IT UNIVERSITY OF COPENHAGEN

AntiCheatPT: A Transformer-Based Approach to Cheat Detection in Competitive Computer Games

01/06/25

Mille Mei Zhen Loo
milo@itu.dk

Gert Lužkov
gelu@itu.dk

Supervisor

Paolo Burelli
pabu@itu.dk

STADS code : KISPECI1SE

IT University of Copenhagen
M.Sc. Computer Science
Thesis project

Abstract

Cheating in online video games compromises the integrity of gaming experiences. Anti-cheat systems, such as VAC (Valve Anti-Cheat), face significant challenges in keeping pace with evolving cheating methods without imposing invasive measures on users' systems. This paper presents AntiCheatPT_256, a transformer-based machine learning model designed to detect cheating behaviour in Counter-Strike 2 using gameplay data. To support this, we introduce and publicly release CS2CD: A labelled dataset of 795 matches. Using this dataset, 90,707 context windows were created and subsequently augmented to address class imbalance. The transformer model, trained on these windows, achieved an accuracy of 89.17% and an AUC of 93.36% on an unaugmented test set. This approach emphasizes reproducibility and real-world applicability, offering a robust baseline for future research in data-driven cheat detection.

Contents

Chapter 1

Foreword

All code, models, datasets can be found on the links provided in the following table:

Name	Type	Link
CS2_demo_scraper	Code	https://github.com/Pinkvinus/CS2_demo_scraper
CS2_cheat_detection	Code	https://github.com/Pinkvinus/CS2_cheat_detection
AntiCheatPT_256	Model	https://www.doi.org/10.57967/hf/5653
CS2CD	Dataset	https://www.doi.org/10.57967/hf/5654
Context_window_256	Dataset	https://www.doi.org/10.57967/hf/5656
Context_window_1024	Dataset	https://www.doi.org/10.57967/hf/5691

Furthermore, due to the frequent occurrence of gaming terminology within this paper, a glossary has been included. To maintain the readability and flow of the text for readers who may be unfamiliar with such terms, definitions are provided as footnotes at their first occurrence.

Chapter 2

Introduction

The gaming industry is one of the most rapid growing markets with a proposed CAGR[1] of 13.4% from 2023-2030. With the online gaming market taking up approximately 45% of the total gaming market share, it is perhaps unsurprising that the market for anti-cheats[2] is rather large as well, with it currently being valued at 21 Billion EUR[2][3]. Considering the scale of the anti-cheat industry, the amount of information freely available online regarding anti-cheat development is *very* limited. It is unusual for a sector of the security industry to exhibit such secrecy, as this contradicts Kerckhoffs's Principle: "*The security of a cryptosystem must lie in the choice of its keys only; everything else (including the algorithm itself) should be considered public knowledge[4].*" This lack of transparency hinders academic research and limits opportunities for collaborative advancement in cheat detection methodologies used for developing anti-cheats.

2.1 Anti-Cheats And Cheat Detection In Games

In the context of video games, an anti-cheat is a piece of software that prevents cheating[1]. Cheating in online multiplayer games constitutes a significant challenge to game developers and players alike, as it encompasses a broad range of malicious behaviours. However, classifying certain actions as cheating is a challenging task, as the definition of cheating is subjective. In the paper "Cheating and anti-cheat system action impacts on user experience", Bryan van de Ven identifies three definitions of cheating in video games as the following:
"

1. *Using additional code or software to modify the game*
2. *Abusing faulty code in the game without using additional code or software.*
3. *Performing certain actions without using or altering code at all, such as looking at your friend's screen.*

" [5, ch. 2, p. 5]

For the purposes of this paper, cheating is defined as "*the use of additional code or software to modify the game*", as this is the type of cheating anti-cheats often are designed to counteract.

Most often anti-cheats are client-side or server-side, however companies can employ both methods resulting in a hybrid anti-cheat method. Client-side anti-cheat software is installed on the user's device. There are many types of client-side anti-cheats with varying access to system layers. Historically, anti-cheats operated in user-mode, which is the outer most layer of the protection ring. This means that the software is unable to access hardware and memory, hence permission must be obtained from the system API before scanning the system. The more invasive counterpart to the user-mode anti-cheat, is the kernel level anti-cheat. This anti-cheat, as the name implies, has access to the kernel level i.e. the inner circle of the protection ring. Due to the deep access within the device hierarchy, this type of anti-cheat is very effective. However, due to the high access level that this software requires, it is possible that a bug or exploit within the software could offer malicious parties a new gateway into the players device, potentially putting the entire system at risk. Therefore, developing an effective anti-cheat is a delicate balancing act done by developers[6][7][1].

[1] Compound Annual Growth Rate.
[2] In the context of video games, an anti-cheat is a piece of software that prevents cheating[1].

As of 2025 it is worth noting that most big video game companies probably utilise kernel level anti-cheat in some capacity[8][9][10]. Server-side anti-cheat systems, in contrast, rely solely on telemetry and gameplay data collected during matches to identify irregularities. These systems are often more resilient to tampering but are limited in their access to low-level data.

2.2 Counter-Strike 2 and VAC

Valve Anti-Cheat (VAC) is the anti-cheat software developed by Valve, which is used in a variety of games such as Counter-Strike 2 (CS2), Team Fortress 2, Dota 2 etc[11]. The design of VAC, like many other anti-cheats, is intentionally obscure despite security concerns[12]. Due to the closed nature of commercial anti-cheat systems, academic research in cheat detection and open-source alternatives remain limited. Although machine learning has been applied in this field, such research is often constrained by the availability of data[13][14][15][16][17]. This scarcity of data poses a significant challenge, particularly for data intensive deep learning models.

The First Person Shooter (FPS) CS2 was selected as the basis of this project, as the game state is saved throughout a match as a repayable `.dem`-file. (This type of file is also commonly referred to as a "demo".) Furthermore, these files can be downloaded, making it feasible to extract behavioural features for analysis. This accessibility, along with the game's popularity, the well-established competitive scene, and lack of efficient cheating detection, makes it an ideal candidate for exploring machine learning-based cheat detection[10].

2.3 Motivation For Using Transformers In Cheat Detection

While machine learning has been applied to cheat detection, it is still a relatively under-explored area. Deep learning methods, and in particular transformer architectures, have seen very limited use in this field. To the best of our knowledge, there exists only one published paper that applies transformer models in behaviour based cheat detection. The reported results are promising, suggesting that transformers could be effective in identifying patterns in player behaviour. Unfortunately, this paper does not go into any architectural or implementation detail, nor does it provide access to the code or data used. Although the authors reference the original "Attention Is All You Need"[18] paper as a basis for their model, no concrete architecture, hyper parameters, nor preprocessing steps are disclosed, thereby making it impossible to assess and reproduce their results[19][20].

This lack of transparency limits the utility of the work and highlights a broader issue in the field: The combination of limited data access and closed-source practices makes it challenging to build on prior research. This thesis seeks to address that gap by developing and documenting a labelled dataset and a transformer-based model for cheat detection using data extracted from CS2 gameplay. In doing so, we aim not only to evaluate the potential of transformers for this task but also to provide a clearer, reproducible foundation for future work in this area, through the development of an open-source dataset containing data from `.dem`-files representing CS2 gameplay.

Chapter 3

Related Work

The recent advancements in large language models (LLMs), exemplified by widely recognised systems such as ChatGPT, have led to a surge in research focused on detecting artificially generated text[21]. While cheat detection has become a prominent and actively explored topic in academic contexts, the field remains comparatively underdeveloped within the domain of video games. This disparity can be attributed in part to the reliance of anti-cheat companies on security through obscurity, which limits the availability of public data and methodologies. Consequently, the body of literature on cheat detection in video games is relatively sparse, with even fewer studies addressing behavioural cheat detection specifically. This chapter therefore outlines the existing research most relevant to the present thesis.

Due to the nature of machine learning, a vast amount of data was required. Unfortunately, the search for a suitable dataset was very quickly halted due to the very evident lack of available datasets. From our research very few datasets are publicly available. Furthermore, most available datasets consist of gameplay images - often with very few details regarding the data collection and data spread. Two notable examples of such datasets include the "pubg Computer Vision Project" by the user "deer"[22] and the "Call of Duty MW2 Computer Vision Project" by the user "Roboflow Universe Projects"[23]. During our search, one dataset containing data found from game logs was found, namely the "CSGO cheating dataset" by the user "emstatsl". This dataset contains the view angles and engagement for a variety of legitimate and cheating individuals. All of this information is stored in two numpy arrays, one for the cheaters and one for the legitimate players. Unfortunately, this dataset is quite poorly documented, as there is very little information regarding what certain values represent. This lack of documentation also leaves the source and collection method of the data equally ambiguous. This notion is further supported by the Kaggle Usability score[1] being 3.75/10, where 0% of the information provided supports the credibility- or the compatibility-score[24].

As it will be apparent throughout this chapter, papers writing in a similar field also run into the problem of datasets, and mitigate it by creating their own datasets, although often times they end up being of very limited capacity[13][14][15][16][17]. Furthermore, only two papers published a dataset with one of them being the paper: "Few-shot Learning for Trajectory-based Mobile Game Cheating Detection". The dataset is available at `https://github.com/super1225/cheating-detection`. The study uses the interactions with touchscreen devices as it's primary data source. Since this paper focuses on desktop FPS-games, the provided dataset was unfortunately outside the scope of the project. The paper "XAI-Driven Explainable Multi-view Game Cheating Detection" by Tao et al. writes that data and code is available online at the link, `https://github.com/fuxiAIlab/EMGCD/tree/master`, however this repository is fairly empty and contains neither the dataset nor the code[19]. The successor to the XAI paper, on the other hand, shares their source to their dataset at `https://github.com/fuxiAIlab/GXAI/tree/master/data`. The usability of this dataset is, however, limited by the lack of sufficient documentation[20].

Due to the scarcity of publicly available datasets, there is a noticeable absence of scholarly work addressing the methodology of manual behavioural cheat detection. This gap in the literature served as the primary motivation for the survey discussed in the following chapter. However, the notion that no relevant information exists is inaccurate. While fragmented and anecdotal, dis-

[1] A Kaggle Usability score is a score given by the website kaggle.com to assess the documentation of a model or a dataset. The three metrics making up this score are completeness, credibility, and compatibility.

cussions on forums and Steam community threads occasionally offer insights, tips, and techniques related to cheat detection. Though rare, some users have also produced informal guides dedicated to this subject[25]. In addition to the threads and forum posts, users frequently express concerns regarding the current state of CS2, particularly criticising the effectiveness of the VAC system. These discussions often highlight the perceived inadequacy of VAC in detecting and removing cheaters from the game. Regarding the effectiveness of VAC, a study was conducted, exploring the effectiveness of several anti-cheat systems, with VAC nearly tying for last place. Furthermore, the prices for cheats that surpass VAC, are the lowest compared to all other game franchises[10].

The idea of fitting a machine learning model to the problem of cheaters in Counter-Strike has been attempted before. Various approaches, greatly varying in model architectures and training data, have been made. One such approach was done by Valve for their VACNet model[26].

An example of Transformer architectures applied to behavioural cheat detection was explored in the context of NetEase titles such as *Knives Out Plus* and *Justice Online*. The study titled "XAI-Driven Explainable Multi-view Game Cheating Detection" investigates this problem by training various models across four distinct data groups: Portrait view, behaviour view, image view, and graph view. The portrait view encompasses user account and hardware information, while the behaviour view captures in-game player actions such as "put on fashion clothes" and "use money". The image view is based on screenshots from the game client, enabling analysis of visual elements displayed on screen. The graph view incorporates social network structures, derived from player interactions including transactions, friendships, team formations, and chat logs. Multiple machine learning models were evaluated for each data group, with the Transformer architecture achieving particularly strong performance on the behavioural data, yielding an AUC of 0.9836 and an accuracy of 96.94%. Despite these promising results, the lack of released source code limits the reproducibility of the study's findings [19].

Another example of machine learning in cheat detection is given in the paper "Cheat Detection using Machine Learning within Counter-Strike: Global Offensive" by Harry Dunham. This paper created a dataset with 400 points of view in total, 200 of which were cheater and 200 not cheater. The data gathering was done through the Overwatch feature, which was present in Counter-Strike: Global Offensive(CS:GO), but is no longer available in CS2[27]. Overwatch was a feature, that allowed trusted players to review matches with suspected cheaters. The investigators would then come to a verdict and report whether or not the suspect was cheating or not[28]. The dataset created and used in this work was not published. A recurrent neural network with long short term memory was used for model fitting. In this work, the probability of cheating was continuously calculated by the model being given sequential data from a `.dem`-file. The sequential data consisted of time steps, where each step contained data from an in game tick. Therefore, specific metrics were calculated on a per tick basis and subsequently concatenated to create the sequential input. A baseline accuracy of 80-90% was claimed. An important observation by the author is that many of their cheater data points included spinbotting[2], which is easy to detect, as will be explored section 4.3.3. This cheat, while still existing in CS2, is no longer as prevalent as it was in Counter-Strike: Global Offensive. Additionally, metrics were derived from a k-fold cross validation set rather than a train, validation, and test set split[29].

Another attempt at fitting a machine learning model to detect cheaters is the paper "Robust Vision-Based Cheat Detection in Competitive Gaming" by Jonnalagadda et al. The study takes a different approach to the problem of cheating detection. In the paper by Jonnalagadda et al, the dataset consists of screenshots rather than data gathered from `.dem`-files. A player's screen is recorded while using cheats and not using cheats. These images make up the cheater and not cheater sets. A total of nine screen recordings were made from two different games. Then, a convolutional neural network was created for cheater classification. The paper claims a precision of 90%. While the paper acknowledges the potential of adversarial attacks, such as applying filters over gameplay footage, it does not address the potential of aim hacks[3] with no overlays, leaving open a potential method of detection avoidance[16].

The gap in the existing literature on cheating in CS2 highlights the need for further investigation.

[2]Type of cheat where trigger bot is combined with anti-aim. This cheat makes sure that the player using a spinbot is even more difficult to be shot at[28].

[3]A class of cheats involving the use of one or multiple aim and/or trigger assisting cheats. These can include but are not limited to recoil control system, trigger aim and trigger bot[28].

Due to the absence of open-source datasets, the creation of one is essential. Due to the lack of such data collection and labelling methodologies, more research is needed in order to determine the best approach. Furthermore, previous work suggests that behavioural cheat detection is possible within the realm of CS2. Additionally, the use of transformers in cheat detection in games have shown great results although they have not been applied to CS2 and no reproducibility is possible. This work is therefore structured into three key chapters: A survey designed to gain a deeper understanding of the cheating detection methodology; The development of a comprehensive open-source CS2 dataset with cheater labelling; And the design and training of a transformer-based machine learning model aimed at detecting cheating behaviour.

Chapter 4

Community Perception of Cheating in Counter-Strike 2

4.1 Introduction

While this thesis ultimately focuses on the cheat detection using transformers, it is essential for dataset creation to understand the methodologies of manual behaviour based cheat detection. Behaviour based cheat detection is the process of determining whether a player is cheating based on behavioural patterns. Manual behaviour based cheat detection was an implemented feature of the CS2 predecessor CS:GO. This feature was called Overwatch and it allowed players to manually perform behaviour based cheat detection by reviewing Overwatch cases. Players investigating these cases would download a `.dem`-file and watch the match from the perspective of the suspected cheater. Players could then submit a verdict where they were required to classify the type of cheating the suspect was displaying. Unfortunately, as of March 22, 2023 the Overwatch feature was removed and has not made a return since[27][30][31][32][33].

Due to the wide-spread practice of manual behavioural cheat detection within the Counter-Strike community, a survey was conducted to gather information regarding cheat detection methodology. To leverage participant engagement, we included an additional section with exploratory questions regarding general attitudes toward cheating in CS2.

4.2 Methodology

To explore player attitudes towards cheating and cheat detecion methodology in Counter-Strike 2 a mixed-methods online survey was used. The research in this chapter aimed at capturing both quantitative trends and qualitative insights from the CS2 community. The survey was self-designed and consisted of both closed- and open-ended questions. Questions were designed primarily to explore how cheating is detected, however some questions focus on cheating in CS2.

In order to sample answers from the CS2 community, the survey was posted on reddit.com on the top 5 Counter-Strike related subreddits in terms of number of members: r/counterstrike, r/counterstrike2, r/cs2, r/GlobalOffensive, r/csgo. The post contained a brief description of the topic and purpose of the survey. No specific inclusion or exclusion criteria were applied aside from requiring that respondents had experience playing Counter-Strike. The survey was distributed via a Google Forms link and remained open for 20 days totalling in 82 responses. Email verification was required, however when the survey ended emails where removed from answers. One response contained an email in plaintext which was redacted.

Quantitative data from closed-ended questions were analysed using descriptive statistics. Open-ended responses were analysed using thematic coding, in which similar responses were grouped into categories to identify common themes related to cheat detection methods and perceptions. The complete list of sections and questions as well as the order in which they appear can be seen in appendix A.

4.3 Survey Results

To ensure the quality of the responses, several questions regarding the participants' experience and skill levels regarding CS2 were included. Figures 4.1, 4.2, and 4.3 illustrate the participants' played hours, years of experience, and premier Elo ranking, respectively. These figures showcase, that the participants were in a wide variety of skill levels and player experience.

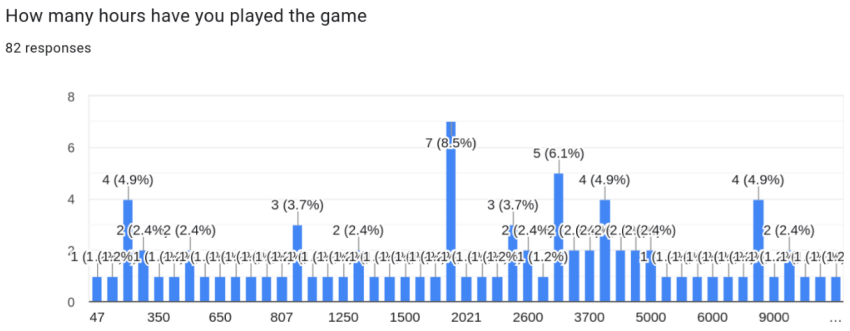

Figure 4.1: Participants played hours

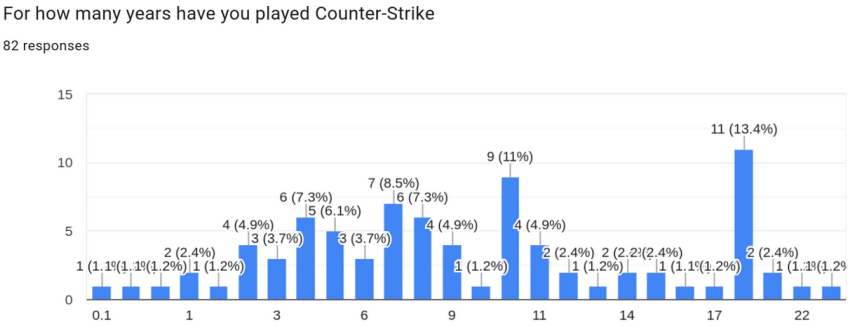

Figure 4.2: Participants played years

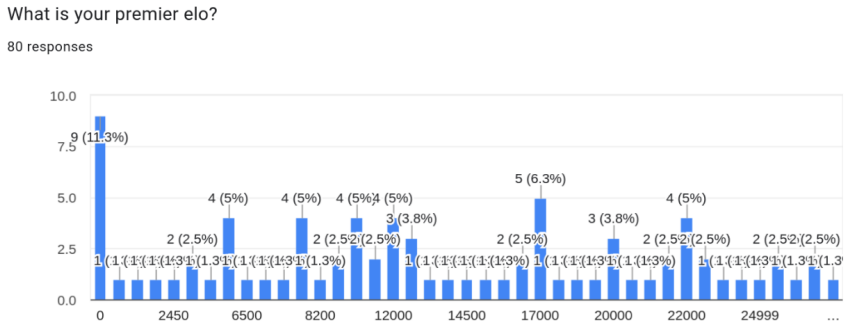

Figure 4.3: Participants premier Elo

4.3.1 Cheating as a problem

When asked "Do you believe that there is a problem with cheaters in CS2", 93.9% of participants answered "Yes" as seen on figure 4.4. When asked "Have you experienced cheaters in Counter-Strike 2?", 96.3% of participants replied "Yes" as seen on figure 4.5. When asked how often the participants encounter cheaters, the vast majority responded "occasionally", as can be seen in figure 4.6. This shows that players semi-regularly encounter cheater, although this result should be viewed critically due to the options given to the participants. For a further explanation regarding this see section 4.4. The answers to these 3 questions clearly show the sentiment that players encounter cheaters more frequently than they would like, although the phrasing of the questions leave the recency of the encounters ambiguous.

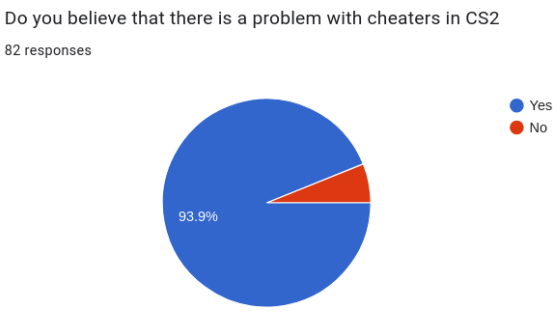

Figure 4.4: Answers to the question: Do you believe that there is a problem with cheaters in CS2

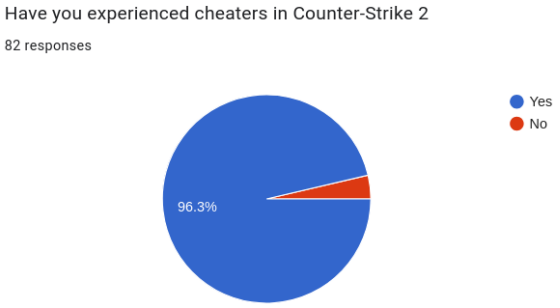

Figure 4.5: Answers to the question: Have you experienced cheaters in CS2

A thematic analysis was conducted for the question "Why do you believe cheating is a problem?" in order to identify recurring patterns and player attitudes towards cheaters. 5 main themes where identified: Unfairness, game integrity and player morale, distrust, lack of effective enforcement, Low barrier to cheating and discouragement from improvement.

Unfairness, game integrity and player morale Many participants emphasised that cheating fundamentally undermines the fairness and integrity of CS2. As a competitive game, players expect a level playing field where skill, strategy, and teamwork determine the outcome. However, respondents described how the use of cheats provides an unfair advantage, which in turn diminishes both the enjoyment and the competitive spirit of the game. Some did not express frustration solely from losing, but rather from the sense that their time, focus, and energy had been rendered meaningless. This in turn, affects the player's motivation to continue playing.

"*Cheating takes all the fun away from other players and as a 28 years old I do not really have the time to play 10 games a day anymore, therefore it makes my experience playing CS undeniably worse.*" - Participant n. 1.

"*It ruins the time commitment of 9 other people looking for a competitive experience for no reason other than a false sense of accomplishment.*" - Participant n. 61.

"*Cheaters ruin the game at higher elo than I am right now. Forcing legit better players to face me instead of people their skill. It also brings terrible vibes to the game as a whole, every play becomes suspicious and everyone accuses everyone, since nobody trust the anti cheat. As for chreating as a whole, it just gives unfair advantage, and garbage braging right*" - Participant n. 72.

"*People who put in the effort to get better at the game are discouraged from playing because they*

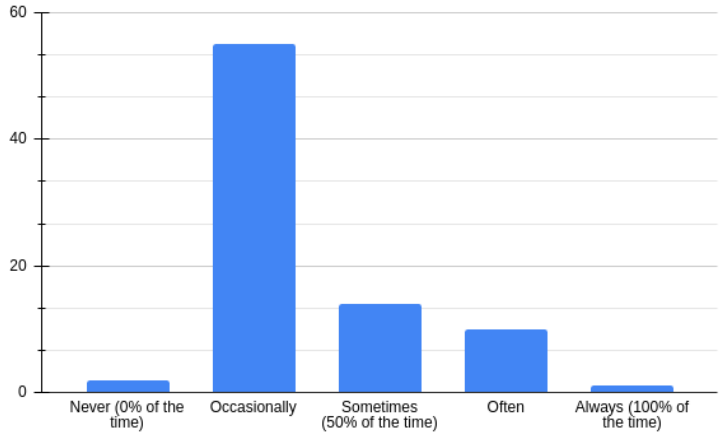

Figure 4.6: The frequency with which players encounter cheaters on a scale from 1-5, where never = 1 and always = 5.

cannot find matches that are a challenge, all they are met with is loss after loss and decide to stop playing the game" - Participant n. 64.

Other reflections underscore how cheating affects not only the fairness of matches but also the emotional and psychological connection players have with the game. It undermines the effort invested by players seeking improvement, devalues fair competition, and creates a sense of futility that can lead to player burnout or departure from the game entirely.

"*It ruins the integrity of the game. It ruins the fun of the game. It's a huge waste of time when someone is cheating*" - Participant n. 19.

"*It ruins both the fun and the competitive integrity of the game, effectively leaving it a joyless experience*" - Participant n. 73.

Distrust When answering this question 11 respondents (13.4%) mentioned that distrust and suspicion are high due to the lack of effective cheating detection methods. Some participants mention how constantly being suspicious of cheating ruins their experience.

"*Even if there is a smaller amount of cheaters, it messes with my mind and i start questioning every death and every round. Turns out after watching demo they are just good, but it ruined my experience.*" - Participant n. 43.

"*The amount of cheaters has caused people to automatically assume everyone is cheating, so often times teammates will give up on a match after losing one round.*" - Participant 28

"*People always accuse each other while playing because there is no way in which you can be sure that your opponent isn't cheating*" - Participant n. 14.

"*It happens so often and so blatant, cheaters do not have to fear consequences. You cant trust anyone anymore*" - Participant n. 33.

"*It removes the ability to trust the people/the game. Sometimes people do have a good day. As it currently is, noone belives anyone is just having a good day/hitting some insane shots. It is always assumed ppl are cheating*" - Participant n. 42.

Lack of Effective Enforcement A number of participants expressed frustration with the perceived ineffectiveness of CS2's current anti-cheat enforcement. The most commonly cited issue was the system's passivity, meaning that cheating behaviour is often not addressed in real time, and matches frequently continue uninterrupted even when a cheater is clearly present. Several respondents also suggested that the introduction of a kernel-level anti-cheat could help mitigate the cheating problem. While they acknowledged the potential privacy and security concerns such a solution could raise, they felt the trade-off may be worthwhile to restore competitive integrity. These concerns are especially pressing given the scale of the CS2 player base. According to SteamCharts, CS2 reached an all-time peak of 1,818,368 concurrent players as of May 6, 2025[34]. Effective enforcement needs to scale with this large and diverse player population, ensuring fair gameplay for all.

"*Easy of entry and good trust factor players, the ones with experience, experience way less and down play the problem, new non prime and Prime players will have a bad time and get tons of bad lobbies. Also „passive" anti cheat (have never seen any game beeing stopped). Obvious wall hack is just invisible to it*" - Participant n. 12.

"*Very difficult to ban cheaters without extremely infiltrative anti-cheat OR letting people know what your anti-cheat is based on (So cheat makers can get ahead)*" - Participant n. 10.

"*No actual anti cheat*" - Participant n. 20.

Low Barrier To Cheating Some participants highlighted that the accessibility of cheats is a major issue contributing to the widespread nature of cheating in CS2. Cheats are often inexpensive or even freely available online, as can be seen in figure 4.7, making it easy for players to obtain and use them without significant risk.

"*The fact that it is so easy to cheat makes people very suspicious. I made some good plays and got accused instantly, whereas of course I do the same sometimes when people get a good shot or get lucky. If we just had more trust that the other player wasn't cheating, the game would become less tilting.*" - Participant n. 81.

"*Unfair playing field. Low bar means everyone is perceived as suspicious*" - Participant n. 71.

"*Easy of entry and good trust factor players, the ones with experience, experience way less and down play the problem, new non prime and Prime players will have a bad time and get tons of bad lobbies. Also „passive" anti cheat (have never seen any game beeing stopped). Obvious wall hack is just invisible to it*" - Participant n. 12.

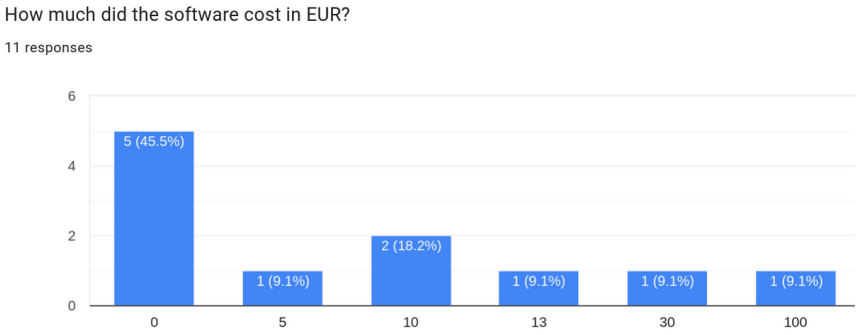

Figure 4.7: The reported prices of cheats according to players who have previously employed cheats.

Discouragement From Improving A recurring theme in participants' responses was the demoralising effect that widespread cheating has on players who strive to improve their skills. For many, the presence of cheaters creates an environment where honest effort and progress are rendered meaningless.

"*People who put in the effort to get better at the game are discouraged from playing because they cannot find matches that are a challenge, all they are met with is loss after loss and decide to stop playing the game*" - Participant n. 64.

"*It ruins the experience for everyone else. Personally for me its very demotivating*" - Participant n. 68.

"*It kills the urge to play the game. For players this is a problem.*" - Participant n. 24.

From the perspective of cheaters, one participant mentioned how cheaters are not motivated by a desire to develop skill or genuinely engage in competitive play. Instead, they often seek shortcuts to success, driven by ego, frustration, or a desire for recognition without putting in the effort that skill-based games like CS2 demand.

"*[...] With counter strike being a relatively simple game, it has a high skill ceiling which might frustrate some players. They may start cheating in order to boost their ego, seek appraisal from others. They may not having the mentality to improve at the game, looking for quick and easy way to feel better about themselves. Also, I believe some simply want to grind xp for weekly drops the easy way. Why is this problematic for us, legit players? Well, we want to be matched against players with similar skill sets, who share our competitive spirit and genuinely want to improve at the game. [...]*" - Participant n. 56.

4.3.2 Overwatch

The next pool of questions came from the section called "Overwatch" and "Overwatch and overwatch users." The answers to the first question, "Did you have access and make use of the overwatch feature when the feature was active" can be seen in figure 4.8. From those answers we can see that 62.2% of participants recall having access to the feature, with 59.8% having reviewed at least one overwatch case. The hypothesis of demo reviewing being a big part of the CS2 community appears to hold since 72% of all participants have reviewed a demo at some point, and 80.5% have rewatched a match due to suspicions of cheating as seen in figure 4.9.

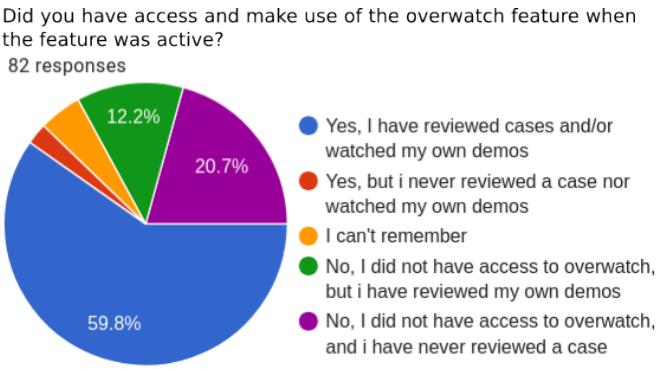

Figure 4.8: Pie chart over the answers to the question: Did you have access and make use of the overwatch feature when the feature was active

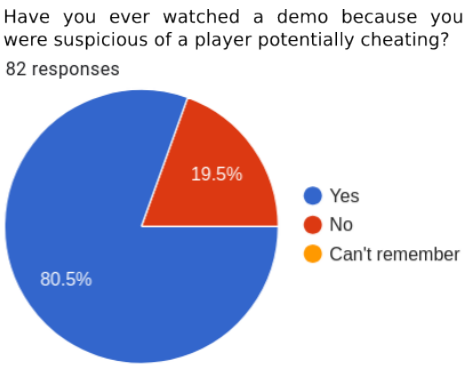

Figure 4.9: The percentage of players who have rewatched a match due to suspicions of cheating.

Of the respondents who replied, that they previously had access to the overwatch feature, 55.1% said that the feature improved CS2, whilst 38.8% responded "maybe", as seen in figure 4.10. Overall, from figure 4.11 it is clear that the majority(75.5%) of participants believe that they aid the community by reviewing Overwatch cases.

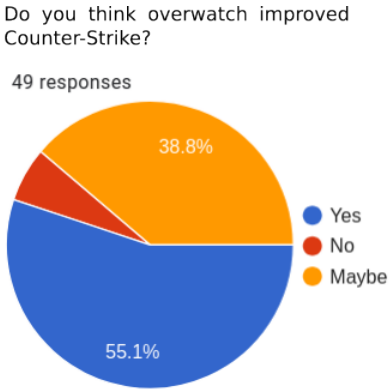

Figure 4.10: The percentage of participants who believe that the overwatch feature improved CS2. Only participants who previously had access to the feature were asked this question.

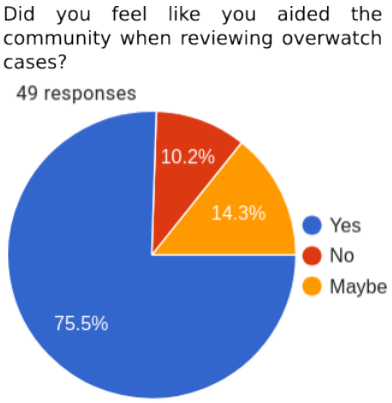

Figure 4.11: Percentage of players who believed that they aided the community by reviewing overwatch cases.

The questions above were followed by the open-ended question: "Is there something else you want to remark regarding overwatch". From these responses the following five themes were discovered: lack of transparency, insufficient selectivity, insufficient rewards, obvious cheaters, difficult judgement.

Lack of transparency Some respondents highlighted concerns regarding Valve's lack of transparency. Players expressed uncertainty about the banning process and how their participation

influences the system. One participant notably asserted that cheaters possess greater knowledge of the system than honest players. The high proportion of participants selecting "maybe"(38.8%) as seen in fig 4.10 regarding the question "Do you think overwatch improved Counter-Strike" could be attributed to this perceived lack of transparency from Valve, which potentially leaves players uncertain about how their contributions impact the game.

"*I would have liked more transparency from the platform. I understand some cases require more in depth analysis to determine whether or not the person was cheating, but most of them were players that were clearly cheating and I have no idea how Valve determines how banning them works*" - Participant n. 1.

"*Cheaters knew more about the overwatch system than legit players.*" - Participant n. 43.

Insufficient selectivity Many participants mention that the overwatch investigator pool should have been more exclusive, as cheaters managed to gain access to the system. Some players believe, that other players were knowingly convicting a suspect even without sufficient evidence.

"*Overwatch access should have been more exclusive; I want to say I had access to it at a high GN*[1] *rank. There were several cases I reviewed where I did not have sufficient expertise to judge the suspect. I usually erred on the side of innocence, but others I was friends with at the time would vote to convict, even in cases where the evidence was not very compelling.*" - Participant n. 48.

"*Overwatch was overused by cheaters to determine who should be banned or not.*" - Participant n. 76.

"*It was stopped due to people abusing the system, but it is needed in Cs2*" - Participant n. 38.

"*Overwatch was great until it wasn't, when it was full of legit player being sent there by actual cheaters. Then valve fixed it and everybody was spammed with turbo spin botter. Then we went back to only legit player there and it became lame. Then it got removed*" - Participant n. 72.

Insufficient rewards Some respondents expressed the view, that greater incentives should be offered for accurately identifying cheaters in Overwatch cases. One participant specifically proposed that such rewards could take the form of in-game skins.

"*A 'vac-store' such as in anomaly's april fools video could be a fun addition. Add a skinline in that store maybe called the 'Enforcer collection' or something like that, then people can redeem skins after doing overwatch cases accurately.*" - Participant n. 2.

"*No rewards given for trying to catch a cheater*" - Participant n. 9.

"*I think they should bring it back but with higher rewards for the viewers*" - Participant n. 24.

Obvious cheaters A recurring theme among responses was the belief, that blatantly obvious cases of cheating should not be included in overwatch reviews. Although such cases were easily identified as violations, one participant expressed the view that if the cheating is so apparent, it ought to have been detected automatically by the VAC system, rather than relying on community members to invest time in reviewing them.

"*It was only really useful when there were obvious hacks involved, but to be honest those should be easy to detect through normal computer algorithms. Suspicious players take so much effort to review that Overwatch really wasn't sustainable. I understand why it is gone.*" - Participant n. 81.

"*[...] most of them were players that were clearly cheating [...]*" - Participant n. 1.

[1]Short for Gold Nova, which is a competitive rank in Counter-Strike.

"*[...] everybody was spammed with turbo spin botter.*" - Participant n. 72.

"*I probably reviewed close to 20 games and every single one was a blantant cheater haha*" - Participant n. 52.

Difficult judgement Several participants voiced that some cases were harder to judge than others. The participants mention several reasons for this including insufficient skill level and closet cheating[2]. One participant mention that the `.dem`-file tick rate of CS:GO was too low to sufficiently make a judgement. Note that the match recording tick rate for CS:GO was 32 ticks per second, whereas the recording tick rate of CS2 is 64 ticks per second.

"*There were several cases I reviewed where I did not have sufficient expertise to judge the suspect.*" - Participant n. 48.

"*The demo tick rate means it is very difficult to discern anything but the most obvious cheaters.*" - Participant n. 71.

"*Some cheaters are doing their best to hide evidences. It is not a perfect solution*" - Participant n. 21.

"*I understand some cases require more in depth analysis to determine whether or not the person was cheating, but most of them were players that were clearly cheating [...]*" - Participant n. 1.

"*It is not always possible to judge right*" - Participant n. 74.

4.3.3 Detecting cheater behaviour

In the survey sections named "reviewing demos", participants, who previously replied that they have reviewed either their own matches or overwatch cases, were asked some questions regarding the process in which they review demos. The most insightful responses were for the question: "Can you in more detail describe how you determine whether a player is cheating or not."

From this question, we gathered two investigation approaches used by the respondents in order to determine whether a player is cheating: Account reviewing and demo reviewing. It is important to note, that one investigation approach does not exclude the other. Both methods are jointly used in multiple cases. Across multiple responses, players start their reviews with the assumption, that other players are not cheating. This presumption frames the investigative process as one where suspicion must be built progressively through observations.

"*First of all, I start with the assumption that the player is not a cheater.*" - Participant n. 18.

"*I'm trying not to jump into conclusions straight away and give them the benefit of the doubt, so unless it's very obvious like spinbotting, shooting through the walls, soft aim [...]*" - Participant n. 56.

"*To be sure someone is cheating beyond a reasonable doubt there has to be a consistent pattern of suspicious gameplay across multiple duels and rounds.*" - Participant n. 61.

Account analysis Players frequently begin the analysis with an assessment of the Steam account of the suspect. Within this account assessment, players investigate Steam profile visibility and in-game hours. Accounts that are new, private, or statistically anomalous (e.g. high skill level with very low playtime) often prompt closer inspection. The account analysis can even go a step further and use account data from the third party website faceit.com. This data would include the FaceIt ELO and the FaceIt rank and would similarly be compared to the playtime.

[2]A cheating technique in which a player, who is deploying cheats, tries to hide that fact that their abilities are mechanically enhanced.

"*[...] I check their profile, stats, and history. Based on that, I decide how closely I'll need to watch their gameplay. For example, someone with thousands of hours in their history and consistent performance is probably not a cheater, and if I don't see anything particularly suspicious in their gameplay, I'll quickly rule them out. On the other hand, if it's a new account—or an old account with many hours but a sudden spike in performance—then I'll be more suspicious and examine the video much more thoroughly.*" - Participant n. 18.

"*Its usually not one thing, but combination of a lot of stuff. Things like 28k elo, no faceit, under 1000 hours on account,[...]. Can't be 100% sure still, but a lot of the time if I think someone is cheating I might follow the steam account for a bit and a lot of them do eventually get banned.*" - Participant n.58.

Although this type of analysis can prove helpful, it is possible that it can be faulty, due to the fact that there is a market for authentic looking Steam accounts. These sites advertise their accounts with thousands of hours in game, commendations[3], badges[4], and by how many other games the account has. The prices of these accounts vary from less than a euro to more than 100 EUR[36][37][38][39].

"*If an account looks fairly new or is private, it is already making it more suspicious. But on the other hand, does it really matter nowadays when you can buy a legit looking account for a few bucks?*" - Participant 56.

Behaviour analysis The second and most indicative investigation approach is manual behaviour based cheat detection (also more commonly called demo reviewing). This type of investigation involves inspecting the suspect's play style by reviewing a demo. Generally there is a sentiment amongst respondents, that not all types of cheats are equally difficult to identify. Cheats like spinbotting and aim hacking are, according to the respondents, easy to identify as multiple participants report this in their responses. Unfortunately, due to the alleged obviousness of these cheats, no participants described how to identify them.

"*Besides spinbotting and aim which are pretty obvious I think the main factor determining whether or not a player is closet cheating is skill gap and overall game sense.*" - Participant n. 1.

"*Aim hacks are easy to spot.*" - Participant n. 36.

The most frequently cited method of evaluating suspicious players is **identifying imbalances in skill expression**. This method is more thorough and requires reviewing a larger portion of the match. Identifying skill imbalances involves evaluating the suspects abilities within several areas of play. An example of this could be a player demonstrating highly accurate aim but simultaneously poor movement, poor utility usage, or generally low game sense. This perceived mismatch suggests, that some form of artificial assistance (such as an aim hack) may be compensating for otherwise underdeveloped gameplay skills. Conversely, experienced players with well-rounded abilities are generally assumed to be legitimate unless their gameplay behaviour suggests otherwise. Skills that are analysed in this manner include:

1. Crosshair placement[5]
2. Recoil control[6]
3. Movement tracking[7]

[3]Commendations in CS2 are a form of peer recognition where players can positively rate their teammates after a match. These commendations fall into three categories: Friendly, Teaching, and Good Leader. They are intended to encourage positive behaviour and sportsmanship within the community.

[4]In the context of CS2, a badge is a type of collectible that are displayed on a users steam profile and within the Counter-Strike game. Examples of these are Service Medals, Operation Coins, Major Trophies and Coins as well as Collectible pins.[35].

[5]Crosshair placement refers to the strategic positioning of a player's aiming reticle (crosshair) in anticipation of enemy presence.

[6]The technique of compensating for a weapon's recoil pattern by dragging the mouse in the opposite direction of the spray.

[7]The ability to follow an enemy's movement smoothly with one's crosshair.

4. Flicking[8]
5. Counter strafing[9]
6. Jiggle peeking[10]
7. Utility[11] usage
8. Game sense[12]
9. Corner clearing[13]
10. Economy[14] knowledge

"*I watch the video. There's a set of skills a player can have: Crosshair placement, recoil control, movement tracking, flicking when it comes to aiming. Also, movement in general, game sense, and how they clear angles. If a player is extremely good at one of these skills but noticeably lacking in the others, I'll suspect they're cheating and will look for concrete evidence. That imbalance between skills isn't proof by itself, but it's a strong indicator that makes me investigate further.*" - Participant n. 18.

"*In general you can see quite quickly if someone is cheating. Crosshair palcement, movement, utility usage, account in general (hours, steam level, ig level, friends, faceit etc.), corner clearing, economy knowlege etc. Its usually not one thing, but combination of a lot of stuff.*" - Participant n. 58.

"*[...] if they're using aim assists it makes it obvious when you compare their ability to aim vs their ability to do everything else.*" - Participant n. 47.

"*Usually their aim and path tracing don't match their movement and positioning. Eg: high headshot accuracy and time-to-kill, but can't throw a nade nor counter-strafe*" - Participant n. 53.

Beyond skill discrepancies, many responses focus on of identifying players utilising wallhacks by their **advantageous timing**. Multiple participants mention, that it is reasonable to assume cheating is at play, when a suspicious player constantly has advantageous timing when performing risky or dangerous moves such as peeking. Examples of advantageous timing include peeking an opposing player, when the opposing player is unable to fire their primary weapon, or when the opposing player is observing a different angle. Although, it is possible to detect, when a player is reloading a weapon due to auditory cues, repeated advantageous timing is a symptom of cheating according to multiple participants.

"*I argue that is really difficult to have the perfect timing to peek or to flank throughout a whole CS game. For example, if someone repeatedly peeks the second that an AWP turns away from the line, I would generally assume that the person is cheating if I was unsure about it beforehand.*" - Participant n. 1.

"*For players that are using a non-detectable version of a cheat, often watching their ability to avoid certain interactions or perfectly time an interaction is key to catching them. Further, unnatural rotations when outplayed that make absolutely no sense but are in fact, the perfect counter to a move. Another indication is when they do very well with no one spectating them and very badly*

[8] Aiming technique where the player rapidly moves the crosshair from one position to another — typically to snap onto a target.

[9] A movement technique used to quickly stop and shoot accurately after moving. It involves pressing the opposite directional key to instantly halt momentum, allowing for precise, accurate shooting with minimal delay.

[10] A defensive technique where a player quickly peeks around a corner in and out of cover to gather information or bait enemy fire without fully exposing themselves.

[11] Collective term for non-weapon equipment such as grenades, flashbangs, smokes, molotov/incendiary grenades, and decoys.

[12] An abstract but essential skill referring to a player's intuitive understanding of the game's flow. It includes predicting enemy behaviour, anticipating rotations, understanding timing, and making strategic decisions based on incomplete information.

[13] A tactical maneuver in which a player systematically checks potential enemy positions—especially tight angles and corners—when entering or moving through areas of the map.

[14] Refers to the in-game currency system that determines a team's ability to purchase weapons, armour, and utility.

with people spectating them, indicating a toggle of assistance." - Participant n. 23.

"*I usually look for the suspect making direct and confident plays multiple times in a row - peeking or checking only the angle that their enemy is on, finding timings and flanks too many times for it to be luck, etc.*" - Participant n. 73.

"*If they consistently have advantageous timings on the opponents that make the others have a disadvantage. I.e. they peek exactly when the target was reloading/grabbing util/looking away, something like that.*" - Participant n. 2.

"*Spidey senses. Some how, these players just know. i believe they are toggling to get the edge. For example, in inferno. I would be hiding in church, say behind the podium. And some how, this person would check that. It's very rare that spot gets check.*" - Participant n. 19.

Closet cheating is a commonly mentioned phenomenon within the responses and refers to a cheating technique, in which a player, who is deploying cheats, tries to hide that fact that their abilities are artificially enhanced. Closet cheating and wall hack[15] often go hand in hand, since wall hacking does not have the same characteristic tell as an aim hack does, thereby making it easier to hide. According to several participants, the skill of a cheater decides how successfully they can appear as a legitimate player. A less skilled cheater may look/trace objects through walls or disregard otherwise logical caution in high risk areas, because they have access to information that they should otherwise not have access to.

"*A noob cheater or one that does not know to hide his/her cheats will normally not clear any angles or jiggle peek, rather they aim at their target through the wall and keep their aim there. However, a skilled cheater who's only walling can be very hard to judge, especially if they have some raw aim skills to go along with their cheats. Judging from just 1 kill or 1 round whether or not someone is cheating can be very hard and so a significant part of the demo must be viewed in order to make judgement*" - Participant n. 46.

"*Closet cheaters can be harder [to tell] but mostly you can tell because they will only ever clear or hard clear the angle where a player is.*" - Participant n. 52.

"*They are too aware of their surroudings, like running with knife in dangerous area.*" - Participant n. 21.

"*Pre-firing and aiming at walls for a line up too often than not in unnatural ways.*" - Participant n. 80.

"*Skipping/rushing through empty areas. Only "checking" THE angles*" - Participant n. 70.

Most of the aforementioned sentiments were echoed by participant n. 18, who wrote the following:

"*First of all, I start with the assumption that the player is not a cheater. Then, I check their profile, stats, and history. Based on that, I decide how closely I'll need to watch their gameplay. For example, someone with thousands of hours in their history and consistent performance is probably not a cheater, and if I don't see anything particularly suspicious in their gameplay, I'll quickly rule them out. On the other hand, if it's a new account—or an old account with many hours but a sudden spike in performance—then I'll be more suspicious and examine the video much more thoroughly. Next, I watch the video. There's a set of skills a player can have: Crosshair placement, recoil control, movement tracking, flicking when it comes to aiming. Also, movement in general, game sense, and how they clear angles. If a player is extremely good at one of these skills but noticeably lacking in the others, I'll suspect they're cheating and will look for concrete evidence. That imbalance between skills isn't proof by itself, but it's a strong indicator that makes me investigate further. Concrete signs that someone is cheating include: They don't check any other angles except the exact one where the enemy is—repeatedly, not just once. They're never surprised by an enemy's position. They shoot at enemies faster than what would be possible with normal human reflexes (prefiring). These are signs of a wallhack. Their bullets constantly hit the target even when their aim is off and their crosshair placement isn't right. That's a sign of an*

[15] A type of cheat that reveals hidden characters and objects through walls, giving an unfair advantage[28].

aimbot."

4.4 Reflection

While the survey gathered responses from 82 participants, this sample size is not sufficient to draw definitive conclusions or generalise findings to the broader community. Nonetheless, the responses provide valuable insights and may serve as an initial indication of broader trends or attitudes, offering a foundation for future, more comprehensive studies.

Firstly, no broad definition of cheating was provided therefore participants may have differing views on what constitutes cheating. It is to mention, that in the counter-strike community there is a broad understanding of what types of cheating can occur. Furthermore, according to the 3 definitions of cheating described in *Cheating and anti-cheat system action impacts on user experience* by Bryan van de Ven, only one of them is consistently possible whilst playing CS2; that being definition number 1: "*Using additional code or software to modify the game*"[5]. Abusing faulty code in the game without using additional code or software is quite rare, as Counter-Strike is a relatively bug-free game[16]. Furthermore, gaining access to otherwise unavailable information without altering or interacting with code as described in definition 3, is not possible as all players on a team share the same information. Furthermore,communication skills are fundamental to the game. Therefore, although participant understanding is central to the survey, we believe that respondents are aware that additional software is implied in the definition of cheating. This is also reflected in the wording of the survey question:"Have you ever deployed external software in order to enhance your gaming abilities?"

A question that could have been asked is about player experience with encountering cheaters. This could have given further information about cheating as a problem from the perspective of a player. By directly asking players to describe their personal encounters with cheaters, the study could have gathered more nuanced and detailed insights, into how these experiences shape their perceptions of the game. Overall, including this question would have enhanced the understanding of how cheating affects the competitive balance of CS2 and the player community's overall experience. Note, that some of this information was given, when asked the open-ended question of why participants believe cheating is a problem, however a more directed question could have been asked.

Another possible question would have been how confident players are in detecting cheaters. This question could have provided valuable insights into the subjective perception of cheating in the game and the trust players have in their own judgment, as often times cheating is easily detectable, but in some cases it is ambiguous. Understanding players' confidence levels in identifying cheaters would shed light on how often players might misinterpret legitimate gameplay as cheating, which could contribute to a culture of suspicion and frustration within the community.

The wording of the question "Why do you believe cheating is a problem?" could have been interpreted in multiple ways. Some participants interpreted the question as asking why cheating occurs in the first place, while others interpreted it as a question about why cheating itself is problematic. The latter participants focused on the impact of cheating on gameplay, fairness, player experience, and community trust. This ambiguity could have been avoided by clarifying the intent of the question by specifying whether it is asking about the reasons behind the presence of cheaters or the effects of cheating have on the community.

The question "How often do you experience playing in a match with a cheater?" had 5 options that the participants could choose from: Never (0% of the time), Occasionally, Sometimes (50% of the time), Often, and Always (100% of the time). One participant noted in the final comment that answering this question proved difficult due to the options presented to them, and that the ranges didn't accurately reflect their answer:"*Question about amount of cheaters in my games wasn't easy to answer as it obviously wasn't 0, but 50% was excessive, so data isn't that useful for you guys.*" An improvement to this question would be to set a time frame for the question as well. With the current version of the question, it is up to the reader to interpret how far back in memory they need to go. Furthermore, having the participant convert recent matches into percentages might not

[16]Note that the 8th of May 2025, a bug was introduced to the game that could be utilised to gain an advantage[40]. However with that said, this bug was also fixed in a subsequent update the 9th of May 2025[41][42]. This illustrates, that although Counter-Strike 2 is not free from bugs, bug are fixed fairly quickly

be concrete enough, and thereby make it difficult for the participant to answer. Therefore, adding that they need to inspect their 10 most recent matches and count how many times the matches included a cheater, could possibly make the question easier to grasp for the participants. A revised version of the question would be the following: "How frequently did you encounter cheaters in your 10 most recent matches?" with the options: Never (0 out of 10 matches), Rarely (1–2 out of 10 matches), Sometimes (3–5 out of 10 matches), Often (6–8 out of 10 matches), Always (9–10 out of 10 matches).

A bug in the flow of the survey was discovered after the survey was closed. The error consisted of user being sent to the wrong section after answering the question: "Do you believe that there is a problem with cheaters in CS2", as highlighted on figure A.1 by the red arrow. Participants answering no to this question should have been sent to the block "Cheaters are not a problem". Instead, all users where sent to the section "Cheaters as a problem". Therefore, no responses where recorded to the question: "Why do you believe that cheaters are not a problem?" However, due to the section "Cheaters as a problem" containing a similar question and an open answer format, some participants voiced their disagreement here. It is however entirely plausible that some simply tried to answer the question given to them, and therefore there is unfortunately some data missing with regard to that.

4.5 Conclusion

The survey appears to have effectively captured insights into cheat detection and general attitudes toward cheating within the Counter-Strike community. Although some questions may have been suboptimally phrased, the inclusion of numerous open-ended questions allowed participants to express their perspectives freely on a range of relevant topics. Participants from a wide variety of skill levels were successfully sampled from the top Counter-Strike communities on reddit.com.

When participants were asked the question of "Do you believe that there is a problem with cheaters in CS2", the vast majority of responses were saying yes. The survey reveals that the overwhelming majority of participants perceive cheating in CS2 as a significant and pervasive problem, both in terms of it's prevalence and it's impact on gameplay. Not only do most players report encountering cheaters regularly, but they also express a strong consensus that cheating severely compromises the integrity and enjoyment of the game. Through thematic analysis, several core issues emerged, including a perceived lack of fairness, disruption of player experience, widespread distrust, inadequate enforcement, and low barriers to cheating. Players voiced frustration over the devaluation of effort and competitive spirit, highlighting how cheating creates an environment of suspicion, discourages improvement, and ultimately diminishes the overall appeal of the game. The findings underscore the need for more effective anti-cheat solutions and greater trust in the game's competitive ecosystem.

Furthermore, the hypothesis that behavioural cheat detection is a pervasive skill within the Counter-Strike community holds, as more than half of participants reported having reviewed an Overwatch case and the majority of participants had previously rewatched a demo due to suspicions of cheating. Whilst conducting a thematic analysis on the remarks on the Overwatch feature the following themes were discovered: lack of transparency, insufficient selectivity, insufficient rewards, obvious cheaters, and difficult judgement. Participants voiced that the work done for Overwatch cases can often be time consuming and was not rewarded adequately. Furthermore, the lack of transparency diminishes the players' ability to clearly interpret how their efforts are being evaluated. Suspects in Overwatch cases were often obviously cheating, and participants believed that an algorithmic approach should be capable of automatically detecting such obvious cheating behaviour. Lastly, respondents indicated that the selection process for Overwatch investigators may have been compromised by malicious individuals, potentially due to insufficient selectivity or inadequate screening criteria.

With regard to standardising behavioural cheat detection methodology, two main approaches were found: Account analysis and behaviour analysis. Account analysis was used as an indicator, but respondents never relied on only this method to determine, whether a player was cheating or not. During an account analysis players will investigate the authenticity of an account by examining age, publicity status, hours in game, activity level, and badges and compare them to the skill level of the corresponding player. Indicators of purchased accounts included: A sudden spike in

activity, high skill level with low playtime, no badges, and private profile. However, account analysis is not an unassailable method due to the cheap prices of authentic looking Steam accounts. Behavioural analysis is, in contrast to the account analysis, based on the behaviour of the suspected cheater in game. This is done by rewatching a match from the perspective of the suspected cheater. Participants describe imbalances in skill expression as an overall indicator of cheating. This method builds upon the notion that an authentic and skilled player has a well rounded set of abilities, whereas a cheater may appear skilled in one area of play, but exhibit significantly lower proficiency in others. Another indicator of cheating was continuous advantageous timing. This approach builds upon the premise that if a player is using wall hacks, they may exploit the additional information to execute high-risk manoeuvrers, while attributing the successful outcomes to chance. The underlying rationale for this indicator is that consistently advantageous timing is statistically improbable, particularly when high-risk manoeuvres are only executed in situations where a favourable outcome (e.g., a kill) is possible. A third indicator of cheating is object tracing, a behaviour often associated with wall hacking. This occurs when a player consistently tracks opponents through obstructions, such as walls, or fixates on seemingly non-strategic locations to maintain peripheral visual contact with hidden players. Lastly, although never described in the responses, multiple participants mention that aim hack and spinbotting are easily detectable. Due to the unfortunate exclusion of these key cheat detection techniques, more research in this area is required.

Although improvements could have been made to the phrasing of certain survey questions, this chapter proposes a standardised method for manual behavioural cheat detection in CS2. Furthermore, the chapter shows that participants were very dissatisfied with the current state of VAC, highlighting a need for more robust and transparent anti-cheat systems.

Chapter 5

Dataset - CS2CD

5.1 Introduction

The majority of machine learning papers writing about cheat detection run into the same major problem: Lack of labelled data. Currently, there are no datasets containing labelled gameplay data. Furthermore, the only easily accessible data source for Counter-Strike gameplay data is through HLTV, and this data source only contains official tournament gameplay[43]. Due to the heavy monitoring of cheats during competitive tournaments, it is very likely that the vast majority of these matches do not contain any cheaters, which would result in an extreme class imbalance. One paper mitigates this problem by exploiting a data leak in the Overwatch feature, however since the removal of this feature, this option is no longer feasible[29].

Due to the lack of a public dataset, the creation of one such dataset was essential. The publication of such a dataset would enable further research into machine learning applications in cheat detection. Hence, this chapter will explore the creation of the Counter-Strike 2 Cheat Detection (CS2CD) dataset[44].

5.2 Methodology

5.2.1 Data collection

The collection of data for the dataset proved to be a challenge, as game data is not public by default. For these reasons, a previous project handled data collection, hence this section will only cover the broad strokes of the process. For more detail, the data collection process is described in "*Counter-strike 2 Game data collection with cheat labelling*" by Mille Mei Zhen Loo and Gert Lužkov[28].

In order to find publicly available `.dem`-files, we needed a third party source. The match sharing service csstats.gg was deemed suitable as a data source, as this website had a page dedicated for the games that have most recently been uploaded to the site[45]. Furthermore, the page indicates whether a user participating in a match had been VAC-banned. This information was used to create a script, that scraped the site for publicly available match sharing codes. The match sharing code, names of the VAC-banned users, and other match information was saved to a `.csv`-file. This file was then used to select a balanced set of matches with respect to the played map type. Unfortunately, this was not possible for matches where VAC-banned users were present, as there was a tendency for cheaters to prefer some maps over others. Additionally, matches with VAC-banned players were in the minority of all scraped data, therefore, all data containing VAC-banned players were added to the `.csv`-file. Hence, the data collection resulted in 795 matches, with 317 containing at least one VAC-banned player and 478 containing no VAC-banned players[28].

Next in the process, was the retrieval of the demos. Downloading `.dem`-files through the in-game match downloading tool was an inefficient process that required downloading matches one by one. The application CS Demo Manager allowed bulk download of `.dem`-files[46]. Using the `.csv`-file with scraped match sharing codes, the codes were extracted into a list, which in turn would be inserted into CS Demo Manager. This application downloaded all of the matches in the list, and added the file name to the `.csv`-file in a new column[28].

5.2.2 Data labelling

The integrity of the labelling system was critical for the quality of the dataset. Taking a subsample ($n = 50$) of the data with no VAC-banned players present, the precision of the not cheater label was 97.2%. This was deemed as a good enough metric. Therefore, no manual review was performed on the subset containing no VAC-banned players[28].

Taking a subsample ($n = 50$) from the set with at least one VAC-banned player showed, that the precision of VAC-bans were 92.6% and the recall was 24.7%. In other words, according to this subsample, around half of users deploying cheats, in games where a VAC-banned user was present, were not yet VAC-banned. The label metrics for this class were quite poor and in order to use this data for machine learning purposes, better labelling was needed. Since we observed a poor recall in the set containing matches with at least one banned player, we chose to manually label the data within this set. Therefore, all 317 demos in the subset which contained at least one VAC-banned player were manually reviewed[28].

In order to determine whether a player is a cheater, several approaches are available as seen in section 4.3.3. Since this project primarily focuses on behavioural cheat detection, we opted not to judge based on accounts but rather behaviour during gameplay. As mentioned in section 4.3.3, there is a certain level of expertise needed when reviewing demos. For this reason, all of the demo reviewing was done by Gert Lužkov, as he had, as of may 2025, over 5600 in game hours, a premier ELO of over 27,500 and a FaceIt ELO of 2707, which is equivalent to level 10[47].

Note, that while section 4.3.3 references the use of spinbotting and aim hacking in the context of cheating behaviours, it does not provide explicit definitions or detailed explanations of these terms. Spinbotting refers to the use of a cheat that causes the player's in-game character to spin rapidly and erratically, making it difficult for opponents to land shots. Aim hacking involves software manipulation that enhances a player's aim precision beyond human capabilities, allowing for immediate and consistent head-shots or damage. Therefore, spinbotting is detected by the player model spinning rapidly and erratically, and aim hacks are detected by minute sharp changes in a players' aim, that are only achievable by the use of software and not by the movement of a hand. Additionally, some types of aim hacks circumvent recoil and imprecisions caused by movement by programming the player shots to go exactly in the opponents direction, even when the player is not aiming directly at the enemy.

As described in 4.3.3, the criteria for player being labelled a cheater was, that there should be evidence of cheating beyond a reasonable doubt. This means, that in the scenario, where a player was performing exceptionally well, but did not display any behaviour of a cheater, the player was not labelled a cheater. In the case of uncertainty, a player was labelled as not cheater.

Whilst labelling the data, outliers were detected. Most notably games where played where no rounds took place. 3 of these outliers were discovered and were subsequently removed from the dataset, as they were found to have provided no value. Furthermore, games containing cheaters in the no VAC-ban subset, that were found during the label quality check, were moved to the cheater set.

5.2.3 Data extraction

`.dem`-files contain sensitive information; Therefore, the raw data could not be published directly. To allow publication, it was necessary to apply anonymisation procedures. Unfortunately, no publicly available software can perform this anonymisation on a `.dem`-file directly, which meant that the data needed to be extracted into another format before anonymisation could begin. This unfortunately means, that matches within the dataset would not be replayable using the CS2 "watch matches and tournaments" feature.

For data extraction from `.dem`-files, there is a wide selection of demo parsers. Some of these parsers aim to extract statistics in order for the user to better understand their CS2 skills and potentially give an insight for the user in how to improve their performance. For this project the python

library, demoparser2, was used, as it provides the means to extract all data from a `.dem-file`[48]. This parser is able to parse CS gameplay into two formats: ticks and events, by using the functions `parse_event()` and `parse_ticks()`. Ticks were stored in a time-sequential Pandas DataFrame. In other words, it was a table-like structure sorted by time. On the other hand, events were stored as a list of tuples containing strings and DataFrames. Using Python type annotation, the following is the data type returned by `parse_event()`:

```
list[tuple[str, pd.DataFrame]]
```

In order to handle this data type better a class called `EventList` was created. This class can be initialised by using a list of tuples containing strings and DataFrames, and has `read_json()` and `write_json()` functionalities. Furthermore, the class also supports the function `__getitem__()` and can therefore be accessed just like any list. Another class that was created to handle data extraction was `CSDemoConverter`. This class was responsible for extracting data from a `.dem`-file and censoring sensitive information. Furthermore, it also added information scraped from csstats.gg, which was stored in the `.csv`-file from section 5.2.1.

This paragraph will go in-depth with the data extraction algorithm. All files regarding this process can be found in the folder `DataExtraction` on the project's corresponding github page[1][49]. The algorithm can also be seen in algorithm 1. The algorithm used for extracting data started by reading the `.csv`-file from section 5.2.1 to a Pandas DataFrame. This file contained the filenames for all downloaded `.dem`-files along with other information scraped from csstats.gg, including the names of cheaters found during data labelling. The script then droped all rows, where there was no file name. Thereby, only the files that were downloaded remained in the DataFrame. The script then iterated through the list of file names. During a single iteration, the list of corresponding cheater names was loaded. Next, a `CSDemoConverter` was instantiated by using the current `.dem`-filename. Then, the `convert_file()` function was used to get the tick data and the event data. Furthermore, this function was also responsible for adding cheater and csstats data, and subsequently responsible for anonymising the data as well. We will go into more detail regarding the anonymisation process in section 5.2.3.1.

Algorithm 1 Data extraction

```
df ← read_csv(scrape_csv_path)
df ← df.drop_row_if("demo_file_name" is na)
for each row in df do
    cheaters ← row["cheater_names_str"]
    converter ← CSDemoConverter(row["demo_file_name"])
    tick_df, events_list ← converter.convert_file(cheaters=cheaters, csstats_info=row)
    write_to_parquet(tick_df)
    write_to_json(events_list)
end for
```

5.2.3.1 Data anonymisation

In order for this dataset to be publishable, it was necessary to apply anonymisation procedures. Otherwise, the data would contain user IDs and other sensitive information, that would breach GDPR regulations[50] . We deemed there were two categories of data: Omittable data and essential data. The omittable data would be removed from the dataset, while the essential data would undergo anonymisation procedures. The following is a complete list of omittable data fields:

1. name
2. user_name
3. names
4. attacker_name
5. crosshair_code
6. assister_name
7. player_name
8. victim_name
9. player_steamid
10. music_kit_id
11. leader_honors
12. teacher_honors
13. friendly_honors
14. agent_skin
15. user_id
16. active_weapon_skin
17. custom_name
18. orig_owner_xuid_low

[1]Code available at: https://github.com/Pinkvinus/CS2_cheat_detection

19. orig_owner_xuid_high
20. fall_back_paint_kit
21. fall_back_seed
22. fall_back_wear
23. fall_back_stat_track
24. weapon_float
25. weapon_paint_seed
26. weapon_stickers
27. xuid
28. networkid
29. PlayerID
30. address

The following is a list containing the sensitive data fields:

1. steamid
2. user_steamid
3. attacker_steamid
4. victim_steamid
5. active_weapon_original_owner
6. assister_steamid
7. approximate_spotted_by

Part of the code used for anonymisation can be see in listing 1. The Demoparser2 library contains the function `parse_player_info()`. This function returns a pandas DataFrame containing the name and `steamid` of every single player participating in a given match. This function was used to create a mapping from player `steamid` to an anonymised `playerid` (e.g: Player_1). This mapping was applied to the tick and event data in the columns mentioned in the sensitive data list. Note, that name related data were not considered sensitive, but removable. This was due to names creating bulk data, as they would otherwise be changed to the same value as the `steamids`. Furthermore, multiple players can have the same name, which would make the mapping approach prone to fail. Lastly, players were always identifiable by their id, but not necessarily by their name. For all of the reasons mentioned above, name related data were not included in our dataset.

Note, that if all players from a single team leave a match, a single bot spawns in their place. The bot has no AI, remaining perfectly still in the spawn site of the team. Therefore, kills for a user with no ID can occur. These are bot kills.

Listing 1 The code used for anonymising sensitive data

```
from utils.sensitive_data_fields import SENSITIVE_DATA_REPLACE

def _replace_sensitive_data_df(self, df:pd.DataFrame):
    df_anonymised = df.copy()

    for d in SENSITIVE_DATA_REPLACE:
        if d in df_anonymised.columns:
            # Check if the column contains any lists
            if df_anonymised[d].apply(lambda x: isinstance(x, list)).any():
                # Apply the anonymisation mapping for each element in each list in the column
                df_anonymised[d] = df_anonymised[d].apply( lambda lst:
                ↪  [self.player_mapping.get(str(id)) for id in lst])
                continue

            # Apply anonymisation mapping to the whole column
            df_anonymised[d] = df_anonymised[d].astype(str).map(self.player_mapping).fillna("")

    return df_anonymised
```

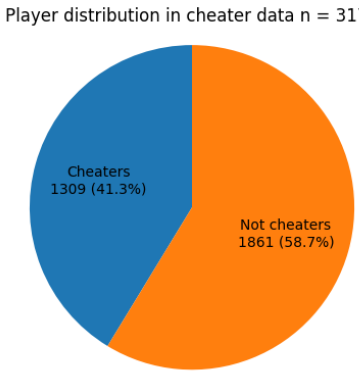

Figure 5.1: Ratio of cheaters to non cheaters in data with cheaters present

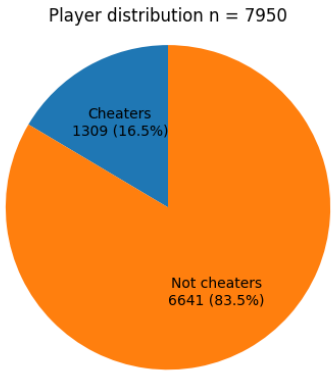

Figure 5.2: Ratio of cheaters to non cheaters in the whole dataset

5.2.4 Data publication

Ticks being stored as a continuous sequence of events make them store a lot of data. To put this into perspective the average size of a tick DataFrame was 725,019 rows by 225 columns. If one wanted to merge the events into ticks, this would add a lot of bloat data to the DataFrame, as a single instance of an event generally happens on a single tick, but would add several columns to the DataFrame. Therefore, it was decided that events and ticks would be stored in two separate files in different formats, in order to be more space efficient. Ticks were stored in `.parquet`-files due to it's quick read/write functions and because of the built-in support from the python library Pandas[51].

Events contained minimal bloat data, therefore they were stored in `.json`-files, as this would allow readers to inspect the files without loading the data. However, due to the the data type of an event, there was no built-in support for saving and loading this data type. Functionality for these purposes was implemented in the file `DataExtraction/utils/EventList.py`[49].

5.3 Results

5.3.1 Descriptive statistics

The size of the dataset is 48.9 GB. The dataset contains 795 `.parquet`- and `.json`-file pairs. 317 of these pairs have at least one cheater present and 478 have no cheater present. The ratio of cheating and not cheating players in cheater data can be seen in chart 5.1 and the whole dataset in chart 5.2.

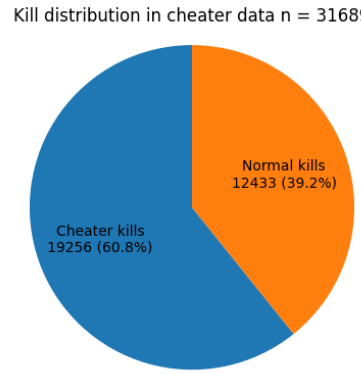

Figure 5.3: Ratio of cheater to non cheater kills in data with cheater present

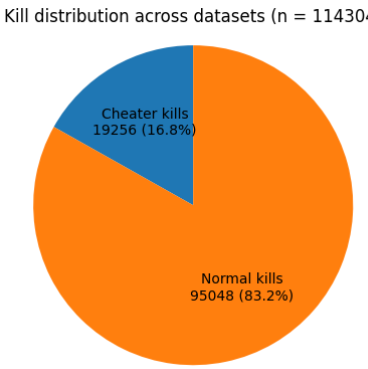

Figure 5.4: Ratio of cheater to non cheater kills in the whole dataset

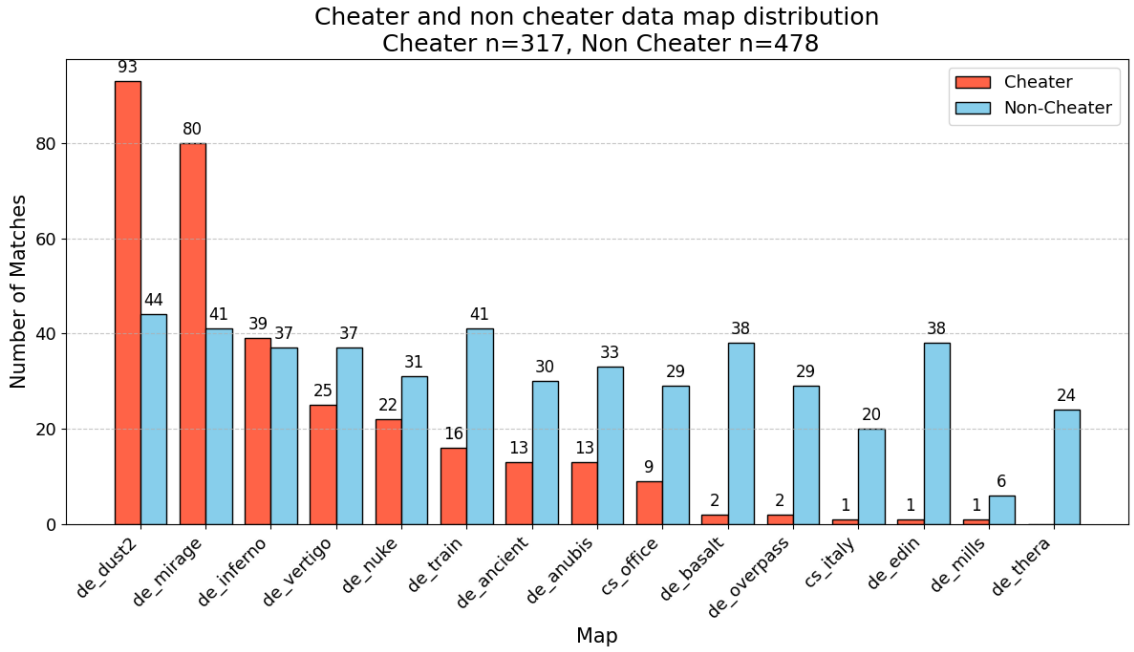

Figure 5.5: Map distribution of dataset

The total amount of gameplay in the cheater dataset is approximately 21,503,010 ticks (equivalent to 93.3 hours of combined player perspectives), while the non-cheater dataset contains around 47,778,477 ticks (207.4 hours). Since each match includes data from all 10 players simultaneously, these figures represent the cumulative gameplay from all player points of view — meaning that one hour of in-game time corresponds to 10 hours of recorded player perspective data.

The data columns present in the `.parquet`-files can be seen in appendix B. In total, there are 225 columns in each `.parquet`-file and an average of 725,019 rows. Examples of data stored per tick are X, Y, Z, pitch, yaw and angle of all players in the match. The list of events that can be found in `.json`-files is listed in appendix C. In total, there are 59 possible events in the `.json`-file, each one containing a DataFrame if it's own. Examples of such events are `player_hurt` and `bomb_planted`.

The total amount of kills within this dataset is 114,304. The distribution of kills by cheaters in cheater data can be seen in chart 5.3 and the whole data in chart 5.4. Comparing the ratio of cheater kills in cheater data to the ratio of cheaters in the cheater data, it is clear that cheaters manage to gain an unfair advantage, as cheaters make up the minority of the players at 41.3% but manage to get a 60.8% of kills.

The played map distribution of cheater and non cheater data can be seen in figure 5.5. Note, that non-cheater matches were intentionally selected to achieve the most balanced possible distribution across maps. However, for cheater data, all matches with VAC-banned players were selected. This causes an imbalance in terms of map distribution for cheater data.

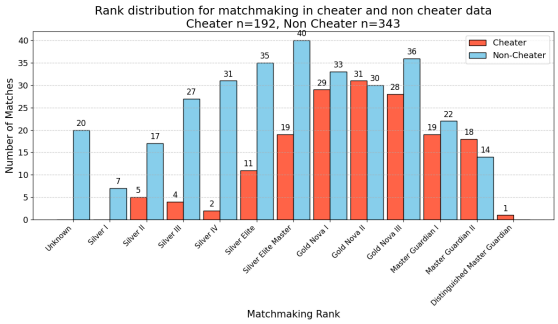

Figure 5.6: The average ranks of matchmaking games in the dataset

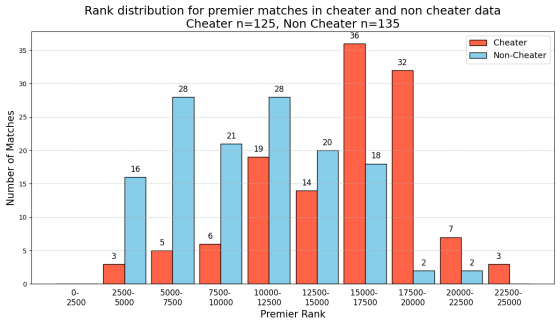

Figure 5.7: The average ranks of premier games in the dataset

The average rank distributions of cheater and non cheater matches in matchmaking and premier can be seen in figures 5.6 and 5.7. The rank distribution for both cheater and non cheater sets have an acceptable variety of different ranks present. It is, however, possible to gather that on average, cheater matches have a higher average rank than non cheater matches. This further highlights the advantage that cheaters gain systematically.

5.3.1.1 Visualising cheater vs not cheater data

This section attempts to visualize some behavioural pattern differences between cheaters and non cheaters within our dataset using a combination of box and violin plots, that illustrate where the majority of the data lies and it's distribution shape.

Kill to death ratio A good metric of performance in CS2 is the kill to death ratio, which is calculated by dividing the total amount of kills by the total amount of deaths for a player. Due to cheaters having unfair advantages, it was hypothesised that the mean kill to death ratio of cheaters would be significantly higher than that of non cheaters. All players were included in the creation of this figure, therefore $n_{cheater} = 1309$ and $n_{non\_cheater} = 6641$. Note, that in the case of a player having zero deaths, a single death was added, due to division by zero being mathematically impossible. This comparison can be seen in figure 5.8, which highlights cheaters achieving a higher median kill to death ratio of 2.1 compared to 0.8 for non cheaters.

Head-shot kill percentage Head-shot kill percentage describes the percentage of total kills that were done by head-shots. We hypothesise that due to cheaters deploying the use of aim hacks, their head-shot percentage would be above that of a normal player. This data excludes players who got no kills and is calculated by taking the number of head-shot kills and dividing with the number of total kills done by a specific player in a specific match. Therefore, the total number of cheater data points $n_{cheater} = 1304$ and non cheater data points $n_{non\_cheater} = 6387$. The results can be seen in figure 5.9, which displays cheaters having significantly higher head-shot kill percentages, with a median of 74.2% compared to non cheater 43.8%.

Shots fired per kill We hypothesised, that due to aim cheats assisting in aiming and therefore requiring less shots to be taken at the opponent. This data excludes players who got no kills and is calculated by taking the total number of shots for a player in the entire game and dividing it with the number of total kills in that game for that player. The number of cheater data points is $n_{cheater} = 1279$ and non cheaters $n_{non\_cheater} = 6524$. Figure 5.10 showcases how cheaters generally fire their weapons less compared to non cheaters, with a median of 10.8 shots per kill, compared to non cheater median of 17 shots per kill.

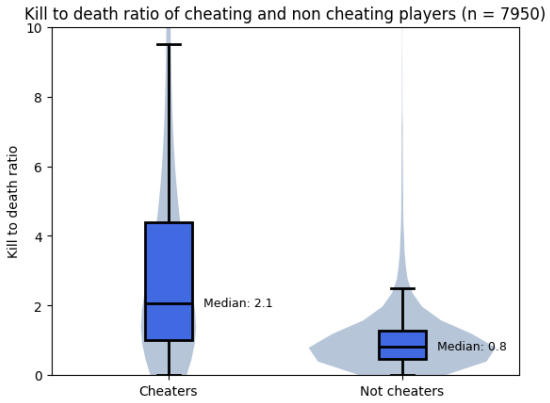

Figure 5.8: Kill to death ratio of cheaters and non cheaters in the whole dataset.

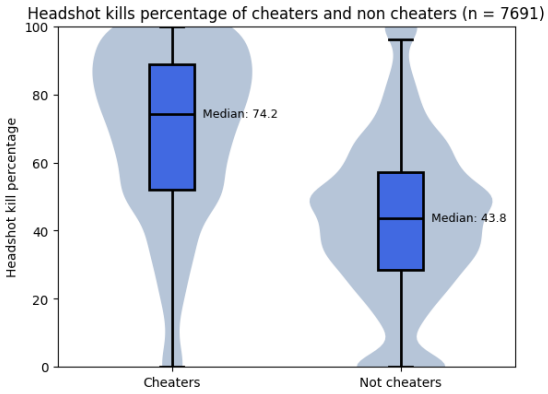

Figure 5.9: Head-shot kill percentage of cheating and non cheating players

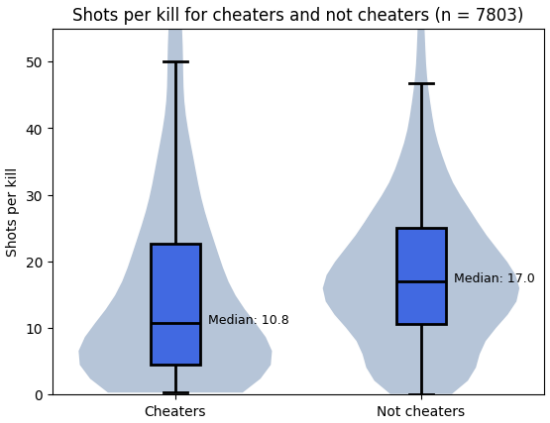

Figure 5.10: Shots per kill for cheating and non cheating players

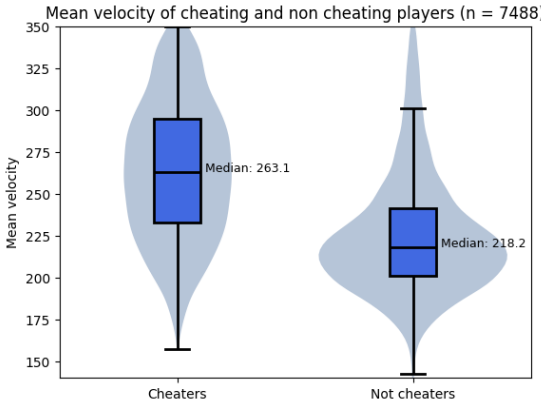

Figure 5.11: Mean velocities of cheating and non cheating players

Average velocity Due to cheaters often using bunnyhop[2] scripts, we hypothesised, that their mean velocity would be above that of non cheating players. This data takes the mean velocity of all ticks where velocity was above zero. All data points where mean velocity was above 350 were excluded, due to outlier cases. One such outlier is a player simply spawning into the map. In such cases velocity can reach extreme values, such as 10,000 or even more. These cases were excluded from this data. The number of cheater data points is $n_{cheater} = 1160$ and non cheater data points is $n_{non\_cheater} = 6328$. Figure 5.11 illustrates how the average velocity for cheating players is noticeably higher, with a median of 263.1, compared to non cheater players' median of 218.2.

5.3.2 Publishing dataset

The dataset set was published to HuggingFace under the name **CS2CD**, short for **Counter-Strike 2, Cheat Detection**. The dataset has the DOI `10.57967/hf/5315` and can be found here `https://huggingface.co/datasets/CS2CD/CS2CD.Counter-Strike_2_Cheat_Detection`.

5.4 Reflection

This project aimed at creating a dataset with cheater annotations for CS2. With 317 demos manually reviewed for cheaters and 1309 players labelled as cheater, it is fair to say that this goal has been met. However, there are certain compromises that had to be made for the sake of time efficiency.

The first point of critique was the fact that the not cheater data was not manually reviewed. It contained 478 matches with no VAC-banned players present. However, this does not mean that there are no cheaters present within that data. In fact, the subsample review of the not cheater data revealed, that small numbers of cheaters were present, as explained in section 5.2.2. Ideally, the not cheater data should have been manually reviewed as well, however due to the time intensity of the task, a review of the data was not done and the wrongly classified players were deemed an acceptable amount of noise within our data.

Secondly, no labelling regarding the type of cheat was done. This extra detailing of the label could prove to be extremely valuable, as it would allow for different machine learning models to be trained for different types of cheating, thereby resulting in more specialised models. However, labelling cheats and when they each occur is a very time intensive task. Furthermore, as mentioned in chapter 4, not all cheats are equally visible. Some cheats are even undetectable judging from behaviour alone. For these reasons, cheat type labelling was not done.

[2] A movement style where a player continuously jumps while moving and upon landing instantly jumps again within the same server tick. Counter-Strike 2 (CS2) uses a friction variable to slow the player down after landing, but if another jump is made upon the same server tick as landing, the player effectively removes all friction. This allows for increased movement speed[28].

Additionally, due to various factors, the cheater label might not be 100% accurate throughout a match. Currently, the labelling operates under the assumption that cheaters are cheating throughout the entire match; An assumption that does not hold for every match in the dataset due to toggling[3]. One possibility to mitigate this would have been to label the tick on when cheating started and ended. This way, the gameplay that happened without using cheats by someone who later in the game enabled cheats would not be classified as cheater in those sections. This would have been an extremely time intensive process and due to that was not done.

An alteration that could have potentially helped with data quality in the non cheater data was creating a social graphing logic. Due to trust factor[4] match making, players with low trust factor are matched with other players of similar trust factor. Players using cheats often have a low trust factor, hence if a player has played multiple matches with banned players, there is a higher likelihood of that player conducting cheater like behaviour[52]. This knowledge could be leveraged in order to flag games with no banned players as potentially cheater games. These matches could then be manually reviewed for cheaters. However, this social graph would be a sizeable development and was therefore not done.

5.5 Conclusion

The goal of creating a labelled CS2 dataset was achieved. This was done by scraping the match sharing service csstats.gg for publicly available `.dem`-files. Matches were categorised as cheater, when there was at least one VAC-banned player present, and not cheater, if there was no banned player present. This resulted in 317 matches with VAC-banned players present and 478 matches with no VAC-banned players present, totalling at 795 `.dem`-files.

The quality of the VAC-ban label was assessed. The results for the subset with a banned player present showed, that while the precision of the cheater label was good at 92.6%, the recall showed poor results at 24.7%. These results meant, that a manual labelling of the subset with banned players had to be conducted. Manual review of the subsample with no banned players present showed, that the precision of the not cheater label was 97.2%. This was deemed acceptable, therefore no manual review of this subset was done[28].

Due to `.dem`-files containing sensitive information, the data needed to be extracted and anonymised. For this, the python library demoparser2 was used[48]. Two types of data could be retrieved from a `.dem`-file - game ticks and events. Data from each `.dem`-file was extracted and subsequently anonymised. Game ticks were written to `.parquet`-files and events to `.json`-files. The total size of the dataset was 48.9 GB. The total amount of gameplay hours within this dataset is 300.7 hours. In total, 1309 players were labelled as cheater and 6641 as not cheater.

In section 5.3.1.1, data visualisations were made to highlight some behavioural differences between cheaters and non cheaters. These visualisations supported anticipated behavioural differences between cheater and non-cheater gameplay, including significantly higher kill to death ratios, greater shooting accuracy, and faster than average movement velocities. It also displayed the quality of the labels, as cheaters had significantly different behavioural patterns compared to non cheaters.

Although improvements could have been made to the scraping algorithm to ensure a greater collection of data containing cheaters and more descriptive labelling, time constraints limited our ability to do so. However, even with these constraints we believe that this dataset will act as a solid foundation for future works regarding cheat detection, and that this chapter along with the paper "Counter-Strike 2 Game data collection with cheat labelling" showcase methods that enables future expansion of the dataset[28].

Finally, the dataset was published on `https://huggingface.co/datasets/CS2CD/CS2CD.Counter-Strike_2_Cheat_Detection` in an organisation dedicated to this thesis.

[3] In the context of video games toggling refers to the action of toggling cheat software on or off.

[4] A hidden variable that aims to enhance players matchmaking experience by matching up players with similar trust factor scores[52].

Chapter 6

AntiCheatPT: Cheat detection using Transformers

6.1 Introduction

Machine learning has previously been used to perform cheat detection in the Counter-Strike franchise. However, papers reporting these findings used simpler models, in the form of recurrent neural networks [29] or convolutional neural networks [16]. Additionally, these papers used substantially smaller datasets, which could result in limited real-world applications. However, with the dataset created in the previous chapter, the barrier for creating machine learning models in this field is reduced.

Additionally, the transformer architecture has proven to be effective for cheating detection in sequential data[20][19][53]. However, the use of transformer architecture machine learning models have not yet been explored for cheat detection in the context of sequential CS2 game tick data. Additionally, the paper "XAI-Driven Explainable Multi-view Game Cheating Detection" by Tao et al. does not share any code, datasets, nor models, meaning that, to our knowledge, there exist no open-source behaviour-based cheat detection transformer models[20].

This chapter aims to explore the usage of the transformer architecture in behaviour-based cheat detection, and provide an open-source and reproducible model. All data used in this chapter was derived from data in chapter 5.

6.2 Methodology

6.2.1 Context windows

The input data to the machine learning model was named context windows, similarly to how they are named in large language models. The game CS2 runs on game ticks - updates to the game state that happen 64 times per second. These updates are all recorded in `.dem`-files, which was where the data was gathered from, as explained in section 5.2.3. A context window contained a series of tick vectors, where each tick vector is comprised of specific data points from the `.dem`-file. A single context window was centred around a single specific kill, which contained data from two players: attacker and victim. A tick vector contained specifically chosen data points from a single in-game tick.

Regarding the models developed for this chapter, named AntiCheatPT, three different size context windows of 1024 ticks (16 seconds), 512 ticks (8 seconds) and 256 ticks (4 seconds) were selected. The exact specifics of where a kill happened within a context window can be seen in table 6.1. Therefore, the input to the AntiCheatPT models were a matrix of 1024 rows by 44 columns for the AntiCheatPT_1024 model, 512 rows by 44 columns for the AntiCheatPT_512 model and 256 rows by 44 columns for the AntiCheatPT_256 model.

The amount of context window data points available depended on how many kills happened within our dataset. The quantity of context windows amounted to 18,423 labelled as cheater and 72,284

Context window size	Ticks before kill	Ticks after kill
1024 ticks (16 seconds)	896 ticks (14 seconds)	128 (2 seconds)
512 ticks (8 seconds)	448 ticks (7 seconds)	64 (1 seconds)
256 ticks (4 seconds)	224 ticks (3.5 seconds)	32 (0.5 seconds)

Table 6.1: Context window specifics

labelled as not cheater. This was less than the total amount of cheater and not cheater kills within our dataset, as can be seen on figure 5.4 in section 5.3.1. This was due to three types of kills being excluded: kills against bots, kills with grenades and kills against teammates. These were found to have been of little value to the problem of cheating detection.

6.2.1.1 Tick vector

The idea of collecting certain data points into a tick vector was inspired by other papers. These papers support the idea of extracting and calculating per tick statistics in order to use them for training in a machine learning model. the paper "A cheating detection framework for Unreal Tournament III: A machine learning approach" employs a similar idea of the attacker and victim relationship, while the paper "Cheat Detection using Machine Learning within Counter-Strike: Global Offensive" uses data from all players whilst focusing on a suspect. Both of these papers use the positional information regarding the included players along with other information such as calculated metrics. the sizes of these vectors vary among these two papers with the first paper using somewhere between 20-50 data points[1], compared to the 170 data points of the latter paper[29][13].

The input of the models could be thought of as a series of vectors. The information stored within a single tick vector can be seen in table 6.3. The first 24 data points in the vector describe the attacker's data and information about the kill within the context window. Especially important are attackers velocity (`attacker_vel`), pitch (`attacker_pitch`) and yaw (`attacker_yaw`), as these data points may highlight two specific cheats - bunnyhop scripting and spinbotting. The velocity data point was essential for detecting bunyhopping, due to the velocity of such a cheater consistently being well above 250, which is the maximum running speed for CS2. The yaw and pitch angles are essential for catching spinbotting, as rapid changes in yaw and pitch angles suggest the presence of spinbotting cheats. This is further supported by the delta pitch and yaw values (`attacker_pitch_delta` and `attacker_yaw_delta`). `attacker_pitch_head_delta` and `attacker_yaw_head_delta` are used to detect multiple types of cheats. For example, an aim hack may continuously lock onto an enemy's head, resulting in these values being close to 0 for an abnormal amount of time. The rest of the attacker data points were there to provide the model with additional information about the kill, as these could also point to other types of cheating being present. An example of such data is kills through vision obscuring smokes, as this may suggest the use of wall hack. The final attacker values were a one-hot encoding of the weapon type held by the attacker.

The following values were about the victim's data, starting with `X`, `Y`, and `Z` coordinates. Another victim related data point was whether the victim made noise. The reason for including noise data, was due to noise making a kill through an obscuring smoke or through a wall more credible, as the attacking player would be able to pinpoint the victim's location via these auditory cues.

Finally, a one-hot encoding of the played map was added, with 15 possible maps present. This was done so that the model could have the necessary map context of `X`, `Y`, and `Z` coordinates.

[1]The specifics of what data points were used in model training was not disclosed

Data point	Description
attacker_X	X coordinate of the attacking player
attacker_Y	Y coordinate of the attacking player
attacker_Z	Z coordinate of the attacking player
attacker_vel	Velocity of the attacking player
attacker_pitch	Pitch angle of the attacking player
attacker_yaw	Yaw angle of the attacking player
attacker_pitch_delta	Change in pitch angle of the attacking player since last tick
attacker_yaw_delta	Change in yaw angle of the attacking player since last tick
attacker_pitch_head_delta	Pitch angle distance from the victim's head
attacker_yaw_head_delta	Yaw angle distance from victim's head
attacker_flashed	Is the attacker currently flashed
attacker_shot	Did the attacker shoot their weapon on this specific tick
attacker_kill	Did the attacker kill the victim on this specific tick
is_kill_headshot	Was the kill a headshot
is_kill_through_smoke	Was the kill through a smoke
is_kill_wallbang	Was the kill through some wall or surface
attacker_midair	Did the kill happen while the attacker was midair
attacker_weapon_knife	Is the attacker holding a knife
attacker_weapon_auto_rifle	Is the attacker holding an automatic rifle
attacker_weapon_semi_rifle	Is the attacker holding a semi-automatic rifle
attacker_weapon_pistol	Is the attacker holding a pistol
attacker_weapon_grenade	Is the attacker holding a grenade
attacker_weapon_smg	Is the attacker holding a submachine gun
attacker_weapon_shotgun	Is the attacker holding a shotgun
victim_X	X coordinate of the victim player
victim_Y	Y coordinate of the victim player
victim_Z	Z coordinate of the victim player
victim_health	Health of the victim player
victim_noise	Did the victim player make noise
map_dust2	Is the played map de_dust2
map_mirage	Is the played map map_mirage
map_inferno	Is the played map de_inferno
map_train	Is the played map de_train
map_nuke	Is the played map de_nuke
map_ancient	Is the played map de_ancient
map_vertigo	Is the played map de_vertigo
map_anubis	Is the played map de_anubis
map_office	Is the played map cs_office
map_overpass	Is the played map de_overpass
map_basalt	Is the played map de_basalt
map_edin	Is the played map de_edin
map_italy	Is the played map cs_italy
map_thera	Is the played map de_thera
map_mills	Is the played map de_mills

Table 6.3: Data points within a single tick vector

6.2.1.2 Calculation and normalisation of data points

All data points were normalised prior to training. The normalisation process was not straightforward and is explained in the following paragraphs

Player X, Y, Z coordinates The coordinates for both attacker and victim were taken directly from the dataset. However, normalisation would not be trivial, as player coordinates can vary from map to map. Therefore, for each map, a maximum and minimum X, Y and Z coordinate were hardcoded and then used to normalise the data. The hardcoded values were found by going into the respective map in CS2 and manually finding the minimum and maximum coordinates a player can reach. The hardcoded minimum and maximum values can be seen in table 6.4. These minimum and maximum values were used to normalise raw coordinate data. After this normalisation, the values are clipped between 0 and 1, to catch potential outliers, such as players standing on top of each other in order achieve unexpected positioning.

Map	min X	max X	min Y	max Y	min Z	max Z
Dust 2	-2300	1800	-1200	3150	-130	400
Mirage	-2700	1500	-2700	1000	-400	150
Inferno	-1800	2700	-800	3600	-100	500
Train	-2200	1800	-1800	1800	-400	200
Nuke	-3000	3500	-2500	1000	-750	100
Ancient	-2310	1400	-2600	1800	-120	400
Vertigo	-2700	100	-1600	1200	11400	12100
Anubis	-2000	1810	-1810	3200	-150	200
Office	-1800	2400	-2200	1300	-350	10
Overpass	-4000	20	-3500	1700	0	800
Basalt	-2100	2000	-1700	2350	-100	400
Edin	500	3700	-350	4300	300	750
Italy	-1550	1100	-2200	2650	-200	300
Thera	600	4300	-2600	2200	-170	300
Mills	-4300	0	-5560	-300	-100	300

Table 6.4: Hard coded minimum and maximum values for map coordinates

Player velocities A player can only have a positive velocity value. The velocity of players is a recorded value found in the tick data. The fastest a legitimate player can move while running in CS2 is while holding the knife at 250 velocity, therefore normalisation was done by dividing the velocity by 250. Note, that when jumping, the velocity could go slightly above 250. This value was purposefully not clipped, as values higher than 1 may suggest the use of bunnyhop scripting.

Player pitch and yaw values The pitch and yaw values are stored in the tick data. Pitch is the vertical angular movement of a player, while yaw is the horizontal angular movement of a player. Therefore, the maximum pitch value is 90° and minimum is −90°. Similarly, maximum yaw value is 180° and minimum is −180°. These values were simply scaled to between 0 and 1 for normalisation.

Player pitch and yaw delta values The attacker pitch and yaw delta values represented a change in view angles between the previous tick and the current tick. These values were normalised by taking the absolute value of the delta and then dividing pitch by 45° and yaw by 90°. These values were chosen, because it is near impossible to change the view angle by more than 45° in 1 tick without using scripts such as spinbot. Although, this is technically possible by using an unusably high mouse sensitivity, a mouse sensitivity capable of moving the view angle by 90° would also prove highly impractical whilst playing the game. However, the values were clipped to between 0 and 1 nevertheless, to catch any unexpected outliers, such as script usage.

Player pitch and yaw head delta values The player's pitch and yaw head delta values quantify how far the attacker's aim is from the victim's head, with a value of 0 indicating a direct aim at the head. These values are calculated by determining the shortest angular difference between the attacker's and the victim's orientation for both pitch and yaw. Normalisation is applied by scaling values under 45 degrees for pitch and under 90 degrees for yaw by dividing them by 45 and

90 respectively. Any values exceeding these thresholds are capped at 1, representing a situation where the attacker is aiming far enough away from the victim's head that their shots should not logically hit the target

Player flashed The player flashed[2] value is calculated by taking the time, that a player was flashed for, from the `flash_duration` column in tick data and linearly scaling it from 1 (meaning a player is fully flashed) to 0 (no longer flashed). Note, that in CS2, the flash effect does not decrease linearly as it did in our context windows, but this was deemed an accurate enough representation.

Player shot This value is 1 on the tick that an attacker had shot a shot. `weapon_fire` is an event and was therefore stored in the event data.

Attacker kill data The values `attacker_kill`, `is_kill_headshot`, `is_kill_through_smoke`, `is_kill_wallbang`, and `attacker_midair` all describe the kill performed by the attacker. `attacker_kill` is 1 on the tick that the attacker got kills. The other values describe the kill that happened by having value 1 if true and otherwise 0. `player_death` is an event and was therefore extracted from the event data.

Weapon one-hot encoding As there is a big variety of weapons in CS2, a one hot encoding for all weapons would add a lot length to the tick vector. Weapons were grouped together based on weapon play styles and types, to form a shorter one hot encoding vector. Table 6.5 show all weapon groupings. The weapon held by the attacker was stored in the tick data in the `active_weapon_name` column.

Weapon group	Weapons
`attacker_weapon_knife`	Zeus x27, knife_t, knife, Bayonet, Bowie Knife, Butterfly Knife, Classic Knife, Falchion Knife, Flip Knife, Gut Knife, Huntsman Knife, Karambit, Kukri Knife, M9 Bayonet, Navaja Knife, Nomad Knife, Paracord Knife, Shadow Daggers, Skeleton Knife, Stiletto Knife, Survival Knife, Talon Knife, Ursus Knife
`attacker_weapon_auto_rifle`	AK-47, M4A1-S, Galil AR, SG 553, M4A4, AUG, FAMAS, M249, Negev
`attacker_weapon_semi_rifle`	G3SG1, SSG 08, AWP, SCAR-20
`attacker_weapon_pistol`	CZ75-Auto, Desert Eagle, Dual Berettas, Five-SeveN, Glock-18, P2000, P250, R8 Revolver, Tec-9, USP-S
`attacker_weapon_grenade`	C4 Explosive, Decoy Grenade, Flashbang, High Explosive Grenade, Incendiary Grenade, Molotov, Smoke Grenade
`attacker_weapon_smg`	MAC-10, MP5-SD, MP7, MP9, P90, PP-Bizon, UMP-45
`attacker_weapon_shotgun`	MAG-7, Nova, Sawed-Off, XM1014

Table 6.5: Weapon groups made for one hot encoding.

Player health The possible values for a player's health range from 0 to 100. Therefore, normalisation was done by dividing the health value with the maximum health value.

Player noise Player noise is determined based on player actions, specifically firing shots and movement speed, as these are the primary sources of player-generated noise in CS2. Each of these actions contributes 0.5 to the `victim_noise` field. For instance, if a victim is both running and shooting, their `victim_noise` field reaches 1. In CS2, players produce footstep sounds when their velocity exceeds 140, so 0.5 is added to the `victim_noise` field whenever this velocity threshold is crossed. An additional 0.5 is added to the field for each tick in which the victim fires their weapon. Note, that there are other actions in the game, that create audible sounds, such as reloading a weapon. However, these were not included as the radius in which a reload can be heard is no as large as footsteps or gunfire.

[2]In the context of CS2, being flashed refers to being blinded by a flashbang, a type of grenade that blinds and deafens players.

Map one-hot encoding The map played is represented by a 15 dimensional vector. Each value in the vector represents a map. The one-hot encoding of the map has the value 1 on the map index of the played map and 0 on all other columns. For example, when the map `de_dust2` is played, the `map_dust2` column has the values 1 and all other map columns have the value 0. The played map is saved in the events data under the key `Csstats_info`.

6.2.1.3 Data augmentation

Due to the class imbalance of cheater kills to not cheater kills, data augmentation was performed on the attacker and victim's `X`, `Y` and `Z` coordinates. This was because, it was not desirable that the models learned specific coordinates, where cheater or not cheater kills happen, but rather a function of relative positioning between two players. Therefore, Gaussian noise was added in small amounts to the coordinates of the attacker and victim. Note, that the exact same amount of random noise was added to the same coordinate of the attacker and victim, so that their relative positioning and distance from each other remained the same. This algorithm can be seen in Algorithm 2.

Algorithm 2 Data augmentation

Input: directory, axes list $[X, Y, Z]$, noise standard deviation $\sigma = 0.01$, augmentation count n_{aug}
for each file f in directory **do**
  $df \leftarrow$ load parquet file f
  for $i = 1$ to n_{aug} **do**
    $df\_aug \leftarrow df$
    for each axis a in ["X", "Y", "Z"] **do**
      $c_{\text{att}} \leftarrow$ `"attacker_"` $+ a$
      $c_{\text{vic}} \leftarrow$ `"victim_"` $+ a$
      $n \leftarrow$ sample from $\mathcal{N}(0, \sigma)$
      $df\_aug[c_{\text{att}}] \leftarrow \text{clip}(df\_aug[c_{\text{att}}] + n, 0, 1)$
      $df\_aug[c_{\text{vic}}] \leftarrow \text{clip}(df\_aug[c_{\text{vic}}] + n, 0, 1)$
    end for
    $new\_file \leftarrow f$.filename + "_aug$i$"
    save df_aug as Parquet to new_file
  end for
end for

The added noise was sampled from a normal distribution with mean 0 and a standard deviation of 0.01. The magnitude of the noise was deliberately small, since it was applied to already normalised coordinate values. A 0.01 change in a given coordinate represents a 1% shift in the map's total length. Additionally, the values were clipped to be between 0 and 1, such that no outliers occured.

In order mitigate class imbalances, the cheater data was augmented more than the not cheater data. Specifically, for every cheater data context window, three augmentations were made, meaning the number of cheater context windows increased from 18,423 to 73,692. For every not cheater context window, one augmentation was made, resulting in the number of context windows with not cheater label increasing from 72,284 to 144,568. Prior to the augmentation, the ratio of cheater to not cheater context windows was ∼1:4. After the augmentation, the ratio was ∼1:2 with the total number of context windows being 218,260. The cheater data was not augmented beyond 3 times, due to the potential risk of overfitting to the training data.

6.2.2 Transformer architecture

The AntiCheatPT's model architecture was a transformer encoder based on the original implementation of the transformer by Vaswani et al.[18]. Various models with three different context window sizes were developed to address the problem of fitting a function for cheating detection. Namely, AntiCheatPT_1024, AntiCheatPT_512 and AntiCheatPT_256 with context window sizes of 1024, 512 and 256 respectively. For all models, the number of features within the tick vector was 44. Various architectures of the transformer models were tested. Parameters, such as feedforward layer dimension, number of attention heads and number of transformer layers were all tested using various training hyperparameters to observe how the models training converged.

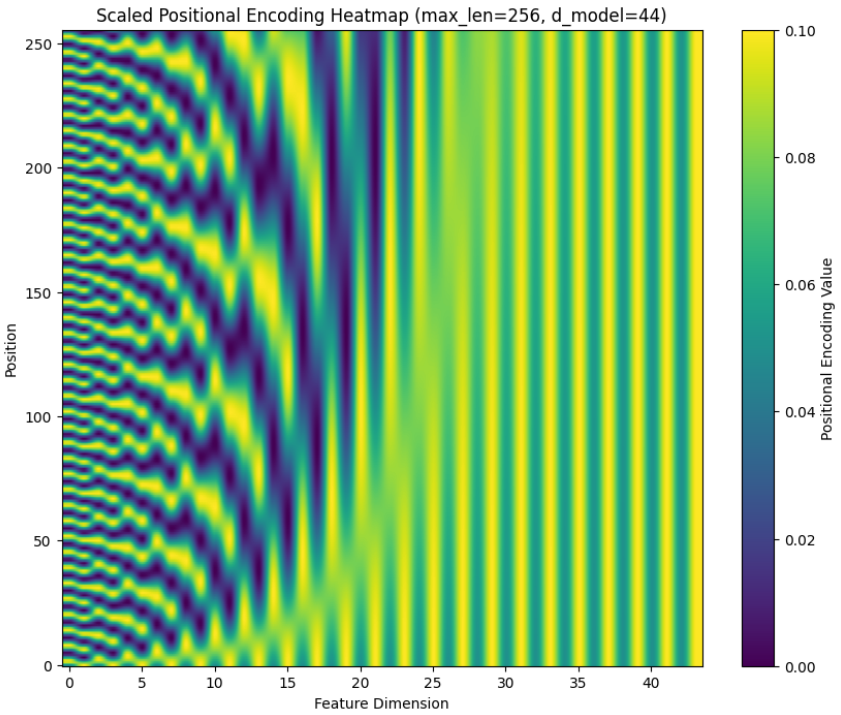

Figure 6.1: Visualisation of the positional encoding

6.2.2.1 Positional encoding

A positional encoding was added to the input data, in order to create positional dependency. The implemented positional encoding module precomputed a set of positional vectors using sine and cosine functions of varying frequencies, following the approach introduced in the original transformer architecture by Vaswani et al. [18][54]. For a given position pos, dimension i and model feature dimension d_{model}, the encoding PE is defined as follows for even and odd i respectively:

$$PE_{pos,2i} = sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{6.1}$$

$$PE_{pos,2i+1} = cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{6.2}$$

The result of this contained values on the range from -1 to 1. Since the positional encoding would be added to a normalised numerical input in the range of 0 to 1, a scaling of the positional encoding was required. This is because the AntiCheatPT model utilises numerical inputs instead of embedding dimensions, which helps preserve the essential information contained within the input. The following is the calculation of the final positional encoding:

$$PE_{out} = \frac{PE_{in} + 1}{2} \cdot PE\_scale \tag{6.3}$$

The PE_scale value was chosen to be 0.1, resulting in all positional encoding values ranging from 0 to 0.1. A visualisation of the positional encoding can be seen in figure 6.1.

6.2.2.2 Model input

The input of the AntiCheatPT model can be seen in figure 6.2. This includes the aforementioned tick vectors, onto which the result from the positional encoding algorithm is added. Note, that the classification token `[CLS]` is prepended to the beginning of the input.

6.2.2.3 Model output

For classification purposes, a classification token is prepended to the context window sequence. After passing through the transformer layers, this token is extracted as the aggregate representation of the sequence. The extracted token is then processed through two linear layers: The first transforms the 44-dimensional feature space into 128 dimensions, followed by a `ReLU` activation.

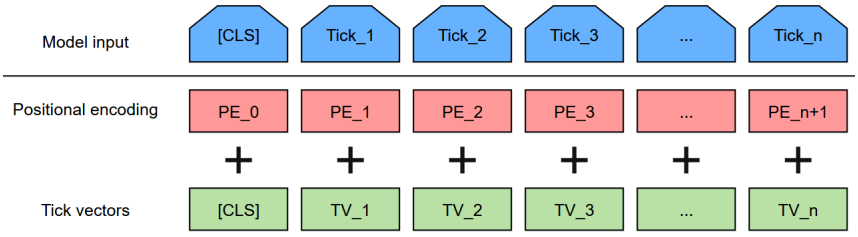

Figure 6.2: Input of the AntiCheatPT model, consisting of a sequence of tick vectors and positional encodings. `CLS` is the prepended classification token.

This was done to allow the model to learn non-linear interactions between the classification token features. Finally, the second linear layer maps the 128-dimensional representation to a single output dimension. This output is subsequently passed through a sigmoid function to produce a probability score, where 1 indicates a cheating label and 0 a non cheating label. It is important to note that the sigmoid activation is not part of the model architecture itself, and must be applied externally. Therefore, the model outputs logits rather than probabilities. The reason for this will be explained in section 6.2.2.4. A visualisation of the final output layers can be seen in figure 6.3.

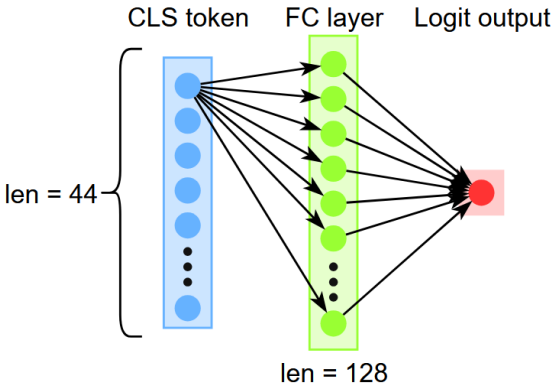

Figure 6.3: Final layers of the AntiCheatPT model

6.2.2.4 Loss function

The model is trained using PyTorch's `BCEWithLogitsLoss`, which combines a sigmoid layer and the binary cross-entropy loss in a single, numerically stable function. This choice is preferred over applying a separate sigmoid activation followed by a standard binary cross-entropy loss, as it reduces the risk of numerical instability during training[55]. This is the reason, why the model outputs logits and contains no sigmoid activation function.

6.2.2.5 Optimiser

The chosen optimiser algorithm was AdamW[56]. This algorithm was chosen for it's reliability and stability in training. The initial learning rate for the optimiser was $1 \cdot 10^{-4}$. However, variations of the models using a lower and a higher learning rate were also tested, namely $1 \cdot 10^{-5}$ and $1 \cdot 10^{-3}$.

6.2.2.6 Scheduler

A scheduler was added to the training to slow down the learning rate as a minima was approached. The chosen scheduler was `StepLR` from Pytorch. This scheduler decays the learning rate by `gamma` every `step_size` epochs. The chosen hyperparameter for `gamma` was 0.5 and `step_size` was 10[57].

6.2.3 Model training

6.2.3.1 Train, validation and test data

The train, validation and test split was done as 70%, 15% and 15% respectively. Note, that the way the data split was done was by using file names from different matches as keys. Those keys

would then be split into train, validation and test sets. Then, from the keys, the context windows were retrieved. This way, no context windows from the same match would end up in different sets. That amounted to 159,592 (54,564 cheater and 105,028 not cheater) context windows in the training set, 28,838 (10,168 cheater and 18,670 not cheater) in the validation set and 12,675 (2,240 cheater and 10,435 not cheater) in the test set. Due to the data being split by matches, rather than context windows, the actual data splits deviated from the desired 70%, 15%, 15% split. The actual split of the data was approximately 73.1%, 13.2% and 13.7% for train, validation and test split respectively. Note, that the 13.7% included augmented data, however the test set in reality excluded any augmented data and was therefore smaller than the validation set. Thereby, the test set contained a realistic scenario of inputs.

6.2.3.2 Training setup

The models were trained for n epoch, saving the model at each epoch. The number of epochs the models trained for varied by the purpose of the model. The random seed was set manually for reproducibility. The tested seeds were 41, 42 and 43. All seeds converged to the same minima within 20 epochs, therefore the seed 42 was chosen. The chosen batch size for training was 128, as that was the maximum that our hardware could allow for all context window sizes.

After each epoch, inference was performed on the validation set to calculate the following metrics:

- Average validation loss
- Accuracy
- Precision
- Recall
- Receiver operating characteristic (ROC) area under the curve (AUC)

These metrics were saved along with the model parameters, optimiser parameters, scheduler parameters, epoch and average training loss as a `.pth`-file. By selecting models with the lowest validation loss, overfitting was minimised.

6.2.4 Model testing

The testing of a model was done by loading the trained parameters and creating the train, validation and test data split the exact same way as in the training process, including using the same manual seed. Note, that the seed for data splitting was different (41) from the seed used for initialising model weights (42). Following that, inference was performed on the model using the test data, and the following metrics are saved to a `.pth`-file:

- Accuracy
- Recall
- Precision
- Receiver operating characteristic
- F1-score
- True positive count
- False positive count
- True negative count
- False negative count
- True labels
- Output logits
- Output probabilities

6.3 Results

This section presents the results of the machine learning experiments conducted for cheater detection in CS2. A transformer-based architecture was trained and evaluated under various parameter settings to assess it's performance, robustness, and sensitivity to key hyperparameters, such as the learning rate. The evaluation metrics used include accuracy, precision, recall, and area under the receiver operating characteristic curve, as these provide a comprehensive understanding of the model's behaviour while training. In total, three different context window sizes were trained, which are 1024, 512 and 256.

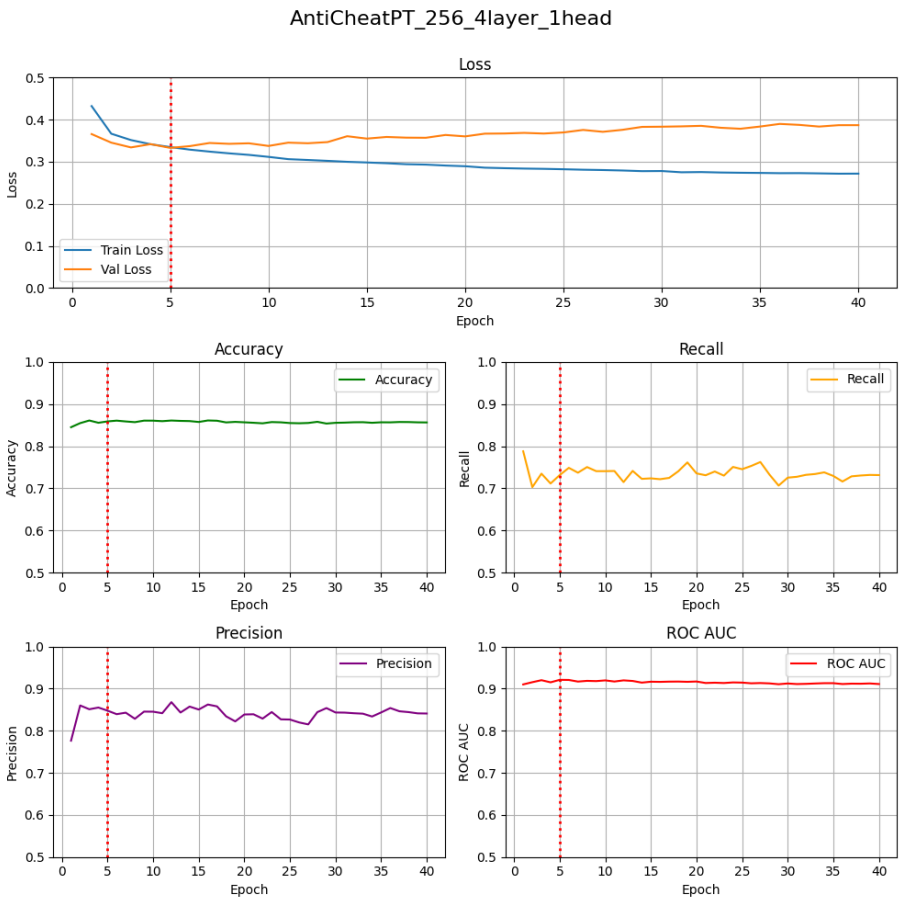

Figure 6.4: Training of AntiCheatPT_256

6.3.1 Context windows 1024 and 512

Ideally, a larger context window should give the model more opportunity to classify cheating. However, getting a larger context window size to generalise well to the problem of cheating detection proved to be a challenge with the available data. Generally, the training loss did converge well, however, validation metrics were not converging well with 1024 and 512 context window sizes. This may be due to an excess of noisy data points within a longer context window. Three different learning rates of $1 \cdot 10^{-4}$, $1 \cdot 10^{-5}$ and were used. A learning rate of $1 \cdot 10^{-3}$ generally began to overfit to the training data too fast, leading to an increase in validation loss. A learning rate of $1 \cdot 10^{-5}$ failed to learn a good generalisation of the data, leading to high losses in both training and validations data. The ideal learning rate seemed to be $1 \cdot 10^{-4}$, as this learning rate lowered the losses of the training and validation sets without immediate overfitting. Due to the quantity of models trained, only the best resulting model will be discussed in length. That was the AntiCheatPT_256 model. Further information regarding these training metrics can be seen in appendix D.

6.3.2 AntiCheatPT_256

The best performing model used a context window size of 256. All the hyperparameters and used components for this model can be seen in table 6.6 and the training metrics can be seen in figure 6.4.

The training loss of AntiCheatPT_256 converged at approximately 0.28, while the validation loss was lowest at epoch 5, with an average loss of approximately 0.34. After epoch 5, the validation loss began to rise and the model began to overfit to the training data. Therefore, the model at the end of epoch 5 was chosen as the final model.

Inference was run on the test set for this model. Note, that the test set contained no augmented data, and therefore had fewer data points than the validation set and had a larger label imbalance of $\sim 1:5$ of cheater to not cheater label. The results can be seen in figures 6.5 and 6.6 and in

Component	Value
Context window size	256
Transformer layers	4
Attention heads	1
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table 6.6: Training components for AntiCheatPT_256

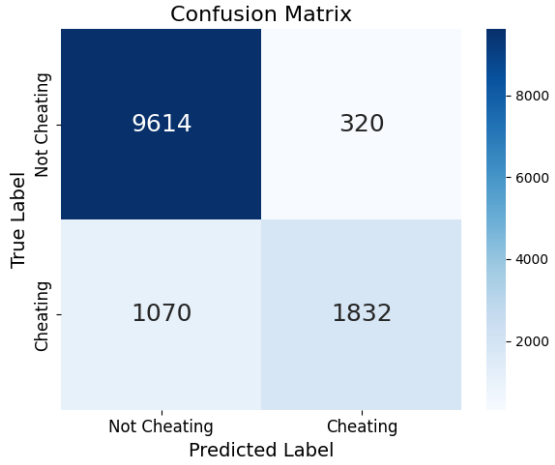

Figure 6.5: Confusion matrix at 0.7 threshold

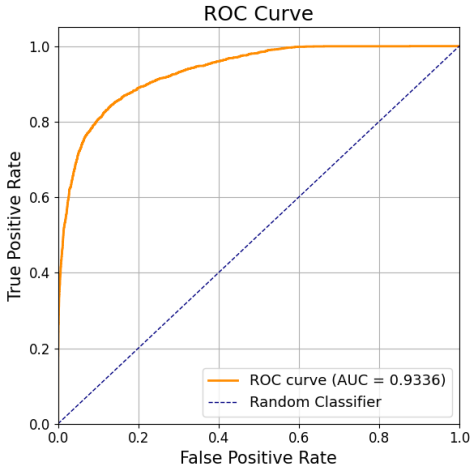

Figure 6.6: ROC curve

table 6.7.

Metric	AntiCheatPT_256	RNN[29]	Transformer[19]
Accuracy	0.8917	0.8-0.9	0.9836
Precision	0.8513	-	-
Recall	0.6313	-	-
F1-score	0.7250	-	-
Specificity	0.9678	-	-
ROC AUC	0.9336	-	0.9694

Table 6.7: Test metrics

The AntiCheatPT_256 model showed good results in classifying the presence of cheating, with very few false positives present. Note, that when manually labelling cheaters, often times a single kill is not enough to label a player as a cheater. Therefore, the rate of false negatives is not a big concern, however, a large false positive rate would be problematic. Therefore, in the case of real world application, we recommend the evaluation of several kills rather than one. An example of cheating probability prediction for players - a cheater and non cheaters in the same match, can be seen in figure 6.7. As can be seen on the figure, there were instances, where a cheating player gets a seemingly legitimate kill, and vice versa, where a non cheating player gets a kill, that raises the cheating probability. However on average, the predicted probability that a cheater was cheating was higher than that of the non cheating player.

6.3.3 Real world usage

In order to further investigate the real world usage of such a machine learning model the inference time of the model (as seen in table 6.8) is used. CS2's record number of concurrent players is 1,862,531 [34]. Using the assumption that 15% of all concurrent players are cheating, we can calculate the number of cheaters playing simultaneously.

$$1,862,531 \cdot 0.15 = 279,379.65$$

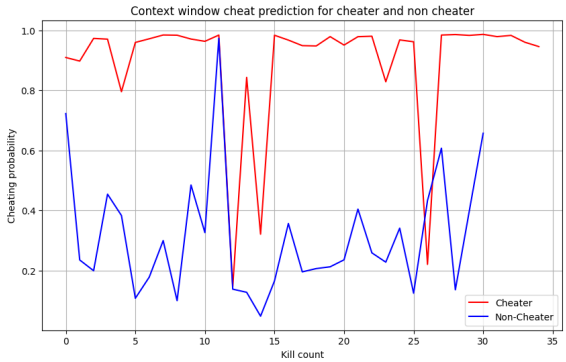

Figure 6.7: Cheating probability of kills for a cheater and non cheater. These context windows are from cheater data, file 18, player 3 (cheater) and players 1, 2, 4 and 6 (not cheater)

Training time (5 epoch)	40.8 min
Inference time (12,675 data points)	42.446 s
AVG inference time (1 data point)	3.3488 ms

Table 6.8: Train and test time for the AntiCheatPT_256 model using an RTX6000 GPU.

Further assuming that in each of these games a cheater killed all 5 players on the opposing team in the least amount of rounds (13) results in the following number of kills per cheater:

$$13 \cdot 5 = 65$$

Using the number of concurrent cheaters and the number of kills per cheater, we can calculate the total number of kills:

$$279,379.65 \cdot 65 = 18,159,677.25$$

From the total number of kills, it is possible to calculate the length of time needed to perform inference using an RTX6000 GPU.

$$18,159,677.25 \cdot 0.00335\text{s} = 60813.13\text{s} = 16.89\text{h}$$

Thereby, if the peak number of concurrent players consisted of 15% cheaters and those cheaters performed 65 kills a game, this would result in 16.89h of inference. Notably, a company such as Valve could possess a significantly larger amount of GPU compute than what was available to us. Furthermore, this number assumes that each cheater was competing in a unique match, however because of trust factor matchmaking, it is more likely that cheaters play against each other [52]. If Valve should be able to handle peak user traffic using this estimation +16 RTX6000s are needed, as CS2 matches can take up to an hour. Conversely, Valve could posses faster GPUs than the RTX6000. All, in all this estimation show that the usability of such a model in real life scenarios is feasible. In terms of viability, this method would require Valve to retrain the model if new cheats are developed that in some way changes the behaviour of the playable character.

6.4 Reflection

During the training of the transformer models, several training components were explored; however, not all possible components were exhaustively tested. Specifically, the selection of loss function, optimiser, and scheduler was not systematically evaluated, and there may exist better configurations. In particular, the scheduler settings were relatively aggressive, and additional experimentation with different scheduler types and parameters could have yielded more stable or efficient training. Note, that for the AntiCheatPT_256 model, the best results were after epoch 5, meaning the scheduler never interacted with the learning rate.

Additionally, a factor for future development would be altering the plot of the loss curves. As can be seen on figure 6.4 and in appendix D, the loss curves do not visualise the learning of the model very well. This may be due to the batch size and the size of the dataset. Since the training dataset consisted of 159,592 context windows and the batch size was 128, each epoch represents over 1200 steps taken with the optimiser. Hence, future visualisations of loss curves for similar projects may

find it helpful to plot the loss every n steps rather than after every epoch.

In creating the context windows for the transformer model, we operated under the assumption that if a player was observed cheating at any point during a match, they were likely cheating throughout the entirety of that match. However, this assumption is not foolproof, as toggling[3] can be a feature of the cheating software used. Furthermore, as discussed in chapter 4, participants noted that they might review larger portions of a match when uncertain about judging potential cheating behaviour. The current size of the context window does not account for this tendency. Generating context windows based on entire matches could better reflect these patterns, but would necessitate a larger dataset than is currently available. Thus, models incorporating longer match sequences will require an expanded dataset.

Although the train, validation, and test splits were intended to be approximately 70%-15%-15%, the actual distribution deviated slightly due to the chosen splitting method. To avoid placing gameplay data from a single match into multiple splits, the data was partitioned by entire matches rather than by individual kills or players. Due to the data being split by matches, rather than context windows, the actual data splits deviated from the desired 70%, 15%, 15% split. The actual split of the data was approximately 73.1%, 13.2% and 13.7% for train, validation and test split respectively. While a more precise split could have been achieved using a knapsack algorithm to optimise the balance, time constraints led to adopting this approximate approach, which still maintains the integrity of the data partitioning by match.

An alteration that potentially could increase the stability of training is artificially reducing the tick rate. Currently all context windows operate at 64 ticks per second. However, some papers report slowing the tick rate to as low as 5 ticks per second with great results, albeit small datasets[13]. This reduction in tick rate would shorten the context window, thereby decreasing the number of self-attention pairs, resulting in faster train- and inference times. However in section 4.3.2, participant 71 mentions that judging cheaters in CS:GO Overwatch cases was difficult, specifically due to the demo tick rate of CS:GO being too low. The `.dem`-file tick rate of CS:GO was 32 ticks per second i.e. half of the CS2 `.dem`-file tick rate. From this statement, we can deduce that lowering the tick rate can result in a lower cheat detection accuracy.

Due to the imbalance of map distributions within the cheater dataset, there is a potential risk that the model may develop a bias, inadvertently associating certain maps with cheating behaviour. Since this imbalance was not explicitly addressed or tested, the model's predictions could be disproportionately influenced by the map being played, potentially misclassifying players. Future work should involve testing the model's performance across various maps and, if necessary, incorporating map specific normalisation or balancing techniques to mitigate this bias.

Furthermore, variations in the tick vector content were not tested, yet they may represent the most crucial architectural component. This aspect could be particularly significant, as features like one-hot encoded map data may introduce noise rather than contributing meaningful context to the model.

6.5 Conclusion

This chapter focused on creating a machine learning model for behavioural cheat detection using the transformer architecture. Furthermore, we aimed to be fully transparent on the development and creation of the model, so recreation of our results would be possible. The final model was called AntiCheatPT_256 with 256 referring to the length of the context window.

The models trained in this chapter required an input. In order to leverage as much data as possible from the CS2CD dataset created in chapter 5, context windows were centred around kills. Data extracted, in order to create each context window, mainly touched on the attacker's position and status. Additionally, context windows included information regarding the victim and the map the game was played on. Context windows varying in length were made: 1024 ticks, 512 ticks, and 256 ticks. Since no reduction in tick rate was done, all context windows shared the same tick rate as the CS2 servers of 64 ticks per second. Thereby, each context window was 16, 8, and 4 seconds

[3] In the context of video games toggling refers to the action of toggling cheat software on or off.

long. Furthermore, training and validation data was augmented to address class imbalances. The augmented class balance became 1:2 post augmentation, rather than being ∼1:5.

Various architectures and model components were trained to assess their ability to generalise to the problem of cheating detection. These included different seeds, learning rates, number of transformer layers, number of transformer heads and dimension of the transformer feedforward layer. The 1024 and 512 context window sizes struggled to find a good generalisation. The 256 context window size proved to be the most stable, with an accuracy of 89.17%, precision of 85.13% and F1-score of 0.7250. The imperfections of the model may stem from the assumption that a cheater was cheating throughout the game. The existence of toggling proves this assumption to be false. Furthermore, during the labelling process not every kill could determine whether a player was a cheater or not. As mentioned in chapter 4, often watching the whole match is necessary in order to determine whether a player is a cheater or not. Hence, it is recommended to use the model on multiple kills rather than only one. For real world usage it is crucial that the False Positive Rate (FPR) is as low as possible. The AntiCheatPT_256 model has a non-zero FPR. Although it can perhaps be expected to have a non-zero FPR, it is also due to the fact that only the subset of data containing a VAC-banned player was manually labelled. From a manually labelled sample from this subset, we know that there are unlabelled cheaters in this subset. Thereby, the label quality of the no cheater subset can attribute to the FPR being elevated.

While the model has certain limitations, this chapter reinforces the idea that a transformer model is suitable for cheat detection in FPS video games such as CS2. Furthermore, this chapter illustrates the viability of such models in real-world scenarios, despite the need for further refinement to achieve full applicability.

Chapter 7

Conclusion

The goal of this thesis was to develop a behavioural cheat detection model using transformer architecture, applied to CS2 gameplay data. In order to achieve this, a three-part approach was taken: A survey on manual cheat detection practices, a data collection and labelling process, and the development and evaluation of a cheat detection transformer model.

The survey results were used to synthesise a manual cheat detection framework based on community input. Respondents commonly cited behavioural indicators such as imbalances in skill expression, advantageous timing, and abnormal view angles. However, responses lacked detailed strategies for identifying spinbots and aimbots.

The data collection chapter outlined the construction of a labelled dataset of 795 matches. Data was collected by scraping match data from the website csstats.gg and labelling matches containing at least one VAC-banned player manually.

Finally, the AntiCheatPT_256 transformer model was developed through iterative experimentation. The model achieved an accuracy of 89% and an ROC AUC of 93%, demonstrating strong potential for detecting behavioural anomalies associated with cheating.

Each part of this thesis builds upon the previous to form a cohesive pipeline for behavioural cheat detection. The survey laid the conceptual groundwork by identifying how players, whom had previously done overwatch cases, detect cheating which informed the labelling process. The data collection and labelling process used these insights to create a dataset that reflects realistic cheating scenarios based on community-validated signals. Finally, the AntiCheatPT_256 model showed the viability of real wold applications of behavioural cheat detection models. Together these components demonstrate the value of integrating domain knowledge and machine learning to tackle behavioural cheat detection.

While promising, the work has several limitations. The most notable being the fact that manual labelling was only performed on matches with at least one VAC-banned player. As a result, the data quality of the no cheater subset may affect the subsequent models abilities to classify cheaters. This effect is partially seen in the FPR and FNR of the AntiCheatPT_256 model. Furthermore, the AntiCheatPT_256 model was only tested using data from within the dataset, however further testing could have been conducted using professional match data from hltv.org. Due to the model primarily being trained on non-professional matches, testing using this data could give an important insight in how the model classifies professional players. Lastly, since no participants described spinbots and aim hacks in the survey, we had to make our own framework for detecting these types of cheats. Although the ease of detection is described in the survey, the method used for detecting these cheats don't necessarily reflect the broader methods used by the community.

This thesis contributes an open-source labelled dataset for cheat detection, thereby removing barriers for further research in the field. Furthermore, this paper introduces the first reproducible transformer model that can perform behavioural cheat detection on CS2 gameplay data. Lastly, this work illustrates that the use of such models is a feasible and viable solution for companies such as Valve.

Chapter 8

Future work

This thesis has investigated the application of Transformer-based models to the task of cheat detection. While one specific approach has been examined, the broader problem remains largely open and invites further exploration thanks to the open source CS2CD dataset. Hence, this chapter outlines several potential directions for future research in this domain.

The AntiCheatPT model offers a promising foundation, but its current implementation only outputs probabilities of cheating within individual context windows. Future work could involve building a secondary model that aggregates these predictions to classify entire players as either cheaters or non-cheaters, based on multiple context windows. Additionally, exploring larger context window sizes, while ensuring model stability during training, may enhance performance, as larger windows are more likely to capture instances of cheating. Employing established techniques such as model warm-up and pre-training could further improve convergence and stability, potentially leading to a more robust and accurate model.

As discussed in section 4.4 the manual behavioural cheat detection methodology is incomplete. Although mentioned by several participants in the survey, no participants described detection of spinbots and aim hacks. Further development of a standardised manual cheat detection method could be done using additional surveying with questions specifically targeted for detection of the various cheating types. Otherwise, this information may be extracted from an analysis of online discourse regarding the topic.

Additional improvements to the dataset quality could be made, such as specific labelling of the cheats used by the cheaters and specific tick markers describing when cheating began and ended. This is a considerably large task, as manual labelling of specific cheats would require more in-depth analysis of the cheaters, including trying to estimate if the cheating began from the beginning of the game or if the cheater performed toggling.

Aside from improvements in the data collection method, a bigger issue to tackle is further data labelling of the CS2CD subset with no VAC-banned players present. Counter-Strike data labelling is a very time consuming task, and therefore it may be plausible to leverage a community effort in labelling the dataset. However, the setup for this to even occur is a large project in it of itself. In order to even share `.dem`-files, it is required to anonymise them as it otherwise could violate GDPR[50, recital 30]. Since `.dem`-files are binary files, it is required to alter these files in some way to remove the usernames and other identifying information. Currently, there is no way of doing this. Therefore, the first step in leveraging community efforts in labelling data requires an anonymisation framework that allows for a `.dem`-file to be altered but remain playable through the client.

As discussed in section 4.3, participants expressed dissatisfaction with the lack of transparency in the cheat detection process. While one study has explored the explainability of comparable cheat detection models, it's description of methods is limited at best[20]. Nevertheless, the methods proposed in that work could potentially be adapted to provide users with more detailed justifications for their bans, offering a meaningful alternative to the current standard practice of issuing bans without explanation.

While this thesis has focused on detecting cheating in Counter-Strike 2 using transformer-based

models, it's broader contribution lies in incorporating player perspectives into the data labelling process. By conducting a survey to inform how gameplay data should be labelled, this work grounds model development in community-informed definitions of cheating. The resulting dataset and model provide a foundation for building more robust and context-aware cheat detection systems in the future. As the field moves forward, prioritising transparency, fairness, and trust in automated moderation systems remains essential, while offering a promising direction for future research and development.

Glossary

aim hack A class of cheats involving the use of one or multiple aim and/or trigger assisting cheats. These can include but are not limited to recoil control system, trigger aim and trigger bot[28]. 5, 15, 17, 20, 22, 27, 31, 44, 45

anti-aim A technique often used in cheats that contorts the playable character in a manner, such that it's hitbox is hard or impossible to target. If cheats are involved, they are often very easily identified due to unnatural movement[28]. 5, 48

anti-cheat In the context of video games, an anti-cheat is a piece of software that prevents cheating[1]. 1–3, 5, 10, 20, 47

badge In the context of Counter-Strike 2 (CS2), a badge is a type of collectible that are displayed on a users steam profile and within the Counter-Strike game. Examples of these are Service Medals, Operation Coins, Major Trophies and Coins as well as Collectible pins.[35]. 15, 19, 20

bunnyhop A movement style where a player continuously jumps while moving and upon landing instantly jumps again within the same server tick. Counter-Strike 2 (CS2) uses a friction variable to slow the player down after landing, but if another jump is made upon the same server tick as landing, the player effectively removes all friction. This allows for increased movement speed[28]. 28, 31, 33

CAGR Compound Annual Growth Rate. 2

closet cheating A cheating technique in which a player, who is deploying cheats, tries to hide that fact that their abilities are mechanically enhanced. 14, 17

commendation Commendations in CS2 are a form of peer recognition where players can positively rate their teammates after a match. These commendations fall into three categories: Friendly, Teaching, and Good Leader. They are intended to encourage positive behaviour and sportsmanship within the community. 15

corner clearing A tactical maneuver in which a player systematically checks potential enemy positions—especially tight angles and corners—when entering or moving through areas of the map. 16

counter strafing A movement technique used to quickly stop and shoot accurately after moving. It involves pressing the opposite directional key to instantly halt momentum, allowing for precise, accurate shooting with minimal delay. 16

crosshair placement Crosshair placement refers to the strategic positioning of a player's aiming reticle (crosshair) in anticipation of enemy presence. 15

CS2 Counter-Strike 2. 3, 5–12, 14, 15, 18–20, 22, 27–31, 33, 34, 38, 40–44, 47

economy Refers to the in-game currency system that determines a team's ability to purchase weapons, armour, and utility. 16

flicking Aiming technique where the player rapidly moves the crosshair from one position to another — typically to snap onto a target. 16

FPS First Person Shooter. 3, 4, 43

game sense An abstract but essential skill referring to a player's intuitive understanding of the game's flow. It includes predicting enemy behaviour, anticipating rotations, understanding timing, and making strategic decisions based on incomplete information. 15, 16

jiggle peeking A defensive technique where a player quickly peeks around a corner in and out of cover to gather information or bait enemy fire without fully exposing themselves. 16

movement tracking The ability to follow an enemy's movement smoothly with one's crosshair. 15

recoil control The technique of compensating for a weapon's recoil pattern by dragging the mouse in the opposite direction of the spray. 15

recoil control system (RCS) A type of scripting cheat that automatically negates the recoil effect of firing a weapon, giving perfect control over the weapon[28]. 5, 47

spinbot Type of cheat where trigger bot is combined with anti-aim. This cheat makes sure that the player using a spinbot is even more difficult to be shot at[28]. 5, 15, 20, 22, 31, 33, 44, 45, 56

toggling In the context of video games toggling refers to the action of toggling cheat software on or off. 29, 42, 43, 45, 48

trigger aim An automation tool that allows for precise aiming. However, as the mouse trigger remains manual, the reaction time of the player remains within human capability. Depending on the settings of the cheat software, trigger aim results in a high shot to kill accuracy and a natural reaction time, but may result in an unnatural aiming behaviour as the crosshair in some instances may appear to snap onto an opposing player[28]. 5, 47, 48

trigger bot An automation tool that assists precise aiming by automatically triggering a mouse click when an enemy is detected in the player's crosshair, resulting in a precise shot and natural aim[28]. 5, 47, 48

trust factor A hidden variable that aims to enhance players matchmaking experience by matching up players with similar trust factor scores[52]. 29, 41

utility Collective term for non-weapon equipment such as grenades, flashbangs, smokes, molotov/incendiary grenades, and decoys. 15, 16, 47

VAC Valve Anti-Cheat. 1, 3, 5, 13, 20–22, 26, 28, 29, 43–45

wall hack A type of cheat that reveals hidden characters and objects through walls, giving an unfair advantage[28]. 17, 20, 31

References

[1] Nisha Ellis. *understand-anticheat*. English. June 2020. URL: https://www.schellman.com/blog/cybersecurity/what-is-anti-cheat (visited on 05/19/2025).

[2] Grand View Research. *Video Game Market Size, Share & Trends Analysis Report By Device (Console, Mobile, Computer), By Type (Online, Offline), By Region (Asia Pacific, North America, Europe), And Segment Forecasts, 2023 - 2030*. English. Market analysis report GVR-4-68038-527-4. The Americas, Europe, and Asia: Grand View Research, 2023, p. 160. URL: https://www.grandviewresearch.com/industry-analysis/video-game-market (visited on 05/19/2025).

[3] Verified Market Research. *Anti-cheat-analysis*. English. 2023. URL: https://www.verifiedmarketresearch.com/product/anti-cheat-software-market/ (visited on 05/19/2025).

[4] Fabien A. P. Petitcolas. "Kerckhoffs' Principle". In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 675–675. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_487. URL: https://doi.org/10.1007/978-1-4419-5906-5_487.

[5] Bryan van de Ven. "Cheating and anti-cheat system action impacts on user experience". English. Bachelor. Radboud University, Jan. 2023. URL: https://www.cs.ru.nl/bachelors-theses/2023/Bryan_van_de_Ven___1024205___Cheating_and_anti-cheat_system_action_impacts_on_user_experience.pdf.

[6] Paul A. Karger and Andrew J. Herbert. "An Augmented Capability Architecture to Support Lattice Security and Traceability of Access". In: *1984 IEEE Symposium on Security and Privacy*. 1984, pp. 2–2. DOI: 10.1109/SP.1984.10001.

[7] Samuli Lehtonen. "Comparative Study of Anti-cheat Methods in Video Games". English. PhD thesis. University of Helsinki, Mar. 2020.

[8] Steamdb. *Technologies*. English. URL: https://steamdb.info/tech/ (visited on 05/19/2025).

[9] HoomanSA7. *EasyAntiCheat Kernel*. Oct. 2024. URL: https://answers.microsoft.com/en-us/windows/forum/all/easy-anti-cheat-driver-incompatible-with-kernel/27da6244-8236-4dd1-ad29-5d7009ae5e50 (visited on 05/21/2025).

[10] Sam Collins et al. "Anti-Cheat: Attacks and the Effectiveness of Client-Side Defences". In: *Proceedings of the 2024 Workshop on Research on Offensive and Defensive Techniques in the Context of Man At The End (MATE) Attacks*. CheckMATE '24. event-place: Salt Lake City, UT, USA. New York, NY, USA: Association for Computing Machinery, 2024, pp. 30–43. ISBN: 9798400712302. DOI: 10.1145/3689934.3690816. URL: https://doi.org/10.1145/3689934.3690816.

[11] Steam. *Games using VAC*. English. URL: https://store.steampowered.com/search/?category2=8 (visited on 05/21/2025).

[12] Gabe Newell. *Valve, VAC, and trust*. English. Feb. 2014. URL: https://www.reddit.com/r/gaming/comments/1y70ej/valve_vac_and_trust/ (visited on 05/21/2025).

[13] Luca Galli et al. "A cheating detection framework for Unreal Tournament III: A machine learning approach". In: *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. Aug. 2011, pp. 266–272. DOI: 10.1109/CIG.2011.6032016.

[14] S.F. Yeung et al. "Detecting cheaters for multiplayer games: theory, design and implementation[1]". In: *CCNC 2006. 2006 3rd IEEE Consumer Communications and Networking Conference, 2006*. Vol. 2. 2006, pp. 1178–1182. DOI: 10.1109/CCNC.2006.1593224. URL: https://www.cse.cuhk.edu.hk/~cslui/PUBLICATION/detect_cheat.pdf.

[15] Anssi Kanervisto, Tomi Kinnunen, and Ville Hautamäki. "GAN-Aimbots: Using Machine Learning for Cheating in First Person Shooters". In: *IEEE Transactions on Games* 15.4 (Dec. 2023), pp. 566–579. ISSN: 2475-1510. DOI: 10.1109/tg.2022.3173450. URL: http://dx.doi.org/10.1109/TG.2022.3173450.

[16] Aditya Jonnalagadda et al. *Robust Vision-Based Cheat Detection in Competitive Gaming*. 2021. URL: https://arxiv.org/abs/2103.10031.

[17] Martin Willman. "Machine Learning to identify cheaters in online games". English. PhD thesis. Umeå University, May 2020. URL: https://www.diva-portal.org/smash/get/diva2:1431282/FULLTEXT01.pdf (visited on 02/12/2025).

[18] Ashish Vaswani et al. *Attention Is All You Need*. 2023. URL: https://arxiv.org/abs/1706.03762.

[19] Jianrong Tao et al. "XAI-Driven Explainable Multi-view Game Cheating Detection". In: *IEEE* (2020).

[20] Jianrong Tao et al. "Explainable AI for Cheating Detection and Churn Prediction in Online Games". English. In: *IEEE TRANSACTIONS ON GAMES* 15.2 (June 2023), pp. 242–251.

[21] Chenhao Niu et al. "Detecting LLM-Assisted Cheating on Open-Ended Writing Tasks on Language Proficiency Tests". In: Jan. 2024, pp. 940–953. DOI: 10.18653/v1/2024.emnlp-industry.70.

[22] deer. *pubg Dataset*. Published: https://universe.roboflow.com/deer-wtuhw/pubg-xmyvk. June 2023. URL: https://universe.roboflow.com/deer-wtuhw/pubg-xmyvk (visited on 05/25/2025).

[23] Roboflow Universe Projects. *Call of Duty MW2 Dataset*. Published: https://universe.roboflow.com/roboflow-universe-projects/call-of-duty-mw2. Feb. 2023. URL: https://universe.roboflow.com/roboflow-universe-projects/call-of-duty-mw2 (visited on 05/25/2025).

[24] emstatsl. *CSGO cheating dataset*. Mar. 2022. URL: https://www.kaggle.com/datasets/emstatsl/csgo-cheating-dataset/data (visited on 05/25/2025).

[25] Forzei. *Steam Community :: Guide :: How to spot CHEATERS*. Dec. 2024. URL: https://steamcommunity.com/sharedfiles/filedetails/?id=3377126116&searchtext=cheater (visited on 05/25/2025).

[26] John McDonald. *Using Deep Learning to Combat Cheating in CSGO*. English. 2018. URL: https://www.youtube.com/watch?v=0bhK8lUfIlc (visited on 05/25/2025).

[27] Counter-Strike Wiki. *Overwatch*. English. Mar. 2024. URL: https://counterstrike.fandom.com/wiki/Overwatch (visited on 12/02/2024).

[28] Mille Loo and Gert Luzkov. *Counter-Strike 2 Game data collection with cheat labeling*. English. Tech. rep. IT-University of Copenhagen, Dec. 2024. URL: https://github.com/Pinkvinus/CS2-demo-scraper/blob/main/Research_project___Counter_Strike_2_dataset_with_labels.pdf.

[29] Harry Dunham. "Cheat Detection using Machine Learning within Counter-Strike: Global Offensive". MA thesis. College of Wooster, 2020.

[30] Valve. *Counter-Strike: Global Offensive \¿ Overwatch FAQ*. English. URL: https://blog.counter-strike.net/index.php/overwatch/ (visited on 12/03/2024).

[31] SteamDB. *Counter-Strike 2 (App 730) Patches and Updates*. URL: https://steamdb.info/app/730/patchnotes/ (visited on 12/03/2024).

[32] SteamDB. *Counter-Strike 2 update for 16 February 2023*. English. Feb. 2023. URL: https://steamdb.info/patchnotes/10562092/ (visited on 12/03/2024).

[33] SteamDB. *Release Notes for 3/22/2023*. Mar. 2023. URL: https://steamdb.info/patchnotes/10832117/ (visited on 12/03/2024).

[34] SteamDB. *Steam Charts - Most Played Games on Steam*. URL: https://steamcharts.com/app/730%5C#48h (visited on 05/24/2025).

[35] Keplerč. *badge_guide*. Mar. 2021. URL: https://steamcommunity.com/sharedfiles/filedetails/?id=2412445225 (visited on 05/15/2025).

[36] g2g. *buy_account_g2g*. URL: https://www.g2g.com/categories/counter-strike-global-offensive-accounts (visited on 05/15/2025).

[37] u7buy. *buy_account_u7buy*. URL: https://www.u7buy.com/counter-strike-2/counter-strike-2-accounts (visited on 05/15/2025).

[38] Eldorado. *buy_account_eldorado*. URL: https://www.eldorado.gg/cs2-accounts/a/20-1-0 (visited on 05/15/2025).

[39] playeracutions. *buy_account_playerauctions*. URL: https://www.playerauctions.com/cs2-account/ (visited on 05/15/2025).

[40] SteamDB. *Counter-Strike 2 Update-bug*. May 2025. URL: https://steamdb.info/patchnotes/18394927/ (visited on 05/11/2025).

[41] SteamDB. *Counter-Strike 2 Update-bugfix*. May 2025. URL: https://steamdb.info/patchnotes/18396610/ (visited on 05/11/2025).

[42] Valve Corporation. *CounterStrike-updates*. English. May 2025. URL: https://www.counter-strike.net/news/updates (visited on 05/11/2025).

[43] HLTV. *Counter-Strike News & Coverage*. URL: https://www.hltv.org/.

[44] Mille Mei Zhen Loo and Gert Lužkov. *CS2CD.Counter-Strike_2_Cheat_Detection (Revision c70448b)*. 2025. DOI: 10.57967/hf/5315. URL: https://huggingface.co/datasets/CS2CD/CS2CD.Counter-Strike_2_Cheat_Detection.

[45] ESL Gaming Online. *csstats*. URL: https://csstats.gg/ (visited on 05/20/2025).

[46] AkiVer. *CS Demo Manager*. English. Dec. 2023. URL: https://cs-demo-manager.com/ (visited on 05/20/2025).

[47] FaceIT. *EscapeR- - FACEIT.com*. English. URL: https://www.faceit.com/en/players/EscapeR- (visited on 05/21/2025).

[48] LaihoE et al. *DemoParser2*. Sept. 2024. URL: https://github.com/LaihoE/demoparser (visited on 11/01/2024).

[49] Mille Mei Zhen Loo and Gert Lužkov. *Github : CS2_cheat_detection*. May 2025. URL: https://github.com/Pinkvinus/CS2_cheat_detection (visited on 05/30/2025).

[50] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*. May 2016. URL: https://op.europa.eu/en/publication-detail/-/publication/3e485e15-11bd-11e6-ba9a-01aa75ed71a1/language-en (visited on 05/26/2025).

[51] Pandas. *read_parquet*. URL: https://pandas.pydata.org/docs/reference/api/pandas.read_parquet.html (visited on 05/31/2025).

[52] Valve. *CS2 - Trust Factor Matchmaking*. URL: https://help.steampowered.com/en/faqs/view/00EF-D679-C76A-C185 (visited on 05/23/2025).

[53] Yueyang Su et al. "Few-shot Learning for Trajectory-based Mobile Game Cheating Detection". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '22. event-place: Washington DC, USA. New York, NY, USA: Association for Computing Machinery, 2022, pp. 3941–3949. ISBN: 9781450393850. DOI: 10.1145/3534678.3539157. URL: https://doi.org/10.1145/3534678.3539157.

[54] Umar Jamil. *Coding a Transformer from scratch on PyTorch, with full explanation, training and inference*. May 2023. URL: https://www.youtube.com/watch?v=ISNdQcPhsts (visited on 05/25/2025).

[55] PyTorch. *PyTorch BCEWithLogitsLoss*. URL: https://docs.pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html (visited on 05/22/2025).

[56] PyTorch. *AdamW*. URL: https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html (visited on 05/26/2025).

[57] PyTorch. *StepLR*. URL: https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.StepLR.html (visited on 05/26/2025).

Appendix A

Survey Questions

This appendix contains the full list of questions answered in the survey along with a UML Activity Diagram displaying the flow between sections in the survey. As seen on the figure A.1 the survey was split into several section that would allow for a tailored questions for the respondents. Each section contained questions related to the topic.

A.1.1.1 Experience with CS2

Description : In this section we will talk a little bit about your experience with the game Counter-Strike 2.

Question	Subtitle	Answer Type	Answer Criteria
For how many years have you played Counter-Strike?	If you have less than a yeas worth of experience enter a decimal.	Short answer	Number >= 0
How many hours have you played the game?	In order to check how many hours you have played a game, check your steam library.	Short answer	Number >= 0
What is your premier elo?		Short answer	Number >= 0
Have you ever deployed external software in order to enhance your gaming abilities?		Multiple choice	

Table A.1: Table showing the question, subtitle, answer type, and answer criteria of the questions in the survey section Experience with CS2.

Options for the question "Have you ever deployed external software to enhance your gaming abilities" were "Yes" and "No".

A.1.1.2 Using external software

Description : In this section we will ask a bit about the use of external software

Question	Answer Type	Answer Criteria
What was the name of this software	Short answer	
How much did the software cost in EUR?	Short answer	Number >= 0
What did the software do?	Checkboxes	
How was your experience using this software	Paragraph/long answer	
Would you categorise the use of the software as cheating?	Multiple Choice	
Please give further reasoning for the answer to the previous question	Paragraph/Long answer	
Do you have any comments you want to share regarding the use of skill enhancing software?	Paragraph/Long answer	

Table A.2: Table showing the question, subtitle, answer type, and answer criteria of the questions in the survey section Using external software.

Options for the question "What did the software do" were:

1. Aim hack
2. Trigger hack
3. Wall hack
4. Grenade projectory preview
5. Spinbot
6. Other

Options for the question "Would you categorise the use of softare as cheating?": "Yes", "No", and "Maybe".

A.1.1.3 Cheaters in CS2

Description : In this section we will ask about your experience with cheaters

Question	Answer Type
Have you experienced cheaters in Counter-Strike 2?	Multiple choice
Do you believe that there is a problem with cheaters in CS2?	Multiple choice
Have you ever watched a demo because you were suspicious of a player potentially cheating?	Multiple choice
How often do you experience playing in a match with a cheater?	Multiple choice

Table A.3: Table showing the question, subtitle, answer type, and answer criteria of the questions in the survey section Cheaters in CS2.

The options for the multiple choice questions "Have you ever experiences cheaters in Counter-Strike 2?" and "Do you believe there is a problem with cheaters in CS2?" were "Yes" and "No". The options for the question "Have you ever watched a demo because you were suspicious of a player potentially cheating?" were "Yes", "No", "Can't remember". The options for the question "How often do you experience playing in a match with a cheater?" were:

1. Never (0% of the time)
2. Occasionally
3. Sometimes (50 % of the time)
4. Often
5. Always (100 % of the time)

A.1.1.4 Cheaters as a problem

Question	Answer Type
On a scale from 0 to 5, how big of a problem do you believe cheating is in Counter-Strike 2	Linear scale
Why do you believe that cheating is a problem?	Paragraph/Long answer

Table A.4: Table showing the question, subtitle, answer type, and answer criteria of the questions in the survey section Cheaters as a problem.

The linear scale used in the question "On a scale from 0 to 5, how big of a problem do you believe cheating is in Counter-Strike 2" goes from 0 (Not a problem at all) to 5 (Big problem).

A.1.1.5 Cheaters are not a problem

Question	Answer Type
Why do you believe that cheaters are not a problem?	Paragraph/Long answer

Table A.5: Table showing the question, subtitle, answer type, and answer criteria of the questions in the survey section Cheaters are not a problem.

A.1.1.6 Overwatch

Description : In this section we will ask about your experiences with overwatch

Question	Answer Type
Did you have access and make use of the overwatch feature when the feature was active	Multiple choice

Table A.6: Table showing the question, subtitle, answer type, and answer criteria of the questions in the survey section Overwatch.

The option for the question mentioned in table A.6 were:

1. Yes, I have reviewed cases and/or watched my own demos
2. Yes, but i never reviewed a case nor watched my own demos
3. I can't remember
4. No, I did not have access to overwatch, but i have reviewed my own demos
5. No, I did not have access to overwatch, and i have never reviewed a case

A.1.1.7 Overwatch users

Question	Answer Type
Do you think overwatch improved Counter-Strike	Multiple choice
Did you feel like you aided the community when reviewing overwatch cases	Multiple choice
Is there something else you want to remark regarding overwatch	Paragraph/Long answer

Table A.7: Table showing the question, subtitle, answer type, and answer criteria of the questions in the survey section Overwatch users.

The answer options for the multiple choice questions were "Yes", "No", and "Maybe".

A.1.1.8 Reviewing demos

Description : In this section we will ask about your tendencies when reviewing demos.

Question	Answer Type
What do you look for when determining whether a player is cheating or not?	Checkboxes
Can you in more detail describe how you determine whether a player is cheating or not.	Paragraph/Long answer

Table A.8: Table showing the question, subtitle, answer type, and answer criteria of the questions in the survey section Reviewing demos.

The pre-written answer options for the question "What do you look for when determining whether a player is cheating or not?" where:

1. Predictive playstyle
2. Unnatural crossair movement
3. Unnatural movement (bunny hopping, spinbotting, etc.)
4. Skill gaps in different areas of play (aim, movement, awareness, timings, etc.)
5. Other

A.1.1.9 Thank you so much for participating

Question	Answer Type
Do you have any comments for us?	Paragraph/Long answer

Table A.9: Table showing the question, subtitle, answer type, and answer criteria of the questions in the survey section Thank you so much for participating.

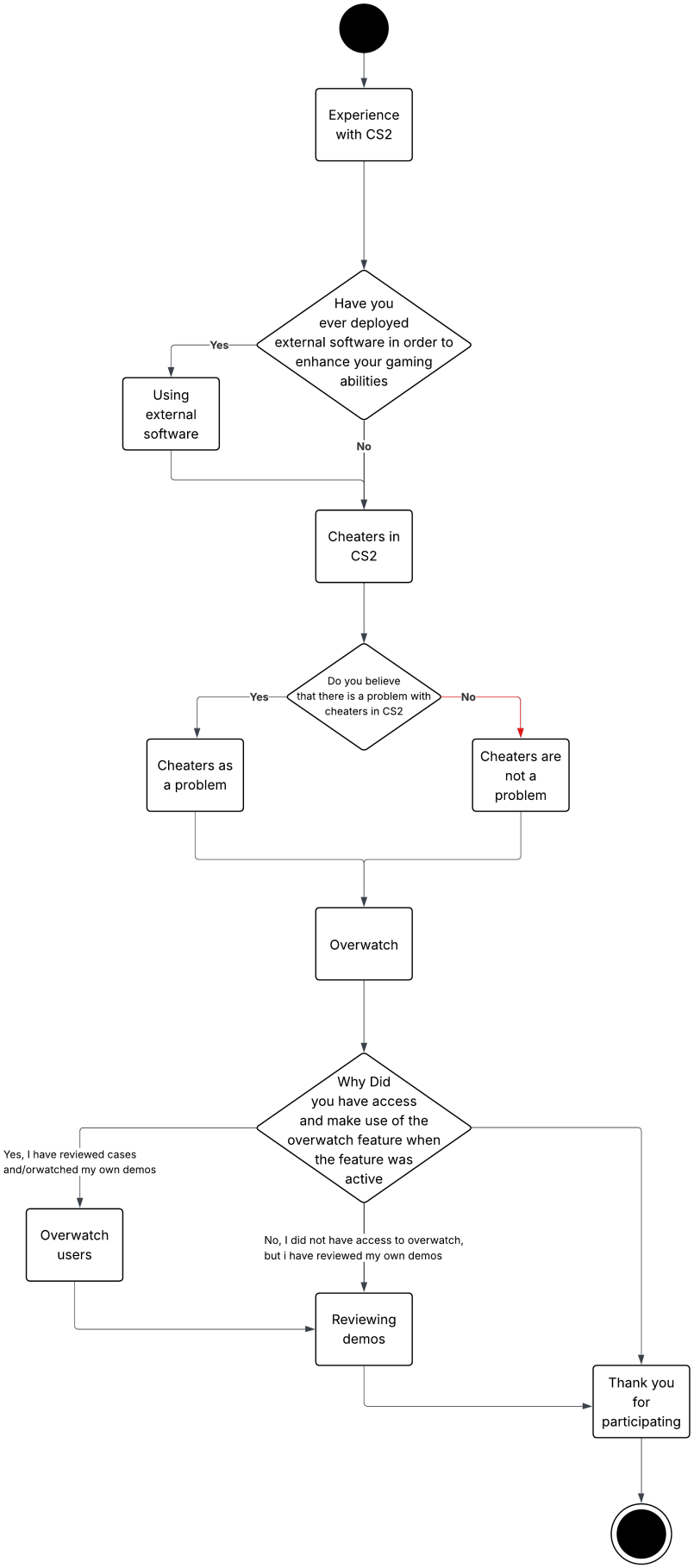

Figure A.1: A UML Activity diagram displaying the flow between sections in the survey.

Appendix B

Parquet file columns

1. inventory
2. usercmd_input_history
3. inventory_as_ids
4. approximate_spotted_by
5. aim_punch_angle_vel
6. aim_punch_angle
7. active_weapon_ammo
8. item_def_idx
9. weapon_quality
10. entity_lvl
11. item_id_high
12. item_id_low
13. inventory_position
14. is_initialized
15. econ_item_attribute_def_idx
16. initial_value
17. refundable_currency
18. set_bonus
19. fire_seq_start_time
20. fire_seq_start_time_change
21. m_iState
22. weapon_mode
23. accuracy_penalty
24. i_recoil_idx
25. fl_recoil_idx
26. is_burst_mode
27. is_in_reload
28. reload_visually_complete
29. dropped_at_time
30. is_hauled_back
31. is_silencer_on
32. time_silencer_switch_complete
33. orig_team_number
34. prev_owner
35. last_shot_time
36. iron_sight_mode
37. num_empty_attacks
38. zoom_lvl
39. burst_shots_remaining
40. needs_bolt_action
41. next_primary_attack_tick
42. next_primary_attack_tick_ratio
43. next_secondary_attack_tick
44. next_secondary_attack_tick_ratio
45. total_ammo_left
46. is_freeze_period
47. is_warmup_period
48. warmup_period_end
49. warmup_period_start
50. is_terrorist_timeout
51. is_ct_timeout
52. terrorist_timeout_remaining
53. ct_timeout_remaining
54. num_terrorist_timeouts
55. num_ct_timeouts
56. is_technical_timeout

57. is_waiting_for_resume
58. match_start_time
59. round_start_time
60. restart_round_time
61. game_start_time
62. time_until_next_phase_start
63. game_phase
64. total_rounds_played
65. rounds_played_this_phase
66. hostages_remaining
67. any_hostages_reached
68. has_bombites
69. has_rescue_zone
70. has_buy_zone
71. is_matchmaking
72. match_making_mode
73. is_valve_dedicated_server
74. spectator_slot_count
75. is_match_started
76. n_best_of_maps
77. is_bomb_dropped
78. is_bomb_planted
79. round_win_status
80. round_win_reason
81. terrorist_cant_buy
82. ct_cant_buy
83. ct_losing_streak
84. t_losing_streak
85. round_in_progress
86. is_connected
87. fov
88. balance
89. start_balance
90. total_cash_spent
91. cash_spent_this_round
92. kills_total
93. deaths_total
94. assists_total
95. alive_time_total
96. headshot_kills_total
97. damage_total
98. objective_total
99. utility_damage_total
100. enemies_flashed_total
101. ace_rounds_total
102. 4k_rounds_total
103. 3k_rounds_total
104. ping
105. is_auto_muted
106. pending_team_num
107. player_color
108. ever_played_on_team
109. is_coach_team
110. rank
111. comp_wins
112. comp_rank_type
113. rank_if_win
114. rank_if_loss
115. rank_if_tie
116. is_controlling_bot
117. has_controlled_bot_this_round
118. spawn_time
119. death_time
120. score
121. mvps
122. health
123. life_state
124. move_collide
125. move_type
126. team_num
127. active_weapon
128. looking_at_weapon
129. holding_look_at_weapon
130. next_attack_time
131. has_defuser
132. has_helmet

133. duck_time_ms
134. max_speed
135. max_fall_velo
136. duck_amount
137. duck_speed
138. duck_overrdie
139. old_jump_pressed
140. jump_until
141. jump_velo
142. buttons
143. fall_velo
144. in_crouch
145. crouch_state
146. ducked
147. ducking
148. in_duck_jump
149. jump_time_ms
150. last_duck_time
151. player_state
152. molotov_damage_time
153. moved_since_spawn
154. flash_duration
155. flash_max_alpha
156. is_rescuing
157. last_place_name
158. in_buy_zone
159. in_hostage_rescue_zone
160. in_bomb_zone
161. time_last_injury
162. is_walking
163. spotted
164. is_scoped
165. resume_zoom
166. is_defusing
167. is_grabbing_hostage
168. blocking_use_in_progess
169. in_no_defuse_area
170. which_bomb_zone
171. shots_fired
172. velo_modifier
173. wait_for_no_attack
174. armor_value
175. current_equip_value
176. round_start_equip_value
177. team_rounds_total
178. team_name
179. team_surrendered
180. team_match_stat
181. team_num_map_victories
182. team_score_first_half
183. team_score_second_half
184. team_score_overtime
185. team_clan_name
186. RELOAD
187. USE
188. FIRE
189. SCOREBOARD
190. BACK
191. INSPECT
192. RIGHT
193. ZOOM
194. FORWARD
195. LEFT
196. RIGHTCLICK
197. pitch
198. usercmd_viewangle_y
199. X
200. yaw
201. usercmd_mouse_dy
202. active_weapon_original_owner
203. velocity
204. Y
205. usercmd_buttonstate_1
206. entity_id
207. velocity_X
208. is_airborne

209. usercmd_buttonstate_3
210. usercmd_impulse
211. Z
212. usercmd_buttonstate_2
213. usercmd_viewangle_z
214. velocity_Y
215. usercmd_viewangle_x
216. usercmd_left_move
217. usercmd_forward_move
218. usercmd_consumed_server_angle_changes
219. is_alive
220. velocity_Z
221. active_weapon_name
222. usercmd_mouse_dx
223. game_time
224. tick
225. steamid

Appendix C

JSON file data

1. round_prestart
2. begin_new_match
3. decoy_started
4. player_team
5. item_pickup
6. cs_round_start_beep
7. decoy_detonate
8. bullet_damage
9. item_equip
10. round_announce_final
11. other_death
12. cs_win_panel_match
13. bomb_begindefuse
14. smokegrenade_detonate
15. rank_update
16. round_poststart
17. round_announce_match_start
18. round_announce_last_round_half
19. round_officially_ended
20. player_spawn
21. inferno_expire
22. flashbang_detonate
23. weapon_fire_on_empty
24. bomb_pickup
25. announce_phase_end
26. bomb_beginplant
27. player_disconnect
28. round_time_warning
29. hltv_versioninfo
30. weapon_fire
31. hostage_rescued
32. bomb_planted
33. buytime_ended
34. cs_pre_restart
35. weapon_zoom
36. round_announce_match_point
37. weapon_reload
38. hostage_hurt
39. inferno_startburn
40. bomb_exploded
41. player_blind
42. hltv_fixed
43. player_jump
44. player_connect_full
45. server_cvar
46. player_footstep
47. player_connect
48. player_death
49. hegrenade_detonate
50. smokegrenade_expired
51. hostage_rescued_all
52. round_freeze_end
53. bomb_defused
54. cs_round_final_beep
55. hltv_chase
56. bomb_dropped
57. player_hurt
58. cheaters
59. CSstats_info

Appendix D

Training metrics

This appendix will show the metric dashboards used when training the models. The accuracy, recall, precision and ROC AUC are calculated on the validation set. As these dashboards are rather large, they will begin on the next page.

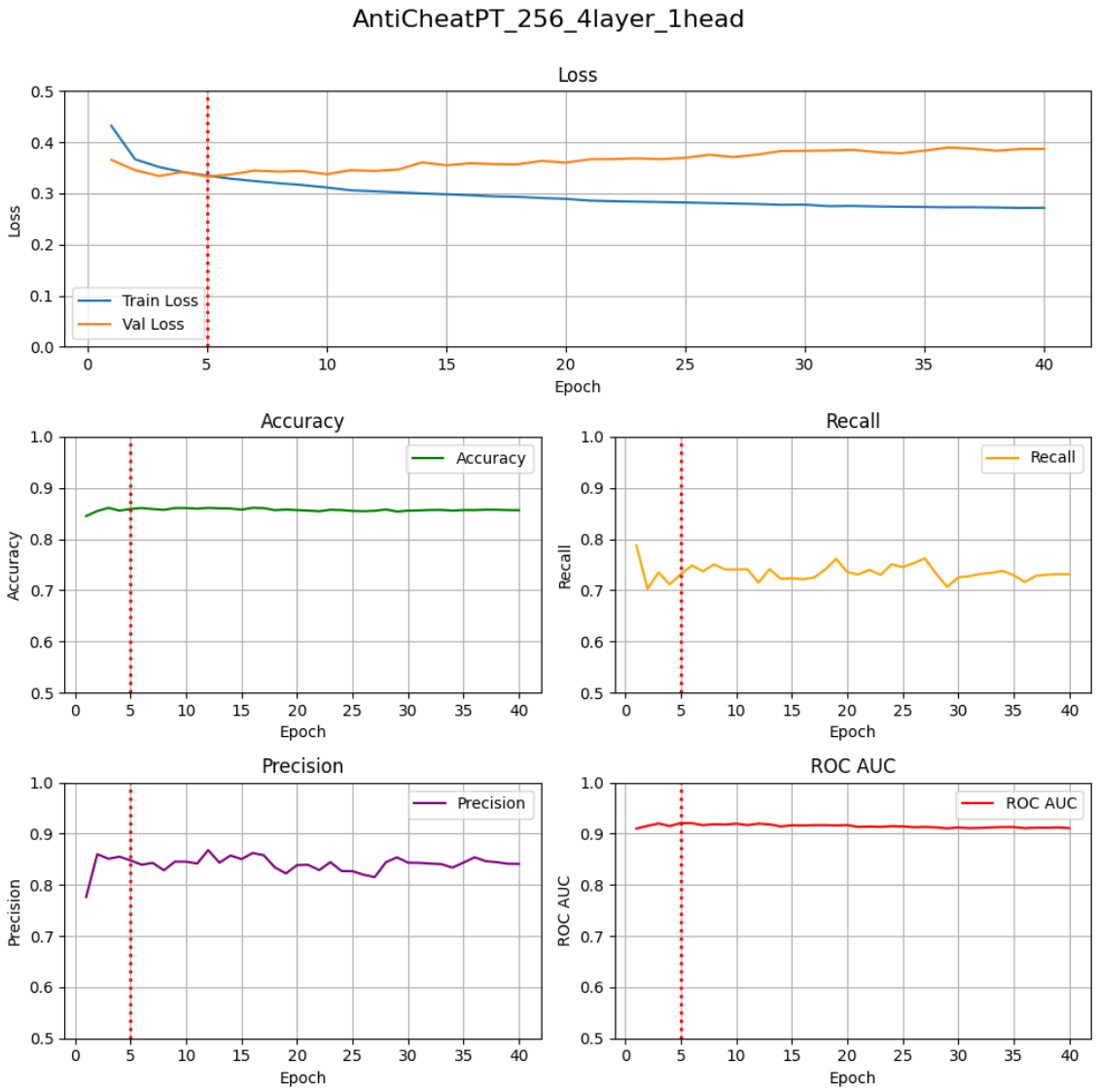

Figure D.1: Training of AntiCheatPT_256

Component	Value
Context window size	256
Transformer layers	4
Attention heads	1
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.1: Training components for AntiCheatPT_256

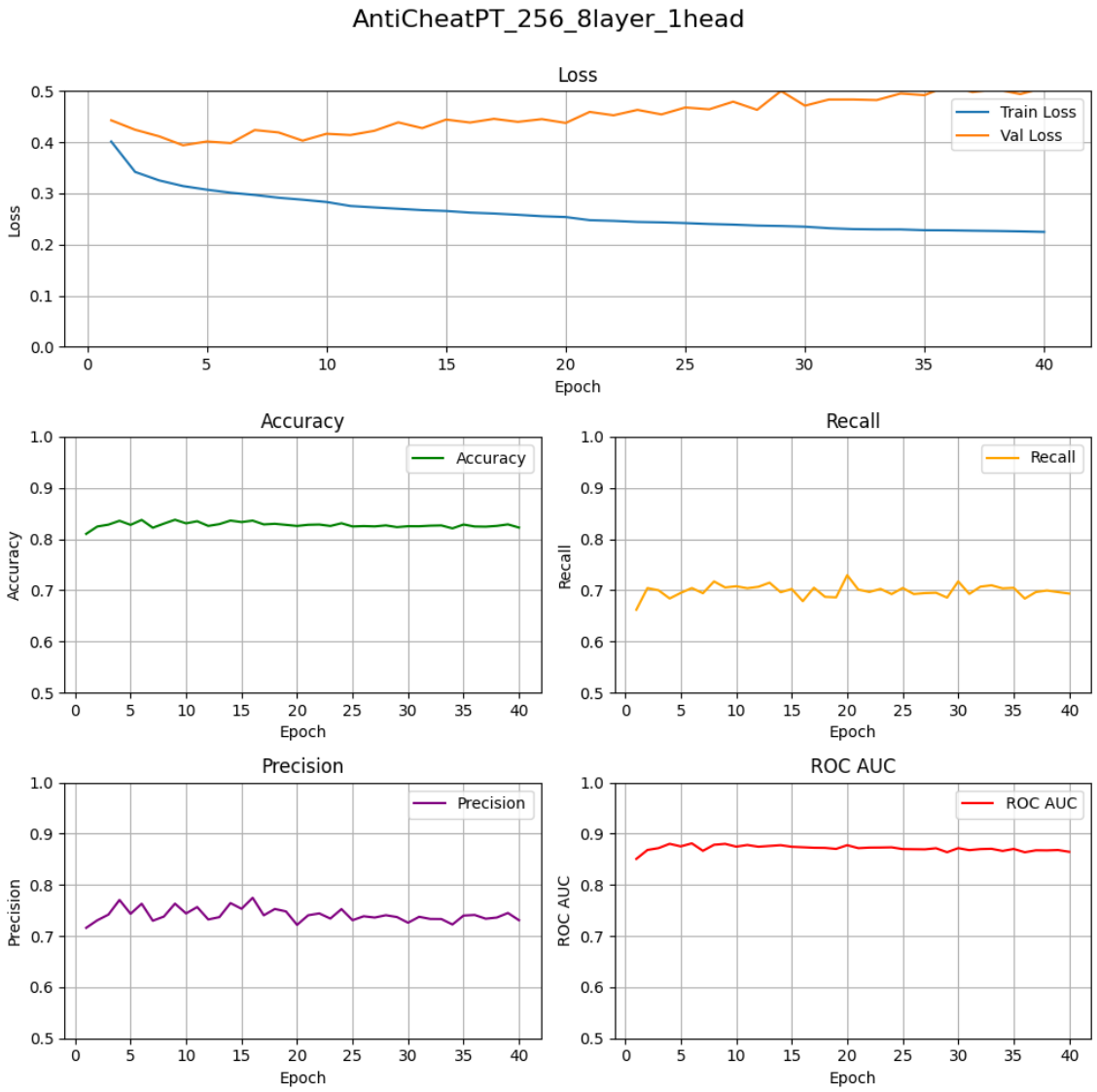

Figure D.2: Training of AntiCheatPT_256

Component	Value
Context window size	256
Transformer layers	8
Attention heads	1
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.2: Training components for AntiCheatPT_256

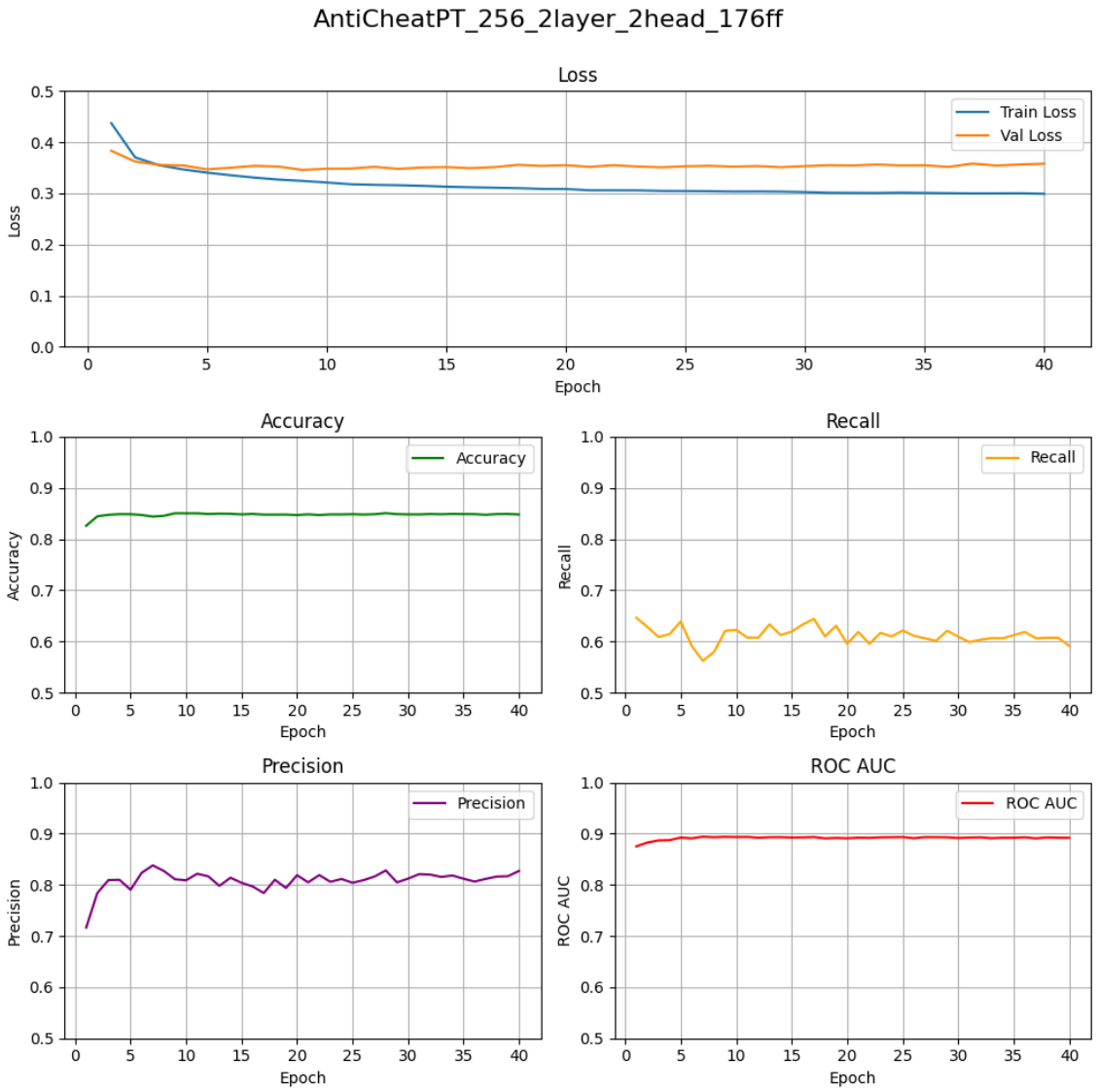

Figure D.3: Training of AntiCheatPT_256

Component	Value
Context window size	256
Transformer layers	2
Attention heads	2
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.3: Training components for AntiCheatPT_256

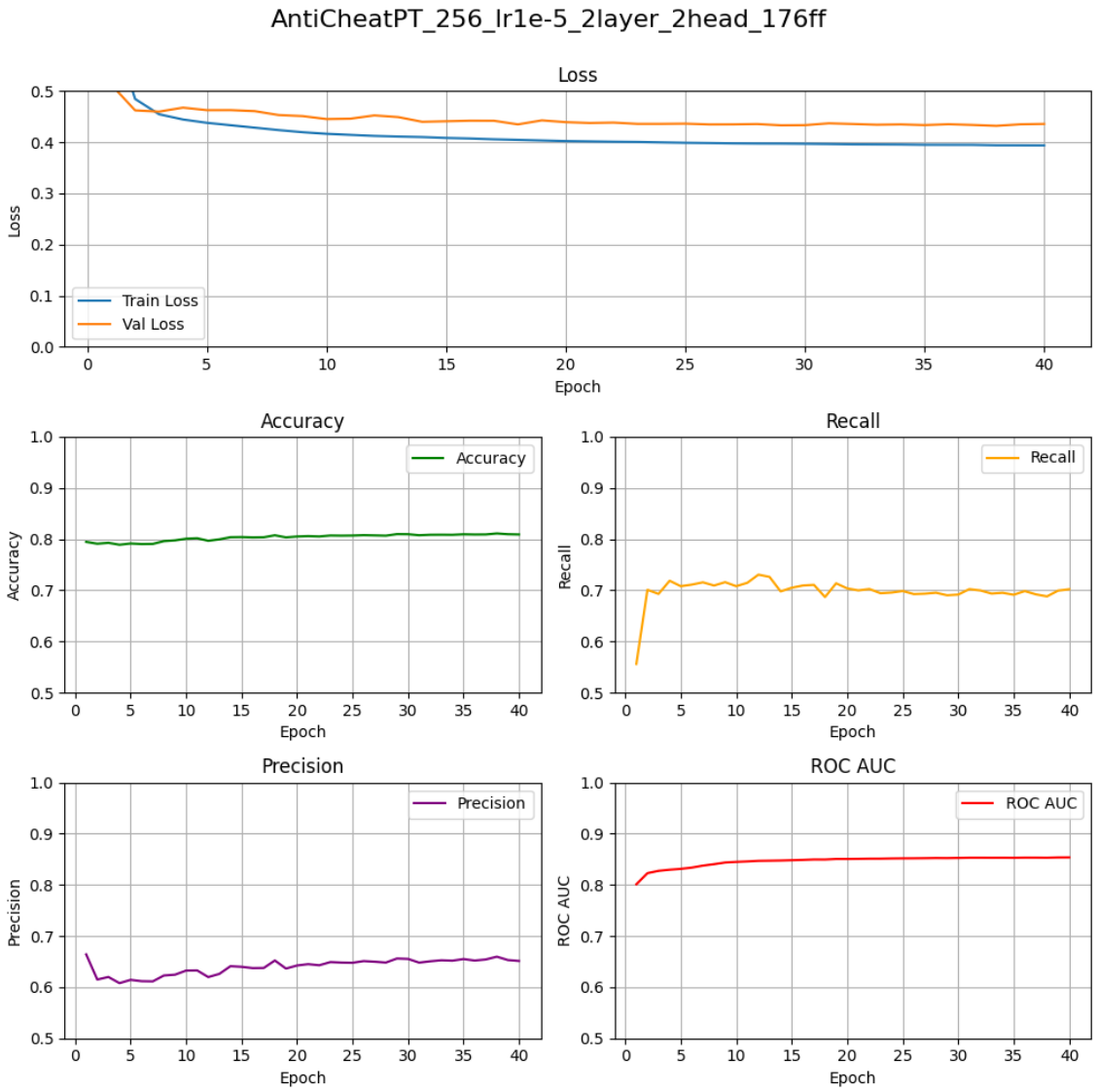

Figure D.4: Training of AntiCheatPT_256

Component	Value
Context window size	256
Transformer layers	2
Attention heads	2
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-5})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.4: Training components for AntiCheatPT_256

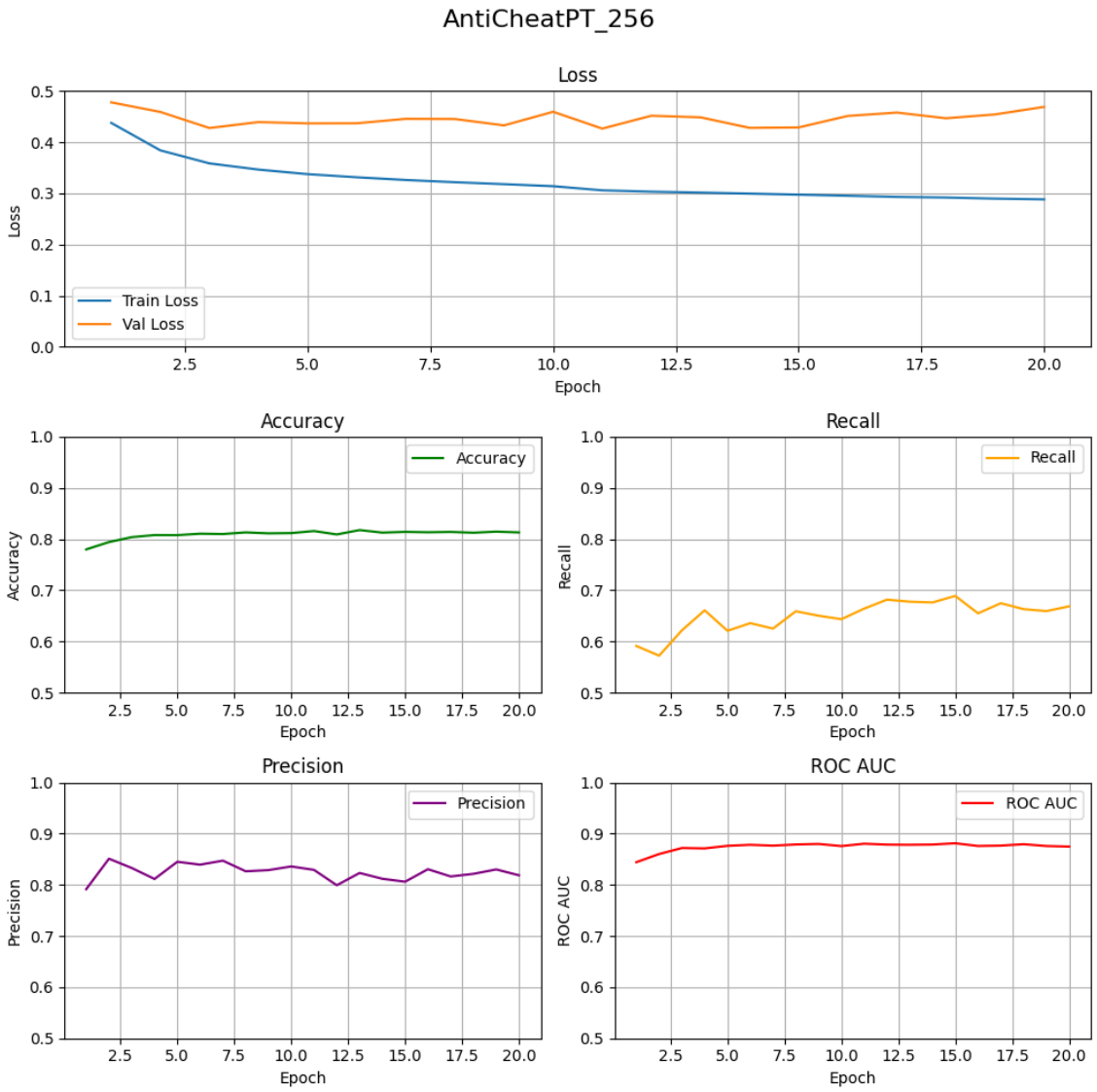

Figure D.5: Training of AntiCheatPT_256

Component	Value
Context window size	256
Transformer layers	6
Attention heads	4
Transformer feedforward dimension	512
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.5: Training components for AntiCheatPT_256

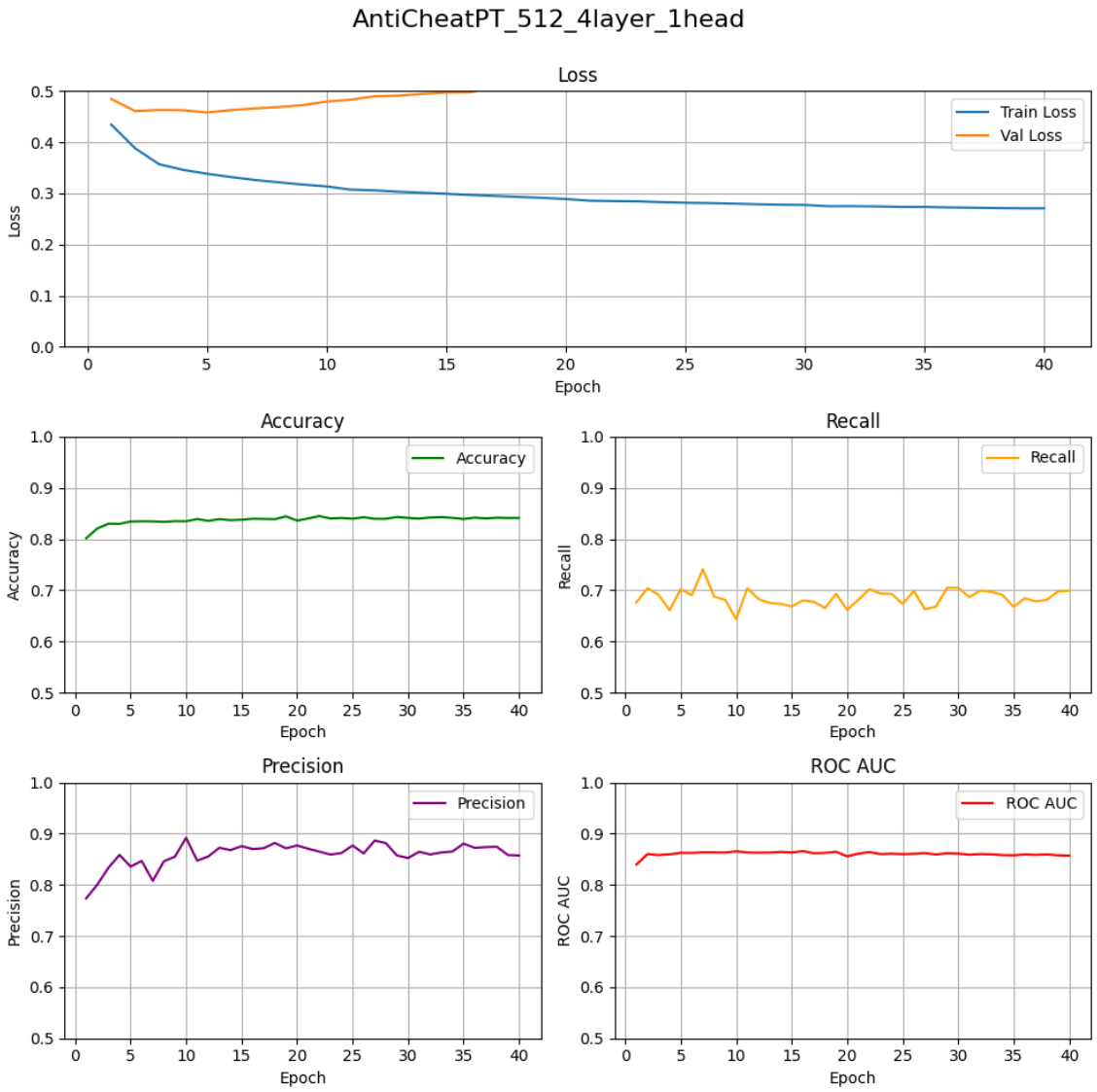

Figure D.6: Training of AntiCheatPT_512

Component	Value
Context window size	512
Transformer layers	4
Attention heads	1
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.6: Training components for AntiCheatPT_512

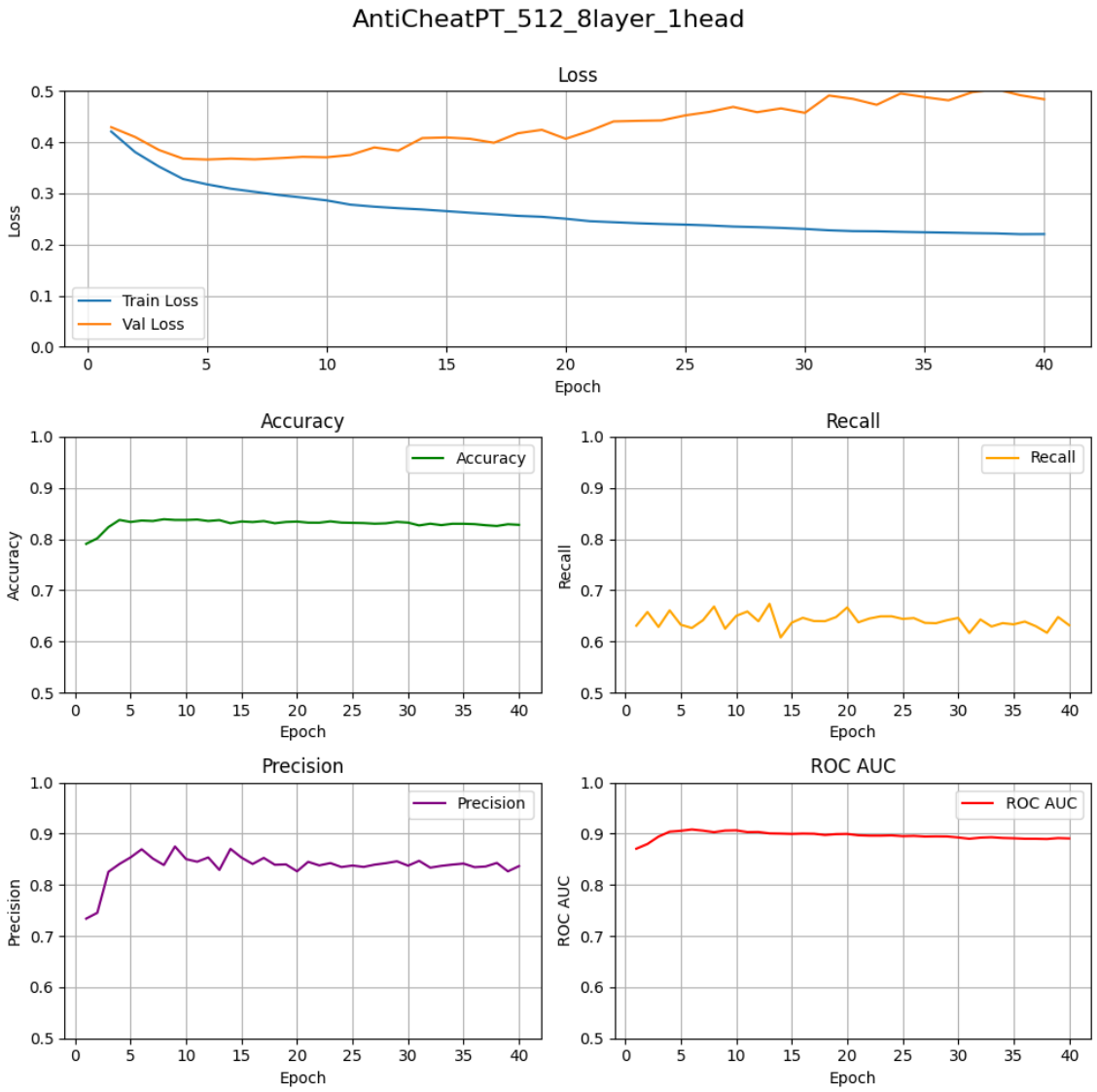

Figure D.7: Training of AntiCheatPT_512

Component	Value
Context window size	512
Transformer layers	8
Attention heads	1
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.7: Training components for AntiCheatPT_512

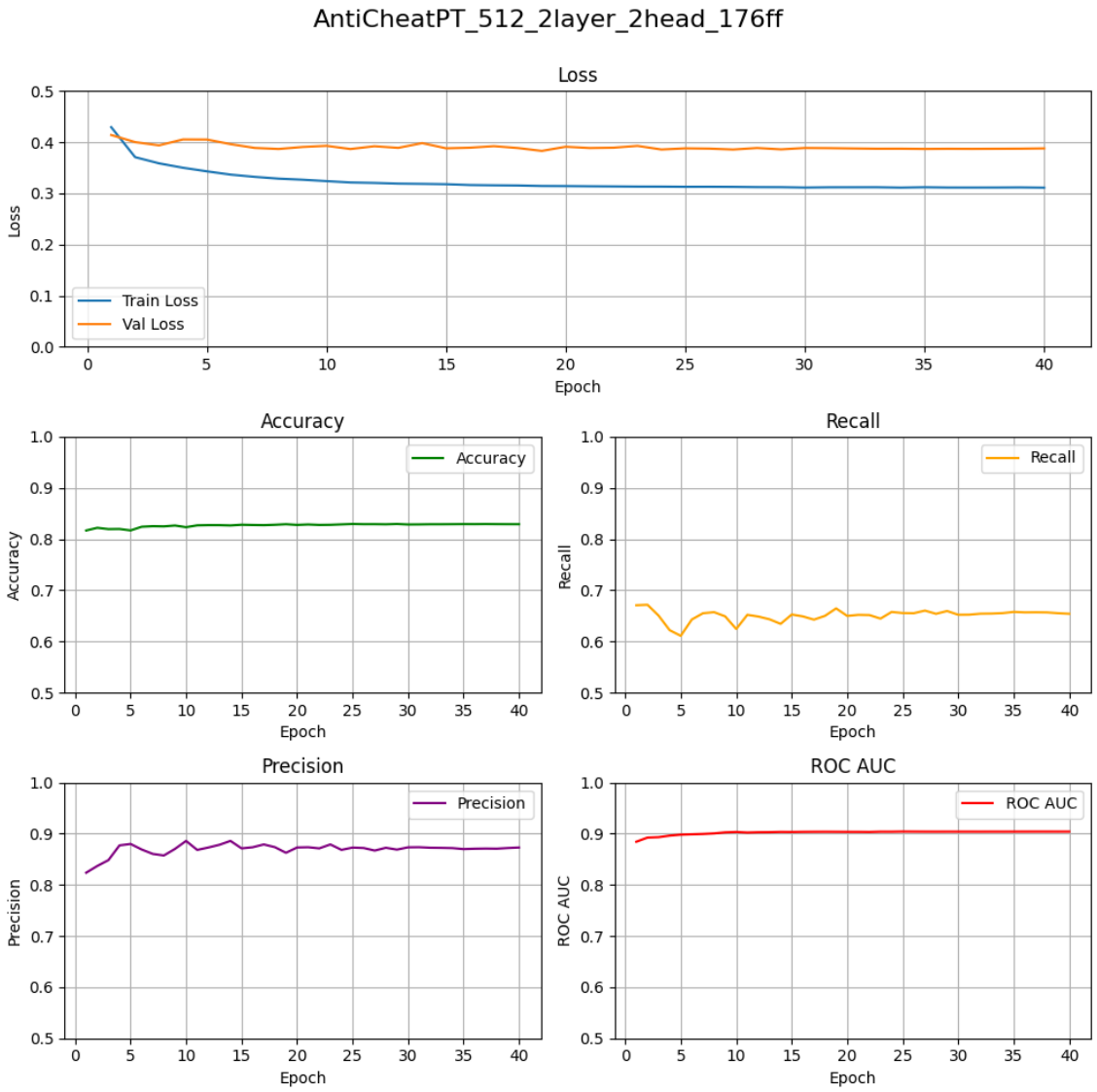

Figure D.8: Training of AntiCheatPT_512

Component	Value
Context window size	512
Transformer layers	2
Attention heads	2
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.8: Training components for AntiCheatPT_512

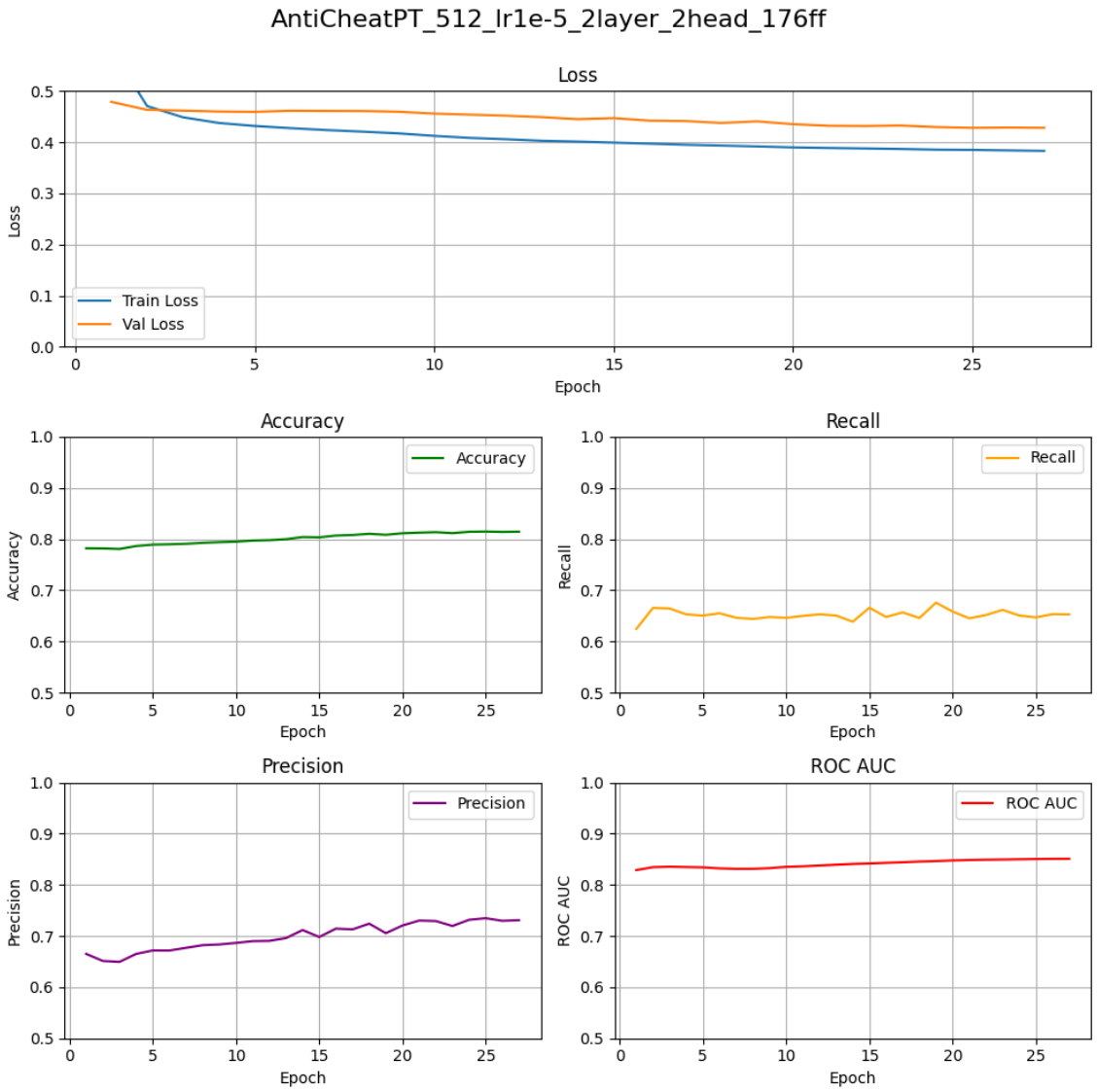

Figure D.9: Training of AntiCheatPT_512

Component	Value
Context window size	512
Transformer layers	2
Attention heads	2
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-5})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.9: Training components for AntiCheatPT_512

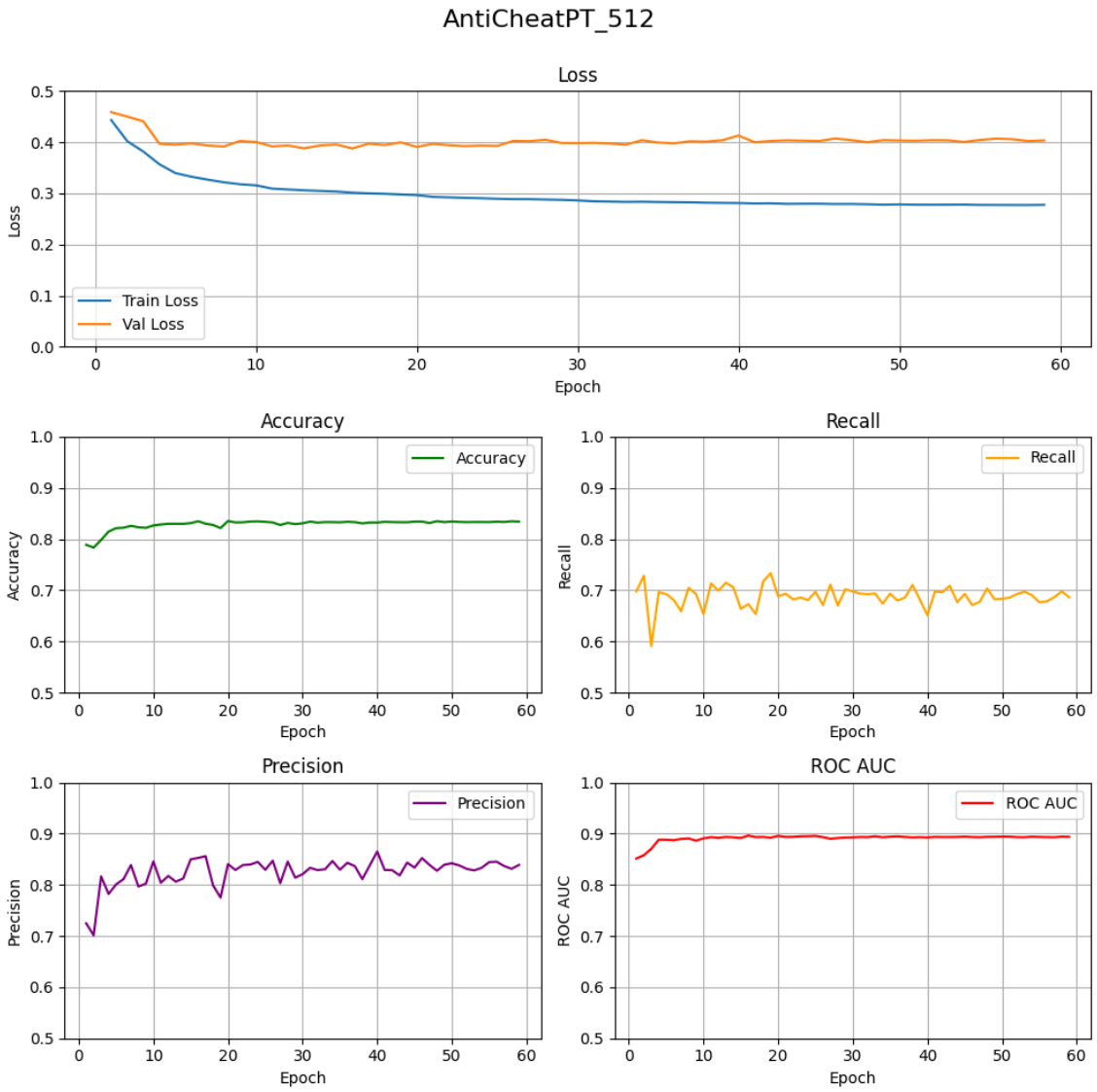

Figure D.10: Training of AntiCheatPT_512

Component	Value
Context window size	512
Transformer layers	6
Attention heads	4
Transformer feedforward dimension	512
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.10: Training components for AntiCheatPT_512

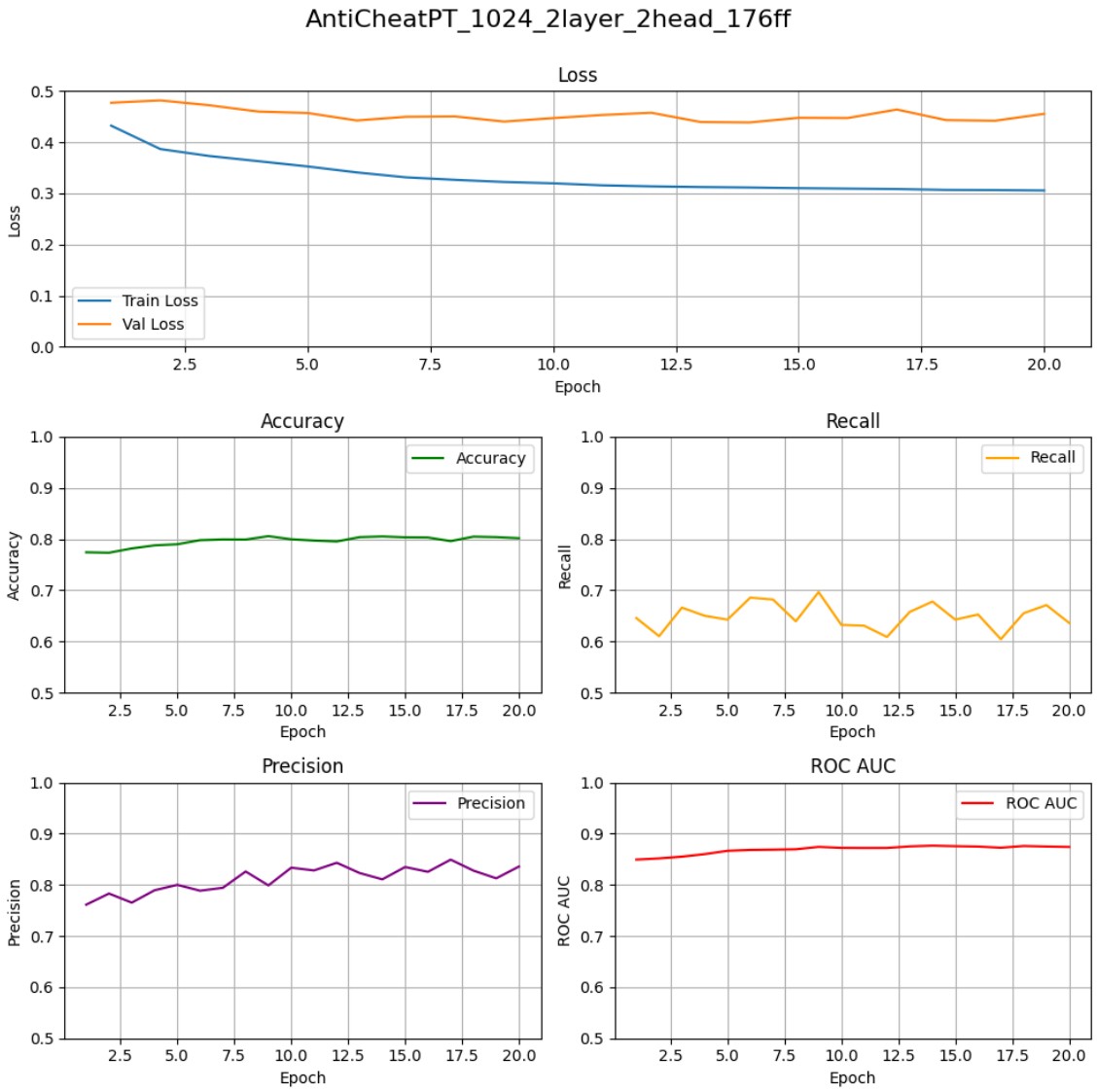

Figure D.11: Training of AntiCheatPT_1024

Component	Value
Context window size	1024
Transformer layers	2
Attention heads	2
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-4})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.11: Training components for AntiCheatPT_1024

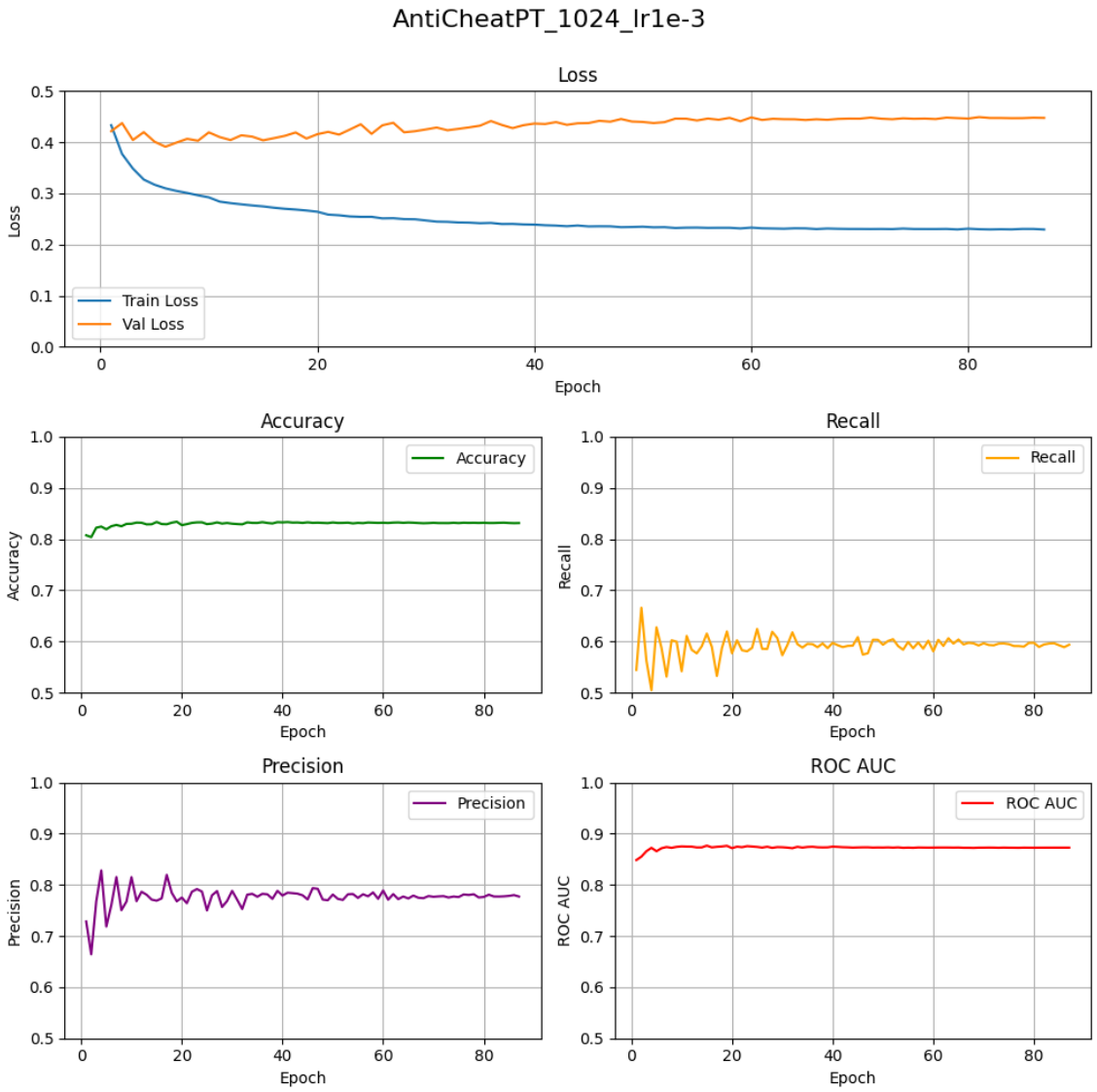

Figure D.12: Training of AntiCheatPT_1024

Component	Value
Context window size	1024
Transformer layers	6
Attention heads	4
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-3})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.12: Training components for AntiCheatPT_1024

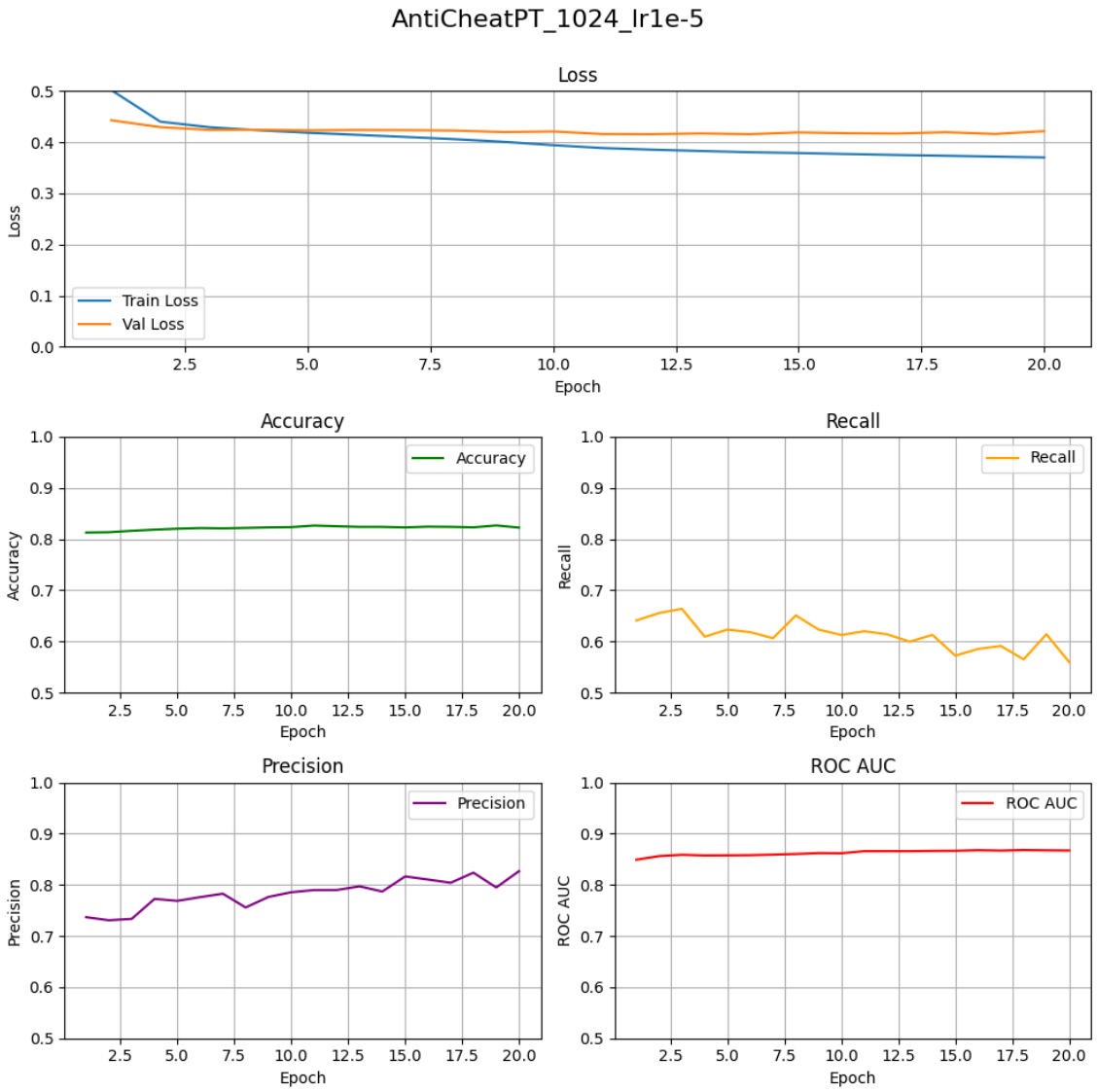

Figure D.13: Training of AntiCheatPT_1024

Component	Value
Context window size	1024
Transformer layers	6
Attention heads	4
Transformer feedforward dimension	176
Loss function	Binary Cross Entropy (`BCEWithLogitLoss`)
Optimiser	AdamW (learning rate = 10^{-5})
Scheduler	StepLR (gamma = 0.5, step_size = 10)
Batch size	128

Table D.13: Training components for AntiCheatPT_1024