# Introduction

Long-range shooting in games with realistic bullet drop (like the game “Rust”) is challenging for two main reasons: finding distant targets in cluttered environments, and estimating the vertical aim offset needed to account for gravity’s effect on the projectile. As an experienced Rust player, I wanted to build a tool that turns my gameplay into useful data and makes aiming more consistent and less error-prone. My approach was to develop an AI-powered crosshair overlay that learns to compensate for bullet drop by combining (1) in-game weapon range statistics and (2) visual feedback via enemy detection and range estimation from gameplay videos, all in an end-to-end learning setup.

# Data

**Collection:** I downloaded ~10 hours of Rust gameplay from YouTube, focusing on videos where weapons and targets are clearly visible.

**Preprocessing**: Using FFmpeg, I split each video into screenshot. I then manually curated frames:

**Gun vs No-Gun**: All frames with any weapon became positive examples; I sampled random frames without guns for negatives.

**Gun Classification**: I sorted weapon frames into folders by gun type (AK, Bolt, MP5, etc.).

**Enemy Detection**: I selected clips with large player groups to get varied poses and scales. Annotations were done in Roboflow, drawing bounding boxes around character models.

## Challenges:

Time-intensive labelling, manually reviewing tens of thousands of frames took ≈15 hours.

Data balance, some rare weapons lacked enough examples.

Small-object detection, enemies at long range occupy only a few pixels, making annotation and learning harder. ​

# Model & Methods

**Framework**: All models use Ultralytics’ YOLOv11 architecture, choosing the “small” (s) variant for a balance of speed and accuracy. I compared v11s vs v11n and found s only slightly slower but notably more accurate; the medium (m) was overkill for real-time goals. ​

**Binary Gun Detector**: A classification model trained on “gun” vs “no gun” frames.

**Gun Type Classifier**: A multi-class classification model that takes frames with guns and predicts one of ~80 % of Rust’s guns (I focused on the most common ones).

**Enemy Detector**: An object detection model trained to localize player characters via bounding boxes.

**Integration**:

Screen Capture: mss grabs the primary monitor at ~15 FPS.

Detection Pipeline (in a PyQt5 QThread):

Run binary detector on a downsampled frame; if positive, run gun classifier to get weapon type → lookup its max range.

Run enemy detector on full-res frame to get bounding boxes → estimate distance from apparent box size.

Crosshair Overlay: A transparent PyQt5 window draws detected boxes, weapon info, and a center crosshair whose vertical offset would be computed from the weapon’s known ballistics and the detected target range. ​

# Results & Evaluation

Gun Detectors: Both binary and multi-class models reached ~85 % accuracy on a held-out test set. In real time its much worse, the main errors were:

False negatives in binary detection(When you are running or looking around in the game only a small portion of the gun is visible witch made it predict there was no gun).

Misclassification of visually similar weapons (e.g. AK vs LR).

Misclassification in low light.

Misclassification of small pistols.

Enemy Detector: Works almost perfectly on enemies at short range, but almost never at very long ranges. Also gives false positives at long ranges.

Range Prediction: I did not implement a range-predictor module for the enemy detector due to time constraints.

### End-to-End Overlay:

Weapon info displays in real time with the range of the weapon.

At this point it just is a stationary crosshair, but with the enemy detector and the display of the weapons range it’s already a little bit easier to calculate yourself for the aiming offset.

Key Findings:

Data volume and variety critically affect small-object detection at distance.

YOLOv11 s strikes a good speed/accuracy trade-off for a real-time overlay.

Manual annotation and data collection/Preprocessing remains the bottleneck.

# Contributions:

* I worked on this project entirely on my own.
* I picked YOLOv11-s after reading it was a good balance of speed and accuracy for detecting weapons.
* I found a Google Sheet, listing all Rust weapons and their ranges (https://docs.google.com/spreadsheets/d/1kh9MV6nFnSOe8OFS3IqdH1qVlI6ZG9uzoM5mhmivgZE/edit?gid=0#gid=0) and used it in my code so the overlay shows the correct range for each gun.
* I used ChatGPT to brainstorm different ways to code overlays, troubleshoot code errors, and get suggestions on speeding up the program. Thanks to its tips, the overlay now runs smoothly with less lag.

# Challenges & Future Work

Data Scaling: Automate frame extraction and annotation via semi-supervised methods.

Distance Estimation:

Future size-based range estimates will fail at extreme distances because it cant detect enemies at those ranges and calculate there size for a range estimate.

Future work: train a scene-regression model that infers range from full-frame context (trees, buildings, horizon cues).

Real-Time Adjustment:

Integrate computed offsets into the overlay’s crosshair position automatically.