Master's Thesis

Submitted to the IT Security Research Group
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

Player Profiling in CS:GO: Leveraging Machine Learning for Mouse Movement-Based Identification

Lars Gansel

Supervisors: Prof. Dr. Patricia Arias Cabarcos
Dr.-Ing. Philipp Terhörst
Date: July 17, 2024

Abstract

The prevalence of smurfing and boosting in online games, wherein skilled players either play on lower-skilled accounts or artificially inflate account ratings, undermines fair matchmaking and diminishes player enjoyment. Since players do not spent money on games which they do not enjoy, developers are incentivized to stop smurfs and boosters. This thesis addresses the need for effective detection of such practices, focusing specifically on the game Counter-Strike: Global Offensive (CS:GO). By utilizing a unique data set derived from publicly accessible CS:GO demos, this research aims to assess the effectiveness of using mouse movement data to verify and identify player identities.

Two research questions guide this study: "How effective are mouse movements for verifying player identities in the game CS:GO?" and "How effective are mouse movements for identifying player identities in the game CS:GO?". To answer these, Deep Learning (DL) models were developed and trained using mouse movement data extracted from professional CS:GO matches which contain 69 years of continuous mouse movement. The verification model achieved an average F1 score of up to 0.957 ($\pm$ 0.026) and an average Equal Error Rate (EER) of 0.036 ($\pm$ 0.020). The identification model achieved an average F1 score of up to 0.941 ($\pm$ 0.051) and an average EER of 0.009 ($\pm$ 0.009). When sequences from a single player and match were grouped, the models achieved an average F1 score of 0.990 ($\pm$ 0.045) and an average EER of 0.001 ($\pm$ 0.004).

Although the proposed models demonstrate significant improvements over the current state of research, further refinement is essential for deploying them in real-world applications to autonomously detect smurfs and boosters. Nevertheless, these models could serve as an initial filter to flag suspicious activity for manual review. The approach and data set of this study can be utilized for future research to advance player recognition techniques using behavioral biometrics, potentially extending beyond CS:GO to other games and applications.

Official Declaration

I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Date	Lars Gansel

Contents

1 Introduction

Why should you play a game when you do not enjoy playing it? Usually, individuals play games to have fun or to achieve a sense of pride and accomplishment. This leisure activity was pursued by 2.4 billion players worldwide in 2023 [8], of which 24 million players played the game Counter-Strike: Global Offensive (CS:GO) monthly [12]. However, sometimes players compete against other players, who ruin their fun by smurfing or boosting [42]. Smurfing is when a skilled player plays on accounts that the matchmaking system predicts as low-skilled. The skilled smurf plays against low-skilled and new players. New players are typically beaten by the smurf in most scenarios. Boosting is a practice in which a skilled player increases the rating of a lower-skilled player's account by playing on it. Furthermore, boosting undermines the selection criterion if gaming tournament organizers select players based on the account's rating. Participating in tournaments generates fame and money for these players. Therefore, tournament organizers should check whether the player's account rating matches the player's skill.

Both of the before mentioned practices undermine matchmaking systems, which aim to establish a balanced match of players of similar skills. Moreover, they reduce the fun of playing the game for players worldwide [42]. If smurfing and boosting become too big of an issue, new and existing players will stop playing the game. Thus, game developers and publishers have fewer potential customers. Many modern online games are often free to start but earn money through in-game transactions such as different looks for the character or equipment as well as other bonuses. Since in-game transactions can be conducted multiple times, they can generate more money than one-time purchases, like paying once for the game itself to play [18]. Despite the financial incentive for developers and publishers to mitigate these issues, only a few cases of effective countermeasures are known [5, 21]. When smurfs or boosters are detected, their accounts are typically banned. However, creating a new account and starting over again is usually a straightforward procedure. Some developers implement a hardware ban, where the hardware ids are tracked and blocked from playing again. Hardware bans can be circumvented by changing the hardware ids or using Virtual Machines (VMs) [24]. Intrinsic human behavior cannot be changed as quickly as hardware ids [45]. Therefore, if a highly accurate recognition system would exist for this, it would be an excellent candidate to recognize individuals by their behavior and ban them.

Although the problem of smurfing is already known to players [42] and developers [5, 21], detecting boosters or smurfs is sparsely covered by scientific research.

Research regarding the recognition of individuals via mouse movement exists [35]. They usually use a verification approach, where one binary classification model is trained per individual. The model shall predict whether the sample belongs to an individual. Although this approach is feasible for verifying a small number of individuals, applying this to a large set is inefficient because each player requires one model to be recognized [52]. To mitigate this, an identification approach could be applied where one model is trained to predict one of multiple individuals through their sample. However, this approach is more challenging since more individuals must be distinguished in one go, and therefore, more nuanced differences must be detected. In addition to the scaling problem, the related work usually involves a controlled environment in which individuals have to use a provided computer setup in a laboratory. This setup changes the usual way an individual interacts with a computer. Furthermore, artificially created tasks like drawing shapes are often used to create data sets. These tasks do not reflect the real-world usage of a mouse during a normal interaction with a computer. Since collecting data sets is laborious, they are often relatively small regarding the number of samples and individuals [35].

This thesis aims to recognize players via their mouse movement. To achieve this, server-side recordings, so-called demos, from the multiplayer video game CS:GO are exploited. The recordings of professional players since 2012 are publicly accessible through the website Half-Life Television (HLTV), and each demo is available as a download. Mouse movements can then be extracted from the demos to compose a new data set. Since the mouse movement is extracted using a real-world application (by usually playing at home) this represents an uncontrolled environment. Although only the game CS:GO is chosen for this thesis, the work can be extended to other games and applications if the data collection methods are adapted. The complete data set contains 153,189 matches played by ten professional players per match. The estimated time of extracted mouse movement is over 69 years. Due to the size of the data set, it was not downloaded and measured in its entirety. The estimations are instead based on the three years from 2020 to 2022. The total playtime is estimated based on the playtime per match from these three years. This data was then utilized to create a Machine Learning (ML) pipeline to train models to verify if the account owner plays on the account. Moreover, another model was trained to identify which player played and produced this sample. The models achieve an F1 score of up to 0.990 ($\pm$ 0.045). In contrast to most of the related work, which extracts hand-crafted features from mouse movement, this thesis focuses on training Deep Learning (DL) models. Not only is defining and extracting hand-crafted features laborious, but more effective patterns for player recognition may not be found. However, DL and especially Convolutional Neural Networks (CNNs) are known for learning how to extract features [52].

To fill the aforementioned gap in the current state of research, the following two research questions are formulated:

RQ1 “How effective are mouse movements for verifying player identities in the game CS:GO?”

RQ2 “How effective are mouse movements for identifying player identities in the game CS:GO?”

To be able to answer the research questions, the scope of the newly created data set is limited. Only professional players of the game CS:GO which play for the best 20 teams are selected for the analysis and training of ML models. Secondly, the period of analysis is narrowed down from 01.01.2020 to 31.12.2022.

The following thesis is organized as follows: Chapter 2 gives a brief overview of the game CS:GO, explains biometrics, DL, and the current state of research. Chapter 3 explains how the demos are acquired, how the mouse movement is extracted, and how the ML pipeline is designed and implemented. The mentioned data set is described in Chapter 4. The implementation is then evaluated in Chapter 5 based on different experiments. These results are discussed in Chapter 6 and also a potential real-world deployment of the proposed system as well as its limitations. This thesis is concluded in Chapter 7, which recapitulates and enumerates various further research as well as experiments based on this thesis.

2 Background

The following chapter covers all the required information to comprehend this thesis. It starts with an introduction to the game Counter-Strike: Global Offensive (CS:GO) to understand where the data for the Machine Learning (ML) models originate from. Moreover, potential differences in the gameplay and thus the mouse movement are explained. Section 2.2 introduces important biometric characteristics and recognition modes. The following Section 2.3 covers the basics of ML and Deep Learning (DL) to understand how a model can learn from data, for example, obtained from CS:GO. The chapter also describes commonly used metrics to evaluate models. The last Section 2.4 provides a brief overview of published papers on mouse movement and player recognition.

2.1 Counter-Strike: Global Offensive (CS:GO)

CS:GO is a video game of the genre First Person Shooter (FPS) and was developed by Valve Corporation and Hidden Path Entertainment [14]. It is the fourth iteration in the Counter-Strike (CS) series. The first one originated as a modification for Valve's game Half-Life. During the preparation of this thesis, a new iteration of CS, called Counter-Strike 2 (CS2), was released on the 27th September 2023 and replaced CS:GO [14]. Although a new iteration exists, the thesis focuses only on the game CS:GO. The main differences between these two versions are mainly improved visuals and the underlying code base. The gameplay is almost identical [13]. Since CS:GO existed for several years, the amount of played CS:GO matches is far higher than that of CS2 matches. A high amount of data in the form of matches is crucial for ML to detect patterns in mouse movement. Typically, a player registers one account on Valves's gaming platform Steam and then plays the game on this personal account.

2.1.1 Gameplay

The game offers multiple modes but the most played one is "Competitive". In this game mode, two teams of five players compete against each other, where one team plays as Terrorists (Ts) and the other as Counter-Terrorists (CTs). Although the name Competitive may imply professional scenarios, the game mode is also played by regular or casual players with varying skill sets. After half-time, the teams switch

Figure 2.1: Layout of the map Dust II. The green zones are the starting zones for Counter-Terrorists (CTs) (top) and Terrorists (Ts) (bottom). The red zones are the two bomb spots where the bomb is plantable [16].

sides, and the requirements to win a round change. Ts win a round if they plant the bomb at one of two predefined spots (so-called sites) and defend it until it explodes. The bomb spots are typically at the edge of the map and enclosed with red boxes (see Figure 2.1). Currently, the bomb takes 40 s from planting until the explosion. In contrast to this way of winning a round, CTs need to prevent Ts from planting the bomb or defuse it before it explodes if it was already planted. Both teams can win if they eliminate the other five players. If the round time, currently set at 115 s, is over, CTs automatically win [15].

To win a match, a team has to win more than half of the available rounds. A game usually has 30 available rounds; therefore, a team needs to win 16 rounds to win the match. If both teams win half of the rounds, the match can end in a draw, or overtime starts [15]. Overtime usually consists of six additional rounds. To ensure a fair match, the sides are switched after three rounds.

At the beginning of each round, all players are placed in the starting areas, usually called spawns. These are depicted as the green boxes in Figure 2.1, which are typically at opposite sites of a map. After the players spawn, the buy phase starts. Each player can buy equipment such as weapons or utilities during this phase but they cannot move [15]. Then, the combat and round time starts. Usually, two CTs defend each site and one is defending the middle area. Ts typically roam around the map to find ways to enter a site and plant the bomb. After any winning condition is fulfilled, the next round starts [13].

The economic system in CS:GO is essential to understand in order to win a round and is briefly summarized in the following. Each player earns money throughout a match to buy items. During regular match time, each player starts a half with $800. In case of overtime, the player starts with $10,000 per half. All players of the team who won the last round receive $3,250 and each player of the losing team earns between $1,400 and $3,400. The money the losing team receives is increasing with every consecutive lost round. In addition to that, each player can earn money individually if they eliminate an enemy, which is usually rewarded with $300. The weapon will be in the player's inventory even if a new round starts. The current weapon can be lost if a new gun is bought or the player dies. To have a better relative estimation of the money's value in the game two examples are mentioned in the following: the typical weapon bought by Ts is the AK-47, which costs $2,700, and the usual weapon bought by CTs is the M4A4 which costs $3,100. In addition to the primary weapon, a Kevlar West is generally bought to reduce the incoming damage from bullets, which costs $650, and is combined with a helmet for $1000. If a character spawns with 100 Health Points (HP) and wears armor like a Kevlar West, they may survive five instead of four hits. This may be the deciding factor to win a duel against an enemy. To further improve the basic equipment, grenades are typically bought to blind enemies with a flashbang for $200 or obstruct their vision with a smoke grenade for $300. Therefore, it is crucial to manage the economic system and plan what and when to buy to not run out of money [16].

If a player or team decides not to buy expensive equipment, it is called an economy round. In this case, only a cheap pistol ($300 - $700) is bought. Thanks to the economy round, the current amount of money is saved and will be available in the next round in addition to the round-ending reward. However, this will probably result in a lost round for the eco-ing team. Another option to save money is to decide mid-round to avoid attacking and to hide instead. Thus, a hiding player may not die and keep their weapon. The saved weapon then does not have to be repurchased, and less money needs to be spent in the following round.

There are several weapons and categories in all CS games. The weapon choice mostly depends on the available money, the usage scenario like short or long ranges, or the personal preference. Due to the differences in weapons, a player might need to adapt to each gun slightly to deal the most possible damage to the enemy. The weapon choice influences the character's movement speed and the required mouse and keyboard usage while playing. All of these factors influence how a player needs to play to win, their personal play style, and their decision-making.

2.1.2 Rating

One new aspect CS:GO introduced is a matchmaking system for the Competitive game mode [14]. In the older game iterations, the player needed to find and manually join a server through a server browser. Then, players with a variety of game knowledge

and skills played together. Players with much skill typically dominated the match. This negatively impacted the experience of less seasoned players because they lack the practice that veterans of the game have acquired. Therefore, CS:GO started assigning a rating to each player's account and automatically matches accounts with a similar rating. This change makes the game more competitive and appealing for players of all skill groups. The rating is influenced by winning and losing matches. The developers keep detailed influencing factors private to mitigate exploitation. The exact rating is not directly displayed to each player. Only the current rating/skill group is shown out of 18 existing options. As a result of the rating and matchmaking, a player is expected to win half of the games on average if a player's skill matches the rating of the player's account. If a player decides to take longer breaks from the game, their rating might remain at a higher value while the player may have lost some of their skill due to a lack of practice. As a result, the player's account would start losing its rating upon returning to playing the game and likely losing their next matches. Upon improving his or her playing skills, the account rating would then be elevated again, if the player happens to win at a frequent rate. This leads to the player's account climbing the ladder of possible skill groups until they win half of the matches.

Smurfing and boosting are practices that circumvent the skill-rating equilibrium. They are prevalent in most competitive games and are not only present to CS:GO [42]. It is easier to win if a good player plays against a bad player, and winning a match can often be fun. Due to the rating system, players with equal skill are matched. A good player can create a new account to play on whereas the newly created account starts off with a low rating. Therefore, the good player plays on the account with a low rating against other accounts in that same skill group. However, most players in this skill group are not smurfing and actually belong in this bracket due to their lack of experience. Then, the good player easily wins, assumingly having more enjoyment and ruining the fun for the opposing team. Another reason is that a player wants to avoid playing against serious competition and possibly just relax after work or school, without having to put much effort into their gaming experience. When a good player plays on a lower-rated account, it is called smurfing [50]. Smurfing thus requires to have access to an account that is rated far lower than the player's original account, and the rather frequent occurrence of such players at very low ranks is commonly known among the player base of the game.

Boosting, on the other hand, is when a bad player who plays on a low-rated account hires a good player to increase the account's rating. This can be through playing on the low-rated account or playing together. Then, the inexperienced player could earn in-game benefits, which are tied to the account's rating, like rare badges [27]. Another reason for boosting is that a high rating implies high skill; therefore, the player is perceived as more worthy in the gaming scene. Furthermore, high-rated accounts may be invited to tournaments that yield prize money [50]. Since both practices harm balance and through this fun the fun of playing for the majority of people, they are often prohibited by the game's terms of use. However, it is not trivial

to detect these. South Korea criminalizes boosting with a fine of up to 20,000,000₩ (~13,340€) or two years in prison. In 2021, Conroy et al. estimate the annual revenue of commercial boosting at around 112,000,000€ for the games League of Legends (LoL), Dota 2, and Overwatch [10].

2.1.3 Demos

Each game server stores a so-called demo of a match, which contains all game-related events that happened while playing on the server. All events are stored to recreate the recorded match and review it at a later point. These are events like a change in a character's current position and viewing direction, which weapon is fired and how much HP a character currently has.

A tick is a discrete-time unit wherein a game server processes all inputs, updates the game state, and publishes it to all connected players. Professional players usually play on servers that have a tick rate of 128 Hz. The snapshot rate at which the events are stored is often equal to the tick rate, but sometimes, not every tick is stored in order to reduce memory usage. Compared to professional matches, the tick rate of the servers used in the Competitive game mode for regular players is 64 Hz, and the snapshot rate is 16 Hz. Higher tick and snapshot rates lead to more accurate mouse trajectories due to the higher sample frequency. All data is stored in a binary file.

2.1.4 eSports

The eSport scene is on the rise, with an increasing number of players aspiring to compete at a professional level. The allure of witnessing these skilled players in action, battling it out in high-stakes tournaments with big prize pools, is a source of excitement for many viewers. CS:GO has been a major eSport title for several years with a prize money reaching $2,000,000 in a single tournament [28]. Prize money and other financial aspects increase players' willingness to make mastering the game their job. To support players participate in eSports, organizations create and manage teams just as in traditional sports. The teams compete in tournaments like in regular sports tournaments. If two teams play against each other, they usually play multiple matches in one session, called a series. A team wins a series if more than half of the maximum number of matches in this series are won. In the case of a Best Of (BO)3, two out of the three matches must be won. Other typical formats are BO1 and BO5. In this regard, eSports are like regular sports, like soccer or football. Another analogy to traditional sports is the problem of cheating and unfair behavior. While drugs can be used in traditional sports to enhance physical capabilities, in eSports programs or cheats can be installed to highlight other enemies or automatically aim at enemies.

Boosting in eSports is also a potential practice to earn more prize money or qualify for a tournament. For example, an excellent player could play instead of a lesser skilled player yet on their account to enable tournament participation or higher prize money. Although boosting is a well-known practice for casual players, only some partially related eSport boosting cases involving money are known to the public. The professional player XiaoWeiXiao confessed to boosting a friend's account to the second-highest skill group for $1,300. Since this is prohibited, he was suspended from the professional team he was playing for [56].

2.1.5 Half-Life Television (HLTV)

Half-Life Television (HLTV) is a news coverage website about professional CS (eSports) and lists series results, articles, interviews, and news. They provide the results of all series played by (semi)professional players and teams. Most of the time, they also provide a demo to download and rewatch the match. Additionally, they offer a rating metric for each player and team and, therefore, create a global leaderboard. This rating is different from the ones in the game CS:GO. The series coverage started in August 2012, the release month of CS:GO [28]. HLTV provides the biometric data through demos for a newly created database to recognize players via their biometric characteristics.

2.2 Biometrics

Biometric characteristics are behavioral or physiological traits that can be reliably extracted from individuals for biometric recognition [32]. Fingerprints have been used as a physiological characteristic for several years in the area of forensics. There, it is used to verify if an individual touched an object with his/her finger. This recognition can then help to find criminals [33]. Nowadays, individuals often use their face as a biometric characteristic to unlock their smartphone instead of using some passphrase [46]. Commonly used behavioral characteristics are handwritten signatures, gait, or computer interaction via mouse or keyboard. Authentication with biometric characteristics yields the potential to be more secure than traditional authentication with a username and password. Individuals tend to choose simple passwords with a small entropy whereas biometrics can yield a high entropy. A higher average entropy indicates that the recognition method might be more secure [45]. However, not every biometric characteristic is more secure and usable than a username and password combination. The implementations must be secure as well.

Seven properties are important to classify how good a biometric will be for recognizing and distinguishing individuals. Not every characteristic will fulfill every property,

but if most are (partly) fulfilled, it might be enough to be employed for a recognition method [33].

Universality Each individual should possess this characteristic.

Uniqueness The characteristic should have enough entropy and should differ from other individuals.

Permanence The characteristic should not change or change slowly.

Measurability The characteristic should be easily sampleable.

Performance The characteristic should be computationally cheap to sample and compare to other samples.

Acceptability Individuals are willing to sample and share their characteristics with a biometric system.

Circumvention The biometric system should be robust against bypasses or fake characteristics.

For example, fingerprints fulfill these properties entirely or at least partially. A fingerprint could changed due to aging or be lost due to an accident, which is nevertheless rare. Other examples are keystrokes and mouse movement. These are not well known to be unique, and analyzing a recording of them is not as computationally cheap as fingerprints. It might also not be helpful to recognize an individual because they must first use a device for several seconds. However, they can be used to ensure that the individual who interacts with the computer still belongs to the claimed identity. Permanence is assumed to be hard to achieve because individuals type in a different rhythm and speed depending on their physical and psychological state [33].

Biometric recognition distinguishes between verification and identification. In the case of verification, an individual claims an identity. Then, the individual has to provide a sample of a biometric characteristic that proves that the individual matches the claimed identity. The system must answer the question: “Is this individual who they claim to be?”. The identity that can be claimed is already registered in the biometric system. In contrast, the provided sample might not belong to an already registered individual in the system. Examples of such biometric systems for recognition are unlocking a single account smartphone with a fingerprint or face [33, 45].

In the case of identification, an individual does not claim any identity. The system has to find the identity of the provided sample on its own. The system needs to answer the question: “Who is this individual?”. In a perfect scenario, only one identity would be found that matches the provided sample. Examples are gaining access to a building while providing biometric characteristics like an iris scan at the entrance and the biometric system finds the corresponding identity without claiming it beforehand [33, 45].

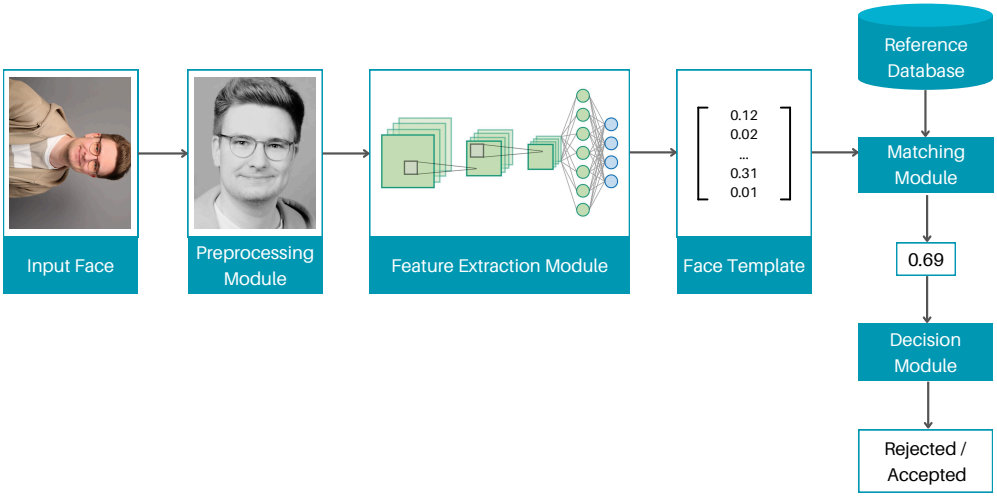

Figure 2.2: Illustration of a biometric system for a face recognition task. A camera (sensor) captured an image which is used to find an enrolled identity with a similar face template.

The provided sample must not belong to an identity registered in the database. Therefore, it is necessary to have the possibility to reject a sample or unknown identity. This case is called open-set recognition because not all provided samples and identities are registered. If all identities, that try to authenticate themselves, are registered, it is called closed-set recognition.

Biometric systems, as the system depicted in Figure 2.2, propose a modular and standardized way to leverage biometric characteristic samples to verify or identify an identity. Typically, a biometric system consists of a sensor, a preprocessing module, a feature extraction module, a matcher module, and a decision module. Furthermore, this system includes a reference database for computed templates from previously registered individuals. The complete system overview is illustrated in Figure 2.2 [45]. The sensor captures biometric characteristics via fingerprint sensors, cameras, or computer mice. This sample is then preprocessed; for faces and fingerprints, this can be centering and aligning the sample. Removing the color channels from images or scaling the values is also a common practice. Additionally, this module can be enhanced to detect if an attacker tries to fool the system and decides to reject the sample. Then, the features are extracted from the preprocessed sample. These features can be manually defined (hand-crafted) or defined and extracted via ML models. The ML models need to be trained first. One way to train the feature extractor is to add a classification layer on top of the feature extraction layers for training only. While being deployed to extract features, the classification layer can

be stripped away from the trained model. Feature extraction is usually a form of compression because the feature template, often called embeddings, uses less data than an image or a time series signal. When an identity is enrolled, the template is stored in the template database to be accessed for the subsequent recognition requests. During verification, the extracted template is compared to the stored template, which belongs to the claimed identity via the matcher module. For identification, the extracted template is compared to the stored templates from all identities. The comparison can be performed by a distance or similarity function like the Euclidean distance or cosine similarity. The score of the distance function should be low and the score of the similarity function should be high, to be considered as belonging to the same identity. Another approach is to train a model which compares both templates. The score computed by the matcher is then forwarded to the decision module, which concludes whether the provided sample is predicted to be a registered identity [33].

Since storing the extracted features from a sample is efficient due to extracting features only once, this can also be a privacy risk if they get stolen [43]. One way to mitigate this issue is not to store the templates directly. In this case, one ML model can be trained to extract the features and decide whether the sample belongs to an identity. Then, no templates are stored. However, then the flexibility of the modular approach is removed and retraining is required if a new individual is enrolled.

2.3 Deep Learning (DL)

DL describes a subset of methods in the area of ML [34, 40]. In ML, data is utilized to create and train models that give predictions based on the learned probabilities. In contrast, in traditional programming, rules are explicitly programmed to transform an input into an output. Supervised ML models are provided with sample inputs and the corresponding outputs, and they determine the relationship between these values [40]. In unsupervised ML tasks, a model is provided sample inputs but no corresponding outputs. For example, a model needs to cluster data and is required to determine the clusters itself [34]. DL models stand out for their potentially good performance. However, they require more data to achieve such performances compared to more traditional or shallow models like K-Nearest Neighbours (KNNs), Support Vector Machines (SVMs), or Decision Trees (DTs). A shallow model often requires hand-crafted features to find relationships when dealing with high dimensional input data. DL models can find these features within raw data; therefore, no hand-crafted features are required [52]. A DL model is characterized by its structure, which includes one input, output, and multiple layers in between. These intermediate layers, often called hidden layers, make the model deep. The good performance results from the high number of parameters. Since the parameters are responsible for forwarding the input, they need to be trained. The more parameters a model consists of, the more

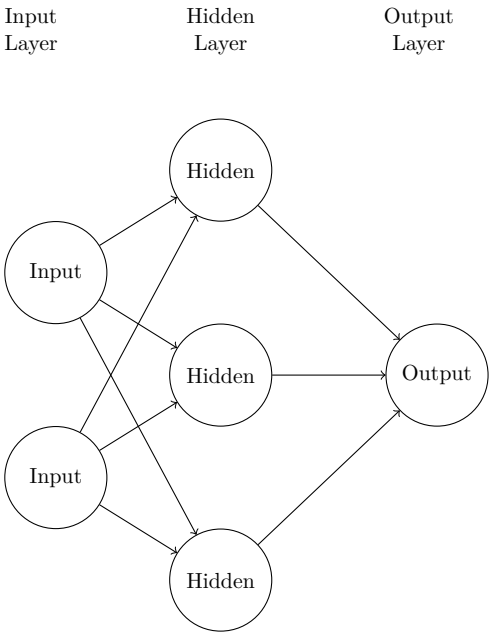

Figure 2.3: Example of a fully-connected Feedforward Neural Network (FNN) which consists of three layers. The model consists of two input, three hidden, and one output neuron.

training and training data are required. After the successful training, the model can correctly predict the outputs of unseen inputs [34, 52]. Typical tasks for DL models are regression (weather forecasting [40]), classification (cats versus dogs [52]), and recreating content (Large Language Model (LLM) [52]).

2.3.1 Artificial Neural Network (ANN)

Artificial Neural Networks (ANNs) are the foundation of DL techniques and are inspired by brain neurons. In the biological context, neurons receive signals and then transmit or act upon them. A neuron activates when the received signals are above a certain threshold and transmits a new signal to the following neurons. In the long run, a signal from the brain can indicate that the muscles in the arm should move. The functioning of a brain's connected neurons is analog to how ANNs work [34].

In the case of ANNs, neurons also receive input, process it, and transmit the processed signal to the following neurons as illustrated in Figure 2.3. The first layer receives the input data, which in this example consists of two numerical values. Each input neuron in the neural network receives a value, which it then forwards to the connected neuron. Per connection, the value is multiplied by a weight and a bias is added. Weights and biases are so-called parameters. The sum of all received and multiplied values can be

directly forwarded or passed into a so-called activation function before forwarding [17]. This procedure is repeated until the output layer is reached [34, 52]. The parameters in the neural network are not static but learned while the model is trained. This dynamic nature of the parameters allows the model to adapt and improve while training. An algorithm called backpropagation is employed to update the parameters. This uses the prediction error defined by a loss function and distributes the approximate error on all parameters [34, 52].

The previously described and illustrated network is a FNN and is usually the type of architecture meant when ANNs are discussed. As the name suggests, the data flow of this network architecture is unidirectional, from input to output. In contrast to FNNs, Recurrent Neural Networks (RNNs) are characterized by a bidirectional data flow. RNNs can also consist of loops that enable the network to have a kind of internal memory or state. Another often-used architecture is Convolutional Neural Networks (CNNs), which is described in more detail in Section 2.3.2 [52]. This is only a partial list of architectures to report a brief overview [40].

Activation functions enable the model to learn non-linear relationships and thus have an influence on the predictions. The functions themselves are usually non-linear. An activation function determines if the received numerical value is forwarded and at which intensity. Sigmoid, Tanh, Rectified Linear Unit (ReLU), and Softmax are commonly used activation functions [17, 34]. The activation functions are illustrated in Figure 2.4. The choice of the function influences the duration until a model converges and correctly predicts the outputs. Tanh and Sigmoid functions have the problem of saturation for high or low values [17]. In the case of a value of 5 or 6, there is almost no difference $\tanh(5) - \tanh(4) = 0.0005$ compared to the ReLU activation function $\text{relu}(5) - \text{relu}(4) = 1$. ReLU is fast to compute and derive. It returns the value if it is positive, but it returns 0 for all negative values, which is bad for backpropagation, which uses derivation to update the parameters. However, the derivation of a constant is 0 which makes learning impossible. A parametrized ReLU can be used, which mitigates the problem regarding the 0 derivation [26]. This version returns a linear function for values below 0, scaled by the scaling factor α. If $\alpha = 0$, the Parametric Rectified Linear Unit (PReLU) becomes the normal ReLU, and if $\alpha = 0.01$, it is called leaky ReLU. Another variant is the Exponential Linear Unit (ELU), which is more robust to noise [9].

One important activation function for predicting categories is called Softmax. This function transforms a vector with arbitrary numbers to a vector with probabilities. Therefore, the sum of all entries in the vector is one, and all entries range from 0 to 1. The advantage of this transformation is that it is easier to interpret the prediction. In turn, this activation function is often used in the last layer to have an interpretable prediction [9, 17, 26]. Softmax is often combined with one-hot encoding. Regarding multi-class classification tasks, it is challenging to train a model to predict a single integer that refers to a class id because the model has to learn an order or priority, which might not be the case for classes. An example is that while distinguishing cats,

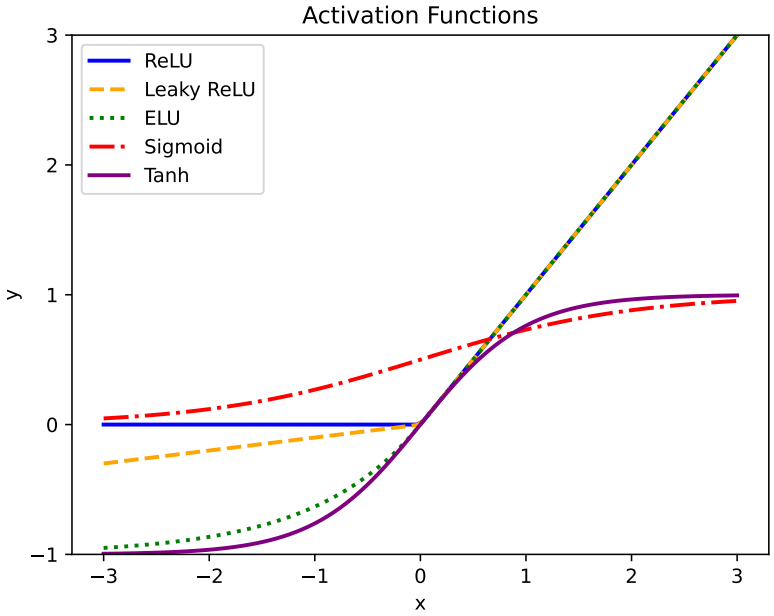

Figure 2.4: A comparison of activation functions, highlighting their distinct characteristics.

dogs, and dolphins, there is no order between these categories. The average of a dog and a dolphin is not a cat. Therefore, one-hot encoding is applied. Here, a vector with a length of the classes is predicted where each entry corresponds to a class. Hence, no order has to be learned and the entry with the highest value might be chosen as the predicted class. Combined with Softmax, the resulting vector denotes the probability that the sample belongs to a specific class.

During the training phase, the prediction error is used to update the parameters via backpropagation [34]. This is commonly referred to as learning. The parameters are initialized with a random but non-zero value. A sample is passed into the model in the forward pass phase and the output is predicted. Then, the loss is computed via a function based on the actual value and the predicted one. The gradient of all connections between the neurons is calculated in the following backward pass phase. The gradient is efficiently calculated via the chain rule to cache performed derivations and reuse them if possible. The gradient can determine the activity of the neurons, and based on the activity, the parameters can be updated in combination with the loss. Updating the parameters after one sample is usually not done due to inefficiency. Therefore, an update is performed after a batch of multiple samples is processed. A batch is a subset of all available training data used for training. It is called an epoch if all batches from the available training data are processed. A model is typically trained for multiple epochs [34, 52].

One common problem of ML is overfitting. Overfitting occurs when a model's performance keeps increasing on the training data but worsens on unseen testing data. In this case, the model only learns the relationship of the training data, which does not help predict unseen data. Therefore, techniques are used to measure and reduce overfitting to verify the model's performance on unseen data. One technique to measure the overfitting is using separate data sets for training and testing the model. The model is trained on the training data set and the parameters are updated. Testing data measures the model's performance on unseen data and indicates how well it will perform when deployed. It would be optimal if the model performs equally well on both data sets [34]. Often, a third data set called validation is used. This data set also does not share any samples with the other two. The validation data set is used to determine the hyperparameters of a model. Hyperparameters are properties like the learning rate or how many samples are used per batch. Although the hyperparameters are not directly used for training, tuning them influences the model's performance.

Cross-validation can be employed to increase confidence that a model generalizes well. It splits the whole data set into k folds. Then, k models will be trained on k-1 folds, but for each model, another fold is held out for training and is only used for testing. Afterwards, the overall model performance can be reported using average and standard deviation.

Another split regarding data is the session split. The session split ensures that the samples in a data set are not shared across sessions. If a sample belongs to session A and is added to the training set, all other samples from session A must not be in the validation or testing set. This split is enforced so the samples do not mix session-specific data, like emotional state. This increases the robustness of a model so that it does not learn based on session-specific information.

If the models perform better on the training data set than the testing data set, techniques must be employed to reduce overfitting. Commonly used regularization techniques are dropout, L2 regularization, and data augmentation [41]. Dropout layers randomly deactivate some neurons during training based on a given probability. During testing or inference, all neurons are active. If some neurons are deactivated, the other neurons have to compensate for the missing ones; meaning that the neurons' usage is distributed more equally. Moreover, redundancy is encouraged and the risk of overfitting is reduced. Using dropout layers increases the time until the model generalizes [41]. L2 regularization penalizes the model for having large parameters ω. In addition to the loss from prediction errors, a penalty for large parameters is added as the sum of all squared parameters. Regularization will result in fewer large parameters and probably fewer tiny parameters [31, 41]. In terms of performance, the model's complexity and variance are reduced. Due to a reduction in variance, the bias might increase, which is called the bias-variance

tradeoff [31].

$$\mathcal{L}_{\text{regularization}}(\mathbf{w}) = \mathcal{L}_{\text{prediction}} + \lambda \sum_{i=1}^{n} w_i^2$$

Despite the good performance of ANNs for different tasks, some drawbacks exist. The more complex a relationship is and therefore the ANN, the higher the hardware requirements tend to be. Training a model requires many computations, which requires good Central Processing Units (CPUs) and Graphics Processing Units (GPUs). Such hardware is not only expensive but also requires a lot of energy [44]. Another critical point is a high data transfer speed and memory capacity because the more complex a model becomes, the more data is required to train it. This data must be stored on disk and loaded into Random Access Memory (RAM). The transfer from disk to RAM is slow and should be minimized to reduce the training time of a model.

Another typical problem is that models are often referred to as black boxes because they usually do not offer a reason for the prediction, respectively, their decision might be incomprehensible. This is essential information when models are deployed for real-world tasks to support decision-making. For example, ANNs determine if or how much loan an individual can receive from a bank. The model will not reason and the individual will not know why it is not loan-worthy [30, 40]. The research area of Explainable Artificial Intelligence (XAI) is trying to mitigate this issue while trying to open the black box [7, 30].

2.3.2 Convolutional Neural Network (CNN)

CNNs are particularly useful when dealing with large inputs due to their efficiency in parameter utilization. As a result, CNNs find extensive use in processing data types such as images and audio, making them a crucial tool in various practical applications [29, 40]. A CNN learns to extract feature maps out of the input and can also semantically compress this data to reduce the number of connections [7]. Therefore, fewer parameters need to be updated, which reduces the time required to train a model. In the case of images, the extracted feature maps could include all edges in the images and various other features. The low-level feature maps can then be used by another CNN layer to combine these into higher-level features. In an image of a human face, the first features could be contours, followed by higher-level features like the mouth or nose. The essential part is that each filter is used on every input value instead of one neuron per input value. This parameter-sharing mechanism in CNNs is a critical factor in reducing the number of parameters that have to be updated [29, 52].

A filter is a sliding window over the input data and computes a value based on the current window. Filters are also known as kernels or feature detectors [29]. Each filter gets randomly initialized and then updated during training. While the

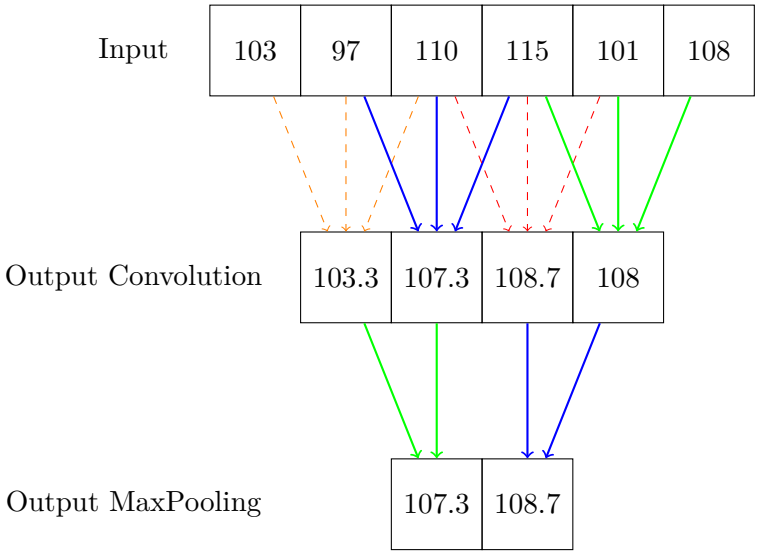

Figure 2.5: Example of a one-dimensional CNN layer followed by a one-dimensional max pooling layer. The CNN layer uses an averaging filter with a filter size of 3 and a stride of 1. The max pooling layer has a pool size of 2 and a stride of 2. Both layers omit padding.

model learns, the filters get updated to capture the features better. Typically, after the feature extraction with CNN layers is performed, other architectures like fully connected ANN layers follow. The following layers make predictions based on the captured features and, in the example above, determine to which human the face belongs [29].

Figure 2.5 visualizes the process for a one-dimensional CNN layer with the average filter. The filter size is 3. It sums up all values, divides them by 3, and stores the result in the feature map. Then, the filter is moved to the right based on the stride of 1. In multi-dimensional inputs, the stride also determines the movement of the filter in all dimensions. A pooling layer is often applied after a CNN layer. The max pooling layer is used in the example but other pooling layers such as average pooling or min pooling, also exist. These reduce the amount of inputs for the next layer. In both layers, no padding is used. The padding adds artificial values to the border of a layer's input to keep the input's size for the output. Some examples of padding are zero padding, which adds zeros to the input border, and the same padding, which duplicates the value at the filter border. If no padding is used, the output of a one-dimensional CNN layer is $n_{\text{inputs}} - w_{\text{filter}} + 1$ when a stride of 1 is used.

2.3.3 Performance Metrics

Functions are necessary to quantify a model's performance. These functions are called performance metrics or short metrics. These metrics enable the evaluation of models and determine if a model successfully solves a given task.

Table 2.6: Confusion matrix for a model which classifies if a sample belongs to player A or not.

		Predicted Class ($\hat{y}$)	
		Player A	$\overline{\textbf{Player A}}$
Actual Class (y)	**Player A**	True Positive (TP)	False Negative (FN)
	$\overline{\textbf{Player A}}$	False Positive (FP)	True Negative (TN)

To introduce the relevant terms to discuss further metrics, the confusion matrix in Table 2.6 is presented. In a binary classification, a positive prediction describes that a model predicts that the sample belongs to the target class. This prediction is called True Positive (TP) if the sample belongs to the target class and False Positive (FP) if not. A negative prediction describes that the model predicts that the sample does not belong to the target class. This prediction is called True Negative (TN) if the sample does not belong to the target class and False Negative (FN) if not [41].

In ML, accuracy is a common and intuitive metric. Accuracy is defined by the percentage of the model's correct predictions. Therefore, an accuracy of 1 would be the best case and 0 the worst. The metric still holds well if the data set contains multiple classes that should be distinguished and all classes occur to the same amount. However, the accuracy is a bad metric if the data set is not balanced. If the data set consists of 90% out of one class, the model always predicts this class and achieves an accuracy of 90%, but the model has not learned the relationship. Consequently, it is essential to ensure the data set is balanced, or that weighting is used [41]. A variation of accuracy is RankN accuracy. RankN accuracy measures the probability that the correct answer is among the top N predictions made by the model. For example, in a classification task with Rank1 accuracy, the metric is the same as the accuracy defined before. Rank5 accuracy reports the percentage of times the correct answer is within the top five predictions. Even if the top prediction is wrong, the actual class might be within the top five predictions, making the model's output helpful.

Additionally to accuracy, precision, and recall can be computed and combined into a new metric called the F1 score. The F1 score is less susceptible to an unbalanced data set than accuracy. Precision is how often the model correctly predicts a TP out of all positive predictions. Recall is defined as how often the model predicts a TP

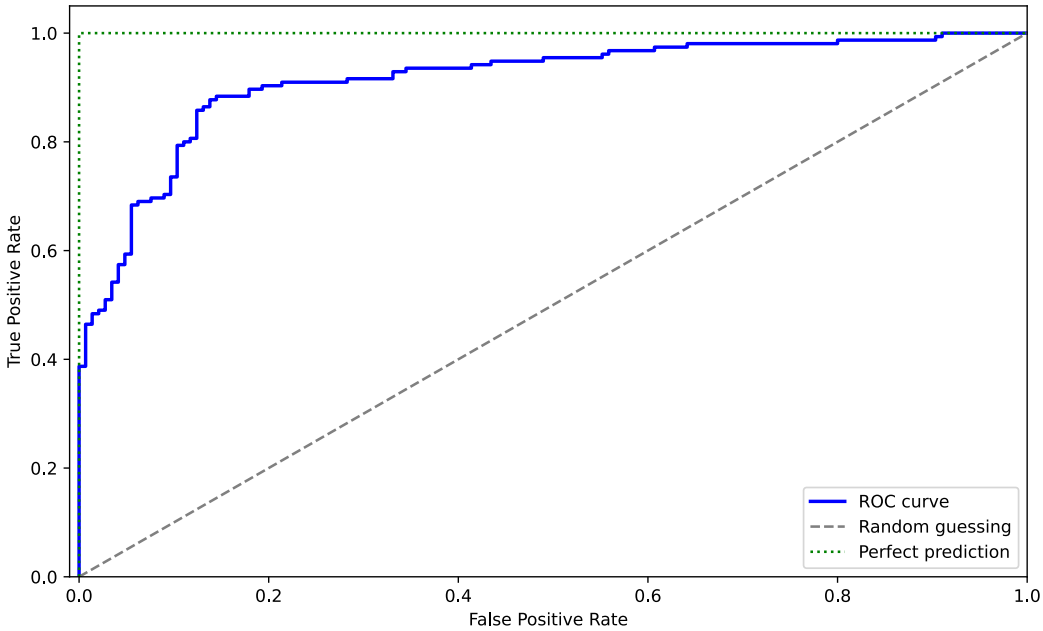

Figure 2.7: Exemplary Receiver Operating Characteristic (ROC) curve. The blue graph is an example of a possible curve when predicting a class. The green dotted line represents a curve based on a perfect prediction. The dashed line reflects random guessing.

out of all actual positive samples. The F1 score is the harmonic mean of recall and precision. For recall and precision, the best value is 1, and the worst is 0. Therefore, the same distribution accounts for the F1 score [41].

False Positive Rate (FPR) is the probability that a negative sample is falsely classified as positive. In recognition, a user would be recognized as another user and can act in the other user's name. A FPR of 1 would indicate that all imposters would be recognized as the user. The goal is to achieve a FPR of 0, thus no imposter is recognized as a genuine user. Another name for FPR is False Match Rate (FMR) or False Acceptance Rate (FAR).

$$FPR = \frac{FP}{FP + TN} = FMR = FAR$$

True Positive Rate (TPR) is the probability that a positive sample is correctly classified as positive. In the case of recognition, a user would be correctly identified as the user. A TPR of 0 would indicate that the user would never be correctly identified as the user. The goal is to achieve a TPR of 1, to always correctly identify and never reject the genuine user. Another name for this metric is recall or sensitivity. This metric is complementary to the metric False Non Match Rate

(FNMR).

$$TPR = \frac{TP}{TP + FN} = Recall = Sensitivity = 1 - FNMR$$

It is the goal to combine and balance the FPR and TPR. They can be combined into a Receiver Operating Characteristic (ROC) curve as illustrated in Figure 2.7. This metric is widely used to evaluate and display the performance of biometric systems. To not only compare the ROC curve visually, the Area Under Curve (AUC) is calculated. If a model were to make perfect predictions, the graph would be rectangular with the corner in the top left, such as the dotted green line in Figure 2.7 indicates. This would result in an AUC of 1 [40]. Random guessing, on the other hand, is visually represented by the dashed line with an AUC of 0.5. To calculate the ROC curve, the test samples are predicted such that every sample has a probability assigned. Then, the decision threshold is varied and the resulting FPR and TPR for each threshold are stored and finally displayed as a curve [40, 41]. Equal Error Rate (EER) is another metric often used when evaluating biometric systems with ROC curves. This is when both FPR and TPR are equal.

2.4 Current State of Research

Research on recognizing players in games is limited. According to ANYBRAIN [5], cheaters can be distinguished from legitimate players based on their play style in FPS games, but there is no supporting data or explanation of how this is achieved. Additionally, no similar research is available for FPS games. While the FPS game Valorant does use a smurf detection system, they have not disclosed any details about it. Instead of banning accounts, they let good players rank up faster and play against more skilled opponents. One of the core components of player identification is identifying a person via their mouse movement, which is discussed in Section 2.4.1. Proposed techniques for identifying players in other game genres exist and will be discussed in Section 2.4.2.

2.4.1 Mouse Movement

Mouse movement is well-studied in the research area of biometric recognition [35]. Researchers use mouse movement data from known data sets like the *Balabeit Mouse Challenge data set* [19] or they collect their own data set [4, 11, 36]. Individuals whose mouse movements are recorded often act in a controlled environment where they must perform tasks in a given application. The recorded mouse movement is then segmented following different rules.

After segmentation, features are extracted depending on the classification methods used. Most current research aggregates the segments to extract features such as

Table 2.8: Overview of verification results based on mouse movements from related work. Predefined tasks require the user to move the mouse such as drawing shapes or moving the cursor to a specific position.

Paper	#Users	Application	AUC	EER
Aksari et al. 2009 [1]	10	Browser Usage	-	0.059
Antal et al. 2021 [4]	120	Predefined Task	0.940	0.150
Gamboa et al. 2004 [20]	50	Memory Game	-	0.002
Garabato et al. 2022 [22]	27	Predefined Task	0.962	-
Zheng et al. 2011 [57]	30	Computer Usage	-	0.013

mouse movement speed, acceleration, direction, curvature, and similar parameters. The advantage of this is that the aggregated features can then be used for shallow ML methods like KNN, SVM, DT, and Random Forrest (RF), which usually handle discrete data points instead of continuous signals. Some research also covers the usage of the continuous signal itself with methods like DL, Multi Level Perceptron (MLP), Long Short Term Memory (LSTM), CNN, and Dynamic Time Wrapping (DTW). The problem with the research is that it often uses different data sets and measures the classification's performance with different evaluation metrics. In addition to this, the data sets often consist of only a few users. Some results from related work are summarized in Table 2.8. These sources were selected due to their well-performing models covering different algorithms and user amounts. The metrics used for evaluation are FAR, False Rejection Rate (FRR), EER, F1 score, and AUC. The best results are reported if multiple data set variations or models are evaluated. However, not all evaluation metrics are provided in the summarized papers. Most of the papers trained one classifier per individual so that a binary classification problem could be solved.

2.4.2 Player Recognition

The effectiveness of smurf detection in the game Valorant has been noted; however, the system cannot identify a specific player [21]. Kim et al. [36] have a similar goal as the developers behind Valorant. They proposed a technique to detect Rankskill mismatch in the game LoL. This method utilizes an LSTM on various features such as enemies eliminated and gold gained, publicly available in 1 min intervals for each game. The achieved AUC was up to 0.8966. The literature on identifying players in a game is limited and this section summarizes papers that examine mouse movement in some form and include features such as events that occur during a game

or that are initiated by a player. The accuracies and used games are summarized in Table 2.9.

A study by Conti and Tricomi [11] used an LSTM to accurately detect players in Dota 2, achieving an accuracy of up to 0.963. It is worth noting that although Dota 2 and CS:GO are both team-based games, they differ significantly in their gameplay mechanics and possible player actions. The research conducted by Conti and Tricomi involved the extraction of various player actions such as cursor and camera position, attack actions, and movement requests from Dota 2 game demos.

The extracted features were aggregated over 0.5 seconds. The longest sequence time evaluated, 120 seconds, yielded the best results. To conduct this study, Conti and Tricomi used 100 matches from 50 players each. The 5,000 selected matches were selected from a pool of 30,000 matches. However, they only analyzed the first 10 min of the 5,000 matches, which can take up to an hour. The reason for capping the matches is that they tend to vary significantly over time and the beginning of the game tends to be more similar.

Silva and Costa-Abreu [49] conducted a study to identify players in a game of the same genre as Dota 2, that is LoL. They utilized features similar to Conti and Tricomi in [11]. The study included 56 users, with 18 playing more than one match used in the study. A binary classification model was trained with this data to distinguish a user from all others. Therefore, for 38 users the evaluation was based on matches that were used for training. The authors aggregated the features for each game in 5 min intervals and achieved an accuracy of up to 0.863 using an MLP approach. They also evaluated a Bayesian network and a Radial Basis Function Network (RBF), but these approaches resulted in an accuracy of around 0.75.

Siddiqui, Dave, and Seliya [48] created a data set featuring ten players playing Minecraft for 20 min which resembles one gameplay sessions. In the recordings, the mouse position and timestamp were both tracked. The author trained a RF classifier for each player, utilizing only ten mouse actions for the training. Due to this low amount of mouse actions, the classifier had an average accuracy of 0.61, a False Negative Rate (FNR) of 0.64, a FPR of 0.21, and an EER of 0.39.

In the study by Wang and Islam [54], they utilized the same data set as Siddiqui, Dave, and Seliya [48] and employed both SVM and RF methods to identify a player. They divided the training data into genuine and imposter data equally to balance it for the binary classification task. The RF method yielded an accuracy of up to 0.782, with a FPR of 0.36 and a FNR of 0.649. Similarly, the SVM method achieved an accuracy of up to 0.765, with a FPR of 0.405 and a FNR of 0.573.

Table 2.9: Overview of the verification results based on mouse movement and keyboard usage. All authors except Conti et al. use a verification approach.

Paper	#Users	Application	Accuracy	EER
Conti et al. 2020 [11]	50	Dota 2	0.963	-
Liu et al. 2013 [39]	-	StarCraft2	0.876	-
Siddiqui et al. 2021 [48]	10	Minecraft	0.616	0.397
Silva et al. 2018 [49]	56	LoL	0.863	-
Wang et al. 2023 [54]	20	Minecraft	0.784	-

2.4.3 Summary of Research Gaps

Some questions still need to be answered in the papers that are reviewed for this thesis. Most related work did not mention whether they allow users to use their own equipment so that the behavior might be altered. Only three papers let the users choose a pointer device and the settings [4, 22, 57]. All player recognition-related papers [11, 36, 48, 49, 54] and the paper by Anima et al. [2] let the users use their private equipment or the used data set implies this fact. In addition to the users' own equipment, all non-player recognition papers collected data in a controlled environment that was not the usual environment of the users.

All papers except the one from Conti and Tricomi [11] did not mention that they trained a classifier for multiple individuals. Note that a binary classifier might lead to better results per player than a multi-class classifier. Furthermore, all papers, except two, did not evaluate an open-set binary classification. Therefore, they only used negative samples of users, which were used during training [11, 22].

Only the paper by Kim et al. [36] explicitly mentioned a session split in the way it was introduced in Section 2.3.1. The mouse movement paper usually collected all data in one session, therefore, a session split would be impossible. However, the other player recognition-related papers potentially collected multiple sessions but did not mention any session split.

3 Design and Implementation

This chapter describes the complete process, from collecting data to recognizing individuals based on their mouse movement. The process with its intermediate steps is visualized in Figure 3.1. Players interact with the game Counter-Strike: Global Offensive (CS:GO) using a mouse, keyboard, or gamepad. The results of these inputs are stored in demos and can be extracted. The demos from Half-Life Television (HLTV) all contain game events played by professional players participating in eSports tournaments. First of all, the demos have to be downloaded. Because no official Application Programming Interface (API) for the website HLTV exists, a web scraper has to be developed. The scraper crawls through each series result page, extracts the demo download link, and stores it in a database. The demo download links from the database can then be used to download the demos to a specified directory.

Subsequently, the mouse movement has to be extracted from the downloaded demos. Walther [53] developed a parser called demoinfocs [53] to extract information from demos. Demos contain game events like shooting or when a match starts. The parser lets developers listen to these events, execute code via defined callback functions, or inspect the game state at every tick. Then, the mouse movement is extracted and stored in another database for player recognition.

To recognize players and to distinguish between players based on the extracted mouse movement, Deep Learning (DL) models are trained. Before the model is trained, the data is loaded into memory and preprocessed to speed up the training and achieve better classification results. Afterwards, the model is trained and evaluated on unseen data.

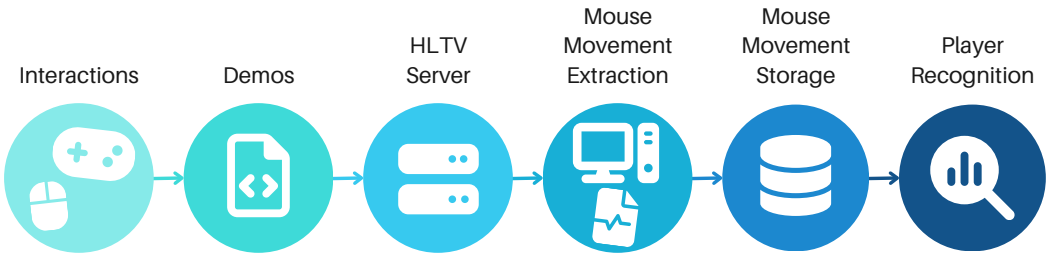

Figure 3.1: Overview of the data flow from recording the player's interactions to using them to train player recognition models.

The following chapter describes the three main components in each section. Therefore, the first Section 3.1 is about downloading demos from HLTV. Extracting the mouse movement from demos is described in Section 3.2 and the preprocessing and model architecture can be found in Section 3.3.

3.1 HLTV Demo Download

The website HLTV hosts a variety of resources regarding CS:GO but does not offer an official API to access these automatically. Thus, an application has to be developed in the scope of this thesis which is called HLTV Demo Downloader. The downloader is the connection from the HLTV server to mouse movement extraction as depicted in Figure 3.1. After running the application, the requested demos are stored in a specified directory on disk. The task is solved in two steps. The first step consists of gathering and scraping all information. The second one is downloading and naming the demos for later use. Although the demos are publicly available, consent by one of the website owners was requested, and the usage is permitted for this thesis.

HLTV uses a Content Distribution Network (CDN) to deliver the web pages faster, checking if a legitimate user or automated bot issues the request. A Captcha service is often used, where a user has to click an object for bot detection. In the case of HLTV, it is done automatically via some not publicly disclosed checks. If the check was successful, some cookies are stored by the client and are valid for 30 min. These cookies are sent to the server with each subsequent client request and serve as proof of being a legitimate user. It is essential to note that these cookies are only valid if the same User-Agent (UA) and Hypertext Transfer Protocol (HTTP) headers are used while retrieving the cookie. The check is successful and the cookies are set if the website is opened with a browser with a Graphical User Interface (GUI). To automate the browsing, the Python version of the library Selenium [47] is used. While Selenium is primarily intended for testing web applications, it can also be used to browse and scrape third-party websites for information extraction. The intention behind this is to test one's web applications, but one can also browse and scrape third-party websites to extract information. One crucial advantage of Selenium is the JavaScript execution, which allows it to render websites fully and not only in some parts, since nowadays websites are built with JavaScript frameworks and not with Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) only. This is one essential part of being detected as a legitimate user. When detected as a legitimate user, the website HLTV is opened so a cookie is set in the browser. Subsequently, the cookie is extracted with Selenium. To obtain the UA and HTTP headers, the website `https://httpbin.org` is used, specifically the /headers endpoint. The shown UA and HTTP headers are extracted from this endpoint. The Python library Requests (2.31.0) sends HTTP requests without the requirement of a browser. Since this automated behavior is detected by HLTV, the obtained

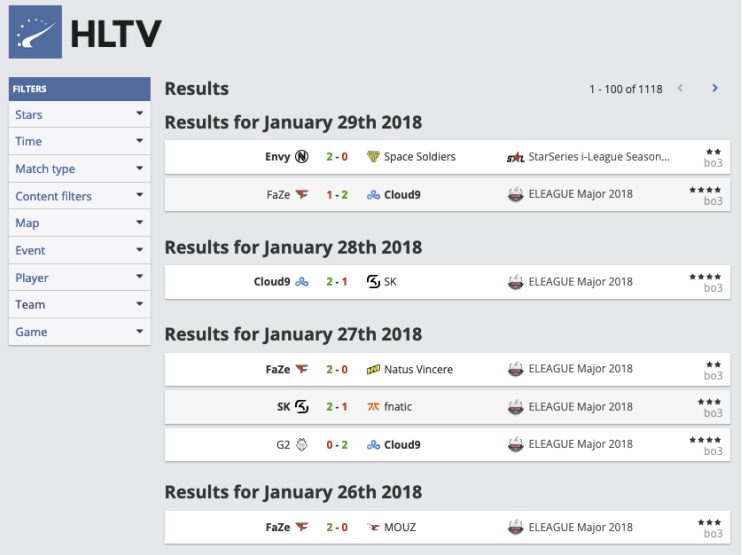

Figure 3.2: A screenshot of the results page of the query used in Listing 3.3. The left sidebar shows the available filters. In the middle are the series results played on several dates. The top right shows the pagination control.

cookies, UA, and HTTP headers are sent with each request. This enables the demo downloader to interact with HLTV automatically.

In the case of HLTV, the results page lists all series that are selected by the filter. A screenshot is presented in Figure 3.2. The default filter selects all series sorted by the date. The filters are stars, time, match type, content filters, map, event, player, team, and game.

The filter stars represent a metric for the importance of a series. As described in Section 2.1.5, HLTV offers a worldwide rating for players and teams. If two teams from the top 3 compete against each other in a series, the series is rated with five stars. Four stars require two teams to be in the top five and three stars require two teams to be in the top 10. Two stars are assigned when both teams are in the top 20 and one star is assigned if one is rated as a top 20. There also exist series without any stars, available in the category named all.

The filter time consists of startDate and endDate, where all series between these two dates are selected. The boundaries are inclusive. In eSports, tournaments usually consist of a qualifier and the final stages of a tournament. The qualifiers are typically played online and the final stages are in an arena with viewers on-site. The filter match type enables to search for only Online or Local Area Network (LAN) series. The filter content type ensures that the queried series has highlights, demos, Video on Demand (VOD) (live stream recording), stats, or all of the above. The filters map, player, and team selects series that contain the selected attribute. The last

filter game differentiates between series from CS:GO and the original Counter-Strike (CS). Although the series is played with Counter-Strike 2 (CS2), they are labeled as CS:GO. All filters are provided as an enum to increase usability. To take all these filters into account, the class UrlBuilder is created, and the usage is exemplary, as depicted in Listing 3.3.

```
from hltv_demo_downloader.downloader import QueryResults
from hltv_demo_downloader.query_url_builder import UrlBuilder, ContentFilter, Stars
from hltv_demo_downloader.driver_management import get_driver_headers,
    get_driver_hltv_cookies

headers = get_driver_headers()
cookies = get_driver_hltv_cookies()

url_builder = UrlBuilder(
 stars=Stars.TWO,
 startDate="2016-11-27",
 endDate="2018-01-31",
 content=ContentFilter.DEMO,
)
url_builder.build()
# "https://www.hltv.org/results?startDate=2016-11-27&endDate=2018-01-31&content=demo&stars=2"

QueryResults(
 url_builder, headers=headers, cookies=cookies
).store_series()
```

Listing 3.3: Usage of the UrlBuilder class to build a query and store the retrieved information by the QueryResults class. The resulting query will list all series from 27.11.2016 until 31.01.2018, during which two of the top five teams competed against each other, and where a demo is available to download.

The QueryResults utilizes the UrlBuilder class. This class navigates through the series returned based on the selected filters. By leveraging the presence of cookies and headers, these requests can be executed directly in the terminal, bypassing the need for Selenium and significantly accelerating the process. HLTV uses pagination to limit the displayed result to 100 series per page, as shown in Figure 3.2. Therefore, after 100 series are scraped, the pagination is used to see the following 100 series.

Figure 3.4 depicts an exemplary series page. Each series page is parsed, and additional information besides the demo id is extracted. The demo download link can be constructed with the demo id. To parse the HTML, the library BeautifulSoup (4.12.3) is used. The button with the text “Demo sponsored by Bitskins” contains the demo download link. All extracted information is stored and organized in a Structured Query Language (SQL) database to be reused later. SQLite is chosen as the database engine because of its high speed and simple usage. Furthermore, all data in the tables is stored in one file, which can be shared. One disadvantage of the database engine is the limited multi-user support, which is however not required

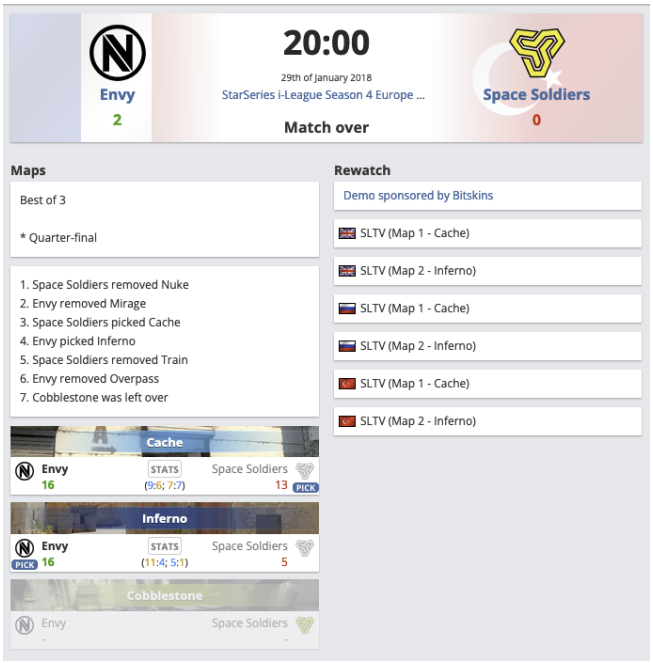

Figure 3.4: A screenshot of the series page. Team Envy competed against Space Soldiers on 29.01.2018 in a Best Of (BO)3 series. The first element in the Rewatch section is the download button for the demos.

for this task [55]. The series table consists of columns like the team names, dates, demo ids, or stars. The matches table has a foreign key to the series table and stores the map name. Since multiple scores per match exist, a separate table is created, which stores these with a foreign key to matches. To query for all distinct players, a table for them is created where the player's name is stored. The last table combines players and matches in a many-to-many relationship. The information is stored after each series page is parsed to prevent it from being lost when the application crashes or unexpectedly shuts down.

Step two consists of querying the database for the demo ids to download the demos. The demo id is the only parameter required to build the demo download link. The following attributes are used to purposeful name the downloaded demos: series id, demo id, event name, date, team names, and the BO amount. These attributes are later used in the mouse movement extractor. If the desired demos are downloaded, the mouse movement extraction can be started.

3.2 Mouse Movement Extraction and Storage

The demos contain all the information required to replay a CS:GO match. Each tick includes the complete game state, including all the characters' viewing angles, positions, current equipment, and more. A parser must interact with all data stored in these binary files to extract this information. This section is depicted as mouse movement extraction and mouse movement storage in Figure 3.1. The parser demoinfocs [53] enables developers to extract information via callback functions from demos. Furthermore, the parser is written using the open-source programming language Go. This compiled language stands out for the built-in concurrency and robust standard library. It is faster than Python and uses fewer resources for the same task [23]. These advantages make it a good candidate for processing high amounts of data. The developed mouse movement extractor uses Go (1.21.3) and demoinfocs (v4.0.0-beta.5). To extract data from demos, developers can listen to an event via a callback function. Upon triggering the callback function, the function can access the current game state.

This enables the extraction of the viewing angles α consisting of yaw and pitch. These are values typically used to describe an object's orientation. The yaw value is the horizontal and the pitch is the vertical orientation of the character's head. The yaw value ranges from 0° to 360°, where 360° is an exclusive boundary and 0° is equal to 360°. The pitch value ranges from -90° to 90°. The pitch values stored in the demos are between 270° and 360° and between 0° and 90° because the demo does not store negative degrees; the first range equals -90° to 0°.

To be precise, the actual mouse movement d_{moved}, meaning the distance the mouse is moved, is not extracted. The difference of two viewing angles α is calculated instead. This difference is the mouse input the game receives. However, it is not the actual distance the player moved the mouse. A mouse has a certain speed, measured in Dots per Inch (DPI). The DPI is then multiplied by the mouse sensitivity s_{OS} of the Operating System (OS). This product is then multiplied again by the mouse sensitivity of the application s_{app}, in this case, the game CS:GO. This resulting multiplier is called effective Dots per Inch (eDPI) and differs for many players. However, most professional players use a relatively low eDPI with a mean of 867.9 ($\pm$252.4; $N = 900$) [37].

$$\text{eDPI} = \text{DPI} \cdot s_{\text{OS}} \cdot s_{\text{app}}$$

The distance between two viewing angles is defined as:

$$dist(\alpha_{\text{i}}, \alpha_{\text{i+1}}) = \text{eDPI} \cdot d_{\text{moved}}$$

Although this calculated distance doesn't represent the actual physical distance the mouse moves, it still captures an individual's mouse movement patterns. Furthermore, in gaming, the eDPI can be viewed as a personal characteristic. Moving the mouse

with a certain eDPI becomes a habit, so-called muscle memory [6]. If the eDPI changes, the player must concentrate on adapting to the new eDPI. Due to the adaptation process, professional players rarely change their eDPI. Incorporating different eDPI values reflects the real-world data as the demos themselves are. Due to such readability, this is still referred to as mouse movement.

To store the extracted mouse movements in an organized and reusable way, a SQLite database is used for the same reasons as mentioned in Section 3.1. The extraction application parses the demo's file name first and creates an entry in the series table. This table consists of the series id assigned by HLTV, a name, date, BO amount, and both team names. Afterwards, the game server rules are extracted and stored. This is essential information because each game server can have different settings such as the maximum round time or the number of rounds played. These are usually kept constant due to unified tournament rules, but rules like the tick and snapshot rates may differ. These values can be used later to filter the mouse movements to acquire the same sampling frequency. After the rules are extracted, the match attributes are added to the database. A match has a foreign key relationship to series and rules. It mainly consists of the demo's amount of ticks and the final score. For each match, the played rounds are stored. In the database, each round consists of the current team scores, the first tick when the round starts, the tick when all characters can move, the tick when no character can move anymore, and the tick when the round is over. The first and last ticks when players can move are important since no mouse movement needs to be extracted when the players cannot move. This reduces the storage usage and removes mouse movement which can be viewed as noise. It is not trivial to determine these four points in time because the game server only sometimes triggers or stores the events. The first tick per round can be determined if the event RoundStart is triggered and the first active tick when the event RoundFreezetimeEnd is triggered. The time when the characters cannot move anymore is more challenging to determine reliably. The events RoundEnd, RoundEndReason, RoundMVPAnnouncement, RoundMVPReason, and RoundEndOfficial are used for this. The four events are required because some are only triggered under certain circumstances. Sometimes, none of the mentioned events are triggered. In this case, the following RoundStart event ends the previous round. Although not all mentioned events co-occur, they occur in quick succession. Not triggering events might result from server plugins interfering incorrectly with the original game server behavior. These plugins alter the match to pause and resume it. Moreover, they sometimes reset and replay a round due to technical difficulties. This will influence the events and make it more challenging to determine the correct order of played rounds because there is no event for pausing or resetting a round. In some demos, a round is over, the next is started, the newly started round is canceled, and the round before is replayed. Because no events exist for this, a simple algorithm has to be employed. Since the match's end score is known from HLTV, the order of the valid rounds is determined backwards. The previous round must have been at least 10 s long, and the score must be exactly one less. The time is chosen as a lower limit

because winning in less than 10 s should be impossible. After this process, multiple rounds with the same score can exist and are removed. The latest round with the same score is always taken.

After the rounds are stored in the database, the ticks will be extracted. The parser is iterating over each player per tick. For potential further processing and filtering properties like current Health Points (HP), armor, ping to the server, whether or not the character is reloading, planting, or defusing are held in memory. While iterating over all ticks, all players and their used weapons are stored in the database. If they already exist, the current id is queried for the player id and weapon id foreign key relationship. The demo tick extractor supports filters that can be applied during the tick extraction. Filters enable users to store ticks only when a player is holding a specific weapon or is not standing still. After the filter is applied, the mouse movement, in the form of yaws and pitches, is extracted per player and round. The angles are then segmented into sequences of a predefined size called sequence length. A sequence is closed or discarded if the difference between two ticks exceeds $(tickrate/snapshotrate) + 1$. This ensures that the sequence does not have too many missing ticks, for example, due to technical issues while storing the demo. During this thesis, one missing tick is allowed between two available ticks. These are not interpolated, but the bigger difference in time is considered in further calculations. The two sequences of absolute angles, yaw and pitch, are then differentiated to become sequences of velocities. When deriving the yaw sequence, the discontinuity between the start and end of the circle must be considered. While a circle has no actual start or end, the numerical values 0° and 360° do not represent it. Therefore, a function has to be used to derivate the yaws. The derivations are then divided by the time difference to the previous tick. Then, the velocity sequences are stored with the foreign keys to players and rounds. Only the velocities are stored because the mouse movement between two ticks is the change or speed in viewing angles.

```
threadAmount: 10
demoQueueLength: 20
useCleanDatabase: true
databasePath: "../dbs/test.db"
demoPath: "../demos"
parsingMode: "grouped-raw" # "grouped-raw", "grouped-features", or "single"
sequenceLength: 2048 #in ticks
```

Listing 3.5: Config file to extract the mouse movement from CS:GO demos to a SQL database. This config would delete the existing database called “test.db”. All demos which are stored in the sibling directory called “demos” would be used for extraction.

The described steps are performed per demo and can be parallelized via the newly developed application. The amount of threads and other configurable parameters can be defined in a YAML Ain’t Markup Language (YAML) file. YAML is a

markup language often used for config files due to its simplicity and readability. Listing 3.5 lists all available parameters. The threadAmount is the amount of parallel parsed and processed demos. The demo queue contains the path for each demo. The demoPath specifies the directory from which all demos are searched in its subdirectories. To reduce memory usage, only a specific number of paths is stored. The option useCleanDatabase deletes the database if the same-named database exists in the target directory. The parsingMode contains multiple modes to store the mouse movements, but only the grouped-raw is described and used in this thesis. These still exist to test different approaches for recognition. The sequenceLength is the number of ticks and data points per sequence. The groupWindowSize groups values in the sequences and calculates the sum.

3.3 Player Recognition

The newly created database, whose structure is described in Section 3.2, is used to train a model. This model should learn to extract patterns from mouse movements to find a relationship between patterns and a player. In general, some fundamental steps must be performed which are described in Section 3.3.1. These steps are performed for each experimental setup.

3.3.1 Loading and Preprocessing

To preprocess the data, the Python (3.11.2) libraries Pandas (2.2.1) and Numpy (1.26.4) are used. To determine which data shall be selected, the amount of stored sequences per player is counted in the database and the result is sorted by the sequence amount. Then, the first N_{player} players are selected. These are the N_{player} players with the most sequences in the database. From this query, the player ids are extracted for further queries. These N_{player} ids are then used to load the sequences into memory. Per player id N_{train} training sequences and N_{test} testing sequences are queried and stored in a Pandas data frame each. A Pandas data frame is a widely used data structure analogous to a single SQL table, offering ease of manipulation and access. It is ensured that every sequence is recorded with a snapshot rate of 128 Hz. After all player ids are queried, the two data frames then contain $N_{\text{player}} \cdot N_{\text{train}}$ and $N_{\text{player}} \cdot N_{\text{test}}$ sequences. While querying the sequences, a session split is enforced. In this case, a series is equal to a session because multiple matches can be played in a row. Usually, up to one series is played continuously. Then, one standard scaler is applied to all velocity sequences to speed up the learning. The standard scaler requires a mean μ and standard deviation σ. These are calculated only based on the training data set. The parameters are not determined by the test data set since they resemble the future and future sequences are not known while training. After this,

the standard scaler is applied to both data sets.

$$z = \frac{x - \mu}{\sigma}$$

As a last step, the player ids are renumerated such that the player ids range from 0 to $N_{\text{player}} - 1$. This is done to ensure compatibility with the Machine Learning (ML) libraries. The resulting data frames contain the following columns: player id, round id, match id, series id, pitches velocity, and yaws velocity. This data frame is the foundation for the following section. While the further steps differ, the sequences will be transformed into Numpy arrays at the end in all cases. Afterwards, the sequence of $((pitch_0, ..., pitch_{\text{sequence\_length - 1}}), (yaw_0, ..., yaw_{\text{sequence\_length - 1}}))$ is reshaped to $((pitch_0, yaw_0), ..., (pitch_{\text{sequence\_length - 1}}, yaw_{\text{sequence\_length - 1}}))$. This is done as a last step so that a data frame's advantages can be used in the meantime. No validation data is used since this would increase the time gap between train and test data. Therefore, the model has to predict even further into the future, which may degrade the performance. Due to this, no hyperparameter tuning is performed.

3.3.2 Base Model Architecture

To build and train the DL model, the Python (3.11.2) library Keras (3.1.1) with the TensorFlow (TF) (2.16.1) backend is used. Keras is a high-level API for multiple ML libraries. It enables developers to build and train models with just a few lines of code. Combined with TF and Python, this results in easy and quick prototyping and evaluation of various models. Due to these advantages, Keras, combined with TF, is used for this thesis. The goal is to train one model to extract features and predict if the sequence belongs to a given individual. The model receives the mouse sequences as input, either predicting true or false in case of verification or the player id as a one-hot encoded vector for identification. The model's architecture is inspired by papers mentioned in Section 2.4 [3, 22]. The first layers are the same for the identification and verification model. A visual overview of the model is provided in Figure 3.6. The model takes $N_{\text{sequence\_length}}$ tuples of yaw and pitch velocities as input. Therefore, the sequence shape is $(N_{\text{sequence\_length}}, 2)$. The input is processed by three groups of Convolutional Neural Network (CNN) layers. All groups use leaky Rectified Linear Unit (ReLU) as an activation function. The leaky ReLU is chosen since the preliminary test showed the fastest convergence time. The following values and compositions are chosen since preliminary tests showed the best performances by the models. The first two groups consist of two CNN layers with 128 filters, a filter size of 3, and the same padding. The two layers are followed by a max pooling layer to reduce the dimensionality and increase the translation invariance [29]. The third CNN group only has one CNN layer, one max pooling layer, and ends with a unit normalization layer such that the L2 norm is enforced. This could help to stabilize the gradients and improve generalization [40]. To reduce overfitting, the CNN groups

are followed by a dropout layer with a probability of 0.4. These are then flattened and followed by a fully connected layer with 256 nodes. This layer can be viewed as the template or embedding output and concludes the base model. In the case of verification, the base model is followed by another dropout layer with a probability of 0.2, which ends in a fully connected layer with one output neuron. This neuron uses the sigmoid activation function to project the input into a range between 0 and 1. 0 represents a rejection and 1 an acceptance. The binary cross-entropy loss function is used to compile and train this model.

In the case of identification, the base model is followed by another dropout layer with a probability of 0.2 and ends in a fully connected layer with N_{player} output neurons. This results in an output vector with N_{player} entries, and together with the Softmax activation function, a one-hot encoded vector is outputted. Since Softmax is used, a value close to 0 indicates that the model predicts that the provided sequence does not belong to the player id. A value close to 1 is the opposite. Although the one-hot encoded output is easy to interpret, and no embeddings or templates need to be computed and stored, the approach has drawbacks. This approach does not optimally scale since, for every new player, the model needs to be retrained. However, it is sufficient for distinguishing between a smaller group of players. As Section 2.2 describes, embeddings or templates are usually stored and compared. Thus, once computed, they can be reused. This modular biometric system was not used to test and develop the model and data pipeline quickly. The proposed pipeline serves as a prototype and a baseline for future work. If the approach is deployed to the real-world, a modular biometric system with embeddings should be used instead. The categorical cross-entropy loss function is used to compile and train this model.

The identification and verification models report their performance via the F1 score, Area Under Curve (AUC), and Equal Error Rate (EER). The identification model also reports Rank1Accuracy and Rank5Accuracy. Both models are trained for 20 epochs, with a batch size 640, and the training data is shuffled. The epochs are limited to 20 since initial testing showed that the model improved marginally after this amount of epochs. The training data is shuffled so each batch contains a uniform distribution of all player ids. If not mentioned otherwise, the default arguments are applied which are present in Keras 3.1.1 and TF 2.16.1. Furthermore, preliminary to each training, the Keras' clear_session function is invoked to free the memory. Since the trainings are performed multiple times in a loop this is necessary to not have full memory.

This model will be used with different $N_{\text{sequence\_length}}$, during Section 3.3. Therefore, Table 3.7 provides a brief overview of the trainable parameters.

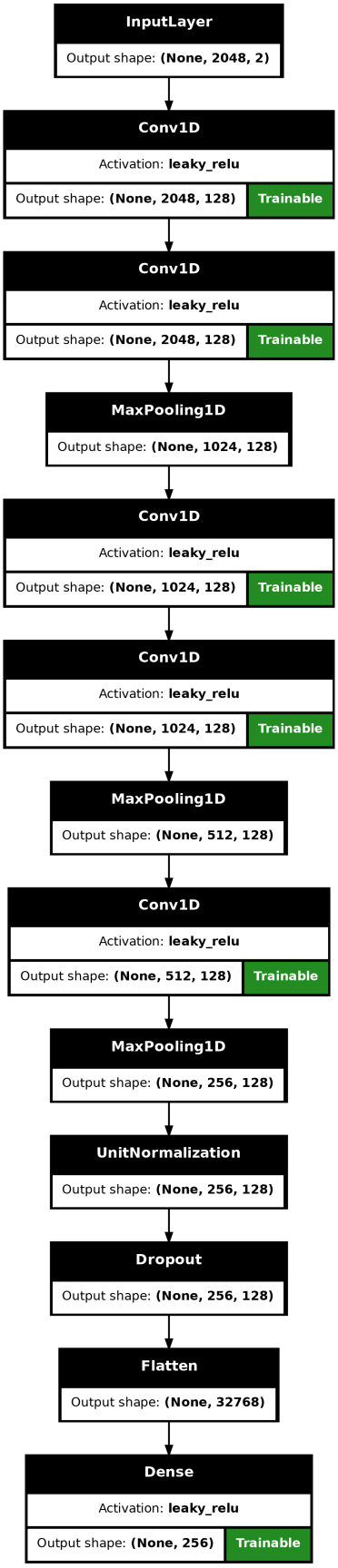

Figure 3.6: The base model visualization shows each layer as one block, including the used name, activation function, and output shape. The figure is generated using the plotting tool Keras provides.

Table 3.7: Amount of trainable parameters the described model has depending on the length of the input with $N_{\text{player}} = 50$.

Sequence Length	**Parameter Amount**
512	2,308,274
1,024	4,405,426
2,048	8,599,730
4,096	16,988,338

4 Data Set

This chapter describes the data set compiled during this thesis. First, the publicly accessible data from Half-Life Television (HLTV) is described and then the data is used for training and testing a model. From 2012 to 2023, HLTV covered 69,354 series for which demos are available to download. These numbers are grouped per year and displayed in Figure 4.1. The amount of matches played by professional players increased over the first five years. This is due to the increasing popularity of gaming, eSports, and Counter-Strike: Global Offensive (CS:GO) itself. On average, a series consists of 2.21 matches and 46.8 rounds. In total, 15,523 unique players have been listed over the years.

Since the data set is large, a subset of it is sufficient to train a model. Furthermore, limiting the data set size saves disk space. Only series from 01.01.2020 to 31.12.2022, which have at least one star according to the HLTV rating, are used to train models. Data from 2023 was not included due to the release of Counter-Strike 2 (CS2) and the lack of filtering options on HLTV, as described in Section 3.1. This database takes up about 163.5 GB on disk, whereas the demos take up about 2.4 TB. Many players have only played a few matches because they played against a team in the top 20.

Figure 4.2 provides examples of how different mouse movements can look despite being from the same player. The models are trained to find patterns in the mouse movements, which seems complicated to find by the human eye.

To estimate how many hours of mouse movement are contained in the complete data set from HLTV, the average time per match from demos with one star and between 01.01.2020 and 31.12.2022 is calculated and extrapolated to the whole data set. Between 01.01.2020 and 31.12.2022, 7,062,656 2,048-long sequences were extracted from 7,913 matches. This results in an average of 892.5 sequences per match. Therefore the estimated amount of mouse movements is $892.5 \frac{\text{sequences}}{\text{match}} \cdot 153,189 \text{ match} = 136,721,182.5$ sequences. Since a sequence is $\frac{2{,}048}{128 \text{ Hz}} = 16$ s long, the data set contains $136,721,182.5 \text{ sequences} \cdot 16 \text{ s} = 2,187,538,920 \text{ s} = 607,649.7 \text{ h} \approx 69.4$ a of continuous human mouse movement. This is the estimated amount of mouse movement the HLTV data set contains until 31.12.2023. An average player contributes $\frac{607{,}649.7 \text{ h}}{15{,}523} \approx 39.2$ h to this data set. It is important to note that the time distribution per player is not equally distributed. Some players contributes several thousands to the data set and some only some hundreds. An average match is 49.3 min long, and 23.8 min of mouse movement per player is extracted per match.

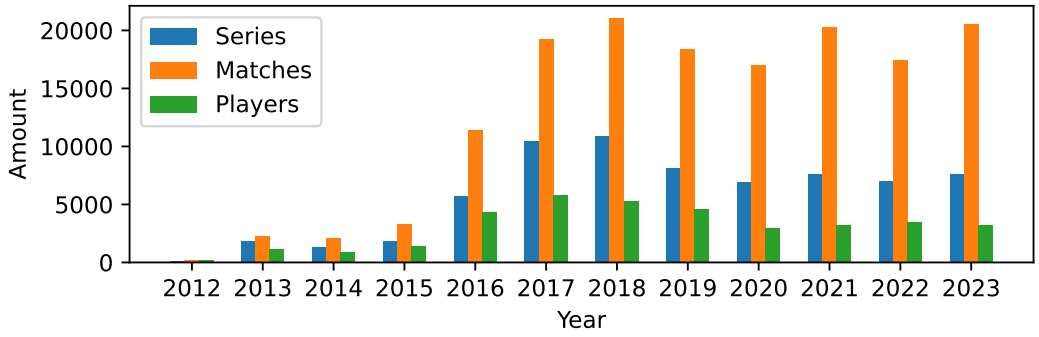

Figure 4.1: The amount of series, matches, and players per year since 2012 listed by HLTV. This data is based on all series between 01.11.2012 and 31.12.2023 which have a downloadable demo.

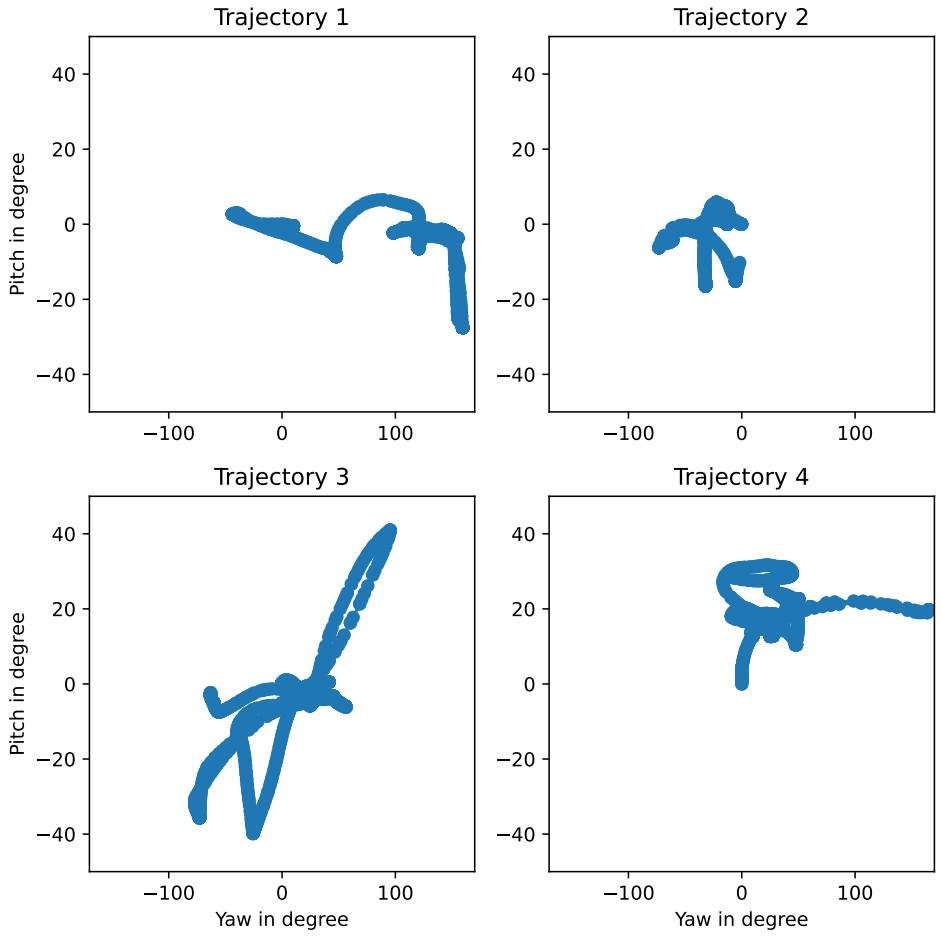

Figure 4.2: Four mouse trajectories which are from the player Mightymax. Each trajectory contains 2,048 data points with a snapshot rate of 128 Hz.

Since only groups of mouse positions (sequences) are extracted for this calculation and not all data points without groups, the actual time of recorded mouse movements should be higher. Because this is only an estimation, this detail is not considered. Out of these 23.8 min, players often wait for an enemy while defending a spot. Although many ticks are recorded, the mouse is not moved at every tick. On average 65% (±11%) of the time, the mouse has not moved from one tick to another at a tick rate of 128 Hz. These pauses, on the other hand, can also be a pattern that differs between individuals.

5 Player Recognition Experiments

To assess the performance and the influence of different parameters like N_{player}, N_{train}, N_{test}, or the sequence length, various experiments are carried out. To answer the research questions, this thesis measures effectiveness by the F1 score, Area Under Curve (AUC), and Equal Error Rate (EER). Due to problems in combination with time series data, cross-validation is not used. Cross-validation splits the data into k folds. Assuming that the testing fold is in the middle between all other training folds, the model trains on sequences before and after testing sequences. Subsequently, the model acquires knowledge of the patterns before and after the testing folds, through the training process. Thus, it makes predicting the correct output easier for the model. This distorts the model's reported performance. Furthermore when dealing with time series data, it does not make sense to predict the past while training on the future. Instead, another data selection method is used, similar to a sliding window. In this case, the first window selects the first $N_{\text{train}} + N_{\text{test}}$ sequences. The following windows are moved in such a way that 80 min of mouse movements are skipped per window movement. The size of the skip is called repetition offset. The repetition offset ensures the model does not converge due to a lucky data selection or weight optimization. Thus, each model is trained and tested with session-split data sets. Most of the experiments evaluate both recognition modes and the last two experiments focus only on identification. The verification mode is designed to match the setup the papers from Section 2.4 use. Therefore, one binary classifier is trained for each player. This is one approach for detecting if the mouse movement related to an account differs from the known patterns. This can detect if another player plays on an account such as when the account is boosted. To consider unknown attackers, the verification models are trained with one group of players, and the evaluation is performed with another group that was not used for training except the player the model is trained for. This is an open-set evaluation. The verification models are trained four times with the mentioned repetition offset. The identification mode is evaluated as closed-set. Therefore, only players used for testing are also used for training. This is one approach to identify the player who plays on another account. This closed-set identification reflects finding the booster's or smurf's main account. The identification models are trained eight times with the mentioned repetition offset. Although the two research questions focus on verification and identification separately, the results are grouped by experiment to compare both recognition modes better and reduce redundancy.

5.1 Influence of Sequence Length

The longer a sequence length is, the more mouse movement is bundled together time-wise. This experiment investigates the correlation between the sequence length and the model's performance. To achieve this, the identification and verification mode with 50 players is trained on sequences of different lengths ranging from 512 to 4,096. This experiment is divided into two approaches. The proportional approach always uses 6,000 training sequences per player for identification and 6,000 positive and 6,000 negative sequences for verification. Therefore, the identification model is consistently trained with 300,000 sequences, and the verification model is trained with 600,000 sequences.

When the sequence length increases and the sequence amount per player remains unchanged, the model sees more mouse movement time-wise. To mitigate this effect, the second approach adapts the sequence amount per player so the product $N_{\text{sequence\_length}} \cdot N_{\text{train}}$ remains constant. This results in a fairer comparison due to the same number of training data points, but the model has fewer examples to learn from despite having more parameters. The constant identification model with a sequence length of 512 is trained with 300,000 sequences. The model with a sequence length of 1,024 is trained with 150,000 sequences to match the same mouse movement time. The constant verification model with a sequence length of 512 is trained with 6,000 positive and 6,000 negative sequences. The model with a sequence length of 1,024 is trained with 6,000 sequences to match the same mouse movement time. The same procedure accounts for the 2,048 and 4,096 models.

The identification models are evaluated based on 100,000 sequences, and the verification models are based on 2,000 positive and 2,000 negative sequences. The negative training sequences are taken from 15 unknown players. Each identification model is trained eight times and each verification model is trained four times per player. In the case of identification, the results are based on $2,000$ testing sequences $\cdot$ 50 players $\cdot$ 8 repetitions $= 800,000$ testing sequences. In the case of verification, the results are based on $(2,000 + 2,000)$ testing sequences $\cdot$ 50 players $\cdot$ 4 repetitions $= 800,000$ testing sequences. Therefore, the reported metrics are based on 800,000 sequences for all four combinations. The left constant subplot in Figure 5.1 shows that the reduction in training sequences gradually decreases the F1 score, although the time of mouse movement seen by the model is constant. This is true for both recognition modes. The best F1 score for identification is, on average, 0.677 ($\pm$ 0.196), and the best F1 score for verification is, on average, 0.860 ($\pm$ 0.098). Both scores peak at a sequence length of 512. The right proportional subplot displays an increasing F1 score with longer sequences. The best F1 score for identification is, on average, 0.850 ($\pm$ 0.112), and the best F1 score for verification is, on average, 0.922 ($\pm$ 0.046). Both scores peak at a sequence length of 4,096. More detailed results are reported in Table A.1 and Table A.2.

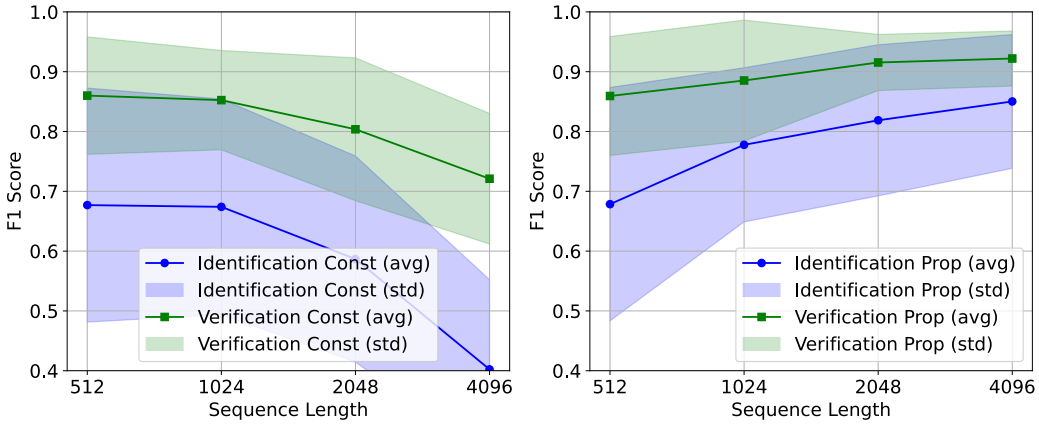

Figure 5.1: The influence the sequence length has on the average F1 score. The left subplot displays recognition models trained after the constant approach. The right subplot displays models trained after the proportional approach. The line and the shades display the average score and standard deviation.

5.2 Influence of Spapshot Rate

The snapshot rate determines the accuracy of the stored mouse movement since the character's view is stored more frequently. The highest possible snapshot rate for Counter-Strike: Global Offensive (CS:GO) is 128 Hz. To assess the influence of the snapshot rate, the sequences sampling rate is artificially reduced by summing every two, four, or eight values. This mimics a snapshot rate of 64 Hz, 32 Hz, and 16 Hz, respectively. Although the sequence length decreases with this summation, the duration a sequence captures remains the same. The identification models are trained to distinguish 50 players using 6,000 training sequences and are evaluated using 2,000 testing sequences per player. Thus, each model is trained with 300,000 training sequences and is evaluated based on 100,000 further sequences. In the case of verification, one model is trained per player. The verification models are trained with 6,000 positive and 6,000 negative sequences and are evaluated based on 2,000 positive and 2,000 negative sequences. The negative training sequences are taken from 15 unknown players. The reported metrics are calculated based on 800,000 sequences for each recognition mode and snapshot rate. Sequences of 4,096 are used as a starting point with a snapshot rate of 128 Hz. This is denoted as 4,096@128. The artificially lowered snapshot rate of 64 Hz reduces the sequence length from 4,096 to 2,048. Then, the models trained with the 2,048@64 sequences are compared to the ones trained with the 2,048@128 sequences. This compares two models trained on the same sequence length and amount. However, the sequence duration differs for all sequence lengths except 4,096 because this is the original sequence. The same accounts for the other comparisons in Figure 5.2.

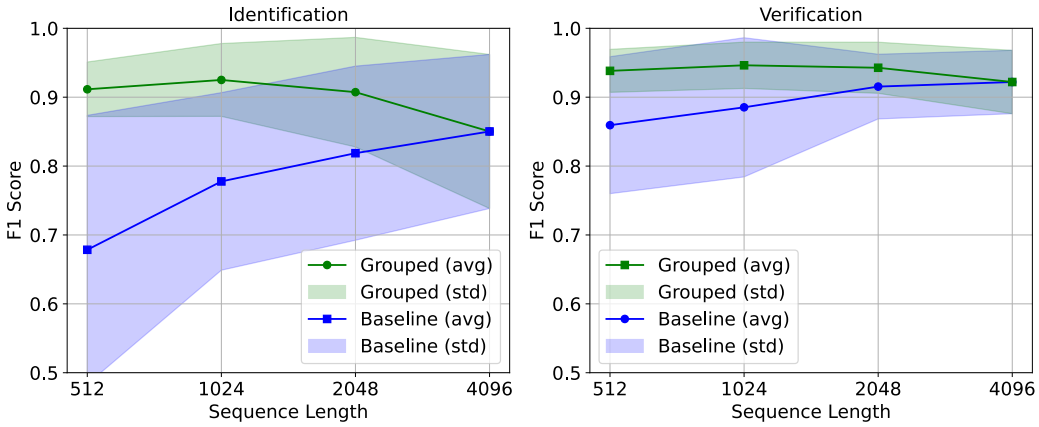

Figure 5.2: The influence of the snapshot rate on the model's performance. The grouped line represents grouping and summing the original 4,096@128 sequence to the sequence length on the x-axis. This reflects the artificially reduced snapshot rate. The baseline is the sequence length without grouping. The line and the shades display the average score and standard deviation.

At a sequence length of 512, the results from the 512@128 models and the 512@16 models are compared. To obtain 512@16 sequences, 4,096 sequences with a snapshot rate 128 are taken, and every eight values are summed up. For both recognition modes, the grouping and summing of every four values results in an optimum. This reflects a snapshot rate of 32 Hz. In the case of identification, the 1,024@32 models achieve an average F1 score of 0.925 ($\pm$ 0.053), and in the case of verification, an average F1 score of 0.946 ($\pm$ 0.034). In the case of identification, the artificially reduced snapshot rate yields an absolute improvement of 0.147, and in the case of verification, it is an improvement of 0.061. The standard deviation (the colored shades) decreases with a lower snapshot rate. The 512@16 identification model has a standard deviation of 0.04 compared to the 0.195 from the 512@128 models. Therefore, the lower snapshot rate reduced the standard deviation by up to 80%. Although lowering the snapshot rate also improves the verification models, the improvements are less significant than those of the identification models. However, the standard deviation is also reduced. More detailed results are reported in Table A.3.

Besides improving the F1 score, the AUC is also improved when reducing the snapshot rate from 0.976 ($\pm$ 0.009) to 0.990 ($\pm$ 0.002). The best AUC is 1. This is the case when the ROC curve is in the top left corner. Figure 5.3 displays the ROC curve when training an identification model and predicting 2,000 sequences for each of the 50 players. In contrast to Figure 2.7, the x-axis is shortened to make some of the 50 curves visible.

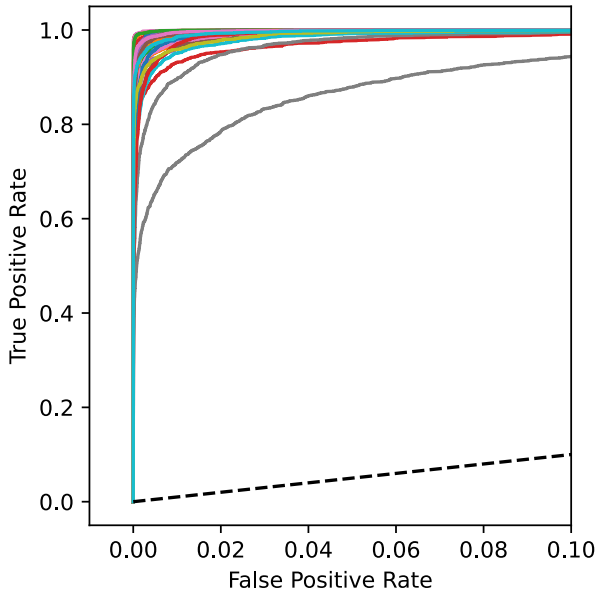

Figure 5.3: The Receiver Operating Characteristic (ROC) curve for each of the 50 players when training and testing a model with 1,024@32 sequences. The black dotted line represents random guessing.

5.3 Influence of Training Sequence Amount

It is essential to know how many training sequences the model requires to effectively find and extract patterns from the sequences to recognize players. To investigate the influence of the training amount, the models for both recognition modes are trained with different amounts of training sequences.

Training and testing utilizes 1,024@32 sequences. The identification models are trained with 1,000 to 9,000 sequences per player and are evaluated with 2,000 sequences per player. Each identification model is trained to distinguish 50 players, resulting in 50,000 to 450,000 sequences for training and 100,000 sequences for evaluation. The verification models are trained with 1,000 to 9,000 positive sequences and an equal amount of negative sequences. Therefore, each player's verification model is trained with 2,000 to 18,000 training sequences and is evaluated with 2,000 positive and 2,000 negative sequences. The negative training sequences are taken from 15 unknown players. The training for each sequence amount is repeated eight times for identification and four times for verification, with a repetition offset of 80 min. The reported metrics are calculated based on 800,000 sequences for each recognition mode and sequence amount.

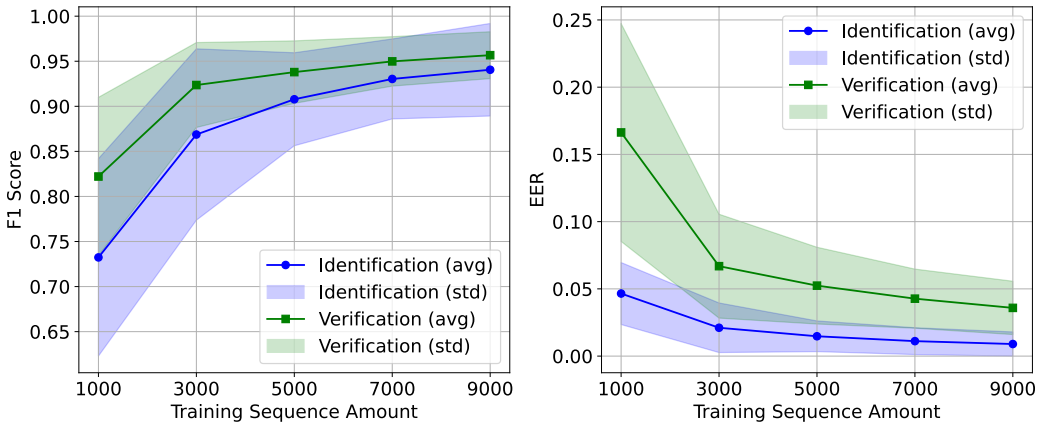

Figure 5.4: The influence the amount of training sequences has on the model's performance. F1 score (left) and EER (right) in relation to the training sequence amount. The lines represent the average and the areas around the lines the standard deviation.

Figure 5.4 visualizes the influence the training sequence amount has on the F1 score and the EER. The F1 score and the EER improve as training sequences increase. Both scores reach their peak at 9,000 training sequences. The F1 score for identification is 0.941 (± 0.051) and 0.957 (± 0.026) for verification. The EER for identification is 0.009 (± 0.009) and for verification 0.036 (± 0.020). More detailed results are reported in Table A.4.

5.4 Influence of Testing Sequence Amount

As mentioned in Section 2.2, permanence is one crucial property of biometric characteristics. This property describes how stable a characteristic is regarding internal changes. Therefore, this experiment varies the number of testing sequences and keeps the number of training sequences constant.

Training and testing utilizes 1,024@32 sequences. The identification models are trained with 6,000 sequences per player and evaluated based on 1,000 to 9,000 sequences per player. The predicted time of the sequences ranges from 8.9 h to 80 h of continuous mouse movement per player. Each identification model is trained to distinguish 50 players, resulting in 300,000 training sequences and 50,000 to 450,000 sequences for evaluation. The verification models are trained on 6,000 positive and 6,000 negative sequences. The evaluation amount ranged from 1,000 to 9,000 positive sequences and an equal amount of negative sequences. The negative training sequences are taken from 15 unknown players. Therefore, each verification model is evaluated based on 2,000 to 18,000 sequences. The training for each sequence

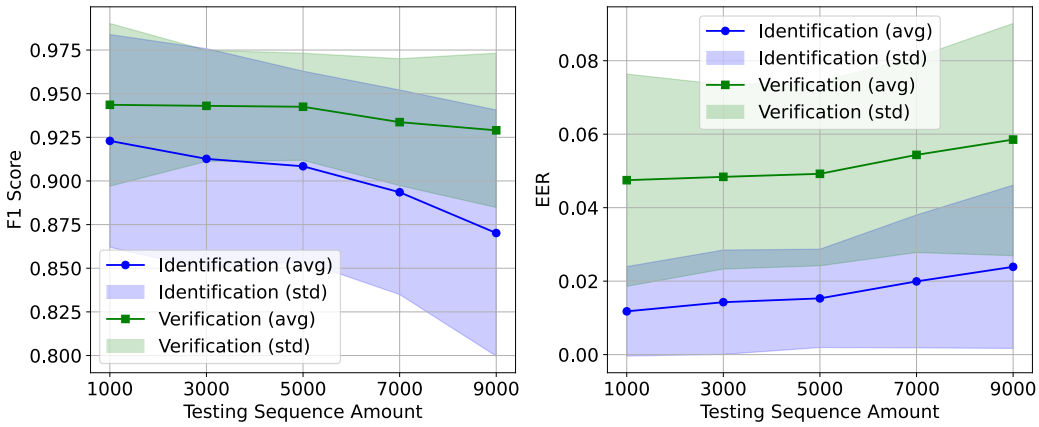

Figure 5.5: The influence of the amount of testing sequences on the model's performance. F1 score (left) and EER (right) depending on the testing sequence amount. The lines represent the average and the areas around the lines the standard deviation.

amount is repeated eight times for identification and four times for verification, with a repetition offset of 80 min. The reported metrics are calculated based on 400,000 to 3,600,000 sequences for identification and verification.

Figure 5.5 visualizes the influence of the testing sequences on the F1 score and the EER. With an increasing amount of test sequences, the metrics worsen. The F1 score decreases by 0.053 in identification from 0.923 (± 0.061) to 0.870 (± 0.070). Verification decreases by 0.015 from 0.944 (± 0.047) to 0.929 (± 0.044). Therefore, verification is less impacted by more testing sequences than identification.

5.5 Influence of Player Amount

When deploying the identification model in the real world, scalability is essential. It is crucial to know how many players the model can distinguish. This experiment evaluates the correlation between the number of players that should be distinguished and the model's performance. Since multiple players are distinguished at once, this experiment is limited to the identification mode. Multiple identification models are trained with player amounts ranging from 10 to 130. 130 players are chosen as an upper limit since the 130th player is the last one with enough sequences for training and testing. The training or testing amounts could have been reduced, but consistency of the amounts to the other experiments are prioritized. If a fixed number of training sequences are used for each player, the training data set grows proportionally to the player number. In contrast to the proportional growth of

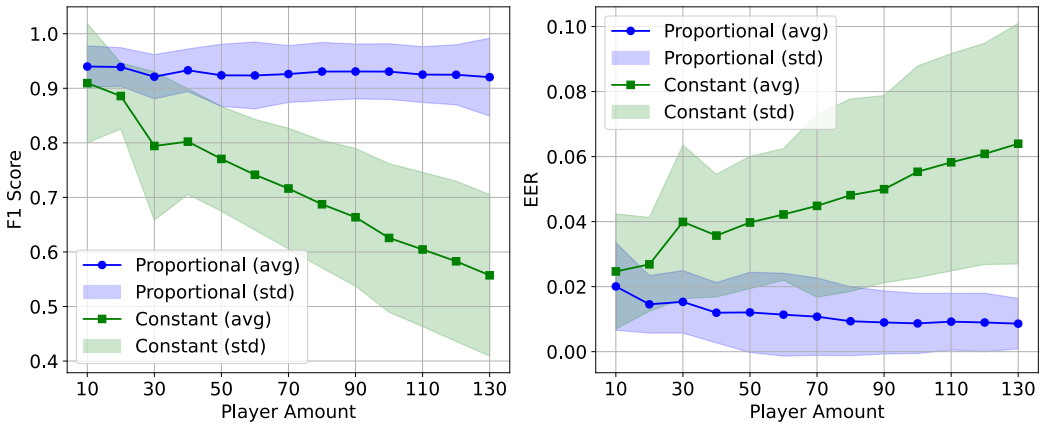

Figure 5.6: The influence the player amount has on the model's performance regarding the F1 score and EER. The lines represent the average and the areas around the lines the standard deviation. The proportional graph uses 6,000 training sequences per player and the constant one 60,000 sequences with 2,048 long sequences.

the training data set, another approach is considered in which the amount of the training data set remains constant. The model is provided with the same amount of training data in the constant data set despite the player increase. The names of the approaches are proportional and constant. The proportional approach uses 6,000 training sequences per player, and the constant approach limits the training data set to 60,000 sequences. Both approach models are evaluated using 100,000 testing sequences. The training is repeated eight times with a repetition offset of 80 min. The models are trained and tested with 1,024@32 sequences.

As illustrated in Figure 5.6, with an increasing amount of players to distinguish from, the average F1 score decreases initially for the constant approach from 0.909 ($\pm$ 0.109) to 0.557 ($\pm$ 0.148). However, in cases where the training data set grows proportionally to the number of players, the F1 score of around 0.93 plateaus at least until 130 players are distinguished. Since distinguishing multiple players, the Rank5Accuracy is also reported. This metric measures how often the player is reported as one of the top five model's predictions. The Rank1Accuracy and Rank5Accuracy are depicted in Figure 5.7. The standard deviation is lower than the one from the F1 score since each model reported the average RankNAccuracy per evaluation, but the models report the F1 score per player. Thus, the outliers that increase the standard deviation are not given and, therefore, are not visualizable. Compared to the more stringent Rank1Accuracy, the Rank5Accuracy is a less strict metric. As a result, the scores it produces are generally better than those of the Rank1Accuracy or F1 score. In the case of the proportional approach, the Rank5Accuracy is around 0.98. Under the constant approach, the Rank5Accuracy decreases to 0.562 ($\pm$ 0.013). This decrease, while not as significant as that of the Rank1Accuracy or F1 score, still indicates a

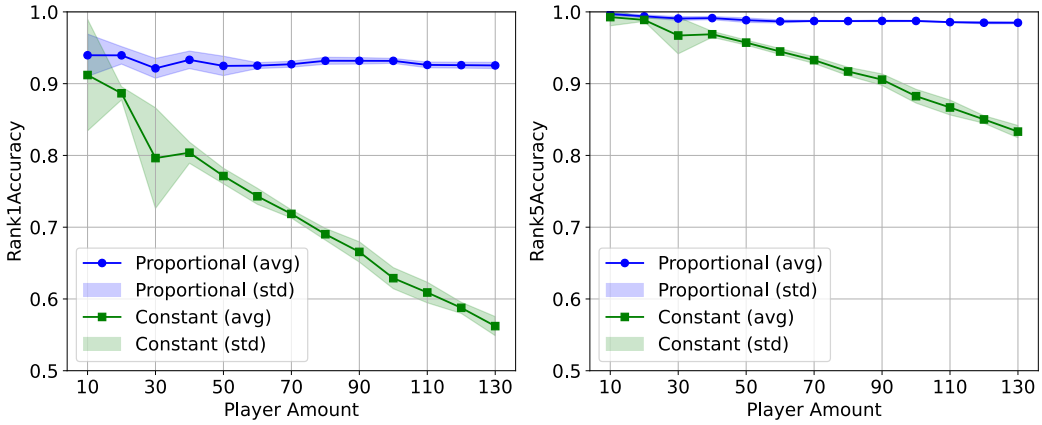

Figure 5.7: The influence the player amount has on the model's performance regarding the Rank1Accuracy and Rank5Accuracy. The lines represent the average and the areas around the lines the standard deviation. The proportional graph uses 6,000 training sequences per player and the constant one 60,000 sequences with 2,048 long sequences.

change in the model's performance. However, the decrease is less significant than the Rank1Accuracy or F1 score. More detailed results can be found in Table A.6 and Table A.7.

5.6 Match Prediction

Since a player typically uses one account during a match, the sequences from one match can be grouped to determine if they were from the same player. Predicting by match changes the recognition question from "Who played during this sequence?" to "Who played during this match?". This experiment is performed for identification. For this experiment, the testing data is grouped by player and match. Then, all sequences from a player in a match are predicted. All predictions (one-hot encoded vector) are summed up and transformed with the softmax activation function. This results in a single one-hot encoded vector per player and match. The metrics are then calculated based on these results.

Training and testing utilizes 1,024@32 sequences. The identification models are trained with 6,000 sequences per player and evaluated using 2,000 and 8,000 sequences per player. These two amounts are chosen to reflect the typical amount of testing sequences (2,000) and a bigger one (8,000) to show the stability. Each model is trained to distinguish 50 players, resulting in 100,000 and 400,000 sequences for the evaluation. The training for both sequence amounts is repeated eight times with a repetition offset of 80 min. The reported metrics are calculated based on 800,000

Table 5.8: The influence grouping sequences per play and match has on the model's performance. The first row 2,000 testing sequences are grouped per match and player and the second row 8,000 are grouped.

	F1 Score	**AUC**	**EER**
2000	0.990 (± 0.045)	0.997 (± 0.002)	0.001 (± 0.004)
8000	0.980 (± 0.039)	0.993 (± 0.001)	0.005 (± 0.014)

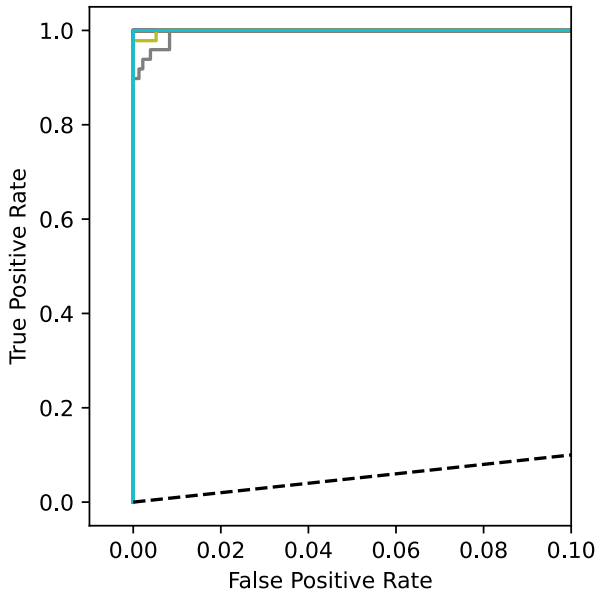

Figure 5.9: The ROC curve for each of the 50 players when grouping all sequences by match and player. The black dotted line represents random guessing.

and 3,200,000 sequences. The results are displayed in Table 5.8. Grouping improves the average F1 score over predicting a single sequence by 0.065 to 0.990 (± 0.045). Additionally, the EER also decreased to 0.001 (± 0.004). For an exact comparison, the values from Table A.5 can be used.

Besides improving the F1 score, the AUC is also improved from 0.990 (± 0.002) to 0.997 (± 0.002) when predicting by player and match. Figure 5.9 displays the ROC curve when predicting 2,000 sequences for each of the 50 players. In contrast to Figure 2.7, the x-axis is shortened to make some of the 50 curves visible.

6 Discussion

The proposed system has been described and evaluated in previous chapters. The gathered results build a foundation to discuss how the proposed system can be deployed into the real world in Section 6.1. Section 6.2 discusses the known limitations of the system and identifies areas for improvement. Section 6.3 compares the proposed system with existing approaches related to mouse movement and player recognition. This chapter ends with some ethical considerations since Personally Identifiable Information (PII) are used in Section 6.4.

6.1 Implications for a Real-World Deployment

The experiments' results described in Chapter 5 are the foundation for discussing a real-world deployment.

The more time a sequence covers, the more accurate the model's prediction is. This result is also intuitive since more data usually results in better model performances. However, one 4,096 sequence could be split into two 2,048 sequences. This may be better in regards to data efficiency. Even if plenty mouse movements are available, they are still not available endlessly. The concept of sequence duration is introduced to better explain the problem of not extracted mouse sequences and therefore data efficiency. In the case of a snapshot rate of 128Hz, the sequence duration is equal to the sequence length divided by 128. Therefore, a sequence with a length of 4,096 is 32 s long. If a player played 50 s in a round and the sequence duration is 32 s, 18 s of mouse movement is lost because the remaining 18 s do not entirely fill a new sequence. If a sequence duration of 16 s is used, only 2 s are lost. Due to the lost mouse movements, sticking to lower sequence lengths may also be useful. This creates a trade-off between improved model performance with longer sequences and reduced training data availability. Since the created database has enough 4,096 long sequences, they are selected for recognition.

The second experiment showed that snapshot rates below 128 Hz help the model to predict sequences. Reducing the sequence length but keeping the sequence duration constant increases the model's average F1 score and a substantially lowered standard deviation. When using 4,096 long sequences as a starting point, a snapshot rate of 32 Hz was found to be the best. Hence, the 1,024@32 sequences were used and recommended for further experiments. A reduced sequence length also results in

fewer trainable parameters as presented in Table 3.7. This leads to fewer required training sequences and fewer computations. The savings on storage and computation lead to less time and money spent on training these models, which in turn helps their scalability. One reason for this behavior could be that on average 65% (±11%) of the velocities in the sequences are 0. This indicates that at a 128 Hz snapshot rate, the mouse is stationary for the majority of the time. Since adding a zero and a non-zero value results in the non-zero value itself, this reduces the number of trainable parameters but keeps almost all information about the velocities. However, the frequencies of pauses are altered by this. Since grouping four values at a time is the experiment's best option, the frequency change does not seem to harm the model's performance.

The third experiment evaluated the required sequence amount to train a model to recognize players by their behavior in the following 17.8 h (2,000 · 1,024@32 sequences) of mouse movement. As expected, the more training sequences are used, the better the model performs. One reason might be that a model can find more patterns and can refine found patterns. An interesting observation is that as the number of training sequences increases, the difference in F1 score between the identification and verification models decreases. The model may improve when using more than 9,000 sequences for training. However, it seems to have reached saturation and diminishing returns. Therefore, a training sequence between 5,000 and 9,000 should be used to still achieve good performance without wasting data. Another interesting observation is that the Equal Error Rate (EER) of the identification models are lower than the one from the verification models despite the verification models achieving better F1 scores. One reason might be that the verification model is in disadvantage due to the open-set setup. Thus, the players are unknown during the evaluation. However, this should also impact the F1 score. Another reason might be that the identification models learn all players' patterns in contrast to verification models. This makes it easier to distinguish multiple players in comparison to only one player, as the verification models do. The same behavior regarding the EER can be seen in the fourth experiment where the permanence is studied.

The results of the fourth experiment give an insight into how stable captured patterns are. The verification models seem to capture patterns better since the performance drop is not as significant as for identification models. Since the verification and identification models are based on the same base model, the verification model might have more internal states to store new and past patterns and recognize a player. When more players are distinguished at a time, even more changes in pattern occur, which requires retraining the model. When predicting 6,000 sequences (53,3 h) of the next mouse movements, the identification model has an F1 score of above 0.9 which is still acceptable as an indication of suspicious behavior. 53,3 h might sound small, but this amounts to the time of consecutive mouse movement. The time difference between the recording of the first and 6,000th sequence, is on average, 163.4 d (±105.4). This time of permanence could be due to the biometric characteristic itself or updates to the game. If the game is updated, the players adapt their

gameplay to win under the new circumstances. Although the permanence regarding the biometric characteristic itself cannot precisely be determined, in the context of Counter-Strike: Global Offensive (CS:GO), the mentioned time can be viewed as permanence since the model requires retraining to hold an F1 score above 0.9. All in all, some biometrics, like fingerprints, change much slower than every hundred days.

The next experiment is crucial due to the required scalability of a real-world deployment. Since the goal is to distinguish 24 million players, scalability regarding the player amount is essential. When deploying a model, training one model to distinguish every player would be optimal. This can be done based on the scale of active pro players participating in a tournament. Since one identification model can distinguish 130 players, the plateauing indicates that the current approach can distinguish even more players. In the last big tournament, 24 teams competed against each other, and five players played for each team [38]. This results in 120 players, which is less than the 130 distinguished players in this thesis. Since 24 million players are significantly more than the evaluated player amount, no research-based answer can be given regarding the possibility of player recognition. The player amount of 24 million can be divided by continents since a player usually does not move between continents, and using a Virtual Private Network (VPN) to another continent would drastically increase the ping, which makes the game almost unplayable. The players could be even more divided, but this might too granular such that the divisions might not be clearly disconnected. The global CS:GO player distribution would be required to discuss more about dividing players.

On average, 36.79 sequences are recorded per player per match. Since the player who plays on an account usually does not change during one match, this information can be leveraged. As described in the fifth experiment, all grouped predictions are combined into a single prediction. This results in an even better F1 score of up to 0.990 ($\pm$ 0.045) with an EER of 0.001 ($\pm$ 0.004). These results seem to be good and hold quite well against the current state of research as discussed in Section 6.3. One may argue it is intuitive that this kind of grouping increases the model's performance. Although the sequences are not independent and identically distributed, the binomial distribution behaves similarly. The crucial point for the distribution is an accuracy better than random guessing. Since the F1 score for a single sequence is up to 0.941 ($\pm$ 0.051), it is significantly higher than random guessing, which would result in an F1 score of 0.02 when distinguishing between 50 players. Grouping is only performed for a single experiment to produce more transparent results. This enables future research to better compare the model results per sequence to their own, since grouping may be impossible due to smaller data sets. In the case of this thesis, the data is available thus the results will be reported. Moreover, this thesis' approach wants to recognize players by their matches, not just by the sequences they play. Although this would result in longer recognition times because more data needs to be recorded before recognition, the better performance outweighs the higher time to

recognition. Furthermore, some of the other papers from Section 2.4 use grouping via majority voting.

Another thought about scalability is to deploy one recognition model to every gaming device. To deploy a system like this in the real world, it would be beneficial to add co-processors to gaming devices that predict the live recorded mouse movement. This would significantly reduce the problem of scalability regarding the required computing power. However, this is a complex task since the model has to be trained and distributed among thousands of devices. If this is possible, the required hardware would need to be assessed. The necessary Central Processing Unit (CPU), Random Access Memory (RAM), and power draw must be determined before implementing this in a gaming device. As a first step, the performance of different Raspberry Pi models could be evaluated while predicting sequences. Other options could be Arduinos or other microcontrollers.

Based on all these experiments, it can be said that potential smurfs or boosters can be found via a real-world deployment. To recognize players, a snapshot rate of 32 Hz should be used in combination with sequences with a duration of 32 s. 6,000 training sequences per player seem to be a good trade-off between the usage of data and the models' performance. Moreover, the grouping by player and match should also be applied. However, the performance might still not be enough to detect a smurf or booster confidently after a single match.

6.2 Limitations

Although the reported model's performance is good compared to the current state of research (see Section 6.3), some limitations exist. First, the proposed system should use the modular biometric system as presented in Section 2.2. This would enable the proposed system to classify more players simultaneously and not require retraining, as it does now. Since retraining for every new player is costly, it is encouraged to use the modular approach. The modular approach can enable the system to distinguish thousands of players instead of up to 130 due to storing the templates explicitly for matching them. This is essential because CS:GO is played by 28 million monthly players.

The proposed model only distinguishes between 130 pro players. Most of the 28 million monthly players are no experts of the game. Therefore, the system may be unable to distinguish 130 casual players. The pro players may play more consistently due to constant training than casual players. Therefore, the pattern-to-noise ratio might be too low to extract patterns accurately. Another aspect is that if the pro players play more consistently and therefore more similar to each other, it could be more challenging for a model to distinguish between the players. Only testing with casual players can give more insights into this.

Another potential limitation is the use of too many training sequences. The model yields many trainable parameters, which require a high amount of training sequences. Since the data is available for CS:GO, this is no problem for this application, but it might be for other applications if the same model architecture is used without any transfer learning.

This thesis does not limit the performance a model has to achieve to use its predictions as a foundation for countermeasures. This limit must be determined before the model is deployed to the real world. For example, if a model with an F1 score of 0.999 is deployed and classifies one match from 1 million players per day, still 1,000 matches would be incorrectly classified. Depending on the measures taken, 1,000 players per day would be banned due to a model's mistake. This would decrease the players' trust in the game and they would stop or not even start playing it. Therefore, these limits have to be discussed and evaluated. A single match might be too little evidence to ban a player, but this example shows the potential problem.

As mentioned in Section 3.2, the effective Dots per Inch (eDPI) is implicitly included in every mouse movement. It could be possible that the models predict solely or mainly based on the eDPI instead of the mouse movement patterns. To assess this, a quick experiment was concluded where the sequences were multiplied with a value between 0.8 and 1.2. The model could still recognize players, but the F1 score was decreased. Even though it was a small-scaled test, this leads to the assumption that the model does not predict solely based on the eDPI. As discussed in Section 6.1, the required hardware for a distribution recognition system must be determined.

One advantage of mouse movement is also a disadvantage: the ease of collectability. An attacker could hijack a website or application and record the mouse movement via injected code. This recorded data can train a model to replicate the user's mouse movement. This model could then be used to fool a biometric recognition system.

6.3 Comparison to Current State of Research

The papers summarized in Section 2.4 are split into mouse movement-only and player recognition papers. The mouse movement papers use only binary classifiers, which resembles the first research question "How effective are mouse movements for verifying player identities in the game CS:GO?". This thesis achieved an average F1 of up to 0.990 ($\pm$ 0.045), an average Area Under Curve (AUC) of 0.997 ($\pm$ 0.002), and an average EER of 0.001 ($\pm$ 0.004). Since only some papers reported every metric, the metrics are partially compared. When recognizing a single sequence, the AUC is better than in all papers that mention the AUC. The EER of 0.009 ($\pm$ 0.009), is worse compared to the paper from Gamboa and Fred [20] with 0.002. In the case of match prediction, this thesis managed to record better overall results

Table 6.1: Overview of verification and identification performances on mouse movement data for player recognition. The bold numbers are the best per category.

Paper	Accuracy	AUC	EER
Aksari et al. 2009 [1]	-	-	0.059
Antal et al. 2021 [4]	-	0.940	0.150
Conti et al. 2020 [11]	0.963	-	-
Gamboa et al. 2004 [20]	-	-	0.002
Garabato et al. 2022 [22]	0.919	0.962	-
Liu et al. 2013 [39]	0.876	-	-
Siddiqui et al. 2021 [48]	0.616	-	0.397
Silva et al. 2018 [49]	0.863	-	-
Wang et al. 2023 [54]	0.784	-	-
Zheng et al. 2011 [57]	-	-	0.013
This thesis - Identification	0.941	0.993	0.009
This thesis - Verification	0.957	0.987	0.036
This thesis - Match Prediction	**0.990**	**0.997**	**0.001**

comparatively. The papers usually do not mention any cross-validation or repeated training, as in this thesis. Furthermore, no session split was mentioned in the other papers. This makes this work stand out regarding the information provided to reproduce these results and the performance. On the one hand, the training data used is far more extensive than those from the current state of research. On the other hand, this thesis also predicts the future in more depth, ranging up to several days of continuous mouse movement. This goes deeper into the future than the papers' training data set contains samples. Furthermore, most papers used hand-crafted features, which scaled poorly with increasing sequence lengths due to more complex patterns. Table 6.1 compares the related work to the achieved performance of this thesis.

The second part in Section 2.4 covers player recognition. Mouse movement data and keyboard usage are extracted while playing video games to recognize players. Except for Conti and Tricomi [11], all papers also used binary classification. Here, the accuracies range from 0.61 to 0.86. Based on the evaluation of this thesis, it is also better than these papers.

Conti and Tricomi [11] trained one identification model which resembles the second research question of this paper "How effective are mouse movements for identifying player identities in the game CS:GO?". They achieved an accuracy of 0.96 while

distinguishing 50 players in Dota 2. They fused keyboard and mouse usage to achieve these results for 120 s long sequences. Furthermore, they used the first 10 min of each match to lower the variance in the training data because during a match, the variance increases. This thesis achieved an average F1 score of up to 0.925 (± 0.053), an average AUC of 0.990 (± 0.002), and an average EER of 0.012 (± 0.012) when identifying layers by a single sequence. In the case of identifying a single sequence, this work could not improve the work done by Conti and Tricomi. However, this thesis match prediction approach outperforms their paper.

Based on demos hosted by Half-Life Television (HLTV), the newly composed database yields over 69 years of mouse movement from 15,523 players. The most used mouse movement data set, Balabeit [19], comprises 20 users. Since the mouse movement of 20 users is recorded over several days instead of years, this data set is much smaller than the one from this thesis. Another data set called SapiMouse [4], also used in related work, consists of 120 users with 4 min of mouse movement. This is also far less than the newly composed data set.

6.4 Ethical Considerations

Ethical concerns mainly regard the collected data in the form of demos. Before collecting the demos, one of the website administrators was asked if they could be used for a university project, and he permitted it. Although the demo distributor grants permission, no direct consent is collected from the players. It is also unknown if the players consented to the tournament organizers for sharing the demos they are part of. Furthermore, even if consent for sharing is given, it is unknown if this includes automatic processing of this data. Since all demos are publicly accessible and the players in them are (semi)professional players, it seems that recognizing these is not as privacy intrusive. The demos contain two PII. The first one is the mouse movement itself. An attacker could train one model on the data from this work and then download demos from HLTV and simply classify these to find the player. Additionally, the data set containing the mouse movements also contains the Steam ids, which are unique identifiers for a player's Steam profile. Since (semi)pro players are publicly known, the Steam profile can be traced back to the natural person. The Steam ids can be removed from the data set for privacy purposes. However, the demos also contain the Steam ids, which are challenging to remove. The removal would only affect the already downloaded demos, not those uploaded by HLTV.

A potential problem with this data set is that it could be used to train a model to detect which players develop health issues related to cognitive abilities or hand/arm injuries. It could be used to train a model to detect carpal tunnel syndrome or if a player has early signs of Parkinson's disease. These are only hypotheses, but this could be possible with more advanced research in Machine Learning (ML).

Considering that the players considered in this thesis were professional players and that health issues could be detected via mouse movement, there is a risk that rumors or information about their health could circulate on social media without their consent. Discussing health concerns on social platforms may heighten the pressure on players dealing with potential health issues. Therefore, there exist some potential cases of misuse.

7 Conclusions and Future Work

To answer the research questions "How effective are mouse movements for verifying player identities in the game Counter-Strike: Global Offensive (CS:GO)?" and "How effective are mouse movements for identifying player identities in the game CS:GO?", a novel data set was created and Deep Learning (DL) models were trained and evaluated. The novel data set is composed of CS:GO demos which are publicly accessible on the website Half-Life Television (HLTV). Mouse movements are extracted from the demos, stored in a database, and used to train models. These models are then evaluated under various experimental conditions.

Both recognition models are trained to recognize a player based on a single sequence of mouse movements. The verification model achieved an average F1 score of up to 0.957 ($\pm$ 0.026) with an Equal Error Rate (EER) of 0.036 ($\pm$ 0.020). The identification model achieved an average F1 score of up to 0.941 ($\pm$ 0.051) with an EER of 0.009 ($\pm$ 0.009). These results are rated as effective due to their high F1 score and low EER which demonstrate substantial improvements over to the current state of research. Notably, the grouping of sequences from a single player and match resulted in an average F1 score of 0.990 ($\pm$ 0.045) with an EER of 0.001 ($\pm$ 0.004), surpassing the performance of single-sequence predictions.

Although the proposed models outperform the current state of research, they may need to perform even better to be deployed in a real-world application to detect smurfs and boosters without human intervention. However, a potential application scenario involves using the models to flag suspicious players for manual review.

The extensive evaluation confirmed that the patterns are useable for recognition up to several days of continuous mouse movement. Furthermore, reducing the snapshot rate improved performance. As a side-effect, this leads to a more efficient system regarding hardware requirements. Moreover, the model's F1 score stays around 0.93 even when identifying up to 130 players simultaneously.

The novel usage of the demos can inspire other researchers to build upon this data set and develop even more powerful approaches to recognize individuals by their mouse movements. This is not only restricted to recognizing players from the game CS:GO, but can also be used to recognize players from other games or users of other applications.

7.1 Future Work

Although the reported system can recognize players, there is still room for improvement and some research fields have not been touched during this thesis. Further research could explore other Machine Learning (ML) architectures that might achieve better results. For example, an ensemble model could be tested, Convolutional Neural Network (CNN) layers with different filter sizes, or stacking even more CNN layers like ResNet [25] does. In addition to architectural changes, embedding learning can be studied with this novel data set. Embedding learning focuses on extracting features from the input such that the resulting embeddings minimize intra-class variation and maximize inter-class variation. This is important when deploying the proposed approach as a modular biometric system, which is also an open task.

The data can also be improved. Some kind of sample quality assessment like isolation forest or SER-FIQ [51] can be employed to detect and remove samples of bad quality. Bad quality means samples which contain noise or do not represent the individual well. This could increase the robustness and performance of the model. Furthermore, other preprocessing methods, like Principal Component Analysis (PCA), can be applied to reduce the dimensionality of the input data, thus reducing the amount of trainable parameters.

This thesis only focussed on time as a criterion to divide mouse movements into sequences. Other ideas are using only mouse movements, starting when an enemy is seen until one of the players die. This ensures that the sequence contains lots of movement. Currently, many sequences include periods of minimal mouse movement, such as when players are waiting for enemies. Another option would be only using sequences from one map.

Another potential source of demos is the matchmaking platform Faceit. This platform does not publicly offer demos, but it stores much more demos compared to HLTV. Moreover, these are from casual players instead of pro players. Using these demos as a data source would better reflect real-world casual players who do not play the game as a profession. This source also enables the recognition of more players at once and the evaluation of this approach.

Since mice are usually combined with keyboards, the data set could be extended with keyboard usage and fuse the information both sensors offer. Then, other games like Valorant could be considered. The developers behind Valorant announced a replay feature like CS:GO offers with demos. Different genres could also be explored, like the strategy and card game Hearth Stone. Another game could be Pokemon Go, where players are incentivized to move outside and use the touchscreen of their digital device. Here, the gait could be combined with some sort of finger movement to achieve good recognition results.

Besides improving the reported approach, the idea behind this thesis can in general be used to detect cheaters in games. It might also be used as a new rating system based on more properties such as the win-loss ratio. Therefore, the skill might be determined by more characteristic like reaction time, precision, or more. As discussed before, the required hardware of an embedded system must be determined to deploy the system or a variation thereof to gaming devices.

Bibliography

[1] Yigitcan Aksari and Harun Artuner. "Active authentication by mouse movements". In: *2009 24th International Symposium on Computer and Information Sciences*. 2009, pp. 571–574. DOI: 10.1109/ISCIS.2009.5291887.

[2] Bashira Akter Anima et al. "User Authentication from Mouse Movement Data Using SVM Classifier". In: Nov. 2016, pp. 692–700. ISBN: 978-3-319-48964-3. DOI: 10.1007/978-3-319-48965-0_47.

[3] Margit Antal and Norbert Fejér. "Mouse dynamics based user recognition using deep learning". In: *Acta Universitatis Sapientiae, Informatica* 12 (July 2020), pp. 39–50. DOI: 10.2478/ausi-2020-0003.

[4] Margit Antal, Norbert Fejér, and Krisztian Buza. "SapiMouse: Mouse Dynamics-based User Authentication Using Deep Feature Learning". In: *2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. 2021, pp. 61–66. DOI: 10.1109/SACI51354.2021.9465583.

[5] S.A. ANYBRAIN. *Anybrain Website*. URL: https://anybrain.gg.

[6] Gabriela Cantarero, Ashley Lloyd, and Pablo Celnik. "Reversal of Long-Term Potentiation-Like Plasticity Processes after Motor Learning Disrupts Skill Retention". In: *The Journal of neuroscience : the official journal of the Society for Neuroscience* 33 (July 2013), pp. 12862–9. DOI: 10.1523/JNEUROSCI.1399-13.2013.

[7] Runjin Chen et al. "Explaining Neural Networks Semantically and Quantitatively". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9186–9195. DOI: 10.1109/ICCV.2019.00928.

[8] J. Clement. *Number of video game users worldwide from 2019 to 2029*. Accessed: 2024-06-16. URL: https://www.statista.com/statistics/748044/number-video-gamers-world.

[9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *Under Review of ICLR2016 (1997)* (Nov. 2015).

[10] Eoin Conroy et al. "Boosting: Rank and skill deception in esports". In: *Entertainment Computing* 36 (2021), p. 100393. ISSN: 1875-9521. DOI: https://doi.org/10.1016/j.entcom.2020.100393. URL: https://www.sciencedirect.com/science/article/pii/S1875952120301014.

[11] Mauro Conti and Pier Paolo Tricomi. "PvP: Profiling Versus Player! Exploiting Gaming Data for Player Recognition". In: Nov. 2020, pp. 393–408. ISBN: 978-3-030-62973-1. DOI: 10.1007/978-3-030-62974-8_22.

[12] Valve Corporations. *Counter Strike 2*. Accessed: 2024-06-16. URL: https://www.counter-strike.net.

[13] Valve Corporations. *Counter-Strike 2*. Accessed: 2024-07-04. URL: https://developer.valvesoftware.com/wiki/Counter-Strike_2.

[14] Valve Corporations. *Counter-Strike: Global Offensive*. Accessed: 2024-07-04. URL: https://developer.valvesoftware.com/wiki/Counter-Strike:_Global_Offensive.

[15] Valve Corporations. *Creating a Classic Counter-Strike Map*. Accessed: 2024-07-04. URL: https://developer.valvesoftware.com/wiki/Creating_a_Classic_Counter-Strike_Map.

[16] Counter-Strike Fandom. *Counter-Strike Wiki*. URL: https://counterstrike.fandom.com/.

[17] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Chaudhuri. "Activation Functions in Deep Learning: A comprehensive Survey and Benchmark". In: *Neurocomputing* 503 (July 2022). DOI: 10.1016/j.neucom.2022.06.111.

[18] Fjedjik. *2023 Year in Review*. Accessed: 2024-07-04. URL: https://csgocasetracker.com/blog/2023-Year-Review.

[19] Á. Fülöp et al. *Balabit Mouse Dynamics Challenge data set*. URL: https://github.com/balabit/Mouse-Dynamics-Challenge.

[20] Hugo Gamboa and Ana Fred. "A behavioral biometric system based on human-computer interaction". In: *Proc SPIE* 5404 (Aug. 2004), pp. 381–392. DOI: 10.1117/12.542625.

[21] Riot Games. *VALORANT Systems Health Series - Smurf Detection*. URL: https://playvalorant.com/en-us/news/dev/valorant-systems-health-series-smurf-detection/.

[22] Daniel Garabato et al. "AI-based user authentication reinforcement by continuous extraction of behavioral interaction features". In: *Neural Computing and Applications* 34 (July 2022), pp. 1–15. DOI: 10.1007/s00521-022-07061-3.

[23] Go. *Go*. Accessed: 2024-05-25. URL: https://www.go.dev/.

[24] *hardware virtualization to spoof*. Accessed: 2024-06-16. URL: https://www.unknowncheats.me/forum/anti-cheat-bypass/605833-hardware-virtualization-spoof.html.

[25] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[26] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *IEEE International Conference on Computer Vision (ICCV 2015)* 1502 (Feb. 2015). DOI: 10.1109/ICCV.2015.123.

[27] Samuel Heaney, Richard Wilcox, and Shailyn Cotten. *Valorant Ranks: Ascendant, Immortal, Radiant Explained*. Accessed: 2024-06-12. URL: https://www.ign.com/wikis/valorant/Valorant_Ranks:_Ascendant,_Immortal,_Radiant_Explained.

[28] HLTV.org ApS. *HLTV.org*. URL: https://www.hltv.org/.

[29] IBM. *What are convolutional neural networks?* Accessed: 2024-05-25. URL: https://www.ibm.com/topics/convolutional-neural-networks.

[30] IBM. *What is explainable AI?* Accessed: 2024-05-25. URL: https://www.ibm.com/topics/explainable-ai.

[31] IBM. *What is regularization?* Accessed: 2024-05-25. 2023. URL: https://www.ibm.com/topics/regularization.

[32] ISO Central Secretary. *Information technology — Vocabulary*. en. Standard ISO/IEC TR 2382-37:2022. Geneva, CH: International Organization for Standardization, 2022. URL: https://www.iso.org/standard/73514.html.

[33] Anil K Jain, Arun A Ross, and Karthik Nandakumar. *Introduction to Biometrics [Elektronische Ressource]*. Boston, MA : Springer Science+Business Media, LLC, 2011. ISBN: 9780387773261. DOI: 10.1007/978-0-387-77326-1. URL: https://dx.doi.org/10.1007/978-0-387-77326-1.

[34] Taeho Jo. *Deep Learning Foundations*. 1st ed. 2023. Cham : Springer International Publishing, 2023. ISBN: 9783031328794. DOI: 10.1007/978-3-031-32879-4. URL: https://dx.doi.org/10.1007/978-3-031-32879-4.

[35] Simon Khan et al. “Mouse Dynamics Behavioral Biometrics: A Survey”. In: *ACM Comput. Surv.* 56.6 (2024). ISSN: 0360-0300. DOI: 10.1145/3640311. URL: https://doi.org/10.1145/3640311.

[36] Haerin Kim et al. “Justice League: Time-series Game Player Pattern Detection to Discover Rank-Skill Mismatch”. In: *2022 IEEE International Conference on Agents (ICA)*. 2022, pp. 42–47. DOI: 10.1109/ICA55837.2022.00014.

[37] Liquidpedia. *List of player mouse settings*. Accessed: 2024-05-25. URL: https://liquipedia.net/counterstrike/List_of_player_mouse_settings.

[38] Liquidpedia. *Perfect World Shanghai Major 2024*. Accessed: 2024-07-09. URL: https://liquipedia.net/counterstrike/Perfect_World/Major/2024/Shanghai.

[39] S. Liu, C. Ballinger, and S.J. Louis. “Player identification from RTS game replays”. In: *28th International Conference on Computers and Their Applications 2013, CATA 2013* (Jan. 2013), pp. 313–318.

[40] Pier Luigi Mazzeo and Paolo Spagnolo. *Deep Learning Applications*. Ed. by Pier Luigi Mazzeo and Paolo Spagnolo. English. IntechOpen, 2021. ISBN: 9781839623769.

[41] Umberto Michelucci. *Applied Deep Learning : A Case-Based Approach to Understanding Deep Neural Networks [electronic resource]*. 1st ed. 2018. Berkeley, CA : Apress, 2018. ISBN: 1-4842-3790-0. DOI: 10.1007/978-1-4842-3790-8. URL: https://dx.doi.org/10.1007/978-1-4842-3790-8.

[42] Charles K Monge and Nicholas L Matthews. "Blaming the smurf: Using a novel social deception behavior in online games to test attribution theories". In: *New Media & Society* 0.0 (0), p. 14614448241235638. DOI: 10.1177/14614448241235638. eprint: https://doi.org/10.1177/14614448241235638. URL: https://doi.org/10.1177/14614448241235638.

[43] Sebastian Pape. "Templateless Biometric-Enforced Non-Transferability of Anonymous Credentials". In: (Jan. 2008).

[44] Dylan Patel, Daniel Nishball, and Jeremie Eliahou Ontiveros. *AI Datacenter Energy Dilemma - Race for AI Datacenter Space*. Accessed: 2024-07-16. URL: https://www.semianalysis.com/p/ai-datacenter-energy-dilemma-race.

[45] Muhammad Sarfraz. *Biometric Systems*. Ed. by Muhammad Sarfraz. Open Access, English. IntechOpen, 2021. ISBN: 9781789844665.

[46] All About Security. *Biometrie: 40% der Deutschen sind zu Fingerabdruck, Face-ID & Co gewechselt*. Accessed: 2024-07-16. URL: https://www.all-about-security.de/biometrie-40-der-deutschen-sind-zu-fingerabdruck-face-id-co-gewechselt/.

[47] Selenium. *Selenium automates browsers. That's it!* Accessed: 2024-05-25. URL: https://www.selenium.dev/.

[48] Nyle Siddiqui, Rushit Dave, and Naeem Seliya. "Continuous User Authentication Using Mouse Dynamics, Machine Learning, and Minecraft". In: Dec. 2021, pp. 1–6. DOI: 10.1109/ICECET52533.2021.9698532.

[49] Valmiro Ribeiro da Silva and Marjory Da Costa-Abreu. "An empirical biometric-based study for user identification with different neural networks in the online game League of Legends". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–6. DOI: 10.1109/IJCNN.2018.8489164.

[50] Nick Speranza. *Gaming the System: The 'boosting' industry is booming — is that good?* Accessed: 2024-06-12. URL: https://dailyfreepress.com/2021/11/02/gaming-the-system-the-boosting-industry-is-booming-is-that-good/.

[51] Philipp Terhörst et al. *SER-FIQ: Unsupervised Estimation of Face Image Quality Based on Stochastic Embedding Robustness*. 2020. arXiv: 2003.09373 [cs.CV]. URL: https://arxiv.org/abs/2003.09373.

[52] Shriram K Vasudevan. *Deep Learning*. CRC Press, 2021-12-24. ISBN: 9781003185635, 9781032028828. DOI: 10.1201/9781003185635. URL: https://dx.doi.org/10.1201/9781003185635.

[53] Markus Walther. *demoinfocs-golang - Counter-Strike 2 & CS:GO Demo Parser*. Accessed: 2024-05-25. URL: https://github.com/markus-wa/demoinfocs-golang.

[54] Yimiao Wang and Tasmina Islam. "Addressing Privacy and Security Concerns in Online Game Account Sharing: Detecting Players Using Mouse Dynamics". English. In: *International Conference on Pattern Recognition Applications and Methods*. Proceedings of the 12th International Conference on Pattern Recognition Applications and Methods - ICPRAM, Feb. 2023, pp. 864–871. ISBN: ISBN 978-989-758-626-2. DOI: DOI:10.5220/0011678300003411.

[55] *What Is SQLite?* Accessed: 2024-07-04. URL: https://www.sqlite.org/.

[56] Jacob Wolf. *XiaoWeiXiao suspended in elo boosting scheme that allegedly involved team manager*. Accessed: 2024-06-12. URL: https://dotesports.com/league-of-legends/news/xiaoweixiao-suspended-elo-boosting-2193.

[57] Nan Zheng, Aaron Paloski, and Haining Wang. "An Efficient User Verification System via Mouse Movements". In: vol. 139–150. Oct. 2011, pp. 139–150. DOI: 10.1145/2046707.2046725.

List of Figures

List of Tables

List of Listings

A Experiment Evaluation Data

This appendix contains detailed evaluation metrics for the experiments described in Chapter 5. The tables below provide a comprehensive overview of various performance metrics for the described experiments, offering more profound insights into the model's performance.

Table A.1: The influence the sequence length has on the average model's performance for identification and verification. These scores belong to models trained after the constant approach.

		Identification			Verification		
		F1 Score	**AUC**	**EER**	**F1 Score**	**AUC**	**EER**
Sequence Length	**512**	0.677 (± 0.196)	0.957 (± 0.005)	0.072 (± 0.100)	0.860 (± 0.098)	0.928 (± 0.082)	0.131 (± 0.083)
	1024	0.674 (± 0.180)	0.949 (± 0.009)	0.068 (± 0.089)	0.852 (± 0.083)	0.923 (± 0.057)	0.136 (± 0.066)
	2048	0.587 (± 0.172)	0.929 (± 0.011)	0.085 (± 0.101)	0.804 (± 0.119)	0.879 (± 0.090)	0.178 (± 0.093)
	4096	0.402 (± 0.150)	0.880 (± 0.005)	0.129 (± 0.076)	0.721 (± 0.109)	0.785 (± 0.110)	0.266 (± 0.103)

Table A.2: The influence the sequence length has on the average model's performance for identification and verification. These scores belong to models trained after the proportional approach.

		Identification			Verification		
		F1 Score	**AUC**	**EER**	**F1 Score**	**AUC**	**EER**
Sequence Length	**512**	0.678 (± 0.195)	0.958 (± 0.006)	0.072 (± 0.100)	0.859 (± 0.099)	0.927 (± 0.084)	0.131 (± 0.082)
	1024	0.778 (± 0.129)	0.976 (± 0.009)	0.039 (± 0.048)	0.885 (± 0.101)	0.947 (± 0.061)	0.103 (± 0.066)
	2048	0.819 (± 0.126)	0.968 (± 0.007)	0.037 (± 0.055)	0.915 (± 0.047)	0.963 (± 0.028)	0.079 (± 0.040)
	4096	0.850 (± 0.112)	0.970 (± 0.003)	0.027 (± 0.031)	0.922 (± 0.046)	0.967 (± 0.027)	0.069 (± 0.039)

Table A.3: The influence the snapshot rate has on the average model's performance for identification and verification. Every group size values are grouped and summed to artificially create a lowered snapshot rate. A group size of 2 corresponds to a snapshot rate of 64 Hz, when starting with a snapshot rate of 128 Hz.

		Identification			Verification		
		F1 Score	**AUC**	**EER**	**F1 Score**	**AUC**	**EER**
Snapshot Rate	**64**	0.907 (± 0.080)	0.984 (± 0.003)	0.015 (± 0.018)	0.943 (± 0.037)	0.980 (± 0.018)	0.047 (± 0.027)
	32	0.925 (± 0.053)	0.990 (± 0.002)	0.012 (± 0.012)	0.946 (± 0.034)	0.982 (± 0.015)	0.045 (± 0.025)
	16	0.911 (± 0.040)	0.993 (± 0.001)	0.014 (± 0.007)	0.938 (± 0.031)	0.979 (± 0.015)	0.056 (± 0.028)

Table A.4: The influence the amount of training sequences has on the average model's performance for identification and verification. Different models are trained based on different training amounts.

		Identification			Verification		
		F1 Score	**AUC**	**EER**	**F1 Score**	**AUC**	**EER**
Training Sequences	**1000**	0.732 (± 0.109)	0.972 (± 0.003)	0.047 (± 0.023)	0.822 (± 0.088)	0.894 (± 0.075)	0.166 (± 0.081)
	3000	0.869 (± 0.095)	0.984 (± 0.002)	0.021 (± 0.018)	0.924 (± 0.047)	0.971 (± 0.025)	0.067 (± 0.039)
	5000	0.908 (± 0.052)	0.989 (± 0.001)	0.015 (± 0.011)	0.938 (± 0.035)	0.978 (± 0.017)	0.052 (± 0.029)
	7000	0.930 (± 0.044)	0.992 (± 0.001)	0.011 (± 0.010)	0.950 (± 0.027)	0.985 (± 0.011)	0.043 (± 0.022)
	9000	0.941 (± 0.051)	0.993 (± 0.002)	0.009 (± 0.009)	0.957 (± 0.026)	0.987 (± 0.011)	0.036 (± 0.020)

Table A.5: The influence the amount of testing sequences has on the average model's performance for identification and verification. Different models are trained based on different testing amounts.

		Identification			Verification		
		F1 Score	**AUC**	**EER**	**F1 Score**	**AUC**	**EER**
Testing Sequences	**1000**	0.923 (± 0.061)	0.991 (± 0.002)	0.012 (± 0.012)	0.944 (± 0.047)	0.980 (± 0.023)	0.047 (± 0.029)
	3000	0.913 (± 0.063)	0.988 (± 0.001)	0.014 (± 0.014)	0.943 (± 0.032)	0.981 (± 0.015)	0.048 (± 0.025)
	5000	0.908 (± 0.054)	0.987 (± 0.001)	0.015 (± 0.013)	0.943 (± 0.031)	0.981 (± 0.014)	0.049 (± 0.025)
	7000	0.894 (± 0.059)	0.983 (± 0.001)	0.020 (± 0.018)	0.934 (± 0.036)	0.976 (± 0.017)	0.054 (± 0.027)
	9000	0.870 (± 0.070)	0.976 (± 0.002)	0.024 (± 0.022)	0.929 (± 0.044)	0.973 (± 0.024)	0.059 (± 0.032)

Table A.6: The influence the amount of players has on the average model's performance for identification. Part 1.

Player Amount	Constant			Proportional		
	F1 Score	**AUC**	**EER**	**F1 Score**	**AUC**	**EER**
10	0.909 (± 0.109)	0.985 (± 0.026)	0.025 (± 0.018)	0.940 (± 0.038)	0.995 (± 0.003)	0.020 (± 0.013)
20	0.886 (± 0.061)	0.988 (± 0.002)	0.027 (± 0.014)	0.939 (± 0.035)	0.994 (± 0.003)	0.015 (± 0.009)
30	0.794 (± 0.136)	0.974 (± 0.020)	0.040 (± 0.024)	0.921 (± 0.041)	0.991 (± 0.002)	0.015 (± 0.010)
40	0.802 (± 0.097)	0.981 (± 0.002)	0.036 (± 0.019)	0.933 (± 0.039)	0.993 (± 0.003)	0.012 (± 0.009)
50	0.771 (± 0.096)	0.977 (± 0.002)	0.040 (± 0.020)	0.924 (± 0.057)	0.991 (± 0.003)	0.012 (± 0.012)
60	0.741 (± 0.102)	0.974 (± 0.002)	0.042 (± 0.020)	0.924 (± 0.061)	0.991 (± 0.001)	0.011 (± 0.013)
70	0.716 (± 0.110)	0.970 (± 0.002)	0.045 (± 0.028)	0.926 (± 0.052)	0.991 (± 0.001)	0.011 (± 0.012)
80	0.687 (± 0.117)	0.966 (± 0.002)	0.048 (± 0.030)	0.931 (± 0.053)	0.992 (± 0.001)	0.009 (± 0.011)
90	0.664 (± 0.126)	0.963 (± 0.004)	0.050 (± 0.029)	0.931 (± 0.050)	0.992 (± 0.001)	0.009 (± 0.010)
100	0.626 (± 0.136)	0.955 (± 0.004)	0.055 (± 0.033)	0.931 (± 0.051)	0.992 (± 0.001)	0.009 (± 0.009)
110	0.605 (± 0.141)	0.952 (± 0.004)	0.058 (± 0.033)	0.925 (± 0.051)	0.991 (± 0.000)	0.009 (± 0.009)
120	0.583 (± 0.147)	0.948 (± 0.002)	0.061 (± 0.034)	0.925 (± 0.055)	0.991 (± 0.001)	0.009 (± 0.009)
130	0.557 (± 0.148)	0.944 (± 0.003)	0.064 (± 0.037)	0.920 (± 0.071)	0.991 (± 0.001)	0.009 (± 0.008)

Table A.7: The influence the amount of players has on the average model's performance for identification. Part 2.

Player Amount	Constant		Proportional	
	Rank1Accuracy	**Rank5Accuracy**	**Rank1Accuracy**	**Rank5Accuracy**
10	0.912 (± 0.078)	0.993 (± 0.012)	0.940 (± 0.029)	0.997 (± 0.002)
20	0.887 (± 0.009)	0.989 (± 0.002)	0.939 (± 0.012)	0.994 (± 0.002)
30	0.796 (± 0.070)	0.967 (± 0.026)	0.921 (± 0.014)	0.991 (± 0.003)
40	0.804 (± 0.015)	0.969 (± 0.004)	0.933 (± 0.012)	0.991 (± 0.002)
50	0.771 (± 0.011)	0.957 (± 0.003)	0.925 (± 0.014)	0.988 (± 0.003)
60	0.743 (± 0.011)	0.945 (± 0.004)	0.925 (± 0.004)	0.987 (± 0.002)
70	0.719 (± 0.006)	0.933 (± 0.005)	0.927 (± 0.004)	0.987 (± 0.001)
80	0.690 (± 0.008)	0.917 (± 0.006)	0.932 (± 0.005)	0.987 (± 0.001)
90	0.666 (± 0.014)	0.906 (± 0.008)	0.932 (± 0.004)	0.987 (± 0.001)
100	0.629 (± 0.015)	0.883 (± 0.010)	0.932 (± 0.003)	0.987 (± 0.001)
110	0.609 (± 0.014)	0.867 (± 0.010)	0.926 (± 0.004)	0.986 (± 0.001)
120	0.588 (± 0.008)	0.850 (± 0.005)	0.926 (± 0.004)	0.985 (± 0.002)
130	0.562 (± 0.013)	0.562 (± 0.013)	0.925 (± 0.004)	0.985 (± 0.001)