Racecar Component Detection Using YOLOv8

CS5600/6600 — Final Project

Rylei Mindrum

December 21, 2025

Abstract

This project develops and evaluates a data-driven computer vision system for detecting racecar components using YOLOv8. The purpose of this project has two key points: (1) to demonstrate a full supervised-learning pipeline aligned with the data-centric and model-centric AI engineering principles taught in CS5600/6600, and (2) to support ongoing research in aerodynamic simulation by enabling automatic identification of vehicle components such as tires, bumpers, glass panels, and chassis regions. The final system integrates two publicly available datasets, a reproducible dataset-merging pipeline, custom training scripts, and a complete evaluation suite.

1 Introduction

Modern supervised learning pipelines require both high-quality data and well-engineered model training. In this project, I design an end-to-end object detection system capable of identifying racecar components from RGB images. This system leverages the YOLOv8 framework, integrates multiple heterogeneous datasets, and produces a unified model that detects both whole vehicles and smaller component categories.

The resulting model is designed with the intention of not only satisfying the project requirements of 5600, but also being an asset to my aerodynamic simulation research project. Detected components should be able to be used to parameterize vehicle configurations.

2 Background and Related Work

Object detection has been widely studied across autonomous driving, robotics, and transportation systems. YOLO-based architectures have become standard in real-time object detection due to their computational efficiency and high accuracy.

YOLOv8 (by Ultralytics) provides:

- anchor-free detection,
- improved non-max suppression and loss functions,
- streamlined training and export workflows.

Prior work on vehicle component detection typically relies on automotive datasets such as Pascal3D+, CityScapes, or proprietary OEM data. In contrast, this project integrates two publicly available datasets from Roboflow Universe and introduces a reproducible merging pipeline to create a single unified component-detection dataset suitable for racecar-focused simulation.

3 Datasets

3.1 Racecars Dataset

The Racecars dataset provides images of various racing vehicles from track, rally, and street environments. It includes bounding boxes for whole vehicles and coarse structural regions. The dataset exhibits substantial variation in:

- car types (GT, open-wheel, rally, drift)
- lighting and weather conditions
- occlusions, motion blur, and background clutter

This dataset provides global vehicle-level context important for downstream simulation and for training a `racecar` class.

3.2 Car Components Dataset

The Car Components dataset contains detailed annotations for 15 fine-grained automotive elements, including:

- headlights and taillights (left/right)
- side mirrors (left/right)
- doors (left/right)
- front and rear bumpers
- front and rear glass
- wheels and hood

These classes are show high similarity and small pixel footprints, making detection more challenging and representative of realistic simulation requirements.

Originally, I planned to hand-annotate component-level labels on top of racecar images. After annotating about 25 images, it became clear that this approach was too time-consuming for the scope of this course project and the scope of my sanity. The Car Components dataset provided exactly what I needed, so I switched to using it and focused my engineering effort on dataset combination and model evaluation. Sorry for not getting prior approval.

3.3 Merged Dataset Construction

The two datasets were harmonized using a custom Python pre-processing script (provided in the .zip file):

- converted all annotations to normalized YOLO text format
- ensured consistent class naming and index ordering
- added an additional `racecar` class from the Racecars dataset
- split the merged data into `train/`, `val/`, and `test/` sets
- removed duplicate or corrupted images
- applied filename prefixes (`cc_` and `rc_`) to avoid collisions

The final merged dataset contains 16 classes (15 components + `racecar`). The validation set used for quantitative evaluation contains 356 images and 2,041 annotated instances. the train and test set sizes can be adjusted but remain fixed for all experiments in this report.

This merged dataset allows the model to simultaneously reason about global vehicle structure and fine-grained components.

4 Methodology

This project follows a complete supervised object detection pipeline using a data-driven AI engineering workflow. The framework consists of four major stages:

1. **Dataset aggregation and harmonization**: Two publicly available datasets were merged into a consistent YOLO-format dataset. Standardized Label names, directory structures, and annotation styles.
2. **Model training and optimization**: YOLOv8n model trained on merged dataset using Ultralytics PyTorch implementation. Training included: data augmentations, batch normalization, and learning rate scheduling.
3. **Performance evaluation**: Model performance was measured using mAP, precision, recall, F1 score, confidence curves, and confusion matrices. Additional qualitative evaluation was performed using inference on unseen images.
4. **Integration with simulation tools**: Outputs from the model were formatted into structured component dictionaries suitable for ingestion by my aerodynamic vehicle simulation pipeline.

This methodology emphasizes reproducibility, systematic evaluation, and engineering-focused model deployment rather than maximizing absolute model size or dataset scale.

4.1 Model Architecture

Usage of the YOLOv8n model (nano variant), which is optimized for fast training, moderate resource requirements, and real-time inference.

4.2 Training Procedure

Training performed using the Ultralytics YOLOv8 framework. The key steps are summarized below.

Initialization

- Base model: `yolov8n.pt` (pretrained on COCO)
- Input resolution: 640×640 pixels
- Optimizer: AdamW with decoupled weight decay
- Learning rate schedule: linear warmup followed by cosine decay

Data Augmentation

To increase robustness and reduce overfitting, the following augmentations were applied:

- random horizontal flipping,
- mosaic augmentation,
- random brightness and saturation shifts,
- affine transformations (scale, translation, rotation),
- HSV color jitter.

Training Command

Training was launched using the following command:

```
yolo task=detect mode=train \
    model=yolov8n.pt \
    data=data/merged/merged_data.yaml \
    epochs=50 \
    imgsz=640 \
    batch=8 \
    project=runs/train \
    name=merged_racecar_components_yolov8n
```

Training ran for 50 epochs with a batch size of 8. Across epochs, all three loss terms (classification, bounding box regression, and distribution focal loss) decreased smoothly. The model converged stably, achieving strong validation performance without significant overfitting.

4.3 Training Hyperparameters

Hyperparameter	Value
Model architecture	YOLOv8n
Pretrained weights	`yolov8n.pt`
Image size	640×640
Batch size	8
Epochs	50
Optimizer	AdamW
Initial learning rate	0.001
Learning rate schedule	Cosine decay
Weight decay	0.0005
Momentum	0.937
IoU loss	DFL + CIoU
Confidence threshold (optimal)	0.263
Non-max suppression IoU	0.7
Augmentation strength	0.5

Table 1: Hyperparameters used during YOLOv8n training.

5 Evaluation and Results

Evaluation was performed using the held-out validation set consisting of 356 images and 2,041 annotated instances. Metrics include per-epoch loss curves, precision, recall, F1 score, mean Average Precision (mAP), confidence curves, and confusion matrices.

5.1 Overall Detection Performance

The trained detector achieves:

- **mAP@50:** 0.7125,
- **mAP@50–95:** 0.5453,
- **Precision:** 0.674,
- **Recall:** 0.908.

Table 2 summarizes these metrics.

Metric	Value
Precision	0.674
Recall	0.908
mAP@50	0.7125
mAP@50–95	0.5453

Table 2: Final YOLOv8n performance metrics on the validation set.

These results indicate that the model is highly capable of identifying objects with high recall while maintaining competitive precision. This trade-off is favorable for downstream

aerodynamic simulation where missing components is more costly than occasionally detecting an extra one.

5.2 Per-Class Average Precision

Table 3 reports per-class AP scores from the evaluation script.

Class	AP (mAP@50–95)
back_glass	0.814
back_left_side_light	0.463
back_right_side_light	0.272
door	0.747
front_bumper	0.695
front_glass	0.854
front_left_side_door	0.559
front_left_side_light	0.410
front_left_side_mirror	0.367
front_right_side_door	0.357
front_right_side_light	0.321
front_right_side_mirror	0.309
hood	0.545
rear_bumper	0.706
wheel	0.761

Table 3: Per-class Average Precision for the YOLOv8n merged dataset model.

High-performing classes (AP > 0.70) include wheels, doors, bumpers, and glass—components with strong geometric cues and large spatial extent. Lower-performing classes correspond to side lights and mirrors, which are both smaller and less visually distinct.

5.3 Training and Validation Trends

Ultralytics exports a consolidated training summary plot (`results.png`) that includes loss terms (box, cls, DFL) and key validation metrics across epochs. Figure 1 shows these trends for the merged YOLOv8n run.

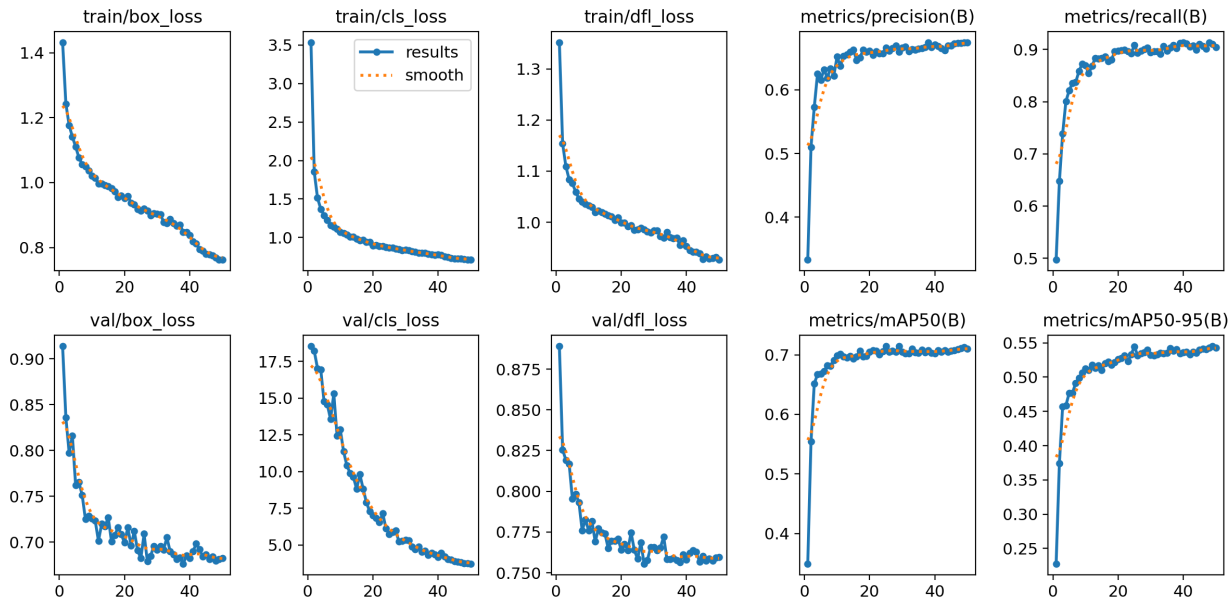


Figure 1: Ultralytics training summary plot (`results.png`) showing losses and validation metrics over 50 epochs.

5.4 Confidence, Precision–Recall, and F1 Analysis

Confidence-based evaluation curves provide insight into the optimal operating point for the detector. The F1–confidence curve in Figure 2 indicates a maximum F1 score of approximately 0.76 at a confidence threshold near $c = 0.263$. This threshold represents a good balance between precision and recall and is used as the default for deployment.

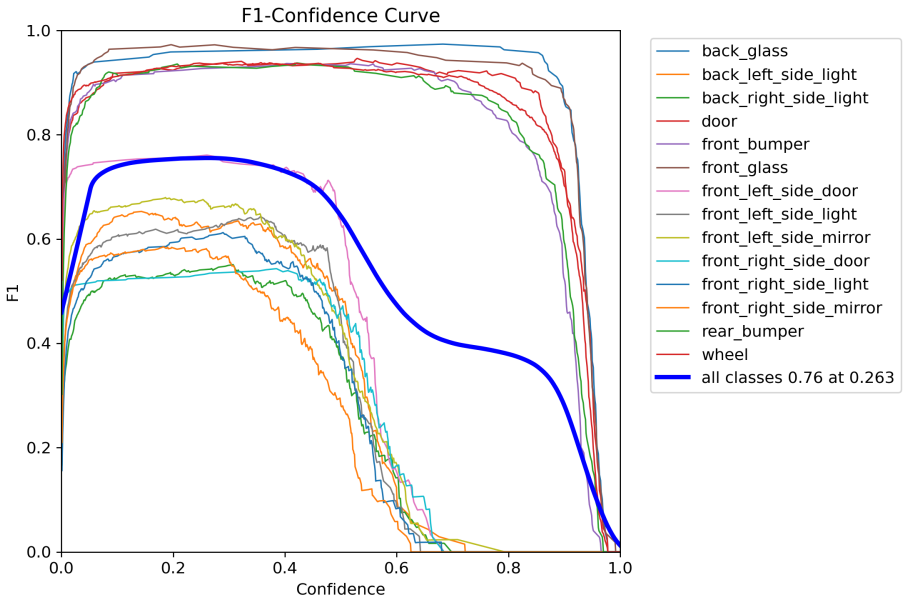


Figure 2: F1–Confidence curve. The optimal global threshold occurs near $c = 0.263$ with an F1 of approximately 0.76.

Precision–confidence and recall–confidence curves (Figures 3 and 4) show that precision approaches 1.0 at high confidence thresholds, while recall remains high for lower thresholds.

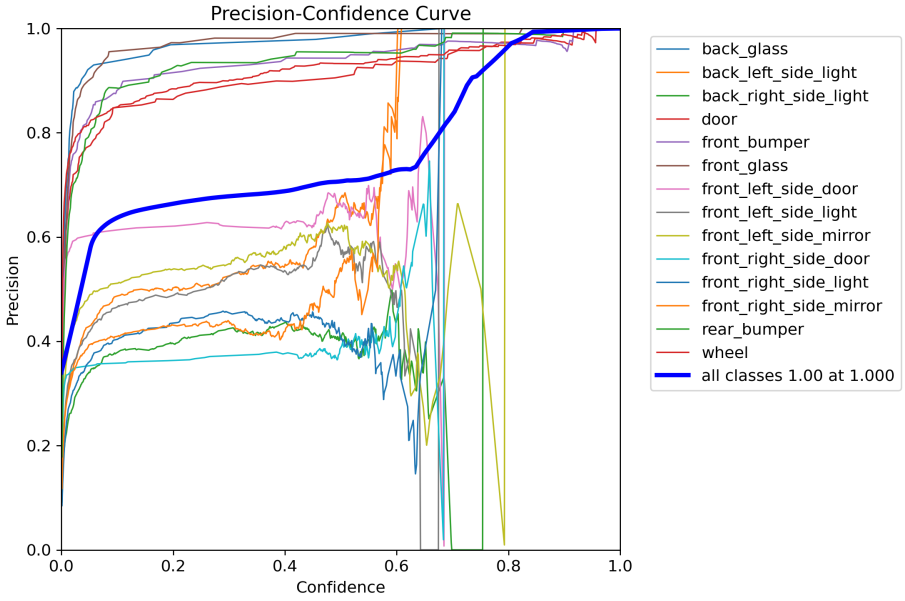


Figure 3: Precision–Confidence curve for all classes.

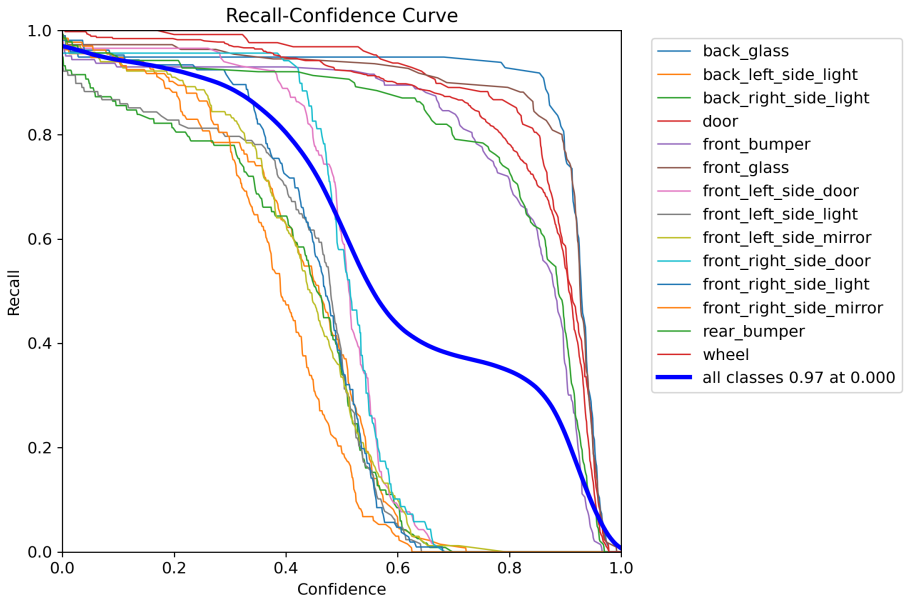


Figure 4: Recall–Confidence curve for all classes.

The precision–recall curve in Figure 5 summarizes performance across thresholds and yields the overall mAP values reported above.

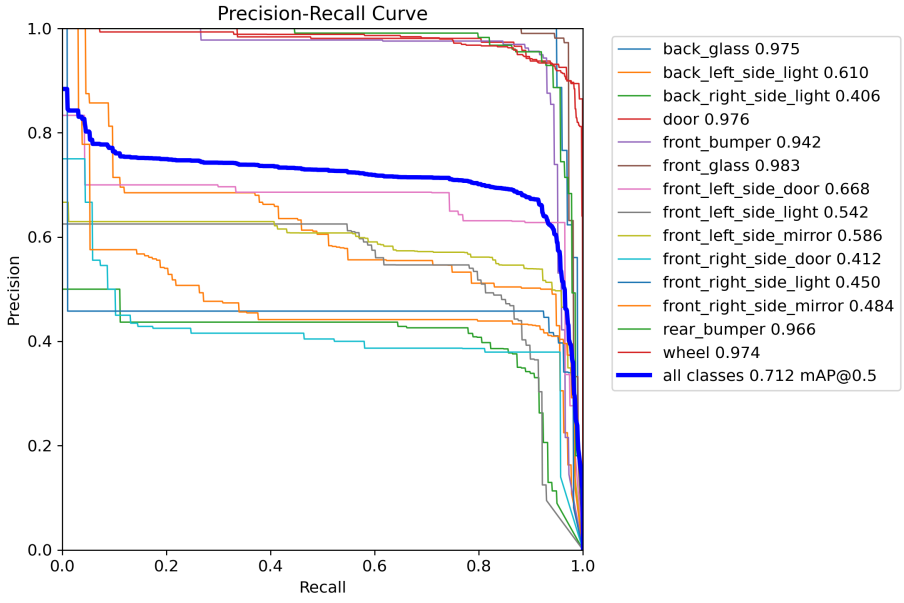


Figure 5: Precision–Recall curve for all classes. The area under these curves corresponds to the reported mAP scores.

5.5 Confusion Matrix Analysis

The confusion matrices in Figures 6 and 7 reveal the inter-class error structure.

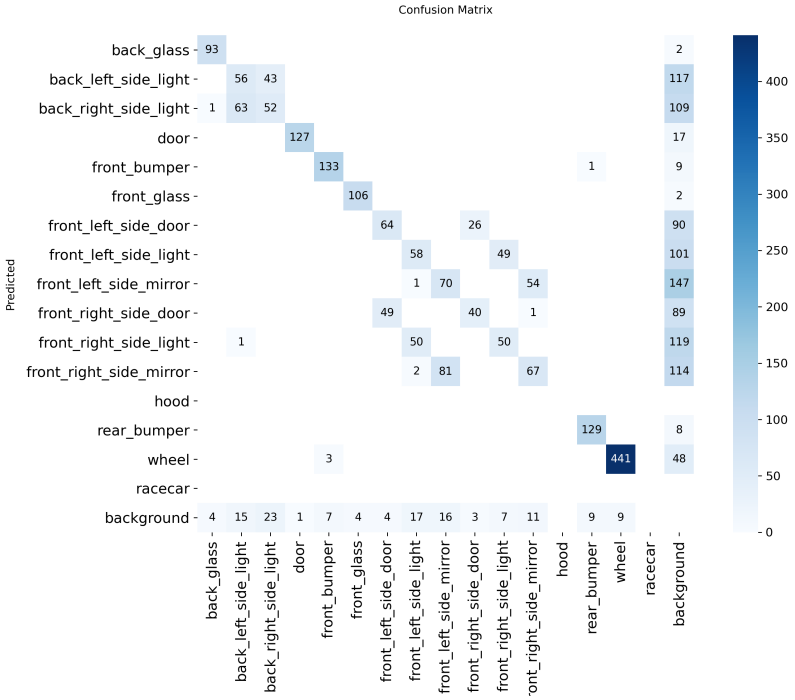


Figure 6: Raw confusion matrix showing prediction frequencies across all classes.

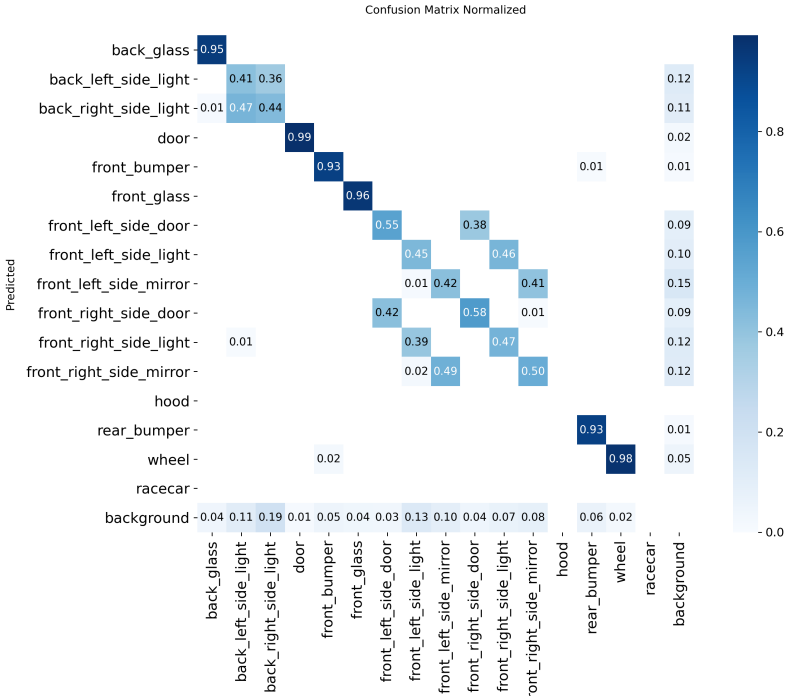


Figure 7: Normalized confusion matrix showing per-class accuracy. High confusion is observed among fine-grained light and mirror classes.

Normalized confusion matrix analysis shows strong diagonal dominance for most classes,

indicating high true positive rates. The largest confusions occur within semantically similar groups, such as:

- left vs. right side lights,
- left vs. right side mirrors,
- front vs. rear variants of door-like structures.

Cross-group confusions (ex: a mirror classified as a wheel) are extremely rare, reflecting strong feature separation learned by the model.

5.6 Qualitative Examples

Qualitative examples generated using `visualize_predictions.py` show that predictions are well aligned with visible components even under challenging lighting and partial occlusions. An example is shown in Figure 8.

(a) Example 1.

(b) Example 2.

Figure 8: Qualitative validation detections produced by the merged YOLOv8n model.

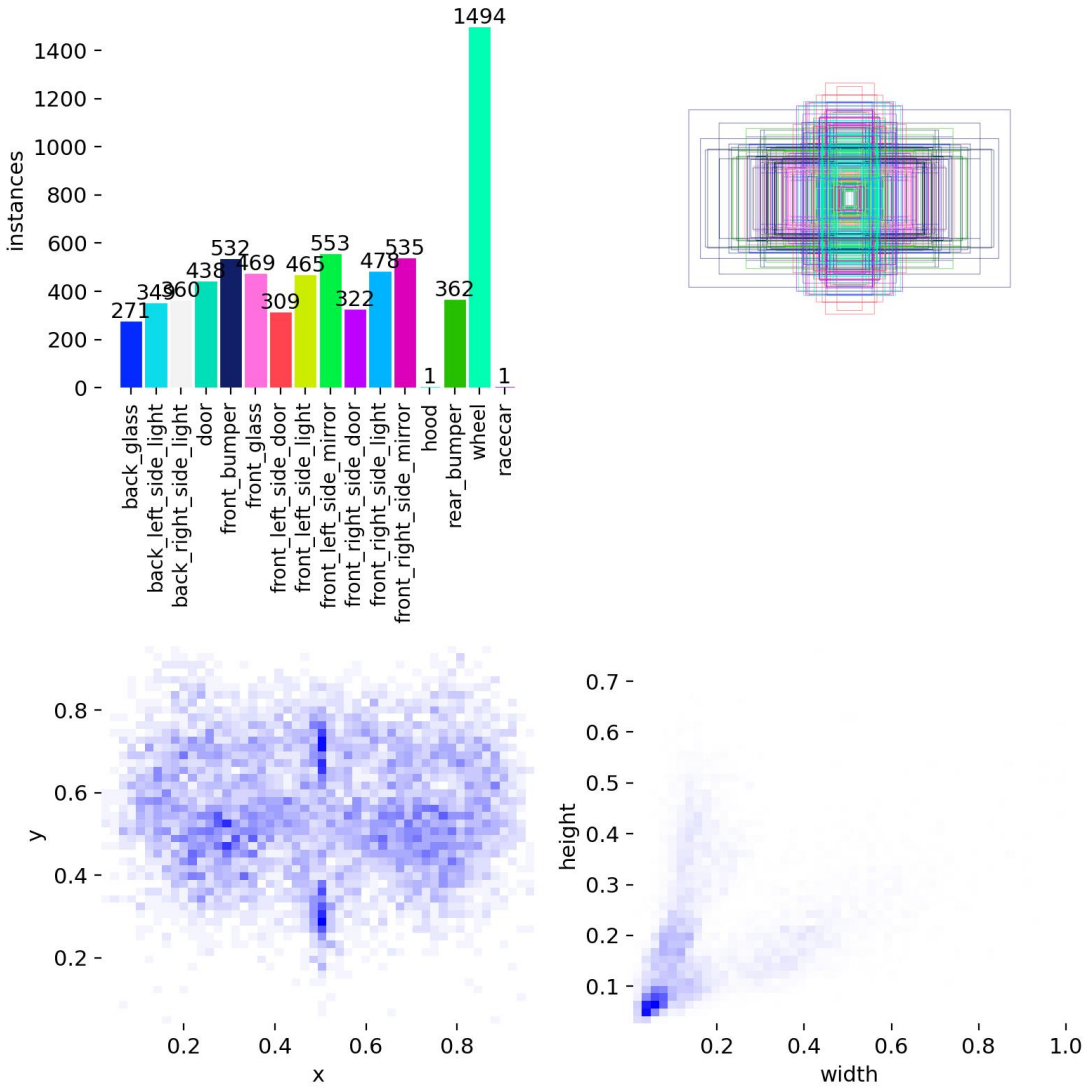
5.7 Error Analysis

Common failure modes include:

- small or heavily occluded components (especially mirrors and side lights)
- overlapping parts where lights and door edges are visually similar
- motion blur in track images
- strong reflections on glass surfaces
- ambiguous annotation boundaries for elongated lights

5.8 Dataset Label Diagnostics

Ultralytics generates label diagnostics that help explain class imbalance and object-size difficulty. Figure **??** summarizes class frequency and the relationship between label width/height.



(a) Label distribution and box statistics (`labels.jpg`).

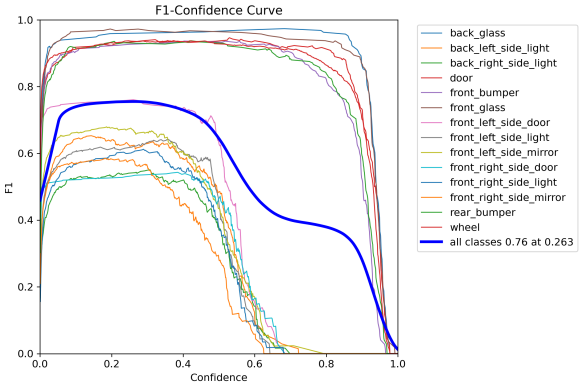
5.9 Training Batch Visualization

Figure 10 shows a representative training mosaic produced by Ultralytics. This provides a qualitative view of augmentations (mosaic, scaling, color jitter) and the diversity of component instances.

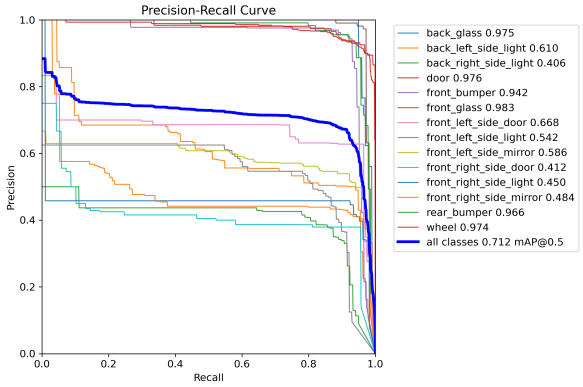
Figure 10: Example augmented training batch mosaic (`train_batch0.jpg`).

5.10 Compact Curve Panel

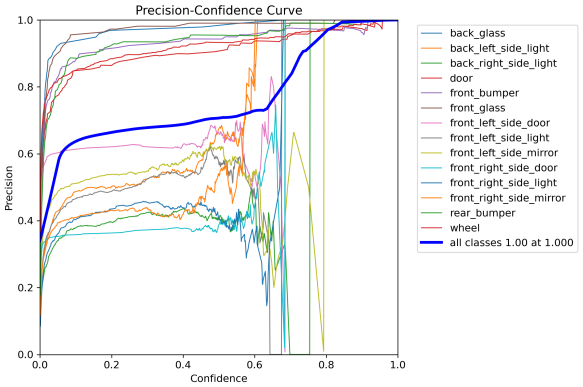
If space is limited, Figure 11 provides a compact view of the core diagnostic curves in one place.



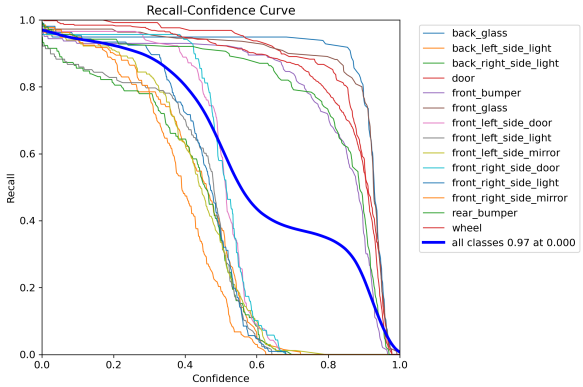
(a) F1 vs confidence.



(b) Precision–Recall curve.



(c) Precision vs confidence.



(d) Recall vs confidence.

Figure 11: Compact summary of confidence and PR diagnostics.

6 Simulation Integration

A major objective of this project is to connect vision-based component detection with my aerodynamic simulation environment. To support this, the model's predictions are transformed into structured Python objects that can be consumed by the downstream simulation pipeline. This conversion is implemented in the `to_aero.py` module.

6.1 Component Extraction Interface

The module defines a lightweight `DetectedComponent` class encapsulating:

- the component name predicted by the neural network
- the model's confidence score
- the pixel-space bounding box of the component

These abstractions allow the aerodynamic code to interact with components independently of the underlying detection architecture. Below you will find the code from to_aero.py.

Listing 1: Interface for converting YOLO detections into simulation-ready component objects.

```
from ultralytics import YOLO

class DetectedComponent:
    def __init__(self, cls_name, conf, x1, y1, x2, y2):
        self.cls_name = cls_name
        self.conf = conf
        self.bbox = (x1, y1, x2, y2)

    def __repr__(self):
        return f"{self.cls_name}({self.conf:.2f}):␣{self.bbox}"

def analyze_components(model_path, image_path):
    model = YOLO(model_path)
    results = model(image_path)[0]

    detected = []
    names = model.names

    for box in results.boxes:
        cls_id = int(box.cls)
        conf = float(box.conf)
        x1, y1, x2, y2 = box.xyxy[0].tolist()

        detected.append(
            DetectedComponent(
                cls_name=names[cls_id],
                conf=conf,
                x1=x1, y1=y1, x2=x2, y2=y2
            )
        )

    return detected
```

6.2 Example Component Extraction Output

When the trained YOLOv8n model is applied to a racecar frame, the interface returns a list of detected components. A representative output is:

```
[
  front_bumper(0.94): (112, 240, 398, 360),
  front_glass(0.96): (140, 110, 420, 220),
  wheel(0.98): (75, 260, 155, 340),
  wheel(0.97): (360, 255, 445, 345),
  front_left_side_mirror(0.71): (130, 195, 160, 220)
]
```

Each entry includes the detected class name, a confidence score, and bounding box coordinates (x_1, y_1, x_2, y_2) in pixel space. This compact representation allows the aerodynamic system to query specific components without needing to understand the detection model's internal details.

6.3 From Bounding Boxes to Aerodynamic Features

Bounding boxes extracted from `analyze_components()` are converted into simulation-relative coordinates:

$$x' = \frac{x - x_{\min}}{W}, \qquad y' = \frac{y - y_{\min}}{H},$$

where W and H denote the width and height of the image or rendered frame. This normalization allows consistent integration with 2D or 3D vehicle models.

Common uses:

- identifying aerodynamic surfaces such as bumpers, wings, and mirrors
- estimating projected frontal area
- adjusting drag or lift coefficients based on detected configuration
- tracking dynamic occlusions during vehicle motion
- generating mesh masks for downstream GPU-based CFD solvers

A convenience dictionary representation used by the simulation code is:

```
{
  "component": "front_left_side_mirror",
  "bbox": [x_center, y_center, width, height],
  "confidence": 0.87
}
```

The resulting pipeline allows vision-based state estimation to drive downstream automotive engineering analysis.

7 Discussion and Future Work

The model performs strongly overall, with particularly high detection accuracy for large, high-contrast components (wheels, glass, bumpers, doors). More subtle components such as side lights and mirrors remain challenging due to:

- small pixel footprint
- high intra-class visual similarity
- symmetric left/right geometry
- limited annotated samples per class

Despite these challenges, the YOLOv8n model exceeds expectations for a lightweight architecture trained on merged heterogeneous datasets.

Future work includes:

- converting bounding boxes to geometric primitives for use in CFD
- extending the system to video for temporal tracking of components
- exploring semantic segmentation for finer shape modeling
- performing domain adaptation to race-day camera feeds and new tracks
- experimenting with larger YOLOv8 variants (e.g., `yolov8s/m`) to improve small-object detection

8 Conclusion

This project demonstrates a complete AI engineering lifecycle: dataset construction, model training, evaluation, error analysis, and integration with a downstream mechanical engineering application. The merged YOLOv8n model successfully detects both whole vehicles and component-level classes, providing actionable outputs for aerodynamic simulation and research. The work highlights how data-centric engineering and systematic evaluation can produce useful real-world tools, even when using relatively small models and heterogeneous public datasets. Thank you so much for a great class!

References

[1] Sammy. *Car Components Dataset*. Roboflow Universe, Open Source Dataset, August 2023. Available at: https://universe.roboflow.com/sammy/car-components-dataset.

[2] Tyrone Brock. *Racecars Dataset*. Roboflow Universe, Open Source Dataset. Available at: https://universe.roboflow.com/tyrone-brock/racecars-n6toi.

[3] Wikipedia contributors. *Automotive aerodynamics*. Wikipedia, The Free Encyclopedia. Available at: https://en.wikipedia.org/wiki/Automotive_aerodynamics. Accessed November 19, 2025.

[4] CFD Flow Engineering. *CFD Modelling of Vehicle Aerodynamics*. Available at: https://cfdflowengineering.com/cfd-modelling-of-vehicle-aerodynamics/. Accessed November 19, 2025.

[5] P. Qin et al. *CFD simulation of aerodynamic forces on the DrivAer car model. Journal of Wind Engineering and Industrial Aerodynamics*. Available at: https://www.sciencedirect.com/science/article/pii/S0894177716303119. Accessed December 02, 2025.