Counter-Strike Character Object Detection via Dataset Generation

Matija Šinko
matija11.sinko1@gmail.com
University of Maribor
Faculty of Electrical Engineering and Computer Science
Maribor, Slovenia

ABSTRACT

This paper addresses the challenge of developing robust object detection systems in the context of Valve's Counter-Strike by introducing a novel, high-quality dataset generated using a complex image generator built within the Unity game engine. This generator mimics the original game's environment and character interactions, capturing the complexity of in-game scenarios. The dataset provides a valuable resource for training models like the YOLOv9 algorithm, which we employ to develop an object detection system that achieves high precision and recall, in turn proving the usability of our dataset. Our dataset and demonstrated model could be used for object detection in future multi-modal autonomous agents, like the one we propose at the end of the paper.

KEYWORDS

Counter-Strike, Object Detection, AI, YOLO, Autonomous Agents, Computer Vision, Data Generation

1 INTRODUCTION

Artificial intelligence (AI) has opened new possibilities in gaming, with achievements like AlphaGo and AlphaStar mastering complex environments [3, 13]. These advances have sparked interest in applying AI to automate popular games. For fans of first-person shooters like Valve's Counter-Strike [12], this raises questions about AI-driven gameplay in fast-paced, strategic environments [4].

The primary problem addressed in this paper is the development of a robust object detection system for Counter-Strike, which is a crucial component for creating autonomous agents capable of human-level gameplay. Traditional approaches have relied on manually labeled datasets, which are time-consuming to create and often struggle to keep up with game updates. Additionally, previous attempts at developing AI agents for Counter-Strike have typically employed single, monolithic neural networks, leading to mixed results in terms of performance and adaptability.

In response to these challenges, we propose a novel solution: a high-quality, automatically generated dataset created using a complex image generator within the Unity game engine. This dataset is designed to train object detection models like YOLOv9 [14], which we use to identify enemy players in Counter-Strike. Furthermore, we suggest that this dataset can be a foundation for multi-model agent architectures, which could offer more robust and adaptable AI systems for gaming.

The structure of this paper is as follows: First, we provide a detailed overview of the methodology used to generate the dataset and train the object detection model. Next, we present the results of our experiments, demonstrating the effectiveness of our approach. Finally, we discuss the potential applications of the dataset in multi-model systems and conclude with suggestions for future research.

2 METHODOLOGY

2.1 Overview of Dataset Generation

This section outlines our approach to generating a dataset for training an object detection model in Valve's Counter-Strike. Using Unity, we closely replicated in-game environments and characters to ensure the training data accurately reflects real gameplay conditions.

2.2 Image Generation Pipeline

Our image generation pipeline was built for flexibility and iterative improvement, allowing quick updates to enhance model training and evaluation. This approach was key to creating a diverse and robust dataset with various in-game scenarios (see Figure 1).

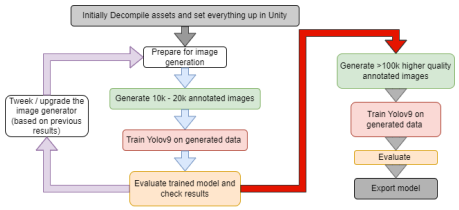

Figure 1: Object detection with image generation pipeline

2.3 Detailed Environment and Character Simulation

A critical aspect of our dataset generation process was ensuring that the simulated environments and character models closely resembled those in the actual Counter-Strike game, similar to approaches using synthetic data in Unity and Unreal engines for realistic object detection [1, 10].

2.3.1 Lighting and Rendering. We used Unity's High Definition Render Pipeline (HDRP) [11] to recreate the complex lighting conditions of Counter-Strike. Realistic lighting made the dataset closely mirror the game, helping the model generalize to real gameplay.

2.3.2 Character Positioning and Inverse Kinematics (IK). To create realistic and varied character poses, we applied Inverse Kinematics (IK) algorithms[15]. IK enabled dynamic posing of characters in natural scenarios like aiming, shooting, and navigating.

2.4 Object Detection and YOLO Algorithm

Object detection is key to validating our dataset, focused on identifying enemy players in the game. We used the YOLOv9 algorithm,

known for its speed and accuracy, to demonstrate the dataset's effectiveness. Training YOLOv9 on our dataset confirmed its utility. Though this study centers on YOLOv9, similar detectors could be used in more complex multi-modal agents in the future.

3 THE PROPOSED METHOD

3.1 Detailed Dataset Generation Process

As discussed earlier, Our object detectior was trained on a large Unity-generated dataset comprising varied scenarios on a detailed Dust 2 map with both Terrorist and Counter-Terrorist characters.

3.1.1 Environment and Character Creation.

(1) **Map Recreation**: The Dust 2 map was decompiled from Counter-Strike 2 game files and imported into Unity 3D. After cleaning up the geometry, we obtained a 1:1 replica of the map. (see Figure 2)

Figure 2: Comparison of Dust 2 in the Counter-Strike 2 game (left) and Unity Recreation (right)

(2) **Character Models**: We extracted and rigged character models for Terrorists and Counter-Terrorists using Blender, then imported them into Unity with various poses and weapons. (see Figure 6 in Appendix)
(3) **Lighting and Rendering**: Unity's High Definition Render Pipeline (HDRP) was employed to set up realistic lighting conditions, using path tracing technology to closely mimic the visuals of Counter-Strike 2. (see Figure 3)

Figure 3: Comparison: Real Counter Terrorist (left) and Unity recreation (right)

3.1.2 Data Annotation and Generation.

(1) **Random Placement**: Characters were randomly placed around the map, and virtual cameras were positioned to capture various perspectives.
(2) **Dynamic Posing**: Using the Final IK plugin for inverse kinematics [7], characters were given dynamic poses aimed at different targets, adding variability to the training data.
(3) **Labeling**: Our system automatically annotated images with bounding boxes for each character, distinguishing between Terrorists, Counter-Terrorists, and their states (alive or dead).

4 RESULTS

4.1 Purpose of the Experimental Work

The goal of our experimental work was to train a YOLOv9 object detector with sufficient accuracy and recall. This would be critical for potential future use in multi-model autonomous agents, where accuracy would be needed to distinguish between friendly and enemy characters, as well as dead and alive ones. High recall would be vital for quickly and reliably spotting characters and differentiating them from the background, which would be essential for agents that rely on object detection models for shooting tasks.

4.2 Comparative Studies and Setups

We iteratively refined our image generator, creating datasets to train and evaluate the object detection model, with each iteration enhancing accuracy and recall:

(1) **Initial Model:** A dataset of 10,000 images featuring only alive characters in various poses, serving as the baseline.
(2) **Addition of Dead Characters:** Introduced separate labels for dead characters to improve recall, especially in distinguishing live enemies from the background.
(3) **First-Person Perspective and UI Overlays:** Added hands and UI elements to reduce misclassification of player arms as enemies.
(4) **Blood Splatter Effects:** Introduced blood effects to enhance precision in differentiating character states.
(5) **Name Tag-Based Identification:** Added name tags to distinguish friendly from enemy characters, as friendlies always have tags above their heads. Some tags without visible characters belong to friendlies behind walls, which we avoid classifying. This approach required training two separate models, one for each team.

These were evaluated for precision and recall improvements.

4.3 Datasets Used for Testing

To ensure the robustness and generalizability of the object detection model, we evaluated it on a variety of datasets:

(1) **Self**: This dataset was generated using the same methods as the training set, serving as a control to measure overfitting and baseline performance.
(2) **Bots CT and T Combined**: Captured from bot games in Counter-Strike, this one did not include unique player skins. This dataset is the closest to a final deployment scenario.
(3) **Batch 3**: This dataset included captures from real multiplayer games, featuring unique player skins and a variety of in-game environments.
(4) **Batch 4**: Similar to Batch 3 but sourced from different multiplayer sessions, providing additional variety in testing conditions.
(5) **Batch 1**: Captured from the Deathmatch mode, this dataset differs most from the intended deployment environment but provides insights into model generalization.
(6) **Bots, Batch 3, Batch 4 Combined**: A comprehensive dataset combining Bots CT and T, Batch 3, and Batch 4, used to test overall model performance across various conditions.
(7) **All Combined**: A super-set combining all the above datasets.

These datasets tested the model's strengths and weaknesses.

4.4 Evaluation Metrics

We measured various evaluation metrics, including Precision, Recall, F1 Score, mAP, and IoU, with detailed results available for each in the Appendix. Our focus was on Precision and Recall. High Precision is vital for accurately identifying enemy characters, avoiding misclassification of friendly units, while high Recall ensures all enemies are detected quickly and distinguished from the backround. These metrics are crucial for creating robust object detection models that could be used for potential future autonomous agents in competitive gaming.

4.5 Announcement of the Experiments Performed

We conducted experiments to evaluate object detection models trained on our dataset, testing their ability to detect and classify characters in Counter-Strike under various conditions. These experiments included assessing different character states (alive, dead) and the impact of dataset sizes and image resolutions, aiming to refine the model's precision and recall for real-world gameplay.

4.6 Detailed Descriptions of Experiments and Results

4.6.1 Image Generator Upgrades.

(1) **Initial Model of Only Alive Characters**: This model showed good precision but lacked recall, which is crucial for our application.
(2) **Addition of Dead Characters**: Improved the model's recall, which is crucial for distinguishing between live enemies and background elements.
(3) **First-Person Perspective and UI Overlays**: addressed the issue of confusing player arms with enemy characters. Improved recall.
(4) **Added Blood Splatter Effects**: improved both precision and recall for the terrorist dead and alive classes.
(5) **Friendly Character Identification via name tag**: Showed best results and vastly improved accuracy and recall . *Batch 1, trained in Deathmatch mode without name tags, performed worse, but this isn't a concern since the dataset is intended for Competitive mode, where name tags are always present.*

4.6.2 Cross Model Examination. The purpose of the cross-model examination is to compare the performance of different model configurations across various metrics.

As observed in Figure 4, our upgrades to the Unity Generator were successful in improving performance. For example, the recall for the counter-terrorist class improved from 0.4 to over 0.6. Graphs for the class Terrorist as well as the F1 score can be found in the Appendix A.3. While the improvements on the dead character classes seen in Figure 14 in the Appendix, were not significant, we observed notable enhancements in the performance for the terrorist and counter-terrorist classes see 12, 13. This distinction is crucial, as our primary objective is for our dataset to be suitable for potential training of agents that can accurately differentiate between alive friendly and enemy characters. The dead characters need only to be distinguished from the living, without requiring detailed differentiation among themselves. Tabled and detailed Data for these cross model examinations can be found in the Appendix A.3.

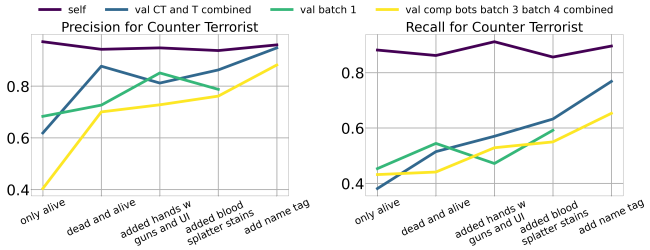

Figure 4: Counter-Terrorist class across all of our generation upgrades on 10k 640x360 images

4.6.3 Testing Different Dataset Sizes. We trained models on varying dataset sizes ranging from 1,000 to 100,000 images to examine the impact on performance. As seen in Figure 15 in the Appendix Larger training sizes improved model performance significantly.

4.6.4 Bigger Image Sizes and a Bigger Model. We explored the effects of training with larger image sizes and using a bigger YOLOv9 model and we came to the conclusion that bigger image sizes and a bigger model lead to better performance all around See Apendix A.5. That's why we trained out best model on a 1280x720 image size and with 20k images

4.6.5 Comparison with Other Counter-Strike Object Detectors. In this section, we compare our best object detection model, trained on a dataset of 20k images at a resolution of 1280x720 with name tags, to three other models developed by different authors:

- **Our Best Model**: (Terrorist team, 1280x720, 20k images).
- **Siromer's Model**: Dataset taken from [8, 9].
- **Python Lessons Model**: Dataset taken from [5, 6]. This dataset was used by Chenyang Dai [2] for object detection in their hybrid imitation training model.

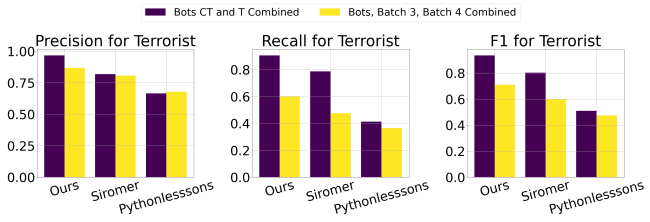

Figure 5: Comparison of object detection models

Our curated bar charts show that our object detector outperforms other methods in detecting Terrorists on the two selected datasets See table 1. For results on additional datasets and classes, see the Appendix A.6. Notably, Chenyang Dai used the 'Python Lessons' dataset to train the object detection part of their multi-model AI. This indicates that integrating our dataset into a multi-model Counter-Strike 2 system could potentially yield superior performance, as our dataset seems better than Python Lessons'. These findings support our initial hypothesis that our dataset could be used to train more complex multi-model agents.

Table 1: Detailed Metrics for Class: Terrorist

Metric	Bots CT and T Combined			Bots, Batch 3, Batch 4 Combined		
	Ours	Siromer	PyLessons	Ours	Siromer	PyLessons
Precision	0.97	0.82	0.67	0.87	0.81	0.68
Recall	0.91	0.79	0.41	0.60	0.48	0.37
F1 Score	0.94	0.80	0.51	0.71	0.60	0.47

4.6.6 Video Demonstrations of Model Performance. We provide video demonstrations of our object detection model's performance for both Counter-Terrorist and Terrorist scenarios. Available on YouTube:

- **Counter-Terrorist**: www.youtube.com/watch?v=u49CLDt8MgU
- **Terrorist**: www.youtube.com/watch?v=u49CLDt8MgU

4.7 Discussion

The improvements through our image generation pipeline have successfully enhanced model performance. The precision achieved by the model is sufficient to avoid mistakenly identifying teammates or dead characters as threats, while the recall is robust enough to reliably detect enemy players. Any remaining inaccuracies could be further refined with reinforcement-based shooting models that adapt to detection patterns when used on our generated dataset. In comparison to previous studies, such as [4], which encountered issues like agents mistakenly shooting dead characters, our approach offers a clear advantage. The dataset we generated, paired with a specific object detection model, can relieve a potential agent from the need to learn the shapes of characters, allowing it to focus more on tasks like shooting accuracy and map traversal when learning. While our model shows promise, it is currently limited to detecting objects within the Dust 2 map and is sensitive to player cosmetic skins. Additionally, the model does not differentiate between body parts, which may limit its application in more precise, action-oriented tasks.

5 CONCLUSION

5.1 Summary of Work and Key Findings

This research focused on the development and validation of a high-quality dataset generated using a Unity-based image generator for training object detection models in the context of Counter-Strike. Through iterative enhancements to our image generation pipeline, we achieved significant improvements in both precision and recall of the YOLOv9-based object detection model. This validated the effectiveness of our approach, demonstrating that synthetic data can effectively train models for complex in-game scenarios.

5.2 Best Results and Contributions

Our study made several key contributions to the field:

- **Versatile Dataset for Object Detection**: Our image generator and datasets are valuable for training object detection models in Counter Strike.
- **Effective Use of Synthetic Data Proven by Object Detection**: We showed that synthetic data can replace real-world data in training models, especially when labeled data is scarce. This was proven by achieving strong results with an object detection model trained on our generated data.
- **Future Applications**: Our work could be incorporated into future autonomous agents or used as object detection teaching exercises.

5.3 Future Work

Looking forward, there are several avenues for enhancing the capabilities of our system:

- **Expansion of Dataset and Model Generalization**: Future work will focus on expanding the dataset to include additional decompiled maps and the introduction of random cosmetic skin patterns to improve the model's generalization. Additionally, incorporating YOLOv9 pose estimation will allow for the identification of specific character body parts, thereby enhancing the model's ability to aim and shoot with greater effectivenes in a potential reinforcement learning framework.
- **Proposed Multi-Model Agent Architecture**: We propose a complex multi-model architecture that could serve as the foundation for developing autonomous agents capable of high-level gameplay in Counter-Strike see Appendix A.7.
- **Planned Dataset Publication**: We plan to publish our dataset on platforms like Kaggle, Hugging Face, and Roboflow, allowing others to use it for developing agents and practicing object detection skills.

5.4 Final Thoughts

This study underscores the potential of synthetic data and iterative model development in advancing AI for gaming. While challenges remain, particularly in bridging the gap between synthetic and real-world data, the progress made here provides a solid foundation for future innovations. The proposed multi-model architecture represents a promising direction for developing more sophisticated and capable autonomous agents, capable of performing at a high level in complex gaming environments like Counter-Strike. As AI continues to evolve, integrating reinforcement learning and advanced detection techniques will be crucial in pushing the boundaries of what these agents can achieve.

REFERENCES

[1] Per-Arne Andersen, Teodor Aune, and Daniel Hagen. 2022. Development of a Novel Object Detection System Based on Synthetic Data Generated from Unreal Game Engine. *Applied Sciences* 12 (08 2022). https://doi.org/10.3390/app12178534

[2] Chenyang Dai. 2021. Counter-Strike Self-play AI Agent with Object Detection and Imitation Training. mailto:qddaichy@stanford.edu. CS230: Deep Learning, Fall 2021, Stanford University, CA. (LaTeX template borrowed from NIPS 2017.).

[3] Guillaume Lample and Devendra Singh Chaplot. 2018. Playing FPS Games with Deep Reinforcement Learning. arXiv:1609.05521 [cs.AI] https://arxiv.org/abs/1609.05521

[4] Tim Pearce and Jun Zhu. 2021. Counter-Strike Deathmatch with Large-Scale Behavioural Cloning. arXiv:2104.04258 [cs.AI] https://arxiv.org/abs/2104.04258

[5] Python Lessons. 2020. TensorFlow 2.3.1 YOLOv4 - CSGO Aimbot. https://github.com/pythonlessons/TensorFlow-2.3.1-YOLOv4-CSGO-aimbot. Accessed: 2024-08-30.

[6] Python Lessons. 2020. YOLOv4 TensorFlow 2.3.1 - CSGO Aimbot. https://pylessons.com/YOLOv4-TF2-CSGO-aimbot. Accessed: 2024-08-30.

[7] RootMotion. 2014. Final IK: The Ultimate IK Solution for Unity. http://www.root-motion.com/final-ik.html. Accessed: 2024-08-30.

[8] Faruk Günaydin (Siromer). 2024. Counter-Strike 2 Body and Head Classification. https://www.kaggle.com/datasets/merfarukgnaydn/counter-strike-2-body-and-head-classification. Accessed: 2024-08-25.

[9] Faruk Günaydin (Siromer). 2024. CS:GO/CS2 Object Detection. https://medium.com/@siromermer/csgo-cs2-object-detection-db234312f9b6. Accessed: 2024-08-25.

[10] et al. Steve Borkman, Adam Crespi. 2021. Unity Perception: Generate Synthetic Data for Computer Vision. arXiv:2107.04259 [cs.CV] https://arxiv.org/abs/2107.04259

[11] Unity Technologies. 2024. High Definition Render Pipeline (HDRP). https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@12.1/manual/index.html. Accessed: 2024-08-30.

[12] Valve Corporation. 2023. Counter-Strike 2. https://www.counter-strike.net/cs2. Accessed: 2024-08-30.

[13] Hado van Hasselt, Arthur Guez, and David Silver. 2015. Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs.LG] https://arxiv.org/abs/1509.06461

[14] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. 2024. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. arXiv:2402.13616 [cs.CV] https://arxiv.org/abs/2402.13616

[15] Chris Welman. 1993. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation.* Simon Fraser University.

A APPENDIX

A.1 Detailed Dataset Generation Process

Figure 6: Various character poses for: Terrorists and Counter-Terrorists each aiming at a target

A.2 Model Upgrades

Figures related to the model upgrades and their detailed results can be found here:

Figure 7: Alive Counter-Terrorist and Terrorist characters

Reference to the image context in the document

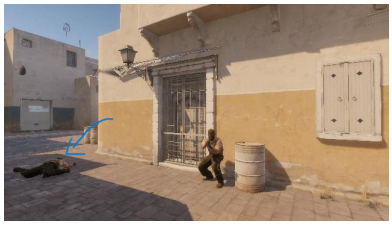

Figure 8: Dead and alive characters

Reference to the image context in the document

Figure 9: First-person hands and UI

Reference to the image context in the document

Figure 10: Blood next to a dead character

Reference to the image context in the document

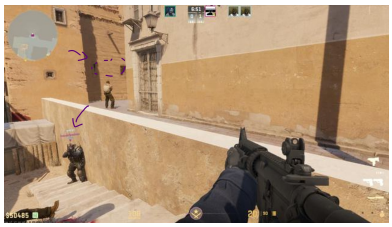

Figure 11: Image for the Coutner Terrorist model with a name tag over a Counter terrorist and a Terrorist with no name tag

A.3 Cross Model Examination data

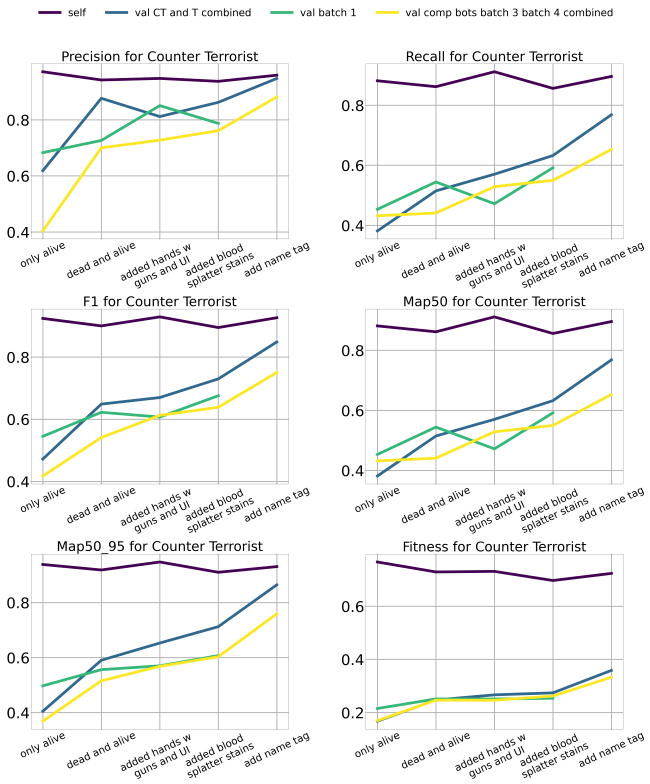

Figure 12: Counter-Terrorist class across all of our generation upgrades

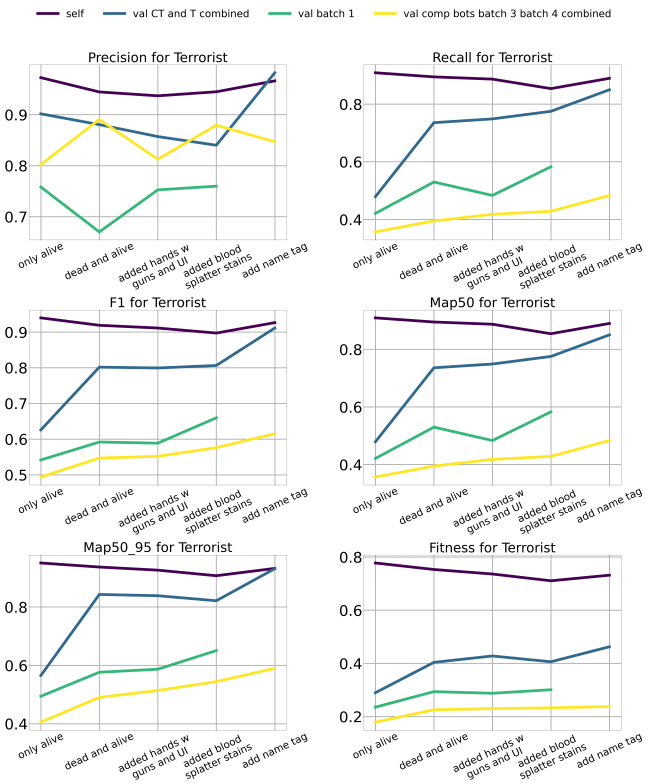

Figure 13: Terrorist class across all of our generation upgrades

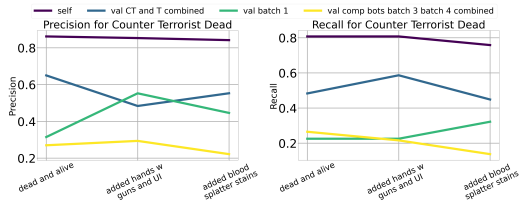

Figure 14: Counter-Terrorist dead class across all of our generation upgrades

Table 2: Precision for Class: Terrorist

Data Set	Self	CT and T Combined	Batch 1	Batch 3 & 4
Only Alive	0.972	0.901	0.758	0.802
Dead and Alive	0.944	0.880	0.670	0.890
Hands w/ Guns & UI	0.937	0.857	0.752	0.812
Added Blood Splatter Stains	0.945	0.840	0.760	0.879

Table 3: Recall for Class: Terrorist

Data Set	Self	CT and T Combined	Batch 1	Batch 3 & 4
Only Alive	0.909	0.479	0.421	0.357
Dead and Alive	0.894	0.736	0.530	0.395
Hands w/ Guns & UI	0.887	0.749	0.483	0.418
Added Blood Splatter Stains	0.854	0.775	0.583	0.428

Table 4: F1-Score for Class: Terrorist

Data Set	Self	CT and T Combined	Batch 1	Batch 3 & 4
Only Alive	0.940	0.626	0.541	0.494
Dead and Alive	0.919	0.802	0.592	0.547
Hands w/ Guns & UI	0.911	0.799	0.589	0.552
Added Blood Splatter Stains	0.897	0.806	0.660	0.576

Table 5: Combined Metrics for Class: All Metrics

Data Set	Self	CT and T Combined	Batch 1	Batch 3 & 4
Blood Dataset 1k	0.919	0.598	0.544	0.448
Blood Dataset 5k	0.920	0.608	0.519	0.416
Blood Dataset 10k	0.916	0.714	0.575	0.536
Blood Dataset 20k	0.919	0.665	0.574	0.464
Blood Dataset 100k	0.905	0.737	0.568	0.460

A.4 Testing Different Dataset Sizes

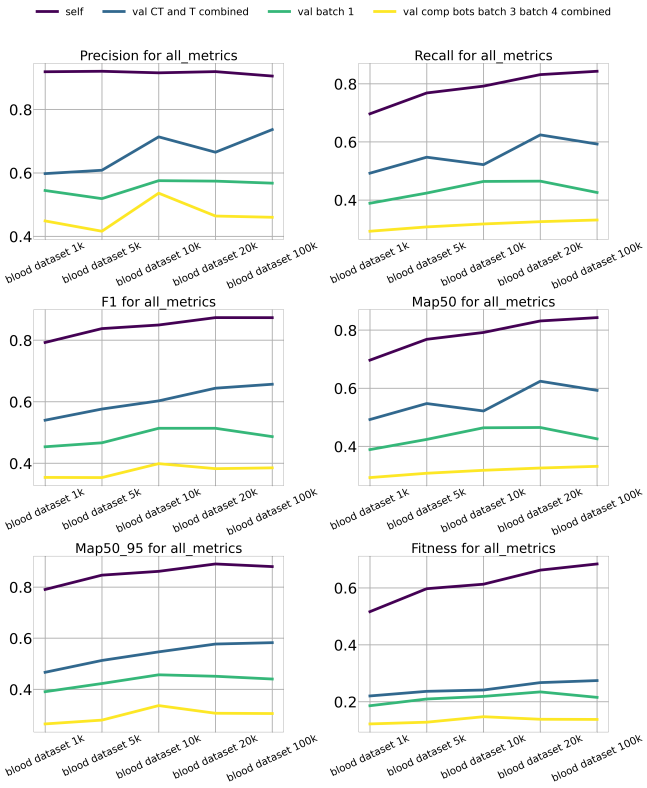

Figure 15: all metrics (classes) combined for different dataset sizes

Table 6: Detailed Metrics for Class: Counter Terrorist Dead

Data Set	Self	CT and T Combined	Batch 1	Batch 3 & 4
Only Alive	-	-	-	-
Dead and Alive	0.861	0.650	0.315	0.270
Hands w/ Guns & UI	0.853	0.483	0.552	0.294
Added Blood Splatter Stains	0.841	0.553	0.446	0.222

A.5 Bigger Image Sizes and a Bigger Model

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	params (M)	FLOPs (B)
YOLOv9t	640	38.3	53.1	2.0	7.7
YOLOv9s	640	46.8	63.4	7.2	26.7
YOLOv9m	640	51.4	68.1	20.1	76.8
YOLOv9c	640	53.0	70.2	25.5	102.8
YOLOv9e	640	55.6	72.8	58.1	192.5

Figure 16: YOLO model sizes

YOLO offers five different model sizes 16, each with a trade-off between accuracy and training/inference speed. All of our experiments so far have been conducted using the smallest model, YOLOv9t, due to its faster inference time. The reasoning is that if our object detector is to be used as part of a real-time autonomous agent, we need the fastest possible response time (inference time). Similarly, we have used images with dimensions of 630x360 for all experiments so far, as larger images provide better accuracy but at the cost of slower processing speeds. We have also explored dataset sizes, which show a similar trend: larger datasets yield better accuracy but result in longer training times. Such adjustments to our dataset, such as changing model size, image size, or dataset size, are dynamic and easy to make when using an image generator like ours but are very rigid and time-consuming in traditional hand-labeling scenarios. In this section, we will examine how different model sizes, image dimensions, and dataset sizes affect accuracy and compare their performance. Here we compare 4 different models to see how effective different methods are (image size, dataset size, model size).

- A 10k YOLOv9 tiny model with image size (640x360)
- A 100k YOLOv9 tiny model with image size (640x360)
- A 10k YOLOv9 model with image size (640x360)
- A 10k YOLOv9 tiny model with image size (1280x720)

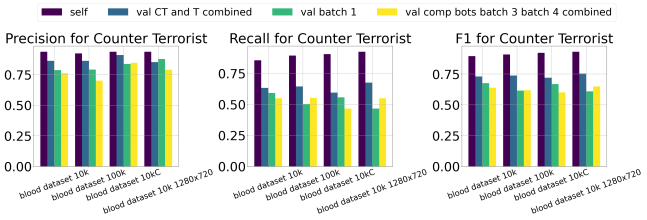

Figure 17: 1280x720 model next to previous models

Table 7: Performance Metrics for Class: Counter Terrorist

Data Set	10k T			100k T			10k C		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
Self	0.937	0.856	0.895	0.924	0.894	0.909	0.937	0.905	0.921
Bots CT and T Combined	0.862	0.632	0.730	0.862	0.645	0.738	0.910	0.595	0.719
Batch 1	0.787	0.592	0.676	0.792	0.503	0.615	0.837	0.556	0.668
Bots, Batch 3, Batch 4 Combined	0.761	0.550	0.638	0.700	0.553	0.618	0.846	0.466	0.601

From the above bar charts, we can deduce that having a bigger dataset, choosing a bigger model, and using larger image sizes helps improve our model's performance. The best performance was observed in the model trained on larger images. Future work could involve training a model on 100k images of size (1280x720).

Check the Appendix to see how the (1280x720) model compares to our previous generation upgrades. A.5.

We can see that the higher image size aligns with our trend of increasing recall, which was our original goal.

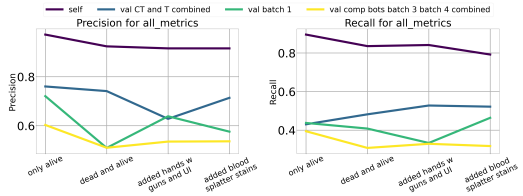

Figure 18: Counter-Terrorist dead class across all of our generation upgrades

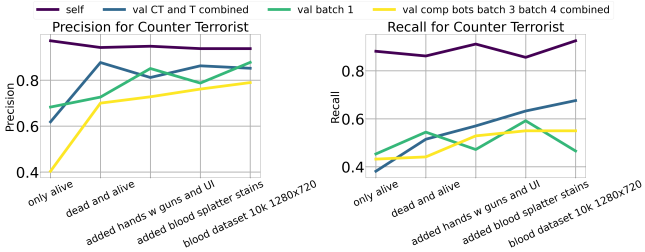

Figure 19: 1280x720 model compared to previous models

A.6 Comparison with Other Counter-Strike Object Detectors

In this section, we present a broader comparison of our model against those trained on Siromer's and PythonLessons' datasets across more metrics and datasets.

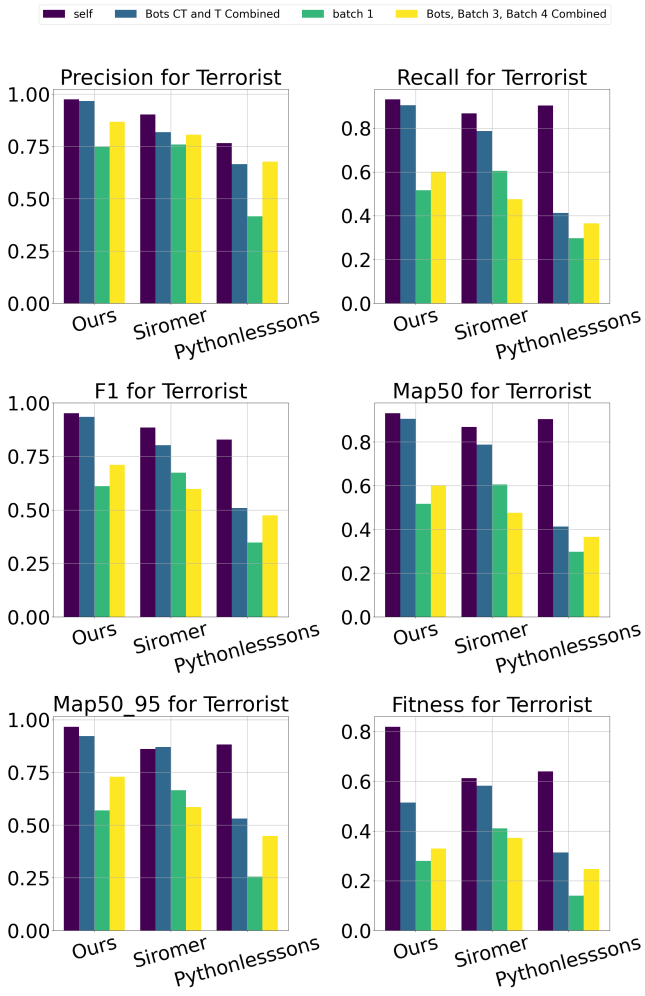

Figure 20: Comparison of object detection models expanded on the class "Terrorist"

Table 8: Expanded Metrics for Class: Terrorist

Metric	Precision				Recall			
Data Set	Self	Bots CT & T	Batch 1	Bots, B3, B4	Self	Bots CT & T	Batch 1	Bots, B3, B4
Ours (1280x720)	0.97	0.97	0.75	0.87	0.93	0.91	0.52	0.60
Siromer	0.90	0.82	0.76	0.81	0.87	0.79	0.61	0.48
PyLessons	0.77	0.67	0.42	0.68	0.90	0.41	0.30	0.37
Metric	F1 Score				mAP50			
Data Set	Self	Bots CT & T	Batch 1	Bots, B3, B4	Self	Bots CT & T	Batch 1	Bots, B3, B4
Ours (1280x720)	0.95	0.94	0.61	0.71	0.93	0.91	0.52	0.60
Siromer	0.88	0.80	0.67	0.60	0.87	0.79	0.61	0.48
PyLessons	0.83	0.51	0.35	0.47	0.90	0.41	0.30	0.37
Metric	mAP50_95				Fitness			
Data Set	Self	Bots CT & T	Batch 1	Bots, B3, B4	Self	Bots CT & T	Batch 1	Bots, B3, B4
Ours (1280x720)	0.97	0.92	0.57	0.73	0.82	0.51	0.28	0.33
Siromer	0.86	0.87	0.67	0.59	0.61	0.58	0.41	0.37
PyLessons	0.88	0.53	0.26	0.45	0.64	0.31	0.14	0.25

As shown in Figure 20 and seen in table 8, our model outperforms the other two models across all metrics, except when evaluated

on the "Batch 1" dataset. This is not entirely surprising. As previously mentioned, our best-performing model relies heavily on the effectiveness of name tags above characters' heads to distinguish friendlies from enemies. This approach led to significant improvements but performed worse on datasets that were trained in game modes like "Deathmatch," where no name tags are displayed above friendly characters. Batch 1, as stated earlier, is trained in such a deathmatch game mode. Therefore, more traditional approaches, such as those using Siromer's and PythonLessons' datasets, generalize better in this scenario. However, this is not a major concern for us since our model is designed for the competitive game mode, where two teams of five players face off and friendly characters always have name tags above their heads.

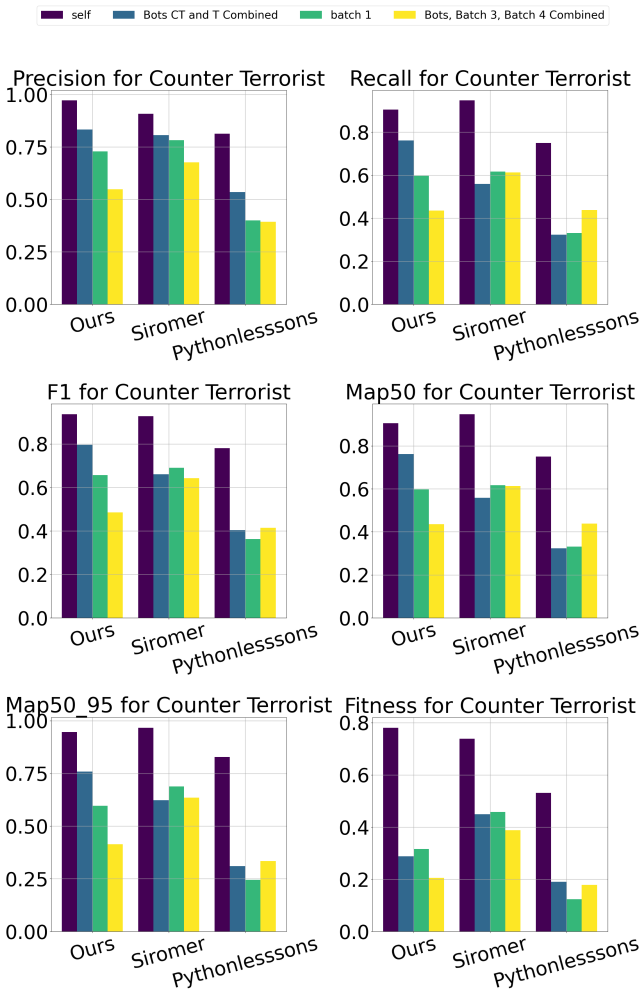

Figure 21: Comparison of object detection models expanded on the class "Counter-Terrorist"

Table 9: Expanded Metrics for Class: Counter-Terrorist

Metric	Precision				Recall			
Data Set	Self	Bots CT & T	Batch 1	Bots, B3, B4	Self	Bots CT & T	Batch 1	Bots, B3, B4
Ours (1280x720)	0.97	0.83	0.73	0.55	0.91	0.76	0.60	0.44
Siromer	0.91	0.81	0.78	0.68	0.95	0.56	0.62	0.61
PyLessons	0.81	0.54	0.40	0.39	0.75	0.32	0.33	0.44
Metric	F1 Score				mAP50			
Data Set	Self	Bots CT & T	Batch 1	Bots, B3, B4	Self	Bots CT & T	Batch 1	Bots, B3, B4
Ours (1280x720)	0.94	0.80	0.66	0.49	0.91	0.76	0.60	0.44
Siromer	0.93	0.66	0.69	0.64	0.95	0.56	0.62	0.61
PyLessons	0.78	0.40	0.36	0.41	0.75	0.32	0.33	0.44
Metric	mAP50_95				Fitness			
Data Set	Self	Bots CT & T	Batch 1	Bots, B3, B4	Self	Bots CT & T	Batch 1	Bots, B3, B4
Ours (1280x720)	0.95	0.76	0.60	0.41	0.78	0.29	0.32	0.21
Siromer	0.97	0.62	0.69	0.63	0.74	0.45	0.46	0.39
PyLessons	0.83	0.31	0.24	0.33	0.53	0.19	0.12	0.18

As seen in Figure 21 and table 9, our model performs noticeably worse on the "Counter-Terrorist" class compared to the "Terrorist" class, as shown in Figure 20. In some instances, it is even outperformed on the "Bots, Batch 3, Batch 4 Combined" dataset. This outcome is also expected because our latest model, which leverages name tags, requires training two separate models: one for when the friendly team is "Terrorists" and another for when the friendly team is "Counter-Terrorists." This approach results in some asymmetry in the detection performance between the two classes. The data shown in the previous two figures was obtained using the model trained when the friendly team was "Terrorists." Results for the model trained when the friendly team was "Counter-Terrorists" showed complementary outcomes, with the "Counter-Terrorist" team being detected more effectively. The differences, however, are minor. We chose to present the results from the "Terrorist" team model because it yielded slightly better results. In future work, it may be more beneficial to average the results across the two team-based models for each class.

Lastly, it is important to emphasize that the most relevant dataset for comparison is the "Bots CT and T Combined," as it most accurately represents competitive environments simulating real matches with bot opponents and teammates. The other datasets introduce more variability in the form of custom player cosmetics, which our generator does not currently support. However, this is unnecessary if our goal is to use our object detector exclusively for bot matches.

A.7 Future Work and Proposed Multi-Model Architecture

Proposed Multi-Model Agent Architecture: We propose a comprehensive multi-model architecture that could serve as the foundation for developing autonomous agents capable of high-level gameplay in Counter-Strike:

(1) **Object Detection for Enemy Players**: This model functions as the agent's eyes, focusing on detecting and classifying enemy players, enabling the agent to focus on shooting accuracy and map traversal.
(2) **Reinforcement Learning for Aiming and Shooting**: Acting as the agent's arms, this model uses reinforcement learning to aim and shoot at detected enemies, adjusting its behavior based on skill levels.
(3) **Object Detection on the In-Game Minimap**: This complementary model identifies player positions on the in-game minimap, providing additional spatial awareness.
(4) **Decision Making for Player Movement**: Utilizing minimap data, this model determines the agent's movement strategy, optimizing its position on the map through supervised or reinforcement learning.
(5) **3D Environment Modeling and Detection**: Enhancing environmental perception, this model employs techniques like SLAM or MiDaS to build a 3D understanding of the game world.
(6) **Dynamic Map Traversal**: Leveraging the 3D environment model, this model navigates the map dynamically, utilizing pathfinding algorithms or reinforcement learning to simulate player inputs.