Realtime Object-Detection In Gaming: Fine-Tuning for Valorant

Liu Chang *
ShanghaiTech University
liuchang2023@shanghaitech.edu.cn

Liu Changyu †
ShanghaiTech University
liuchy2023@shanghaitech.edu.cn

Qiu Zhaolin
ShanghaiTech University
qiuzhl2023@shanghaitech.edu.cn

Abstract

1 Introduction

The confluence of deep learning and interactive entertainment has catalyzed a paradigm shift, unlocking possibilities that were once the domain of science fiction. Video games, now a dominant global cultural and economic force, represent a uniquely challenging and fertile ground for the application of real-time computer vision. The ability to perceive, understand, and react to dynamic, visually complex game environments in milliseconds is a cornerstone of human play. Replicating this capability with artificial intelligence opens a dual-edged sword: it presents an unprecedented threat to competitive integrity while simultaneously offering powerful new tools for accessibility and intelligent agent design.

A significant and pressing challenge emerging from this technological advancement is the rise of vision-based cheating in competitive games, particularly in the First-Person Shooter (FPS) genre. Traditionally, cheating involved direct memory injection or network packet manipulation—methods that could be reliably identified by anti-cheat software scanning for unauthorized code and memory access. However, modern vision-based cheats operate on a fundamentally different principle. By capturing the game's video output—the same visual information a human player sees—and processing it with a convolutional neural network (CNN), these systems can identify and aim at opponents with superhuman precision. Because they interface with the game solely through screen capture and simulated mouse/keyboard inputs, they mimic legitimate human interaction, rendering them exceptionally difficult to detect with conventional anti-cheat mechanisms. This new class of "AI aimbots" poses a profound threat to the fairness and long-term viability of the multi-billion dollar esports and competitive gaming industry.

Conversely, the very same technology holds immense promise for positive transformation. Vision AI is proving to be a powerful enabler of game accessibility, helping to break down barriers for players with disabilities. A recent, award-winning example is the work by NetEase, which utilized vision AI to provide real-time screen narration and touch-based assistance, making complex game interfaces accessible to visually impaired players [1]. This application highlights a path where real-time object and UI element detection can be harnessed to create more inclusive gaming experiences, from automated highlighting of critical game objects to providing auditory cues for environmental threats.

*Group Leader, pipeline implementation, model training, presenter, and paper writer.
†Model training, fine-tuning, and evaluation.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Furthermore, the integration of vision is poised to revolutionize the design of in-game Non-Player Characters (NPCs). For decades, game AI has been largely dependent on brittle and predictable systems like behavior trees and finite state machines, which often rely on privileged access to the game engine's internal state. This leads to enemy behaviors that, while functional, lack the adaptability and emergent complexity of human opponents. In contrast, an AI agent that perceives the game world through vision—akin to a human player—can be trained via reinforcement learning (RL) to develop sophisticated, human-like strategies. Seminal work in complex games like StarCraft II, often driven by the player community, has demonstrated that visually-grounded RL agents can achieve grandmaster-level performance, showcasing a future where NPCs can provide more engaging, challenging, and realistic interactions [2, 3].

Despite this clear duality, a systematic and public investigation into the practical performance of modern object detection models within the real-time constraints of a live game environment is lacking. The core challenge is multifaceted: any viable system, whether for nefarious or beneficial purposes, must achieve high accuracy while maintaining minimal inference latency and a low computational footprint to avoid disrupting the game's performance. This necessitates a move beyond general-purpose benchmarks to a specific analysis of lightweight, efficient architectures operating under the unique visual conditions and temporal demands of gaming.

This paper aims to bridge this gap. We present a comprehensive study focused on implementing and evaluating a real-time object detection pipeline for video games. Our work is guided by the need to understand the capabilities of vision-based exploits while simultaneously laying the groundwork for positive innovation. Our primary contributions are as follows:

1. **Feasibility Analysis of Vision-Based Cheating**: We develop and empirically evaluate a real-time object detection pipeline tailored for FPS games. By fine-tuning and deploying this system, we provide a quantitative baseline for the performance and viability of vision-based automated targeting, offering crucial data for the anti-cheat research community.
2. **Benchmarking for Real-Time Performance**: We conduct a comparative analysis of several state-of-the-art, lightweight object detection architectures (e.g., YOLOv8, FastSAM, RT-DETR). We systematically benchmark their performance on custom-annotated game footage, focusing on the critical trade-offs between mean Average Precision (mAP), inference speed (FPS), and resource utilization on consumer-grade hardware.
3. **A Foundation for Positive Applications**: We detail a streamlined methodology for data collection, annotation, and model fine-tuning. The resulting models and robust pipeline serve as a valuable reference and an open-source foundation for researchers and developers aiming to build assistive technologies for accessibility or to create the next generation of intelligent, vision-driven RL agents. Through this work, we provide a foundational study on the practical application of computer vision in modern gaming, addressing its most immediate challenges and illuminating its most promising opportunities.

2 Related Work

2.1 Object Detection and Segmentation Models

YOLOv8 [4] is one of the most popular object detection models of the YOLO (You Only Look Once) family, known for its speed and accuracy. It employs a single-stage architecture that predicts bounding boxes and class probabilities directly from full images in one evaluation, making it suitable for real-time applications. YOLOv8 introduces several improvements over its predecessors, including better feature extraction through CSPNet, enhanced anchor-free detection, and improved training techniques that allow it to achieve high mAP while maintaining low inference latency.

FastSAM [5] is a lightweight segmentation model designed for real-time applications. It builds upon the principles of SAM (Segment Anything Model) but optimizes for speed and efficiency, making it suitable for scenarios where computational resources are limited. FastSAM uses a simplified architecture that reduces the number of parameters and computational overhead while maintaining competitive segmentation performance. Its design allows for quick inference times, making it ideal for applications requiring rapid object detection and segmentation.

RT-DETR [6] (Real-Time DEtection TRansformer) is a real-time object detection model that adapts the DETR (DEtection TRansformer) architecture for faster inference. DETR revolutionized object detection by treating it as a direct set prediction problem, using transformers to model relationships between objects in an image. RT-DETR modifies this approach to improve speed, making it feasible for real-time applications. It achieves this by optimizing the transformer architecture and reducing the computational complexity, allowing for efficient processing of images while maintaining high accuracy.

2.2 Anti-Cheat Methods

Traditional Anti-Cheat Methods There are three main categories of traditional anti-cheat methods: player client detection, game network communication detection, and game remote server detection. However, these methods are often limited by the need for privileged access to the game client or server, making them ineffective against vision-based cheats that operate solely through screen capture and simulated inputs.

Deep Learning for Anti-Cheat To combat vision-based cheats, researchers have explored the use of deep learning techniques for cheat detection. For instance, Spijkerman and Marie Ehlers [7] applied Support Vector Machines (SVMs), decision trees, and Naive Bayes machine learning models to analyze players' mouse and keyboard operations. By integrating learning features into SVM models, they achieved superior cheating detection results. However, relying solely on SVMs for cheating detection may overlook crucial information such as players' aiming frequency, hit rates, and pre-aim positions.

3 Method

3.1 Data Collection and Annotation

We collected a dataset of game *Valorant* from Santyasa dataset from Roboflow [8], which contains 3771 images of various game scenes and annotations of enemy heads and bodies. The annotations are in the form of bounding boxes, which fit the COCO format. The dataset is split into training, validation, and test sets, with 80% for training, 10% for validation, and 10% for testing.

Also, we collected a corner case dataset of 100 images from gaming records, where the players are at tricky angles or positions, such as the edge of the screen, behind obstacles or inside a smoke grenade. This dataset is used to test the robustness of the model in real-time scenarios.

3.2 Training

As mentioned, we will fine-tune three different object detection models: YOLOv8, FastSAM, and RT-DETR. The training process involves the following steps:

Loading the Dataset and Models We load the training and validation datasets using our custom dataset loader. Then we load the pre-trained weights of the models we want to fine-tune. The pre-trained weights are obtained from the ultralytics repository.

Training with Parameters We adjust the training parameters such as learning rate, batch size, number of epochs, and optimizer. The learning rate is set to 0.01, the batch size is set to 16, and the number of epochs is set to 100. We use the Auto Optimizer from the ultralytics repository, which automatically selects the best optimizer for the model. The training results can be seen in Section 6.

Loss Function The default loss function for YOLOv8 is

$$\mathcal{L}_{YOLO} = \alpha_{box}\mathcal{L}_{box} + \alpha_{cls}\mathcal{L}_{cls} + \alpha_{dfl}\mathcal{L}_{dfl}$$

where $\mathcal{L}_{box}$ is the bounding box regression loss, $\mathcal{L}_{cls}$ is the classification loss, and $\mathcal{L}_{dfl}$ is the distribution focal loss. The hyperparameters α_{box}, α_{cls}, and α_{dfl} are set to 9.0, 0.3, and 1.8 respectively.

3.3 Model Fine-Tuning

Transfer Learning with Frozen Layers We apply transfer learning by freezing the initial layers of the models and only training the last few layers. This allows us to leverage the pre-trained weights while adapting the models to our specific dataset. The frozen layers are determined based on the model architectures. For YOLOv8, we freeze the first 10 backbone and feature extract layers, while for FastSAM and RT-DETR, we freeze the first 5 backbone layers. This helps to retain the general features learned from the pre-trained models while fine-tuning them with a faster convergence.

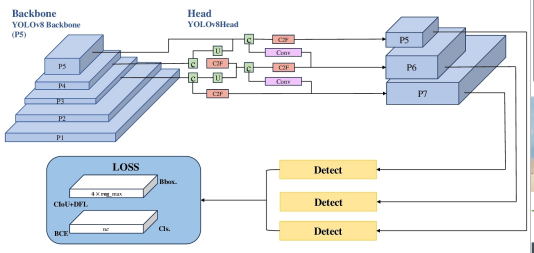

Figure 1: Architecture of YOLOv8. We freeze the backbone layers only.

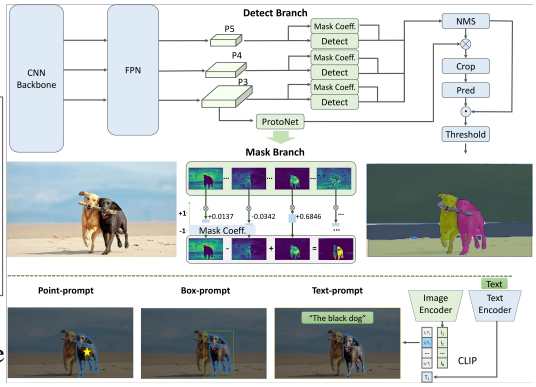

Figure 2: Architecture of FastSAM. We freeze the backbone layers only.

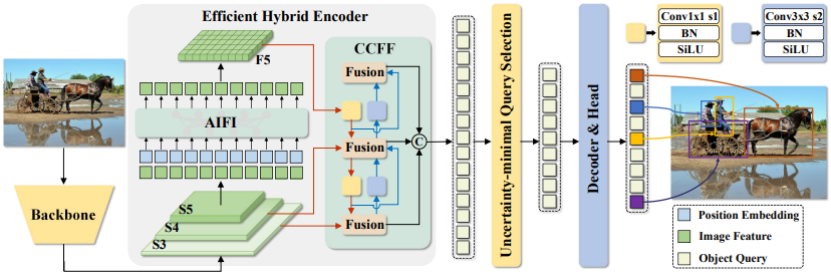

Figure 3: Architecture of RT-DETR. We freeze the backbone layers only.

Figure 4: Three images arranged horizontally.

3.4 Dataset Augmentation

To enhance the robustness of our models, we apply various data augmentation techniques during training. These include:

- **Random Rotation**: Randomly rotating the images in $[-15°, 15°]$ to simulate different viewing angles.
- **Noise**: Adding random Gaussian noise to the images to simulate different lighting conditions and camera quality.
- **Brightness and Contrast Adjustment**: Randomly adjusting the brightness and contrast of the images in $[0.85, 1.15]$ range to simulate different lighting conditions.

Therefore, the total number of training images is increased to 5934.

3.5 Model Evaluation

The metrics used to evaluate the performance of the models are:

- **Precision**: This metric measures the accuracy of the model:

$$\text{Precision} = \frac{TP}{TP + FP}$$

where TP is the number of true positives (correctly detected objects), and FP is the number of false positives (incorrectly detected objects).

- **Recall**: This metric measures the ability of the model to find all the relevant objects. The way to calculate recall is

$$\text{Recall} = \frac{TP}{TP + FN}$$

where FN is the number of false negatives (missed objects). Recall indicates how many of the relevant objects were actually detected.

- **F1 Score**: This metric combines precision and recall into a single score. The way to calculate F1 score is

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1 Score provides a balance between precision and recall, especially when the class distribution is imbalanced.

- **Mean Average Precision (mAP)**: This metric measures the accuracy of the model in detecting objects. It is calculated as the average precision across all classes, taking into account both precision and recall.

3.6 Real-Time Inference Pipeline

Different Input Sources The pipeline supports different input sources, including video files, images, and live window capture. Use the command argument '–input_method' to specify the input source.

Different Model Types The pipeline supports different model types, including YOLOv8, FastSAM, and RT-DETR. Use the command argument '–model_type' to specify the model type.

Prune The pipeline allows for pruning the model to reduce its size and improve inference speed. Using command argument '–prune_percent=0.3' will prune 30% of the model's parameters.

Results See results in Section 6

4 Experiments

4.1 Experimental Equipment

The experiments were conducted both on a cluster of NVIDIA 3090 GPUs and on a PC with a NVIDIA RTX 4090 GPU.

4.2 Tests and Results

We conducted a series of tests to evaluate the performance of the models from their baseline to the fine-tuned version. The tests were conducted on the same dataset, and the results are shown in Table 1. The * indicates the frozen layers of the model.

Then we add our corner case dataset to the training set and fine-tune the original models with the same parameters again. We evaluate the performance on the corner case test set, and compairing with the best models shown in the Table 1 above, we can see that the performance of the models trained on mixed dataset is significantly improved. The results are shown in Table 2. The * indicates the models are trained with the corner case dataset.

Model	Epochs	Train Time(s)	#Params	Precision	Recall	F1-score	AP^{50}	AP^{50-95}
YOLOv8	0	0	6.9M	0.001	0.041	0.0019	0.0004	0
YOLOv8	100		6.9M					
YOLOv8*	100	8904.2	6.9M	0.7664	0.5825	0.6706	0.6478	0.3160
FastSAM	0	0	72.2M	0.0002	0.052	0.0004	0.0001	0
FastSAM	100	8750.78	72.2M	0.7799	0.7991	0.7894	0.8620	0.4112
FastSAM*	100	8050.18	72.2M	0.8667	0.5657	0.6846	0.8548	0.3950
RT-DETR	0	0	45M	0.173	0.168	0.171	0.095	0.046
RT-DETR	100	10284.6	45M	0.619	0.483	0.543	0.465	0.1829
RT-DETR*	100	9700.3	45M	0.667	0.360	0.467	0.375	0.170

Table 1: Performance comparison of different models.

Model	Epochs	Precision	Recall	F1-score	AP^{50}	AP^{50-95}
YOLOv8	100					
YOLOv8*	100					
FastSAM	100	0.011	0.030	0.016	0.0071	0.0036
FastSAM*	100	0.829	0.451	0.584	0.497	0.206
RT-DETR	100	0.0005	0.0627	9.99e-04	0.0004	5.29e-05
RT-DETR*	100	0.552	0.539	0.545	0.476	0.253

Table 2: Performance comparison of different models.

5 Conclusion

6 Appendix

6.1 Training Results

References

[1] NetEase Games. Ai eyes: Enhancing visual accessibility using ai, 2024. Winner at iF Design Award 2024.

[2] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[3] Yanyan Li, Yijun Wang, and Yiwei Zhou. Multiagent deep reinforcement learning algorithms in starcraft ii: A review. *IEEE Access*, 2024.

[4] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. Real-time flying object detection with yolov8, 2024.

[5] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything, 2023.

[6] Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. Detrs beat yolos on real-time object detection, 2023.

[7] Ruan Spijkerman and Elizabeth Marie Ehlers. Cheat detection in a multiplayer first-person shooter using artificial intelligence tools. In *Proceedings of the 2020 3rd International Conference on Computational Intelligence and Intelligent Systems*, CIIS '20, page 87–92, New York, NY, USA, 2021. Association for Computing Machinery.

[8] Roboflow Community. Santyasa dataset, 2021.