Detect Cheater in Online Gaming using AI

MSc Research Project
MSc. In Cyber Security

Sparsh Bajaj
Student ID: X0228392

School of Computing
National College of Ireland

Supervisor: Dr. Ross Spelman

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sparsh Bajaj		
Student ID:	X0228392		
Programme:	MSc. In Cyber Security	**Year:**	2022
Module:	Research Internship		
Supervisor:	Ross Spelman		
Submission Due Date:	15/10/2022		
Project Title:	Detect Cheater in Online Gaming using AI		
Word Count:	4500	**Page Count:**	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Sparsh Bajaj

Date: 14/12/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Detect Cheater in Online Gaming using AI

Sparsh Bajaj
X2028392

Abstract

With the increase in popularity, gaming has become one of the major industries, and cheating in video games is also increasingly common nowadays as everyone wants the top spot. Reduced game traffic is usually caused due to the drastic increase in cheaters, and, at the same time, cheats sold online, resulting in the lower income generated for the studios.

Because of the cheaters in the gaming industry, game developers have developed and deployed numerous anti-cheating strategies. Installing an application with kernel-level access to the client's system is mandatory for a few anti-cheat strategies, and most gamers are concerned about their security and privacy in this aspect. A few issues with the OS caused by the anti-cheat software have been documented; the registered users have claimed that the application reportedly leads to upgrades failing.

With machine learning methods and statistical analysis, this study seeks to identify cheaters from players while protecting one's privacy and personal information.

1 Introduction

1.1 Motivation and Goal to be achieved

Selling cheats for gaming is a multi-million-dollar industry. The sellers charge more than $500 for an elite cheat, according to Santiago Pontiroli, a security researcher at Kaspersky Lab. Santiago Pontiroli began looking into malware-like cheats in video games after becoming tired of hackers in online matches. The "as-a-service" model for cheats was a prevalent business model for selling malware on dark web underground forums. There are certain subscription services that keep the money for the malware authors while giving customers updates and new services. In an interview, a man who was involved in using cheats told the world that sellers don’t care about the service; in particular, they exaggerate about the cheats used on hacked accounts to promote the use of cheats.

Cheats and malware share the same core and are very similar in many aspects. Because, in essence, it’s an untrusted code that tries to hide under the radar and exploit information just like malware.

Cheat developers must be skilled in coding since they protect and hide unwanted code from anti-cheats. Anti-cheats are not only used for gaming but are also used in other areas, such as online exams. For example, during exams, anti-cheats can limit the tasks which are performed during the test. If it detects anything suspicious, it reports it to the authority and terminates the test.

Most contemporary anti-cheating software may violate players' privacy and are usually ineffective. Considering this, this research aims to investigate machine learning methods for identifying cheaters and hackers using non--invasive data sandboxed methods.

The objective is to identify cheaters using machine learning models and game data. Our AI model was trained with 400 images of the dataset. This model was trained using YOLOv55 and can be used to detect enemies and players in real time. The results from this model would help add a layer of security checks that helps anti-cheat software’s traditional approach.

1.2 Cheating in online gaming and its impact

Due to the increase in gaming influencers, the online gaming world now has more players than ever. And since the number of gamers has bloomed, cheating in these games occurs more frequently than before. According to a global survey conducted by Irdeto in 2018, approximately 88% of online gamers have experienced foul play, or their gaming experience has been negatively impacted due to cheaters at least once in their lifetime.

The results from this survey, which was conducted in six different countries, also show that 77% of gamers in China would stop playing a game once they realize that the other players involved are cheating, and 48% of these gamers are seen to buy less content of the game which ultimately shrinks the revenue generation of the gaming companies. 68% of people surveyed in South Korea said it’s very frequent to face a cheater in today's online gaming era. (*Widespread Cheating in Multiplayer Online Games Drives Gamers in Asia Pacific Away - Irdeto*, n.d.). As a result of widespread cheating, the South Korean government took legislative action to make cheating illegal and punishable with fines and prison time (*South Korea Cracks down on Cheaters with Law Targeting Illicit Game Mo*, n.d.). In addition, there have been at least a hundred arrests of cheaters in China in 2018. These arrests resulted from cooperation between the game company Tencent and the Chinese police.

2 Literature Review and Background

2.1 Literature Review and Related Work

To this day, the academic research conducted on the various anti-cheat techniques is approximately lower than the amount of research conducted on any other related topics. This could be because almost all anti-cheat research is considered a proprietary right in private-sector companies. Most companies maintain their anti-cheat methods as a secret to ensure that it is hard to be hacked; failing to do so could compromise the security of the games developed.

In the lack of academic research, many companies like Riot took it upon themselves to keep their customers updated about their anti-cheat technologies by sharing general-level information through blogs or posts. (Papadopoulos et al., 2017; Thurimella & Mitchell, 2009; “Valve Challenged over Anti-Cheating Tools,” 2014)

Websites such as Hackmag discuss cybersecurity in detail. These websites mention methods to reverse engineer anti-cheat techniques. (*Deceiving Blizzard Warden – HackMag*, n.d.)

The writers of these websites always use a pen name to publish their findings since there is a high chance of facing legal consequences if the gaming companies ever decide to take action on them. The fear of facing legal consequences is one of the other main reasons for not having many published academic research papers on this topic. The legal consequences could be more difficult to deal with if the findings are about any commercial item.

Because of the low amount of research done directly on anti-cheat methods during gaming due to the reasons explained above, this thesis has limited resources to explore.

The authors of the paper "Player behavioural modelling for video games" by Sander C.J. Bakkes, Pieter H.M. Spronck, and Giel van Lankveld employ a method to forecast user behaviour in various scenarios. Four models—Action, Tactical, Strategic, and Player profiling—are used to do this. The authors discovered that while foreseeing a player's next move in a board game like chess or ludo may sound intriguing, it is not always possible in more complex gameplays like first-person shooters (FPS). (Bakkes et al., 2012)

The authors of another paper on "Behaviour-Based Cheat Detection in Multiplayer Games with Event-B," by Tian, H and Brooke, P.J in 2012, used their framework to distinguish between biassed and unbiased gameplay players while also providing a proactive strategy to defend the gameplay's fairness rather than a passive one. Despite not being cent percent accurate, they did demonstrate that behavioural models might be used as an anti-cheating mechanism and that this strategy will help ban cheaters. (Tian et al., 2012)

The paper "Cheat-proof playout for centralised and distributed online games" by N.E. Baughman and B.N. Levine discusses real-time cheating detection. However, it has the drawback of network lag. This does safeguard distributed server-less architecture as well as server-client architecture. An anti-cheating protocol is proposed in the paper. They also enhanced the asynchronous synchronisation protocol, which is serverless, has provable anti-cheating guarantees, is resilient in the face of packet loss, and significantly improves communication performance. (Baughman & Levine, 2001)

P. Laurens, R. F. Paige, P. J. Brooke, and H. Chivers published a paper titled "A Novel Approach to the Detection of Cheating in Multiplayer Online Games" that included one of their efforts on the subject. The authors of this work offered a proof-of-concept approach to spot dishonesty and cheating by examining gamers' behaviour. The design has different vulnerabilities and attack techniques than other anti-cheat systems. The proof-of-concept successfully distinguishes between most cheating and non-cheating players, according to their demonstration. (Laurens et al., 2007)

We now have a better grasp of all the anti-cheat methods currently in use thanks to a paper by Helsingin yliopisto. The author discusses how all the comparative analyses were conducted not only between various techniques but also between different implementations of those techniques that were compared and evaluated according to numerous criteria. The authors examined how machine learning methods and cloud gaming will impact anti-cheat technology in the future. (Lehtonen, n.d.)

2.2 Background

Client-server networking is used in modern first-person shooter games. The server, which controls the game and specifies the game's initial state, is an authoritative host. Any player who connects to the server to play the game is considered the client. The server's primary duty is to maintain the game state up to date; as often as feasible, game state updates are made to improve the gaming experience.

2.2.1 Client and Server Example

Most FPS (First Person Shooting) games have similar architecture, as shown in figure 1. For instance, let’s take the matchmaking consisting of 10 players, with 5 in each team competing against each other for victory. The aim of the game is simple: with 24 rounds, any team to win 13 rounds wins the game. Being in a shooting game, each player must try and kill the enemy team with a different set of abilities like flash (stun grenade) or shields, etc., and are allowed to have various weapons each round as per their budget after each round. The game server maintains all this so that fairness is maintained.

For each winning round or kill, there is a bonus of economy through which they can buy expensive guns or gear.

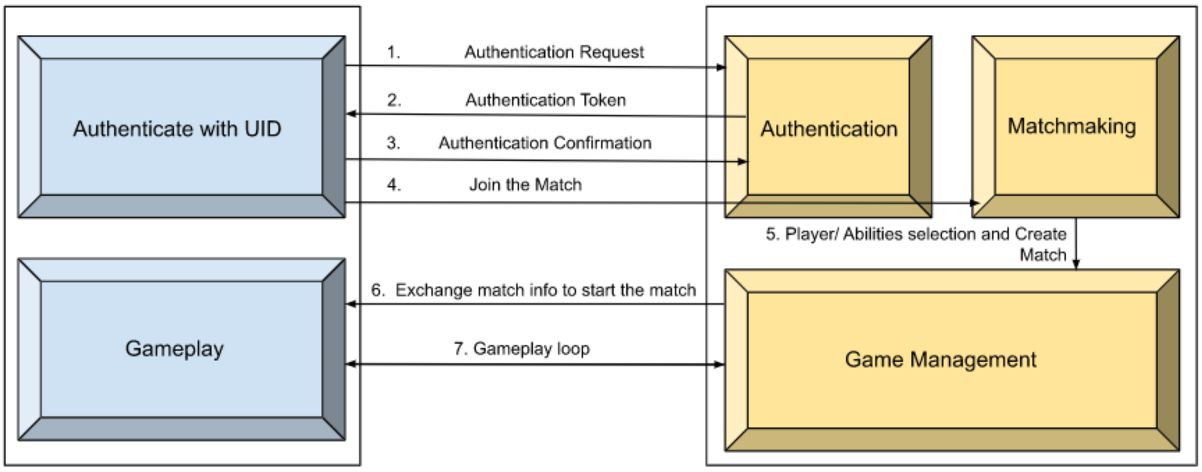

Authenticate with UID
Gameplay
1. Authentication Request
2. Authentication Token
3. Authentication Confirmation
4. Join the Match
5. Player/ Abilities selection and Create Match
6. Exchange match info to start the match
7. Gameplay loop
Authentication
Matchmaking
Game Management

Figure 1: Architecture of FPS Game Example

3 Research Methodology

Fig 1 Explains the flow and architecture of the game. In steps 1 to 3, Authentication, the Player signs in with his unique email and password, which is then authenticated by the server and generates a token or UID (unique Identity) to distinguish each player.

In step 4, matchmaking, the client sends a request to the server that it wants to be placed in the matchmaking queue. In step 5, the player is teamed up with four others and five in the enemy team. They must select an agent out of many agents with different abilities. Step 6 initiates the game, where the server rechecks all the info and ensures fairness is maintained. 7, the Gameplay loop is the main game loop that consists of constant information sharing between the client and the server. Loop is held till the game has reached its conclusion. The server's main task is updating the game state as frequently as possible; the tick rate is maintained to do this. Tick Rate controls how frequently the server updates its game state per second.

3.1.1 Tick Rate

A tick rate, for example, a server with a 35 tickrateS , will update its game state every 1 / 35 ≈ 0.028 seconds.

A tick is nothing but a single snapshot of the game state. It can be considered a unique incremental timestamp where total ticks equal {tick(0) = 1, tick(1) = tick(0) + 1, …, tick(N) = tick(N-1) + 1 }; such that tick(0) gives the initial game state and tick(N) gives the final game state.

The client and server will almost always be on separate ticks because of network latency. Figure 2 highlights this. Given a server with a tick rate of 35. While the server is at tick 120 (giving current_tick(S) = 120), the client is still at tick 115 (giving current_tick(C) = 115) due to the traversing of previous ticks still happening.

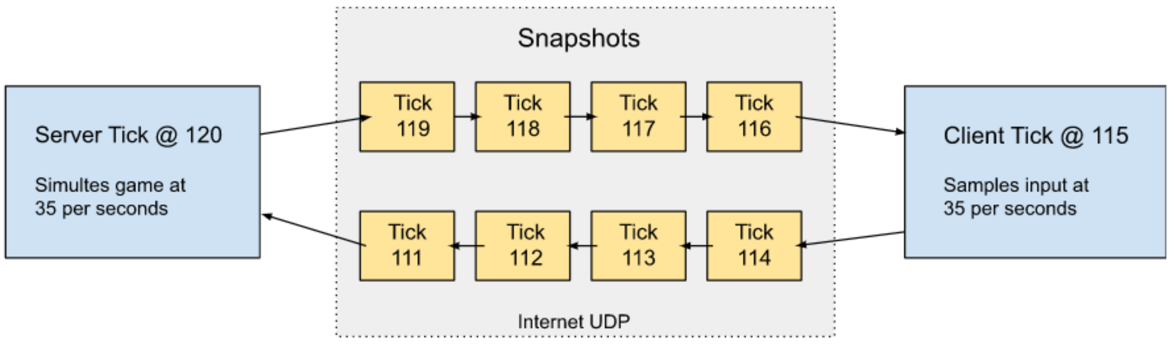


Figure 2: The architecture of FPS Game (Example)

Similarly, the server's latest response received from the client is for tick 115 (response(C) = 115).

This means the total latency between the server and client is: latency(S, C) = current_tickS - response(C) = 120 - 115 = 5 ticks.

Elaborating further in this example: for a 35 tick rate server, the amount of time the client has to wait before getting a response is 1 / tickrate(S) * latency(S, C) = 1 / 35 * 5 = 0.14 seconds Having to wait that long before a game responds to a client's input makes the experience laggy and slow for the player. To counteract this, FPS games implement a feature known as the client-side prediction.

3.1.2 Client-side prediction

In client-side prediction, the player's local inputs are simulated by the client while the server responds. Let's imagine a client sends information to advance. The client will anticipate the game state based on the player's input rather than waiting for the server to react. Any discrepancies in the client's prediction will be fixed to match the servers after receiving the server's answer. These contradictions often result in jump scenarios but usually only happen when the ping or tick rate is very high.

3.1.3 Recoil Pattern

In almost all FPS games, when the trigger is pulled, it is known to cause recoil and is expected to distract the player from their aim. This also reduces the player's accuracy and is measured by the time the gun is fired. This method is considered one to weed out unreliable players and keep the more accurate ones in the game.

The game discussed in this thesis is valorant. This game has distinct, yet random recoil patterns distinctively set for every weapon. As seen in figure 3 that the recoil pattern from the beginning causes the player to focus on the center and then veer in different directions like upwards, downwards, and sideways. Figure 3 also shows how the players adjust their focus to oppose a gun's recoil. It has been noted that even if the players are highly skilled, it becomes impossible for them to compensate for recoil, and it is a noted difference between a player and a cheater since cheaters can compensate for recoil without much trouble. Through a few reports, it can be confirmed that cheaters can use AI, which would help them to either predict the recoil patterns or, in some cases, even avoid the recoil stage all at once using high-level cheats.

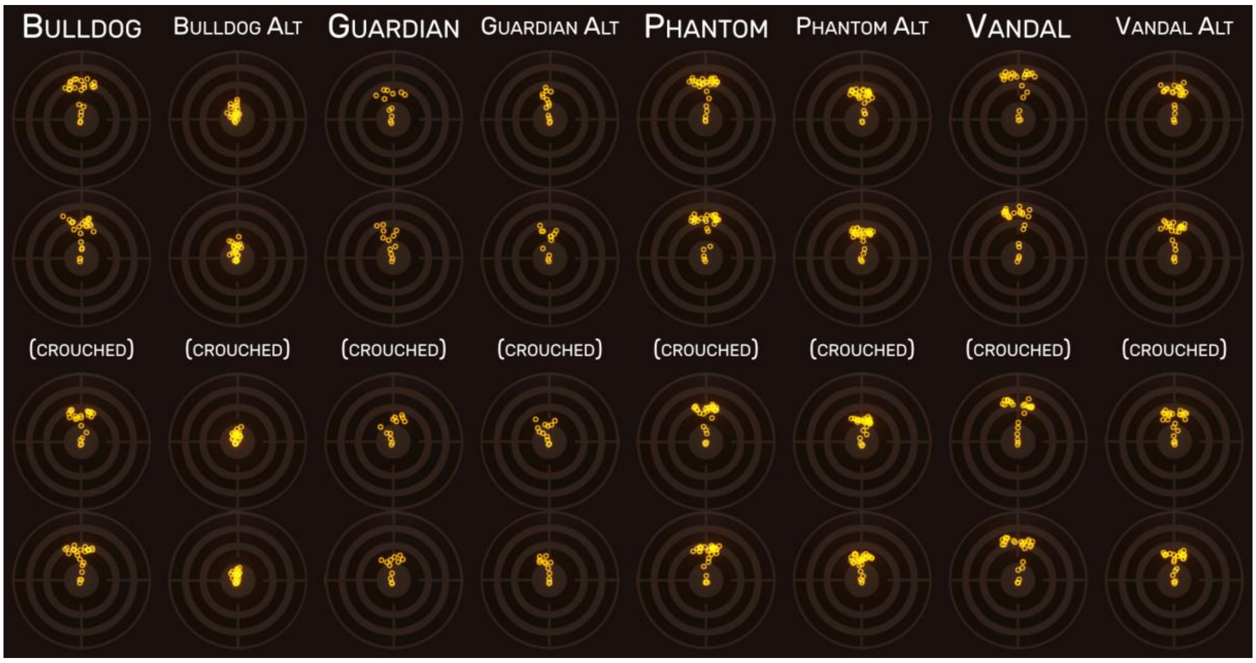


Figure 3: Recoil pattern for each gun (source - se7en.ws)

3.1.4 Types of Cheat this thesis aims to counter -

A. Wallhack

Wallhacks gives the cheater the ability to look through the walls, As the AI can detect enemy and as it's running a lightweight model on the client machine, it sees what the cheater can so if the cheater is using wallhacks the AI would be able to detect it within seconds.

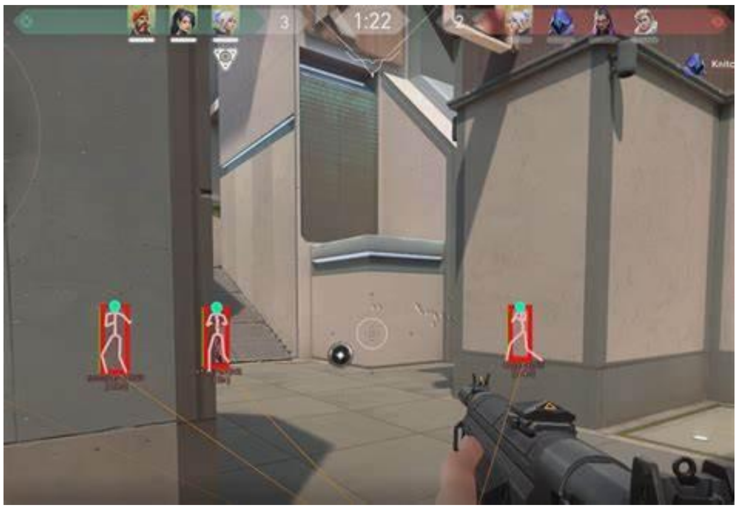


Figure 4: Wallhack (Source- oyunhacker.com)

B. Aimbot

Aimbots allow the cheater to always hit a shot with 100% accuracy. AI could detect this by simply calculating if the player gun and enemy head were in the range of a one-tap instant kill. Most of the weapons are known to decrease the damage if it has been shot from a certain distance or if the line of sight matches the enemy head and player gun or vice versa.

In this project, AI was ready to detect enemy heads and player guns. However, the cheat detection at this stage couldn't be completed due to time constraints and lack of information availability.

C. Recoil Removal

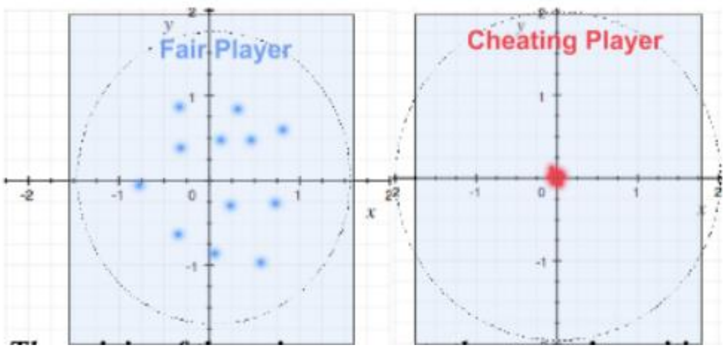


Figure 5: Recoil comparison B/W fair and cheating player

As Fig 3 and 5 above shows, each gun has unique (recoil handling) yet random (bullet spread and time) bullet spray; as the player gun is detected, the movement of the player can be tracked to see the recoil is happening and is within limits, if not and there's no recoil even when the gun is fired continuously it would be detected easily.

D. Speed Hacks

It gives the user a speed advantage that can be used to gain an advantage over the enemy. It can be detected using simple math while detecting enemies and their updated tick rate or even from frames per second.

E. Prediction based on statistical analysis

The server can gather a variety of player information, including the number of victories, kills, and deaths, as well as any other useful information the game may have. For instance, in the first person.

In a shooter, the game server may keep track of the player's kill totals per game and compare them to game averages and the player's historical kill totals. The system may flag a player for further assessment by a human reviewer if their kill total sufficiently deviates from the average for the world or their prior performance. The issue with these systems is that it is frequently difficult to draw a firm conclusion from statistical evidence. So, it should only be used as a preliminary test, and then other tests should be performed to rule any judgment.

One such anti-cheat system that employs these kinds of server-side statistical techniques is FairFight. It is highly efficient and included in many well-known games, including Titanfall 2, Tom Clancy's The Division, and Battlefield V.

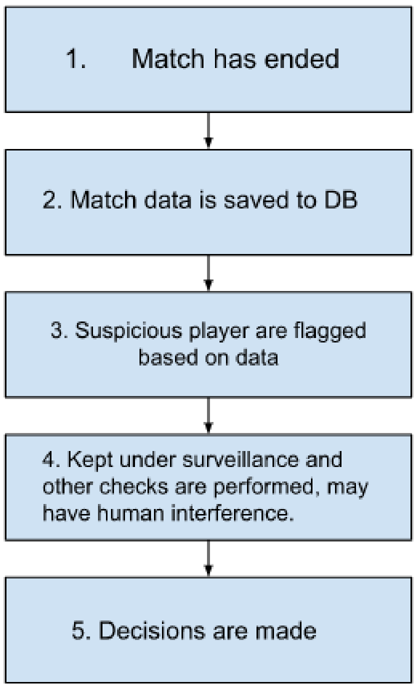

1. Match has ended
2. Match data is saved to DB
3. Suspicious player are flagged based on data
4. Kept under surveillance and other checks are performed, may have human interference.
5. Decisions are made

Figure 6: Theoretical flow for statistics-based anti-cheat layer

3.2 Exploratory Data Analysis:

The dataset used in the research is original and captured from the game called Valorant. It comprises 400 total images. To create the AI model, Yolov5 is used; it stands for - **You Only Look Once**. The most popular vision AI in the world, YOLOv5, is a product of Ultralytics open-source research on upcoming vision AI techniques. It incorporates best practices developed over thousands of hours of research and development. To annotate all the images and create a valid data set, MakeSense is used. The following labels were used to annotate what AI should look for.

Label	**Info**
Enemy	Detects the enemy with a red aura around them. For Speed or wall hacks,
Enemy_head	Detects the enemy’s head. For any instant, kill without any intended bullet hit.
Player_Gun	Detect the player’s gun to check the player's pointing position and recoil.

Table 1: Labels

Fig 7 Labels density depicts through a bar graph the total number of labels it found, through colorogram concerning xywh space where it found the label, it is density where most of the labels are found. In plots.py (Collab notebook by YOLOv5):

def plot_labels(labels, names=(), save_dir=Path('')):
top-right is the rectangles you labelled. (x,y,w,h) is the label of each instance label.

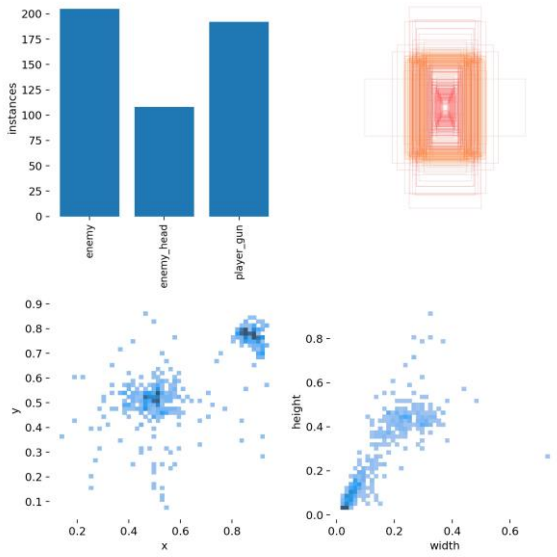
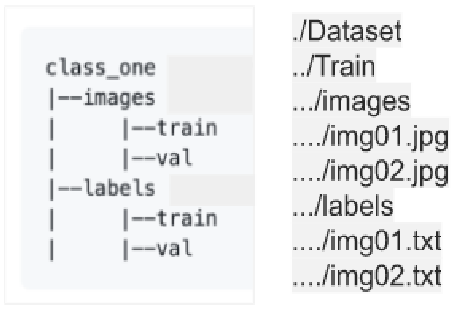


Figure 7: Labels Density

Folder structure - YOLOv5 to read the data set requires a specific folder structure, as shown in figure 8.



```
class_one
|--images
|      |--train
|      |--val
|--labels
|      |--train
|      |--val
```

```
./Dataset
../Train
.../images
..../img01.jpg
..../img02.jpg
.../labels
..../img01.txt
..../img02.txt
```

Figure 8: Folder Structure

The train folder contains training data, and val contains validation data. In this research, each folder had 400 different images split between the two folders; these were enough to create a good enough model that could detect labels within a reasonable rate.

3.3 YOLOv5 variants

YOLOv5 has four different variants of models — s, m, l, and x. Each model offers a distinct detection accuracy and performance, as demonstrated below.

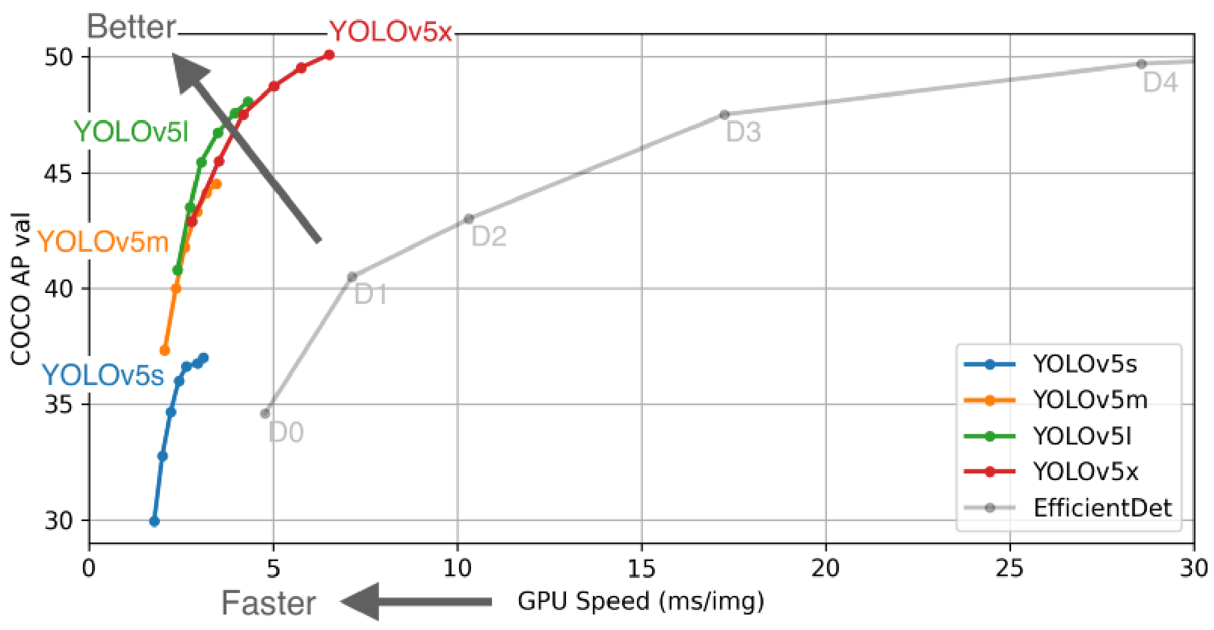
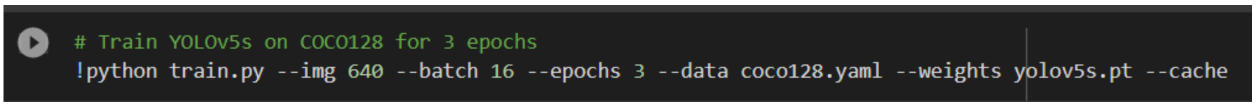


Figure 9: Source：(ultralytics, n.d.)

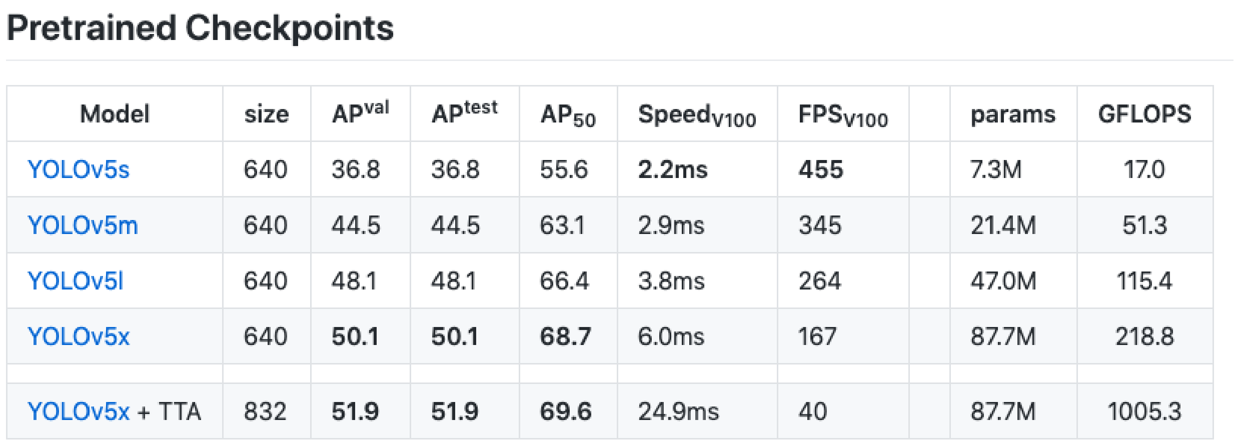
The accuracy (mAP) of v5 s is 55.6 with 17GFlops (computational power). For the research testing purpose, the Yolov5s variant is used as it's the fastest, a
nd it would work pretty well for just testing just 3 objects detection.



```
# Train YOLOv5s on COCO128 for 3 epochs
!python train.py --img 640 --batch 16 --epochs 3 --data coco128.yaml --weights yolov5s.pt --cache
```

Figure 10: YoloV5 command used to train the model

As in fig 10, the command to train the model depicts how many images should be in the batch (16), image resolution (640) with 3 epochs and using YOLOV5s.



Pretrained Checkpoints

Model	size	AP^{val}	AP^{test}	AP_{50}	$Speed_{V100}$	FPS_{V100}		params	GFLOPS
YOLOv5s	640	36.8	36.8	55.6	**2.2ms**	**455**		7.3M	17.0
YOLOv5m	640	44.5	44.5	63.1	2.9ms	345		21.4M	51.3
YOLOv5l	640	48.1	48.1	66.4	3.8ms	264		47.0M	115.4
YOLOv5x	640	**50.1**	**50.1**	**68.7**	6.0ms	167		87.7M	218.8
YOLOv5x + TTA	832	**51.9**	**51.9**	**69.6**	24.9ms	40		87.7M	1005.3

Figure 11: (Source – *Ultralytics/Yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite*, n.d.)

As a comparison, YOLOv3-416 had a mAP of 55.3 for 65.86 GFlops.

YOLOv5 s achieves the same accuracy as YOLOv3-416 with about 1/4 of the computational complexity.

The output from YOLOv5
When given a 640x640 input image, the model outputs the following 3 tensors.

(1, 2, 81, 81, 86) # anchor 0
(1, 2, 41, 41, 86) # anchor 1
(1, 2, 21, 21, 86) # anchor 2

The breakdown of the output is [cx, cy, w, h, conf, pred_cls(80)]. (ultralytics, n.d.)

4 Implementation

The current study focuses on accurately detecting each object in real-time using a custom data set with 400 images (frames from a recorded gameplay video, and then all the non-essential frames are manually deleted for a good data set). Images are stored and annotated into subfolders using makesense.ai, as YOLO requires. YOLO detects objects based on predictions in a region and then scores them using a pre-trained model. (Redmon & Farhadi, 2018).

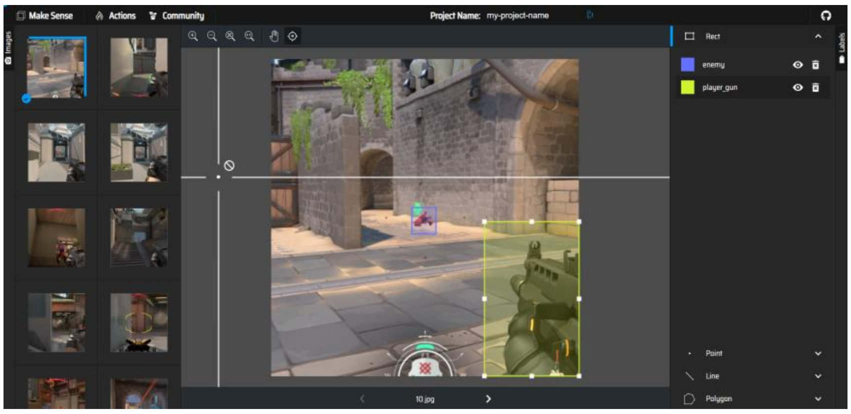


Figure 12: Classification of images using Makesense.ai

After the dataset goes through Yolo, it outputs and weights all the results to the output folder after the ai is trained. Weights are then used to run AI locally using a small python script that captures the screen and then passes through the AI to detect the Player, Player gun and Enemy. Further implementation could not be completed due to the low bandwidth of time available and technical disadvantages at this stage. As explained above, each cheats the paper tends to solve is possible with more time.

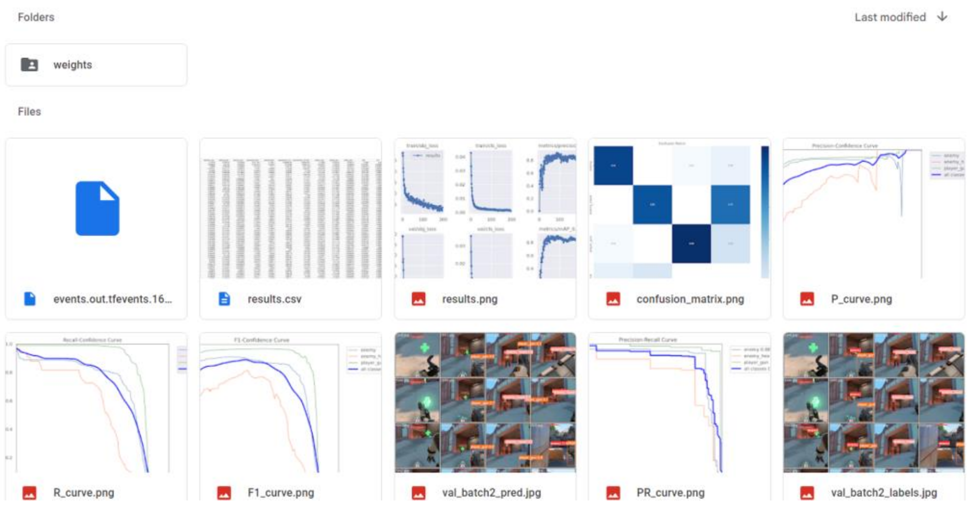


Figure 13: Output/ Results folder

5 Evaluation

With 200 images in Training folder and 200 in val folder, 400 images with three labels were used to train the model, that was manually cleaned for a good result. All the outputs below from the same data set. The performance metric used to compute the performance is selected as Accuracy along with precision, recall and f1-score for each label class.

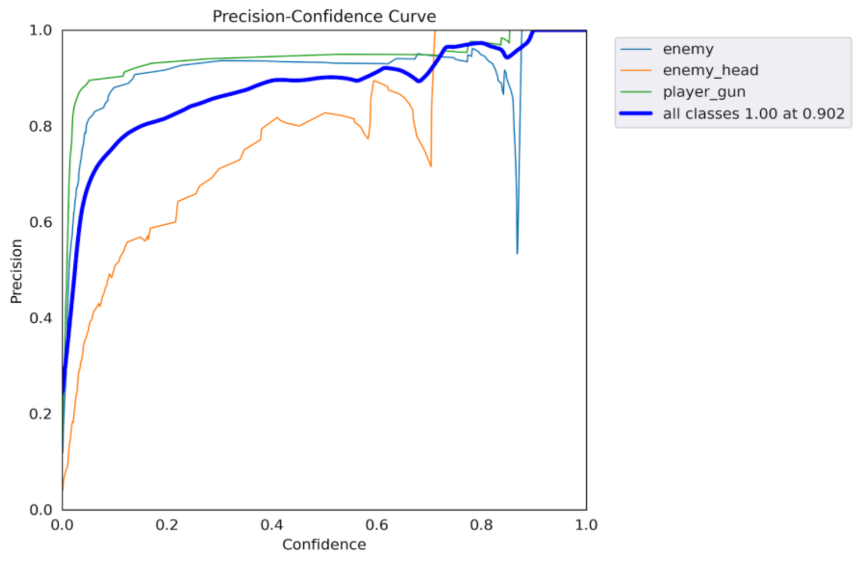


Figure 14: P Curve

P curve or precision curve shows positive predictions, that can be calculated by true/ correct positives divided by sum of true and false positives. Through the graph – most of the values lie between the region 0.8 and 1 (higher the better) so the model is pretty accurate given small dataset used.

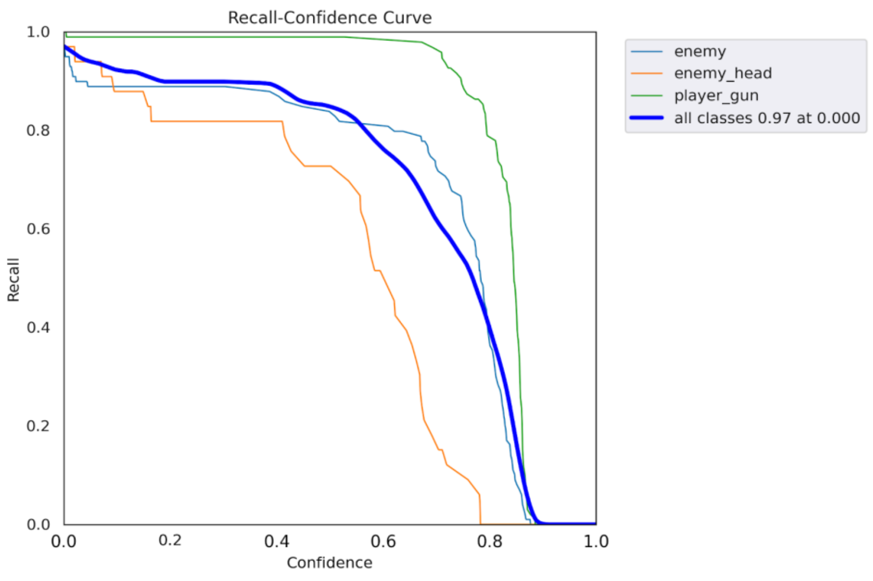


Figure 15: R_Curve

Just like P-Curve, R Curve can be calculated true positives divided by sum of true positives and false negatives. Ranging between 0 and 1 (higher the better), according to the graph above it is observed again that the model is very precise.

P-R curve gives the better understanding of the precision by combining both the above graphs, with most values above 88%. Enemy head accuracy is low due to its small size on the screen compared to other objects. It can be improved over time, but still pretty good.

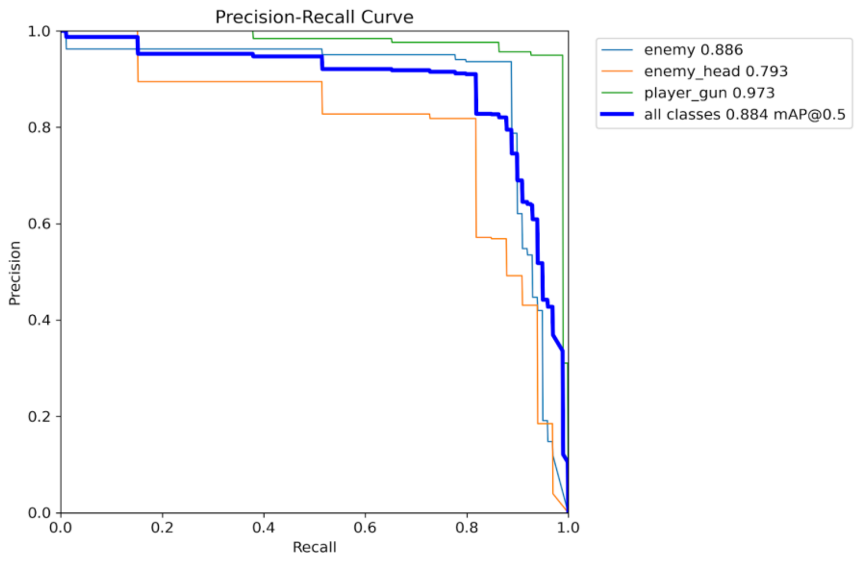


Figure 16: PR Curve

With F1 curve same trend is followed, most values above 80% and the best value is 0.89 or 89% at confidence 0.389.

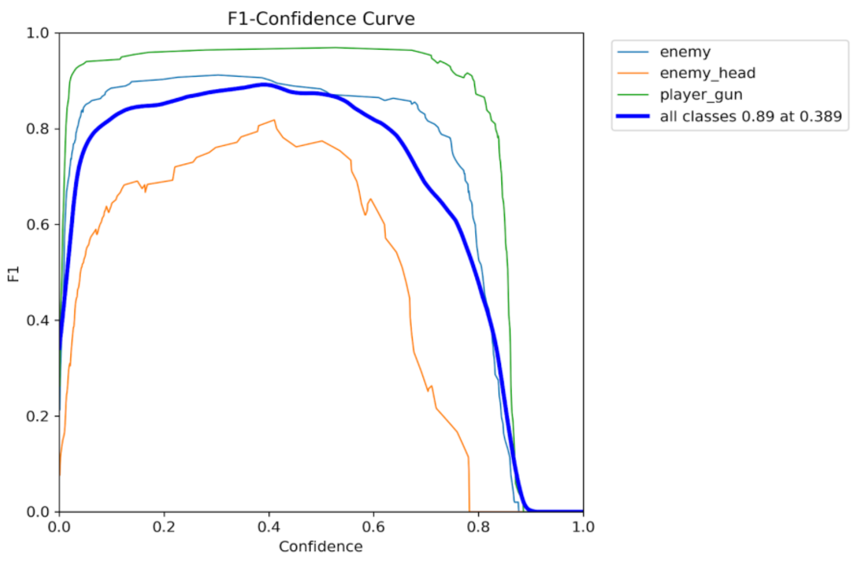


Figure 17: F1 Curve

Confusion matrix below helps in visualizing accuracy or performance of the algorithm. It again follows the same trend with pretty accurate separated values for each of the three labels.

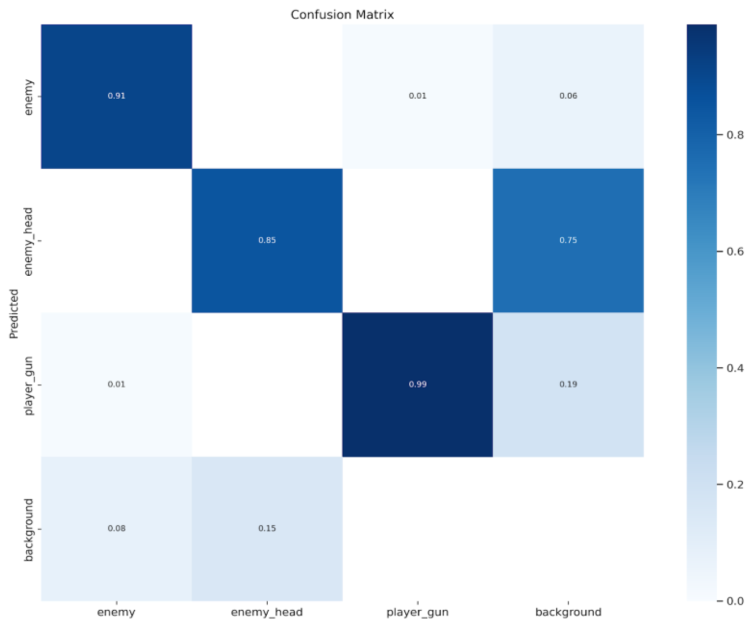


Figure 18: Confusion Matrix

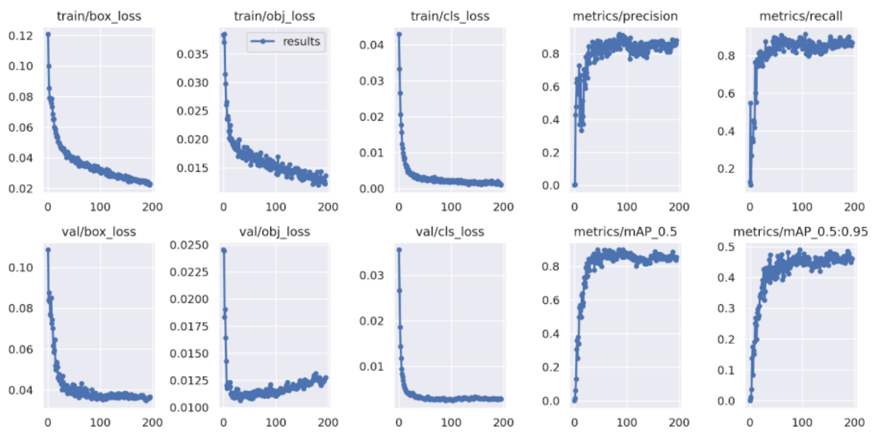


Figure 19: Training Results

5.1 Discussion

Below are the training results, visually showing what AI detects, it is clear in almost all cases it accurately detects all the labels even tiny heads of the character in some cases. But during the actual live test using training best.pt model, it may sometimes give false positives.

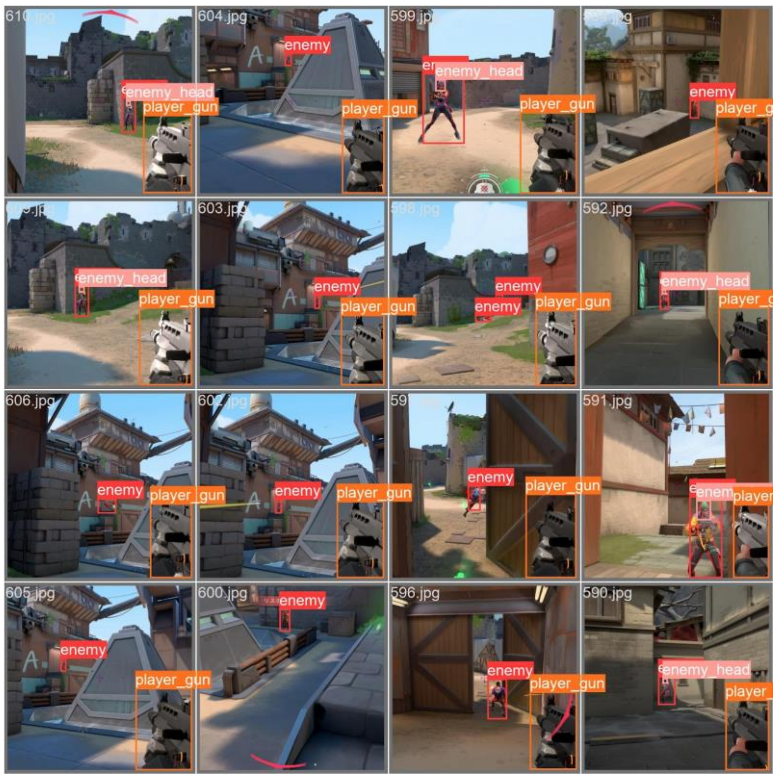


Figure 20: Training results

There’s no to almost negligible performance loss when GPU is being used by both AI and the game.

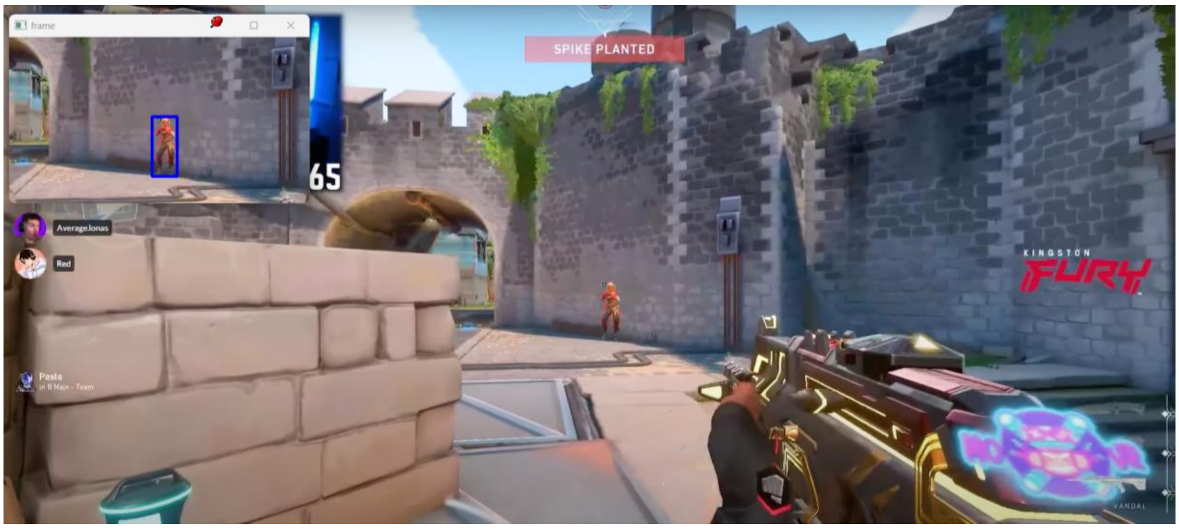


Figure 21: live capture during match (detections on the top left)

Again, the motive is to implement this using client server architecture or going completely on server using cloud gaming that would eliminated even the slightest lag. But that comes with its own challenges. These challenges can be resolved with ever growing technologies and transfer speeds but it's a task for future work.

6 Conclusion and Future Work

This study presents a practical way to differentiate cheaters from actual gamers using AI. The paper it is presented what cheats could be detected, i.e., Wallhack, Aimbot, No Recoil, Speed hacks and skill parity or cheater detection using behaviour analysis is also suggested in the paper.

Due to a lack of information and papers on anti-cheat, unless the proprietary companies implement these methods, it would have to be built on top of anti-cheat by the developers, as top multiplayer games use almost no open-source anti-cheat engine.

With the time constraints and lack of information available on the web, this paper concludes that with AI, it is possible in future to get rid of all the cheaters and cheat-selling organisations that are feeding on the company's fortune till the cheats get more complex.

Future work would include, writing the code to use the AI data to detect the cheats in real time, removing the client server architecture and going only cloud and finally implementing behavioural analysis that would help detect and suspicious player.

Opinions: Our research shows that AI holds immense promise in the field of anti-cheat and can be a game-changer for the industry.

Considerations: While AI offers a potential solution, the implementation and deployment of AI-based anti-cheat systems will require careful consideration and planning, especially with regards to privacy and data protection.

Challenges: The implementation of AI-based anti-cheat systems faces several challenges, including the need for real-time detection, the requirement for cloud-based architecture, and the complexity of behavioural analysis.

Application: AI-based anti-cheat systems have the potential to be applied across various gaming platforms and genres, providing a unified solution to the issue of cheating.

Scale of Impact: If successfully implemented, AI-based anti-cheat systems have the potential to significantly impact the gaming industry, increasing the fairness and integrity of gaming experiences and reducing the financial losses incurred by cheating.

In conclusion, this study highlights the potential for AI to revolutionize the anti-cheat landscape. Further research is needed to address the challenges and considerations outlined above, but we are optimistic about the future of anti-cheat technology.

References

Bakkes, S. C. J., Spronck, P. H. M., & van Lankveld, G. (2012). Player behavioural modelling for video games. *Entertainment Computing*, *3*(3), 71–79. https://doi.org/10.1016/j.entcom.2011.12.001

Baughman, N. E., & Levine, B. N. (2001). Cheat-proof playout for centralized and distributed online games. *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, *1*, 104–113 vol.1. https://doi.org/10.1109/INFCOM.2001.916692

Deceiving Blizzard Warden – HackMag. (n.d.). Retrieved November 7, 2022, from https://hackmag.com/uncategorized/deceiving-blizzard-warden/

Laurens, P., Paige, R. F., Brooke, P. J., & Chivers, H. (2007). A Novel Approach to the Detection of Cheating in Multiplayer Online Games. *12th IEEE International*

Conference on Engineering Complex Computer Systems (ICECCS 2007), 97–106. https://doi.org/10.1109/ICECCS.2007.11

Lehtonen, S. (n.d.). *Comparative Study of Anti-cheat Methods in Video Games*. 128.

Papadopoulos, P., Vasiliadis, G., Christou, G., Markatos, E., & Ioannidis, S. (2017). No Sugar but All the Taste! Memory Encryption Without Architectural Support. In S. N. Foley, D. Gollmann, & E. Snekkenes (Eds.), *Computer Security – ESORICS 2017* (pp. 362–380). Springer International Publishing. https://doi.org/10.1007/978-3-319-66399-9_20

Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement* (arXiv:1804.02767; Version 1). arXiv. https://doi.org/10.48550/arXiv.1804.02767

South Korea cracks down on cheaters with law targeting illicit game mo. (n.d.). Retrieved November 7, 2022, from https://www.gamedeveloper.com/console/south-korea-cracks-down-on-cheaters-with-law-targeting-illicit-game-mods

Thurimella, R., & Mitchell, W. (2009). Cloak and Dagger: Man-In-The-Middle and Other Insidious Attacks. *International Journal of Information Security and Privacy (IJISP)*, *3*(3), 55–75.

Tian, H., Brooke, P. J., & Bosser, A.-G. (2012). Behaviour-Based Cheat Detection in Multiplayer Games with Event-B. In J. Derrick, S. Gnesi, D. Latella, & H. Treharne (Eds.), *Integrated Formal Methods* (pp. 206–220). Springer. https://doi.org/10.1007/978-3-642-30729-4_15

ultralytics. (n.d.). *ultralytics/yolov5: YOLOv5*. Retrieved November 7, 2022, from https://github.com/ultralytics/yolov5

Valve challenged over anti-cheating tools. (2014, February 18). *BBC News*. https://www.bbc.com/news/technology-26240140

Widespread Cheating in Multiplayer Online Games Drives Gamers in Asia Pacific Away—Irdeto. (n.d.). Retrieved November 7, 2022, from https://irdeto.com/news/widespread-cheating-in-multiplayer-online-games-drives-gamers-in-asia-pacific-away/