the round on the server. At the round beginning, this method starts a bidding phase of the game. In the bidding phase, AI and human players repeatedly bid in each river stage. After the end of the round, the *get_winner* method is called, the game returns to the idle state, and the whole process can start again.

5.1.2 Player

The *Player* object stores and manages all the relevant data about a player. It stores static parameters defined in creation, such as a player name, id and whether the player is a human or computer player. Besides that, it manages variables which are changing each round and provides the basic interface to operate with them. These values are, for example, the current player's hand, the last bidding operation and the bank value.

5.1.3 Card generator

The *CardGenerator* class shuffles the cards at the start of each round and deals them to players. During the game round, shuffled cards are used to build a river. Once the *CardGenerator* object is created, all the cards are initialised. After the creation, the *shufflecards* method can be used to reshuffle the cards.

All of the methods for dealing cards are designed to take cards from the top of the deck, so they keep the order from the round's initial shuffle. Thanks to that, dealing cards work the same way as in a real poker game.

5.1.4 Winner checker

The *WinnerChecker* class is used to evaluate the player's hand in combination with river cards. It effectively finds the highest card combination and returns the evaluation in the form of a list of numbers. The first number in the list represents the value of the combination (pair, flush, etc.). The other numbers represent details of the combination (e.g. highest card). To find a round winner, all players' hands are evaluated and compared.

5.2 ML package

This package provides all the modules used for getting the actions of computer-controlled opponents. It consists of two Python modules and a folder with trained models saved as serialised Python objects in binary format.

5.2.1 AI player module

The *ai_player* module manages all the computer players' necessary data and game actions. When the *AI_player* object is created, the related Player object is initialised. A model object with an assigned trained model is created based on the player parameters. This is the only module the *Game* class uses to communicate with the machine-learning part of the service directly.

5.2.2 Model module

The *model* module is responsible for loading and using trained machine-learning models. When the *Model* object is created, it loads the previously trained Scikit-learn machine-learning model based on the parameters given in the constructor method. After the initialisation, the *get_operation* method is used to determine computer-controlled

player operation based on the given game state. The game state is represented as a pandas *DataFrame* object with exactly one row of the same format model used for training.

5.3 API package

This package contains all the API-related Python modules. The process of API creation can be divided into three separate parts. The actual API definition, creation of the functions related to each endpoint, and the start of the service by the Uvicorn server.

5.3.1 API definition

The *api.py* module defines the entire API with all the necessary parameters. For the specification of the API, the fast API library was used. The entire API is formed in the below-defined steps.

At the start of the module, a new *FastAPI* object is created. After that, the list of allowed origins, methods and getters is set, so the Frontend service can properly communicate with the API. In the end, all the necessary endpoints are defined and connected to matching functions from the *Synchronizer* class.

5.3.2 Endpoint functions

To keep the API module minimalistic, each of the endpoint functions contains only the call of the assigned method of the *Synchronizer* object. The synchroniser module provides an additional layer, which manages all the necessary parameter and other variable checks and ensures that each endpoint operates correctly under specific circumstances.

After these checks, the synchroniser calls the appropriate functions from the modules stored in the *core* package. Due to this functionality, each part of the core package has its elementary purpose and does not need to do any other checks.

5.3.3 Start of the API

For the start of the actual API, the Uvicorn ASGI web server is used. It is launched with a path to the Python module and created *FastAPI* object. The API is running with a host IP address 0.0.0.0 and port 8000.

5.4 Constants package

The *constants* package contains two files, *constants.py* and *type_defs.py*. Both of these files provide important definitions used across the entire project.

5.4.1 Constants

The purpose of the *constant.py* file is to define and store all of the constants used by the backend application. Values of these constants are shared between other project parts. It is used, for example, for specifying the game's limitations.

5.4.2 Type definitions

The *type_def.py* file provides all of the necessary type definitions for other project parts. It contains definitions of custom enumeration classes and type aliases. Enumeration classes are used to limit the allowed set of values of the important game variables. Because of it, the application is more secure against errors caused by wrong variable values, and the code is also more readable.

For example, *GamePhase* enumeration class specifies all game life cycle phases. It is primarily used by frontend service to determine the current game state to render. This class also contains a custom method *increment*() for switching to the next game phase without the need to know anything about the internal value representation. To handle and manage different phases of the round, the *BiddingPhase* class was declared similarly to the *GamePhase* class.

Besides the enumeration definitions, this file also provides type aliases for the Python typing library. These aliases are used to document all of the functions with type annotations. This feature does not

directly affect the service functionality, but it ensures the code is more readable and easily manageable.

6 Frontend service

The frontend service was created as a React single-page web application. It was built to provide a user interface (UI) for playing poker against computer-controlled AI players. This service was built as a thin client, only communicating with the backend service and rendering the game status. Most of the game logic is implemented on the backend.

6.1 UI design

The main design goal was to build a simple and intuitive UI. Because of that, the single-page application was created for a frontend application. Users can easily open the game in their web browser and immediately see all the actual information. Because of that, the application is also resistant to accidental refreshes since the UI always renders the current game situation.

The whole graphical user interface consists of just three main components, the game status bar, game table and panel with player operations. Alternatively, when no game round is running, an information bar with a start button appears. In **Figure 6.1**, UI rendering the game results is shown. **Figure 6.2** shows the panel visible after the application startup.

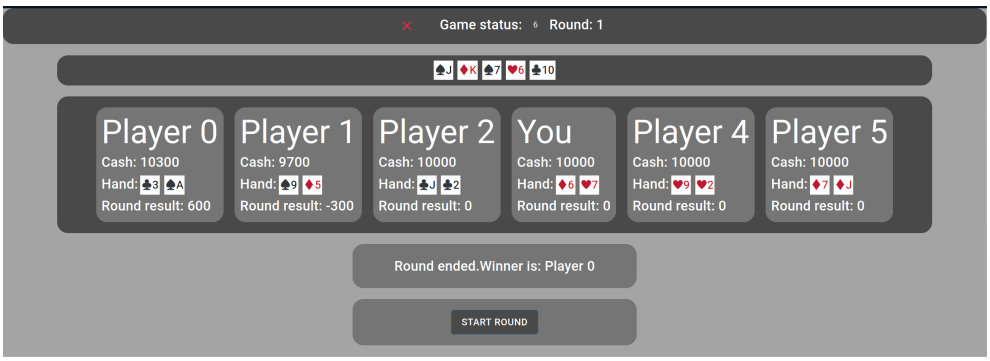

Figure 6.1: UI at the end of round.

6.1.1 Status bar

The status bar is the first component the player can see after the application startup. Its main goal is to show the current game status and inform the player when he is expected to play. Besides that, it also indicates the current round number. On the left side of the panel, there is also a button for terminating a game.

6.1.2 Table panel

The table panel is the main visible component of the application. It contains all of the game players and river cards. It is also maintaining the results after the end of each round. Unlike in real-life poker, computer players' cards are always shown at the end of each round. Due to that, players can see how the game AI works.

6.1.3 Operations panel

The status bar is used for the bidding actions of a human player. It is dynamically updated based on the current round context, so the player cannot do any invalid operation.

6.1.4 Between rounds panel

A special type of panel is displayed between rounds or before the game starts. This panel contains a brief description of a game status and a button for the start of the next round.

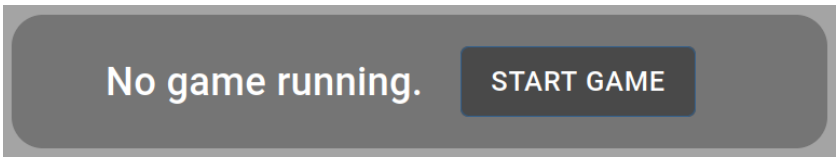

Figure 6.2: Panel after application startup.

6.2 Source code

The source code of the frontend service is divided into three folders. Components folder with all of the React components, the folder utilities with type declaration end HTTP call functions, and the styles folder used for styling UI elements.

6.2.1 Components

All of the game elements are represented as React function components. The Main component operates as a synchroniser and manager of all other sub-components. The synchronisation with the backend server is done by a single game loop. This loop sends an HTTP GET request to get the game status from the server each second. After each iteration, rendered content is refreshed based on updated status.

To share information between components, React state hooks are used. Most of these states are set up in the main game loop. To load additional information, some components contain functions for loading data from connected API endpoints. To execute requests on these endpoints, React effect hooks with configured state hooks are used 6.3.

The only two components sending POST requests are the Main component and the OperationPanel component. All the other components are passive and use only GET requests to refresh their content. The main component is using POST requests only to coordinate AI player operations at a speed suitable for humans. The OperationPanel sends a POST request every time a player does some bidding operation. Due to that, the client application is lightweight, and most of the game logic is provided by the backend server.

6.2.2 Utilities

The utility folder files contain the interfaces type definitions, game constants and utility functions. The interface type definitions specify the input parameters of React components.

The utility file contains shared functions used by many components. These functions have been moved to one specialised file to improve the readability of the code and avoid code duplicity.

```tsx
function PlayersPanel( { gameStatus, refresh, setRefresh, renderResult }: PlayersPanelProps) {

    const [players, setPlayers] = useState<PlayerProps[]>([]);

    const getPlayers = async () => {
        try {
            axios.get<PlayerProps[]>(API_ENDPOINT + 'players')
        .then(response => {
            console.log(response.data);
            setPlayers(response.data);
        });
        } catch (error: any) {
            console.log(error);
        }
    };

    useEffect(() => {
        setRefresh(false);
        getPlayers();
    }, [gameStatus, refresh]);

    return (
        <Box sx={playersPanel}>
            {players.map((player, index) => (
                <Player
                    name={player.name}
                    cash={player.cash}
                    hand={player.hand}
                    bid={player.bid}
                    action={player.action}
                    isHuman={false}
                    playerIndex={index}
                    gameStatus={gameStatus}
                    renderResult={renderResult}
                />
            ))}
        </Box>
    );
}
```

Figure 6.3: Example of UI component with React hooks.

6.2.3 Styles

The styles folder contains all the necessary files to specify the application's design. All the styling is done by three different files. The first file index.css defines the styling of an application root file index.html. The theme.ts file defines the global styling of all Material UI components in the application. The last file main styles.ts contains the definition of the style of all other UI components.

7 Machine learning preparation

The main goal of the training is to create a model for a computer player, which will be reliable enough to simulate a real human player. This makes the model different from the other poker AIs since they are created with the goal of beating human professional players.

The training of the machine learning model was done in three steps. At first, the dataset was parsed into a specialised format. After the parsing, three different machine-learning models were chosen and trained. In the end, models were tested with different parameter settings, and the final results were evaluated to select the most suitable model for the computer player.

7.1 Dataset choice

Records of the games played by professional players against Facebook's Pluribus AI were used as a training dataset [13]. More details about the Pluribus bot are described in the Background section of this thesis.

This dataset was chosen because of its significant benefits over the other datasets available online. The biggest advantage of this dataset is that it contains information about all the player's hands. The players' hands are typically missing in most datasets since they are usually gathered by passively observing online games.

Another benefit of this dataset is the expertise of players who played the games. Since all the players are professionals, the model learned from their plays can be much more consistent than models trained from amateurs games.

7.2 Dataset parsing

Before training models, the dataset had to be parsed from the original format to a format suitable for learning.

7.2.1 Original format

The original dataset contains records of sixty-seven games. In total, it includes information on ten thousand game rounds. Each game was played by five human players and one AI player. Not all the games were played by the same five players, but most of the rounds (over 70 per cent) were played by five main players.

The records of the games were divided into multiple files. Each game was saved as a separate text file, but if the game was too long, the record was split into two files. Each line of the game log contains information about exactly one game round.

Each part of the round data is separated by a character ":". It splits data into six parts. The first two parts specify that the row represents a log of the single game round with a given ID. The next part contains all of the round betting actions. Actions are separated by characters "/" to differentiate between different phases of the round. The fourth part of the log contains all of the player's cards and cards of the river. In the last two parts, the round results and the players' names are specified. Since data are in the same order as the players' names, all of the played actions can be easily assigned to a matching player.

7.2.2 Format after parsing

As the first step, records of the game were parsed to a format usable for training machine-learning models. The actions of the Facebook AI were ignored to prevent models from imitating existing AI. Only the actions of professional players were used for learning. Records of the losing games were kept in the training data, so the model creates an impression of a real player, which is not 100 per cent perfect.

The dataset was parsed to a single CSV file which can be easily loaded to data frames and used for training the models. The goal of parsing was to create a data vector for each game situation where the player needed to decide the next action. After that, the operation played by a matching player in this situation was assigned to the vector. The training vector needs to store all the relevant information about the game situation.

After the dataset parsing, each training vector contains the following information:

- Player order in the round
- Phase of the round
- Number of players who folded
- Bids of all the players
- Bids of other players
- Total cash held by all of the players
- Computer player cards
- River cards

```
||| Loading sample_game_100_68:
STATE:68:fr200ffcc/ccc/r600ff:9sKs|Ac7d|Jh7c|JcKd|7h5h|2d4h/5s2s2c/Th:400|-200|0|-200|0|0:MrWhite|MrPink|MrBr
own|Pluribus|MrBlue|MrBlonde
||| Parsed data:
0.5,0,0,0,50,100,0,0,0,8998.0,6599.0,15401.0,6250.0,15942.0,6810.0,111,70,0,0,0,0,0,f
0.8333333333333334,0,1,0,50,100,0,200,0,15942.0,6599.0,15401.0,8998.0,6250.0,6810.0,71,51,0,0,0,0,0,f
1.0,0,2,0,50,100,0,200,0,6810.0,6599.0,15401.0,8998.0,6250.0,15942.0,23,41,0,0,0,0,0,f
0.16666666666666666,0,3,50,100,0,200,0,0,6599.0,15401.0,8998.0,6250.0,15942.0,6810.0,92,132,0,0,0,0,0,c
0.3333333333333333,0,3,100,200,0,200,0,0,15401.0,6599.0,8998.0,6250.0,15942.0,6810.0,140,73,0,0,0,0,0,c
0.16666666666666666,1,3,200,200,0,200,0,0,6599.0,15401.0,8998.0,6250.0,15942.0,6810.0,92,132,52,22,20,0,0,c
0.3333333333333333,1,3,200,200,0,200,0,0,15401.0,6599.0,8998.0,6250.0,15942.0,6810.0,140,73,52,22,20,0,0,c
0.16666666666666666,2,3,200,200,0,200,0,0,6599.0,15401.0,8998.0,6250.0,15942.0,6810.0,92,132,52,22,20,101,0,r
0.3333333333333333,2,3,200,600,0,200,0,0,15401.0,6599.0,8998.0,6250.0,15942.0,6810.0,140,73,52,22,20,101,0,f
```

Figure 7.1: Example of data parsing.

7.2.3 Missing information

Since the training vector needs to be of a fixed size, some of the information has been lost during the parsing. In the training vector, the detailed context of players' operations is missing. On the other hand, these operations can be estimated from the values of the players' total bids.

Because the trained model is focused only on the classification of the proper computer operation, the exact value of the raised money is missing in the training data. Predicting the value of bids would require an additional machine learning model and could be done as possible future work on this thesis.

7.3 Models selection

After the creation of the training dataset, data still needs some further preprocessing before the start of the model training. After this prepro-

cessing, basic models can be trained and tested for suitability for the game.s

7.3.1 Data preprocessing

The dataset was randomly split into the training part and test parts in a ratio of 80/20. After that, StandartScaler was used for the transformation of all numerical values. Due to that, models sensitive to value fluctuations can still work properly.

7.3.2 Basic models comparison

Eight different models have been tested with default parameters to find the ideal model for poker player AI. After these tests, three models with the highest accuracy on test data were chosen for further testing and improvement.

After the evaluation of results, models MLPClassifier, RandomForestClassifier and SVC were selected for additional investigation due to their highest accuracy. Advanced tunning of these models and their evaluation is included in the next chapter, "Models tunning" 8.

8 Models tunning

All three selected models were trained with various combinations of hyperparameters. It was done to find an optimal setting of each model for the purpose of poker player AI. After the choice of the best model hyperparameters, multiple versions of the datasets were tested. Each version of a dataset contained only slight adjustments to the original.

At the end of the testing phase, the models' results were compared based on various criteria, and the best suitable model for the computer player was chosen.

8.1 Parameter tunning

A Scikit-learn tool called GridSearchCV was used to find the best hyperparameters setting. GridSearch automatically trains and tests the model with all of the possible subsets of given hyperparameters. In this thesis, all models were compared by their accuracy.

To eliminate possible distortion in training results, GridSearch automatically uses cross-validation. Because testing of all the models' hyperparameters took many hours for each model, the cross-validation was done only in two rounds.

8.1.1 MLPClassifier tunning

Model MLPClassifier was tested with different settings of parameters *activation*, *solver*, *alpha*, *learning_rate* and *max_iter*. At first, models were compared with 1-3 hidden neuron layers with all the combinations of sizes 32 a 64.

After this initial testing, best-performing hyperparameters were selected for testing with more variations of neuron layers. Best performing parameters *activation=tanh*, *activation=relu* and *solver=adam* were fixed for additional layer testing. This testing was done with 1-3 hidden neuron layers with all the combinations of sizes 32, 64, 128, 256. After this testing, the best-performing model had hidden neuron layers (128, 128, 128). **Table 8.1** shows the comparison of the models with the highest accuracies.

Table 8.1: Results of MLP grid search.

model	activation	alpha	layers	iter	Grid-acc.	acc.
Best tanh	tanh	0.0001	(32; 64; 32)	50	78.56%	79.12%
Best relu	relu	0.05	(32; 32; 64)	60	78.63%	78.83%
Best logistic	logistic	0.0001	(32; 64; 32)	60	75.85%	75.74%
Best identity	identity	0.05	(64; 32; 64)	50	73.10%	72.1%
Best with 2 layers	tanh	0.05	(64; 32)	60	78.48%	78.84%
Best with 1 layer	relu	0.05	(64)	60	77.03%	77.48%
Advanced 3 layers	tanh	0.05	(128; 128; 128)	50	78.88%	78.88%
Advanced 2 layers	tanh	0.05	(256; 128)	50	78.62%	78.76%
Advanced 1 layer	relu	0.05	(256)	60	77.57%	77.44%

Based on the grid serach results, a model called "Advanced 3 layers" was chosen as the best one for further testing. The model had slightly lower accuracy on the final training dataset, but it was caused by overtraining since the dataset is larger and can be fixed by adjustment of the *max_iter* parameter.

8.1.2 RandomForestClassifier tunning

For the model RandomForestClassifier hyperparameters *n_estimators*, *max_features*, *criterion*, *min_samples_leaf*, *min_samples_split max_depth* were tested.

After the first run of GridSearch, the best-performing models always had parameters *max_features=None*, *criterion=entropy* and the other four parameters on the highest tested values. Because of that, the first two parameters were fixed, and additional testing was done with the adjustment of the remaining four parameters. **Table 8.2** shows the comparison of the best RandomForest models. Based on test results, the model with parameters *criterion="entropy"*, *max_depth=40*, *max_features=None*, *min_samples_leaf=6*, *min_samples_split=21*, *n_estimators=1000* was chosen as the best one for further testing.

Table 8.2: Results of RandomForest grid search.

Name	depth	min_leaf	min_split	estimators	Grid-acc.	acc.
Depth 10	10	4	2	800	77.37%	76.8%
Depth 20	20	4	10	800	77.58%	77.27%
Depth 30	30	4	10	200	77.54%	77.51%
Depth 40	40	4	10	800	77.55%	77.36%
Depth 50	50	4	10	800	77.55%	77.39%
Advanced	40	6	21	1000	77.71%	77.58%

8.1.3 SVC

To find the best hyperparameters for support vector machine models, multiple types of kernels with different parameters were tested. In **Table 8.3** the most successful models for each kernel are compared.

Table 8.3: Results of SVC grid search.

Model	C	gamma	coef0	degree	GridSearch-acc.	acc.
Default (RBF)	1.0	scale	-	-	-	74.89%
Linear	0.001	-	-	-	70.10%	67.67%
RBF	10	scale	-	-	73.20%	75.42%
Poly	1	scale	1	4	72.90%	75.08%
Sigmoid	0.01	auto	-1	-	70.92%	67.66%

The best-performing of the tested models was a model with a linear kernel with changed parameter C from the original.

8.1.4 Hyperparameter testing results

After the testing of different hyperparameters, the best setting for each model was chosen. The accuracy of all models was improved by approximately 0.5-2%. **Table 8.4** shows the comparison of accuracy between default and upgraded models. These models were further used with different versions of the dataset in the next part of the chapter.

Table 8.4: Results of hyperparameter tunning.

Model	Default accuracy	Improved accuracy
MLPClassifier	77.38%	78.88%
RandomForestClassifier	76.10%	77.58%
SVC	74.89%	75.42%

8.2 Dataset tunning

After finding the best settings for all three models, these models were tested with different versions of the dataset. Since the default dataset contains all data columns, the new versions were created by dropping some of the original columns or merging multiple columns into a single one. Even though some information is missing after the dropping, it may still lead to better results since learning from smaller-sized vectors makes the entire process much easier.

8.2.1 Columns merging

As a first dataset adjustment, merging related values to a single column was tried out. At the start, all the opponent bid values were summed up and saved to a single column instead of the five previous columns. After that, the same operation was done with the values of players' banks. In the end, a combination of these adjustments was tested. In **Table 8.5** results of the model testing on a dataset with merged columns are summarised.

Table 8.5: Models' accuracy after merging columns.

Model/dataset	Default dataset	Merged bids	Merged banks	Bids + banks
MLP	78.88%	78.53%	78.74%	78.97%
RandomForest	77.64%	77.99%	78.53%	78.66%
SVC	75.42%	75.46%	75.60%	75.65%

Since merging both column types caused a slight improvement in the accuracy of all models, this changed version of the dataset was used in the final part of testing.

8.2.2 Columns dropping

As another dataset adjustment, the dropping of one or multiple columns was tested. For testing, all of the subsets of the following columns were dropped:

- Player_order
- Opponent bids
- Player bank
- Opponent banks
- Count of players who folded
- Total cash held by all of the players
- Computer player cards.

The number of all possible tested subsets for seven columns was $2^7 = 128$. The best results for each category are described in **Table 8.6**. The accuracy started to decrease after dropping more than four columns. The best results were achived after droping columns current_river, my_bank, and opponent_banks, from the previous dataset.

Table 8.6: Model's accuracy on simpified dataset.

DropedColumns	MLP acc.	RandomForest acc.	SVC accc
0	78.97%	78.66%	75.65%
1	78.99%	78.62%	75.66%
2	79.03%	79.22%	75.80%
3	79.14%	79.24%	76.01%
4	79.02%	78.57%	75.53%
5	77.73%	78.17%	74.66%

8.3 Final model choice

For the final version of the AI, MLPClasifier was chosen as the most suitable model. Although RandomForestClassifier achieved slightly higher accuracy, it was rejected due to its large size, as the trained and serialised model takes a total of 355.8 MB of computer disk space compared to only 845.5 kB of MLPClassifier.

9 Application deployment

9.1 Service architecture

The Poker application consists of two fundamental parts, the backend and the frontend. The main part, providing all the game logic and synchronisation, is the backend server written in the Python programming language. This part also contains a machine learning model representing a computer player and ensures communication with the frontend by delivering a simple REST API. The other part is a web application providing users with a simple graphical interface to play the actual game and manage effective communication with a server.

Both the backend and frontend run as entirely separate services. Each service runs in a standalone container and communicates by an agreed set of HTTP requests 9.1. The Docker platform was used to set up the entire application architecture and its deployment.

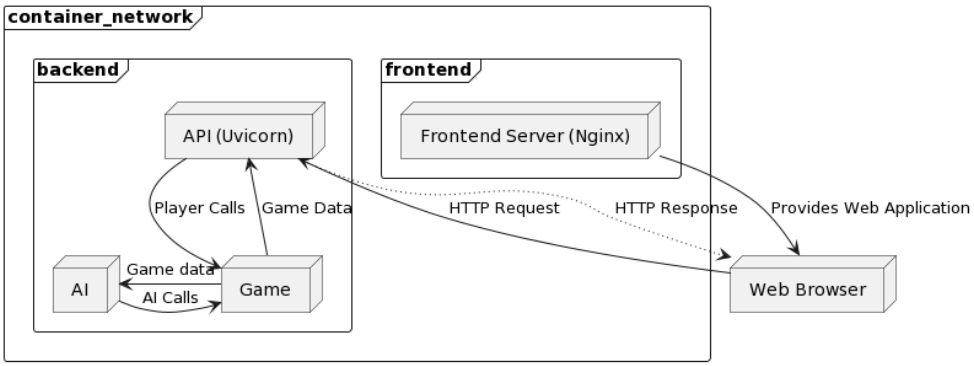

Figure 9.1: Application deployment.

9.2 Container orchestration

The source code of the poker game is split into two separate folders. One with the backend and the second one with the frontend. Each of

the services runs in a different container, defined by a related Dockerfile.

The backend service uses the python:3.9.4-slim container[1]. It is an official container image for running python applications. The slim version of the image includes only minimal packages required to run python. When the container is created, it installs all the necessary python packages specified in included requirements.txt file.

For a build and run of the frontend service, two containers are required. The first container, created from node:18-alpine[2] image, handles the installation of the necessary node packages and the build from the typescript/react source code to a runnable web application. After the build, application files are copied to a new nginx:alpine[3] container and the old build container is removed. The new container contains an Nginx web server, which hosts the frontend application and makes it accessible via a web browser. Container images running on Alpine Linux have been chosen for their simplicity and small system size.

Launch of all the containers is managed by a single docker-compose file. On the backend server, REST API is launched by an Uvicorn[4] ASGI web server. The API is listening for the HTTP request from the frontend services. After the frontend application is launched, it is available via web browser on port 3000. Both of these services share the same virtual network, so they can easily communicate and transfer game data.

1. `https://hub.docker.com/_/python`
2. `https://hub.docker.com/_/node`
3. `https://hub.docker.com/_/nginx`
4. `https://www.uvicorn.org/`

10 Future improvement

This chapter describes possible future improvements to the poker application and its AI.

10.1 Additional model tunning

Even though most of the hyperparameters were already tested, there is still a possibility that further testing could discover more optimal model settings. When training the MLPClassifier, more combinations for a network's hidden layers could lead to better model performance.

Besides that, there is also space for improvement in better testing of different dataset versions. Since all the hyperparameters were tested only on a default dataset format, and only the best models were tested with adjusted files, the optimal model setting could be missed. This could be eliminated by additional testing of all hyperparameter options on all of the dataset versions. This testing was not done in this thesis since it would take approximately a dozen hours of computation time on a personal computer with standard hardware, and it may only bring a slight improvement.

In addition, testing hyperparameters with more cross-validation runs could be beneficial for eliminating the risk of choosing the wrong setting due to random dataset splitting. On the other hand, it would significantly prolong the parameter testing time.

10.2 ML for bid values

The computer player AI could be further improved to learn also the value of the player bids. Since the current model was trained on an operation classification, it is only capable of predicting whether to raise or not. For predicting exact bid values, another machine learning model would have to be trained. This model would require some regression algorithm and would be trained only on records of situations when a player raised a bid.

10.3 Player dependent models

In a current poker application version, all five computer players use the same machine learning model. It was trained on games of multiple players, so it plays more as an "average" of these players. Although this model can have different accuracies on different player actions, so the word "average" should be taken with a grain of salt.

For a better user experience, separate models for each player could be learned as an improvement. These models would have a play-style characteristic to the specific player. On the other hand, this additional training would probably require a more extensive dataset since there are significantly fewer records of individual player actions.

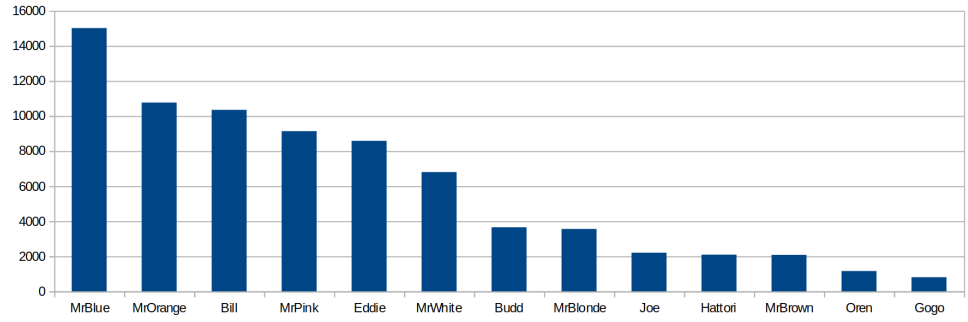

Figure 10.1: Count of records of each player situations

11 Conclusion

The goal of this thesis was to create a poker application for playing against computer-controlled opponents. Requirements have been to use client-server architecture with all the necessary game logic implemented on the server and a thin client for playing the game. The computer-controlled was required to be based on a machine-learning algorithm.

In the first chapters, the thesis described the history and previously achieved successes in the field of machine learning and artificial intelligence. In the next section theory behind the architecture of this thesis's application was covered. After the description of application design, the theory behind used machine learning models was explained. At the end of the theoretical part of the thesis, the technologies used in this thesis are introduced.

The implementation part of the thesis consisted of programming the client-server application and training the machine learning model. After the creation of the application, the ML model was trained and implemented in the application's backend part. To select the suitable model for a computer-controlled player, multiple machine-learning algorithms were tested with different hyperparameters and training files. After testing multiple algorithms, the best-performing model based on neural networks was integrated into the final application. After the model integration, the application was deployed to containers, managed by a single docker-compose file.

As the final product of this work, a functional client-server application for playing poker against ML-based computer-controlled opponents was created. Users can easily launch the entire application with a single command and play poker against five AI players. By that, the requirements for this thesis were satisfied. In addition, multiple machine-learning methods were investigated for their possible usage for game AI.

Bibliography

1. DOERR, By Sebastian; GAMBACORTA, Leonardo; SERENA, Jose Maria. How do central banks use big data and machine learning. In: *The European Money and Finance Forum*. 2021, vol. 37, pp. 1–6.
2. ACHER, Mathieu; ESNAULT, François. Large-scale Analysis of Chess Games with Chess Engines: A Preliminary Report. *CoRR*. 2016, vol. abs/1607.04186. Available from arXiv: 1607.04186.
3. ROSENBLOOM, Paul S. A world-championship-level Othello program. *Artificial Intelligence*. 1982, vol. 19, no. 3, pp. 279–320. ISSN 0004-3702. Available from DOI: https://doi.org/10.1016/0004-3702(82)90003-0.
4. SCHAEFFER, Jonathan; LAKE, Robert. Solving the game of checkers. *Games of no chance*. 1996, vol. 29, pp. 119–133.
5. CAMPBELL, Murray; HOANE, A.Joseph; HSU, Feng-hsiung. Deep Blue. *Artificial Intelligence*. 2002, vol. 134, no. 1, pp. 57–83. ISSN 0004-3702. Available from DOI: https://doi.org/10.1016/S0004-3702(01)00129-1.
6. NEWBORN, Monty. *Beyond deep blue: Chess in the stratosphere*. Springer Science & Business Media, 2011.
7. CHESS.COM. *Stockfish - Chess Engines* [online]. [visited on 2022-12-05]. Available from: https://www.chess.com/terms/stockfish-chess-engine.
8. ASSOCIATION, British Go. *History of Go-playing Programs* [online]. 2018. [visited on 2022-12-05]. Available from: https://www.britgo.org/computergo/history.
9. DEEPMIND.COM. *AlphaGo* [online]. [visited on 2022-12-05]. Available from: https://www.deepmind.com/research/highlighted-research/alphago.
10. SILVER, David; HUANG, Aja; MADDISON, Chris J; GUEZ, Arthur; SIFRE, Laurent; VAN DEN DRIESSCHE, George; SCHRITTWIESER, Julian; ANTONOGLOU, Ioannis; PANNEERSHELVAM, Veda; LANCTOT, Marc, et al. Mastering the game of Go with deep

neural networks and tree search. *nature*. 2016, vol. 529, no. 7587, pp. 484–489.

11. SILVER, David; SCHRITTWIESER, Julian; SIMONYAN, Karen; ANTONOGLOU, Ioannis; HUANG, Aja; GUEZ, Arthur; HUBERT, Thomas; BAKER, Lucas; LAI, Matthew; BOLTON, Adrian, et al. Mastering the game of go without human knowledge. *nature*. 2017, vol. 550, no. 7676, pp. 354–359.
12. DEEPMIND.COM. *MIT Technology review* [online]. [visited on 2022-12-05]. Available from: `https://www.technologyreview.com/2017/01/23/154433/why-poker-is-a-big-deal-for-artificial-intelligence/`.
13. BROWN, Noam; SANDHOLM, Tuomas. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*. 2018, vol. 359, no. 6374, pp. 418–424. Available from DOI: `10.1126/science.aao1733`.
14. BROWN, Noam; SANDHOLM, Tuomas. Superhuman AI for multiplayer poker. *Science*. 2019, vol. 365, no. 6456, pp. 885–890. Available from DOI: `10.1126/science.aay2400`.
15. AI, Noam Brown - Meta. *Facebook, Carnegie Mellon build first AI that beats pros in 6-player poker* [online]. 2019. [visited on 2022-12-05]. Available from: `https://ai.facebook.com/blog/pluribus-first-ai-to-beat-pros-in-6-player-poker/`.
16. OLUWATOSIN, Haroon Shakirat. Client-server model. *IOSR Journal of Computer Engineering*. 2014, vol. 16, no. 1, pp. 67–71.
17. GOCLOUD. *Thin Client vs thick client: the pros and cons* [online]. [visited on 2022-12-05]. Available from: `https://www.gocloud.co.uk/glossary/thin-client-vs-thick-client-the-pros-and-cons`.
18. EDUCATION, IBM Cloud. *Thin Client vs thick client: the pros and cons* [online]. 2021. [visited on 2022-11-25]. Available from: `https://www.ibm.com/cloud/learn/containers`.
19. HECHT, Lawrence E. *TNS Research: The Present State of Container Orchestration* [online]. 2016. [visited on 2022-11-25]. Available from: `https://thenewstack.io/tns-research-present-state-container-orchestration`.

20. MILLER, Ron. *Google invests $1B in CME Group as part of long-term Google Cloud deal* [online]. 2021. [visited on 2022-11-25]. Available from: `https://techcrunch.com/2021/11/04/google-clouds-1b-investment-in-cme-group-part-of-long-term-infrastructure-services-deal/`.

21. GOOGLE. *Containers in Cloud* [online]. [visited on 2022-11-25]. Available from: `https://cloud.google.com/containers`.

22. REDHAT. *Red Hat Container Certification* [online]. [visited on 2022-11-25]. Available from: `https://connect.redhat.com/en/partner-with-us/red-hat-container-certification`.

23. GOOGLE. *Kubernetes - Production-Grade Container Orchestration* [online]. [visited on 2022-11-25]. Available from: `https://kubernetes.io`.

24. BURNS, E. In-depth guide to machine learning in the enterprise. *Techtarget. March*. 2021, p. 17.

25. MITCHELL, Tom M; MITCHELL, Tom M. *Machine learning*. Vol. 1. McGraw-hill New York, 1997. No. 9.

26. MAHESH, Batta. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*. 2020, vol. 9, pp. 381–386.

27. GUPTA, Sakshi. *Regression vs. Classification in Machine Learning: What's the Difference?* October, 2021.

28. YANG, Feng-Jen. An extended idea about decision trees. In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2019, pp. 349–354.

29. RÉNYI, Alfréd et al. On measures of entropy and information. In: *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability*. Berkeley, California, USA, 1961, vol. 1. No. 547-561.

30. VAN ROSSUM, Guido; DRAKE, Fred L. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

31. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.

32. FASTAPI. *FastAPI* [online]. [visited on 2022-12-05]. Available from: `https://fastapi.tiangolo.com/`.

33. TYPESCRIPT. *TypeScript is JavaScript with syntax for types* [online]. [visited on 2022-12-05]. Available from: `https://www.typescriptlang.org/`.

34. SCHOOLS, W3. *JavaScript Tutorial* [online]. [visited on 2022-12-05]. Available from: `https://www.w3schools.com/js/`.

35. META PLATFORMS, Inc. *React - A JavaScript library for building user interfaces* [online]. [visited on 2022-12-05]. Available from: `https://reactjs.org/`.

36. UI, Material. *Matterial UI* [online]. [visited on 2022-12-05]. Available from: `https://mui.com/`.

37. SCHOOLS, W3. *CSS Tutorial* [online]. [visited on 2022-12-05]. Available from: `https://www.w3schools.com/css/`.

38. TEAM, The pandas development. *pandas-dev/pandas: Pandas 1.3.5*. Zenodo, 2021. Version 1.3.5. Available from DOI: `10.5281/zenodo.5774815`.

39. DOCKER, Inc. *Docker: Accelerated, Containerized Application Development* [online]. 2022. [visited on 2022-12-05]. Available from: `https://www.docker.com/`.

40. DOCKER, Inc. *Docker Hub* [online]. [visited on 2022-12-05]. Available from: `https://hub.docker.com/`.

41. UVICORN. *Uvicorn* [online]. [visited on 2022-12-05]. Available from: `https://www.uvicorn.org/`.

42. NGINX - F5, Inc. *Nginx* [online]. [visited on 2022-12-05]. Available from: `https://docs.nginx.com/nginx/admin-guide/web-server/`.

43. VITE. *Vite - Next Generation Frontend Tooling* [online]. [visited on 2022-12-05]. Available from: `https://vitejs.dev/`.

A An appendix

A.1 Electronic attachments

This work consists of the electronic version of the thesis and a zip archive *poker_application.zip* with the entire application and the source codes used for the training of machine-learning models. Both these files are publicly available in the Information System of Masaryk University.

The archive contains a *README.md* file with instructions for launching the application and the list of its requirements. The *pokerGame* folder contains folders: frontend, backend and machine-learning. The frontend and backend folders contain both of the application services. The folder machine-learning contains source code used for training the models, notes from testing of these models and a folder with serialized trained models.