

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology



أكاديمية كاوست
KAUST ACADEMY

CAMelFree

A standalone edge CAMera for CAMel-Free safe roads

Munthir Aljafar
munthiraljafar@gmail.com

Yazan Almakhamel
almakhamelyazan@gmail.com

Abdulaziz Alsaihati
azizalsaihati@gmail.com

Wesam Alsadah
wesam.alsadah@gmail.com

Hisham Alsadah
hisham.sadah@gmail.com

Mentor: Mojtaba AlShams
mojtaba.alshams@kaust.edu.sa

August 27, 2025

Abstract

Camel–Vehicle Collisions (CVC) are a serious safety and economic concern in Saudi Arabia, with more than 600 cases reported annually and a fatality rate six times higher than other traffic accidents. To address this challenge, we present **CAMelFree**, an edge-based computer vision system designed to detect camels on desert highways and provide early alerts to prevent collisions. The system leverages a YOLOv8n deep neural network. The dataset was constructed in stages: starting with camel-only images from Kaggle and web scraping, then expanded with a COCO subset for generalization, and finally augmented with frames extracted from a camel documentary using a custom script with pseudo-labeling and manual correction in Roboflow. Training was conducted locally on an NVIDIA RTX 4070 GPU using Ultralytics YOLOv8 implementation. The final model achieved strong validation results, including a precision of 0.9781, recall of 0.9637, and mAP@0.5 of 0.9858. Deployment experiments across multiple platforms revealed that laptops achieved real-time performance (average 33.67 FPS, 40.62 ms latency), while Raspberry Pi 3 and 5 devices experienced significant latency (200–300 ms) due to CPU-bound inference. These findings confirm that YOLOv8n provides accurate detection and is deployable on edge devices, though real-time performance on low-power hardware requires further optimization. Future improvements such as quantization, pruning, or hardware accelerators will enhance feasibility for large-scale deployment. CAMelFree demonstrates a scalable, low-cost step toward reducing CVC and improving safety on remote desert highways.

Contents

Abstract **1**

1 Introduction **3**
1.1 Motivation . . . 3
1.2 Team Contributions . . . 3

2 System Overview **3**
2.1 Resources and Tools . . . 5
2.2 Evaluation Procedure . . . 5
2.2.1 Initial Model: Camel-Only Training . . . 5
2.2.2 Improved Model: Combined Dataset . . . 6
2.2.3 Data Augmentation Trials . . . 7
2.2.4 Final Model: Augmented Dataset with Documentary Frames . . . 7
2.3 Evaluation Metrics . . . 8

3 Results Discussion **9**
3.1 Model Training Performance . . . 9
3.1.1 Overview and Metrics . . . 9
3.1.2 Loss Behavior and Convergence . . . 10
3.1.3 Overfitting Assessment . . . 11
3.1.4 Training Setup . . . 12
3.1.5 Comparison to Previous Models . . . 12
3.1.6 Impact of Data Augmentation . . . 12
3.1.7 Interpretation and Limitations . . . 13
3.2 Component-Level Evaluation . . . 13
3.2.1 Raspberry Pi 3 . . . 13
3.2.2 Raspberry Pi 5 . . . 14
3.2.3 Laptop Environments . . . 14
3.3 Comparison to Benchmarks . . . 14
3.4 Identified Bottlenecks and Suggested Improvements . . . 14
3.5 System-Level Evaluation . . . 15

4 Conclusion and Future Work **15**

1 Introduction

1.1 Motivation

In Saudi Arabia, camel–vehicle collisions (CVC) are a major threat on roads, especially intercity desert highways, leading to tragic loss of life and costly damages. More than 600 cases of CVC are reported annually, with a fatality rate of approximately 0.25, six times the rate of all traffic accidents [4] In addition, CVC causes significant economic losses as a result of damage to cars, road infrastructure, and valuable wildlife. Although several studies have explored using computer vision to build camel detection models as a cheap alternative solution, they ultimately face two issues: the harsh remote locations where these models may be deployed and the weather and daytime conditions that may impair model function. Our project, CAMelFree, provides a lightweight model, adapted to any weather or daytime condition (more than 90% of reported CVC occur during nighttime), that can be deployed on small edge devices like a Raspberry PI. This offers a cheap, accurate solution that can easily be scaled and deployed.

1.2 Team Contributions

Throughout the development of the project, our team was divided into the following roles: Munthir and Yazan worked on expanding and curating balanced, high-quality datasets. They then designed and implemented an augmentation pipeline to train the model on low-visibility conditions. Abdulaziz and Hisham worked on training the model using the augmentations and balanced datasets. Then they conducted testing on video and image inputs. They also analyzed failure cases to understand the limitations of the models and documented necessary improvements. Wesam took charge of the hardware aspect of the project by setting up both the Raspberry Pi 5 and the camera module. He also troubleshot SSH and hardware-related issues, and conducted live inference testing.

2 System Overview

The system is a small, locally operated setup that runs strictly on a Raspberry Pi. It detects camels on the road or near it by using a camera to capture live video, which is then processed using a machine learning model on the Raspberry Pi. In order to warn oncoming vehicles and prevent accidents, especially during low visibility or poor weather, when the system detects a camel it activates an alert, such as flashing lights. A visualization of the system can be seen in Fig. 2. Because the entire system is powered by a solar panel, it can function in remote locations without requiring access to the power grid or an Internet connection. Unlike Fig. 1, the actual deployment would not need an Ethernet connection, nor USB receivers and a display connection. All it would need is an efficient power source.

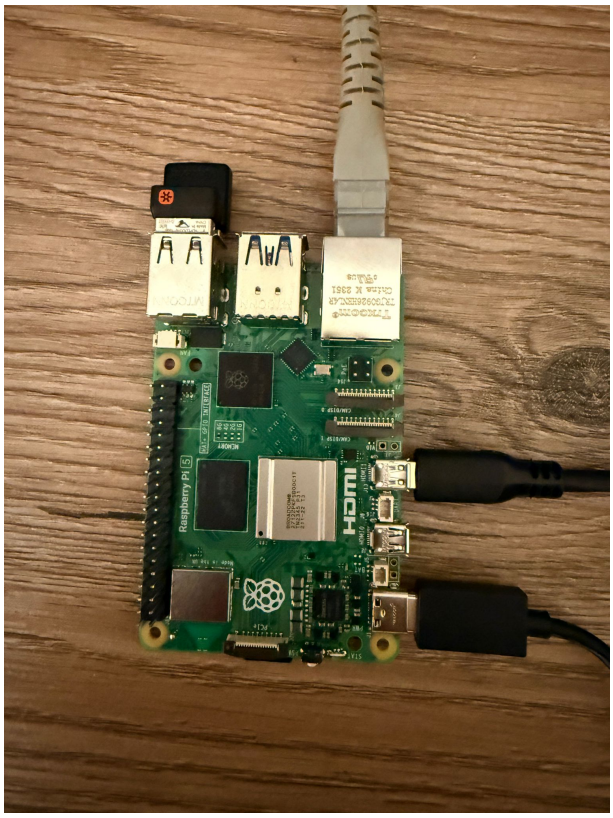


Figure 1: Raspberry Pi 5 board, fully connected to power source, HDMI, Ethernet cable, and keyboard/mouse USB receivers.

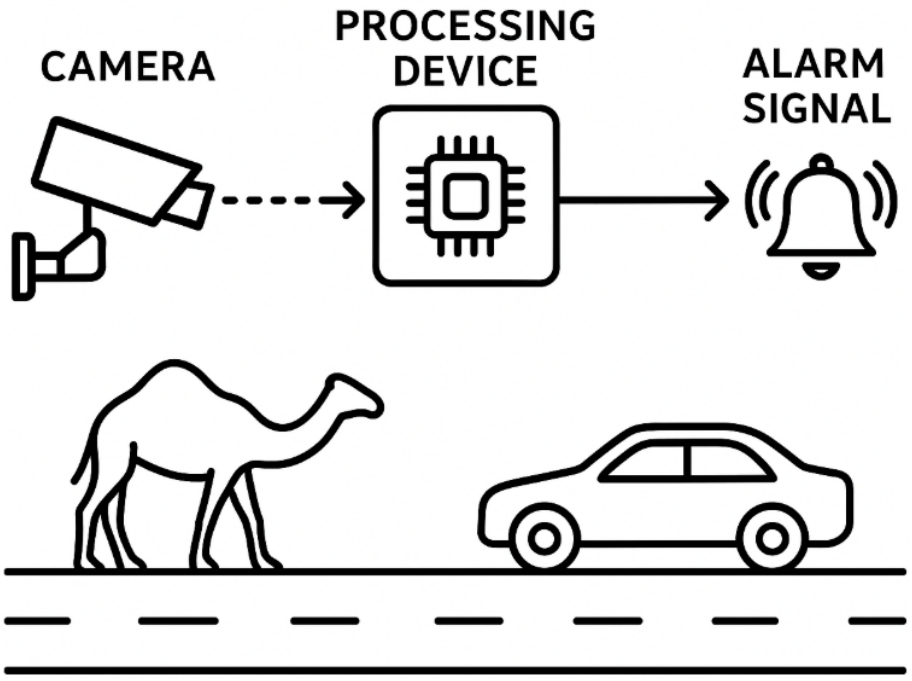


Figure 2: CAMelFree System Overview

2.1 Resources and Tools

- YOLOv8n (Ultralytics, PyTorch backend)
- Hardware: RTX 4070 GPU (local machine)
- Libraries: OpenCV (for frame extraction), Roboflow (for manual labeling), Python scripts
- Frameworks: Ultralytics YOLO CLI

2.2 Evaluation Procedure

2.2.1 Initial Model: Camel-Only Training

The first dataset was constructed using camel images from a Kaggle dataset [1], from which we selected approximately 150 usable images. To expand the dataset, we also developed a Python script that scraped camel images from the Internet. However, many of these images contained heavy watermarking or were otherwise unsuitable, and only around 80 images were retained. Because the scraped images were not in YOLO format, they had to be manually annotated in Roboflow [3]. The two sets were combined to form the initial training dataset, consisting of roughly 230 camel images.

This dataset was used to train the YOLOv8n model. The results appeared promising at first: when tested on webcam input or camel photos, the model was able to correctly identify camels (Figure 3). However, it quickly became clear that the model was severely overfitting. When presented with other objects, such as a cat, the model incorrectly labeled them as camels (Figure 4). This occurred because retraining the model on such a small, single-class dataset effectively overwrote the general object priors from its COCO pretraining. Since the COCO dataset contains hundreds of thousands of diverse images compared to our 200 camel images, directly combining them would have led to class imbalance and weakened camel detection performance. This motivated the development of an improved dataset construction strategy.

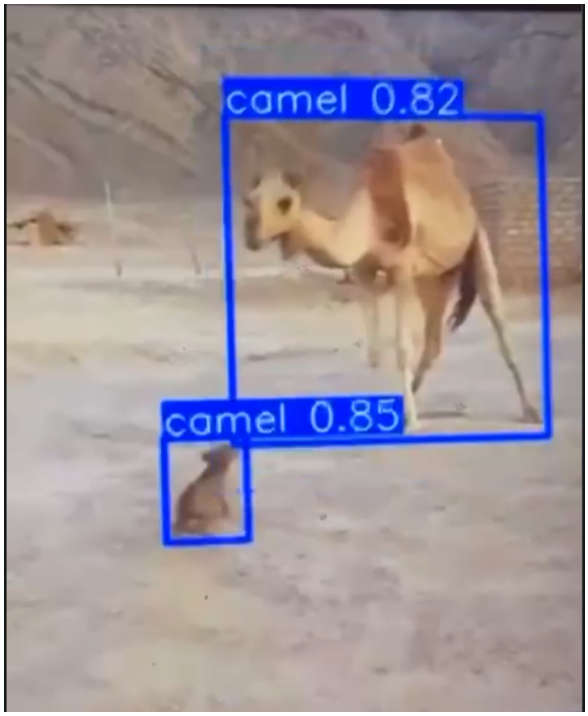


Figure 3: Correct detection in the initial model: a camel successfully identified, despite overfitting.

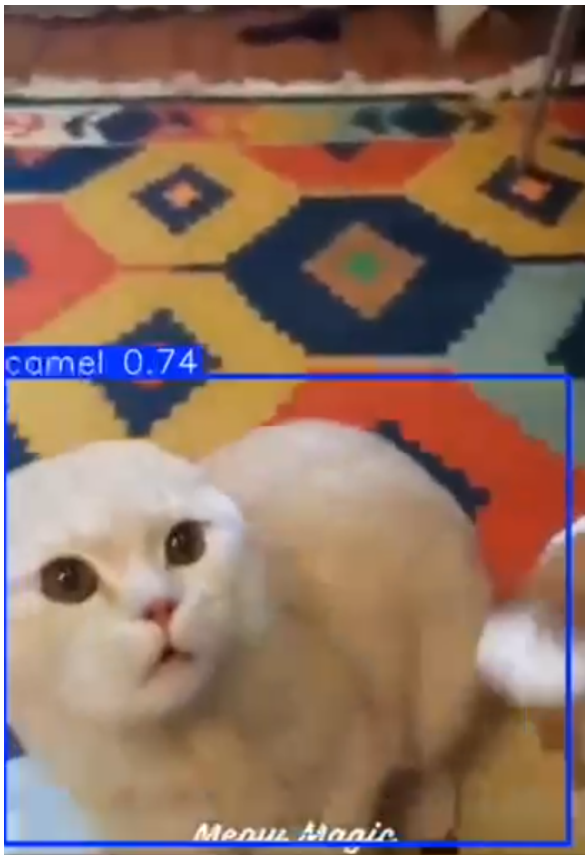


Figure 4: Overfitting in the initial model: a cat misclassified as a camel.

2.2.2 Improved Model: Combined Dataset

To address the overfitting issue observed in the initial camel-only model, we incorporated a subset of the COCO dataset using a Roboflow-prepared mini-COCO dataset [5]. Instead of using the full COCO dataset—which contains hundreds of thousands of images and would have overwhelmed our 200 camel images—we used a reduced 20-class variant. Although this dataset still contained thousands of images, it was already annotated in YOLO format, which simplified integration. A custom sampling script was then used to select approximately 50 images per class, resulting in around 1000 COCO images. Combined with the camel dataset, the total training set contained approximately 1200 images.

Training YOLOv8n on this combined dataset for 50 epochs produced clear improvements. The model developed the ability to distinguish camels from non-camel objects and background clutter. During video inference, for example, when set to detect only camels with 70% confidence, the model no longer misclassified a cat as a camel (Figure 5). Performance improved in terms of recall and reduced false positives, although certain errors remained. Specifically, the model occasionally misclassified animals with similar coloration—such as brown dogs—as camels.

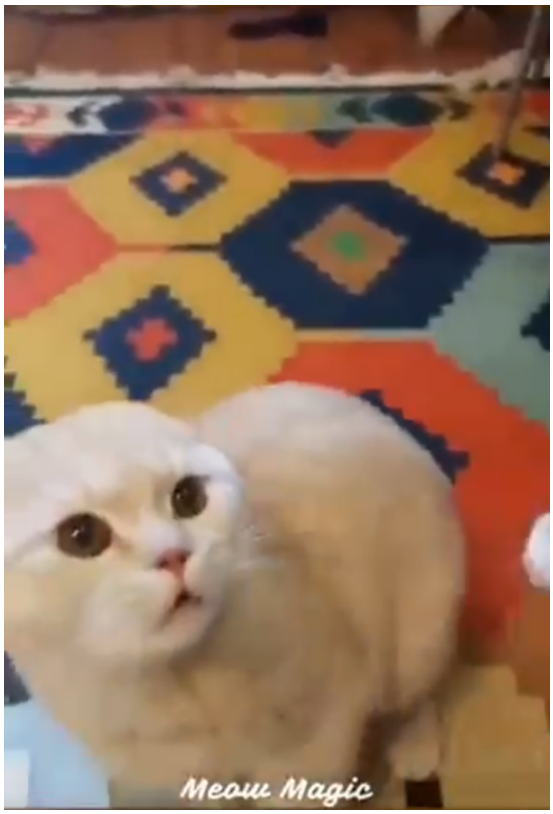


Figure 5: Example from the improved model: the cat, previously misclassified as a camel by the initial model, is now ignored.

2.2.3 Data Augmentation Trials

In addition to expanding the dataset with external sources, we experimented with data augmentation techniques to improve robustness under challenging conditions. The motivation was to prepare the model for deployment in desert highways, where camels may appear in low-visibility environments such as sandstorms, glare, or poor lighting.

Four augmentation strategies were tested: nighttime, sandstorms, duststorms, and fog. These were applied before training to artificially expand the dataset and increase variability in the input images. The expectation was that such augmentations would reduce overfitting and improve generalization to real-world scenarios.

The outcomes of these trials are reported in Section 3.1.6.

2.2.4 Final Model: Augmented Dataset with Documentary Frames

To further improve performance, additional camel images were added, along with more examples of dogs to help the model distinguish between the two and reduce false positives. Frames were manually extracted from a five-minute camel documentary [6] using a custom Python script that saved every 30th frame. A total of 70 frames were collected. Of these, approximately 30 were suitable for pseudo-labeling with the improved model, since the bounding box predictions were consistent and reliable. The remaining 40 images were manually annotated using Roboflow [2]. An example of an extracted frame is shown in Figure 6.

In addition, a custom sampling script was developed to prioritize dog images with

multiple bounding box annotations when selecting from the mini-COCO dataset. This ensured that the dataset contained more diverse and higher-quality dog examples, strengthening the model's ability to distinguish camels from visually similar animals such as brown dogs.

The final augmented dataset therefore contained both richer camel representations and stronger dog coverage. Training YOLOv8n on this dataset for 100 epochs produced the highest performance across all key metrics, including precision, recall, mAP@0.5, and mAP@0.5:0.95, representing the strongest generalization capability of all training stages.



Figure 6: Example frame extracted from the short camel documentary [6].

2.3 Evaluation Metrics

To assess model performance, several standard object detection metrics were used. Precision measures the proportion of predicted camel detections that were correct, while recall measures the proportion of actual camels correctly identified. These two metrics capture the trade-off between avoiding false positives (e.g., labeling a dog as a camel) and false negatives (missing camels entirely).

Mean Average Precision (mAP) was also reported at two thresholds. mAP@0.5 evaluates whether predicted bounding boxes overlap with ground-truth boxes by at least 50%, providing a measure of coarse localization. mAP@0.5:0.95, which averages across stricter thresholds, reflects the model's ability to produce precise bounding boxes around camels.

Loss curves (box loss, classification loss, and distribution focal loss) were tracked during training to monitor convergence and detect signs of overfitting. Finally, inference speed was measured in frames per second (FPS), since very low FPS would prevent real-time detection in highway environments. This metric is especially important in safety-critical applications where delays could reduce the effectiveness of the system.

The specific values for these metrics, along with comparisons between different model versions, are presented in Section 3.1.5.

3 Results Discussion

3.1 Model Training Performance

3.1.1 Overview and Metrics

A dataset split of 80% training, 10% validation, and 10% test was used to train the YOLOv8n model over 100 epochs. The final model achieved a precision of 0.9781 and a recall of 0.9637 on the validation set (Figures 7 and 8), Table 1). These results indicate that the model is highly effective at detecting camels with minimal false positives or missed detections. Additionally, the model achieved an mAP@0.5 of 0.9858 and an mAP@0.5:0.95 of 0.8558. The high mAP@0.5 score reflects the model's ability to correctly localize objects with at least 50% intersection-over-union (IoU), while the mAP@0.5:0.95 demonstrates that bounding boxes are drawn with high precision across a range of IoU thresholds. Although data augmentation was tested during early experiments, the unaugmented dataset produced more stable training and higher final performance. This may be attributed to the need for more tailored or higher-quality augmentations suited to desert environments.

Table 1: Validation metrics of the YOLOv8n model over 100 epochs.

Metric	Value
Precision	0.9781
Recall	0.9637
mAP@0.5	0.9858
mAP@0.5:0.95	0.8558

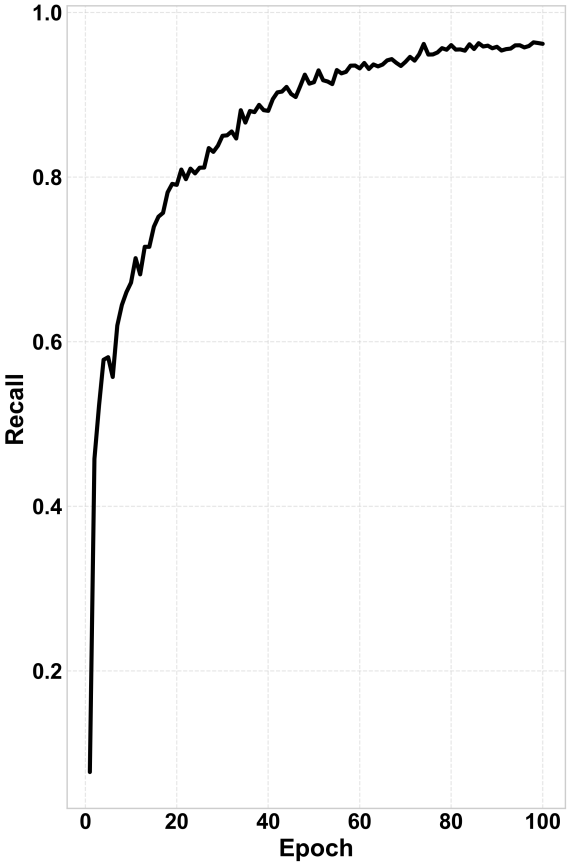


Figure 7: Validation recall across 100 epochs.

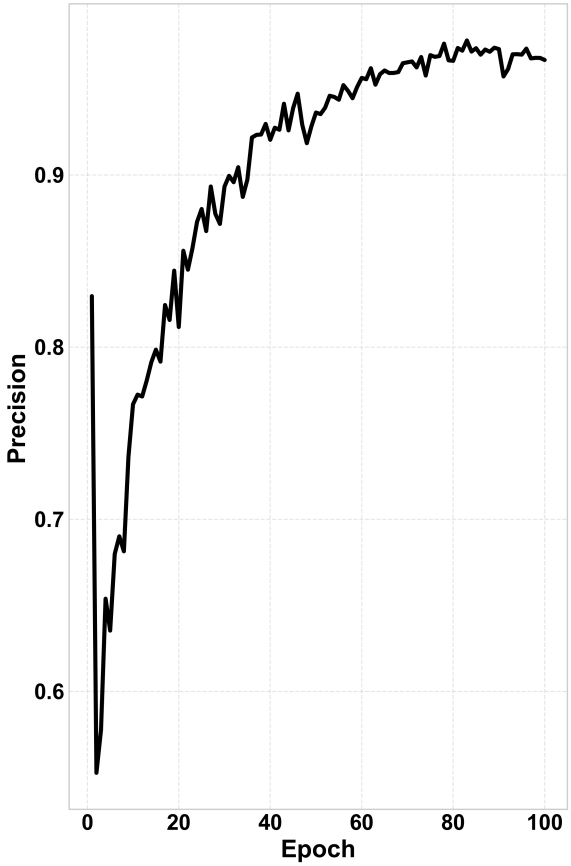


Figure 8: Validation precision across 100 epochs.

3.1.2 Loss Behavior and Convergence

Training losses — including classification, box, and distribution focal loss — decreased steadily over the 100 epochs with no signs of instability, indicating stable convergence. Validation losses followed a similar downward trend, as shown in (Figure 9). Among the three components, classification loss decreased more rapidly than both objectness and box regression losses. This trend can be attributed to the presence of visually distinct classes in the dataset (e.g., camels versus COCO subset classes), which likely enabled the model to converge faster once it began accurately localizing objects.

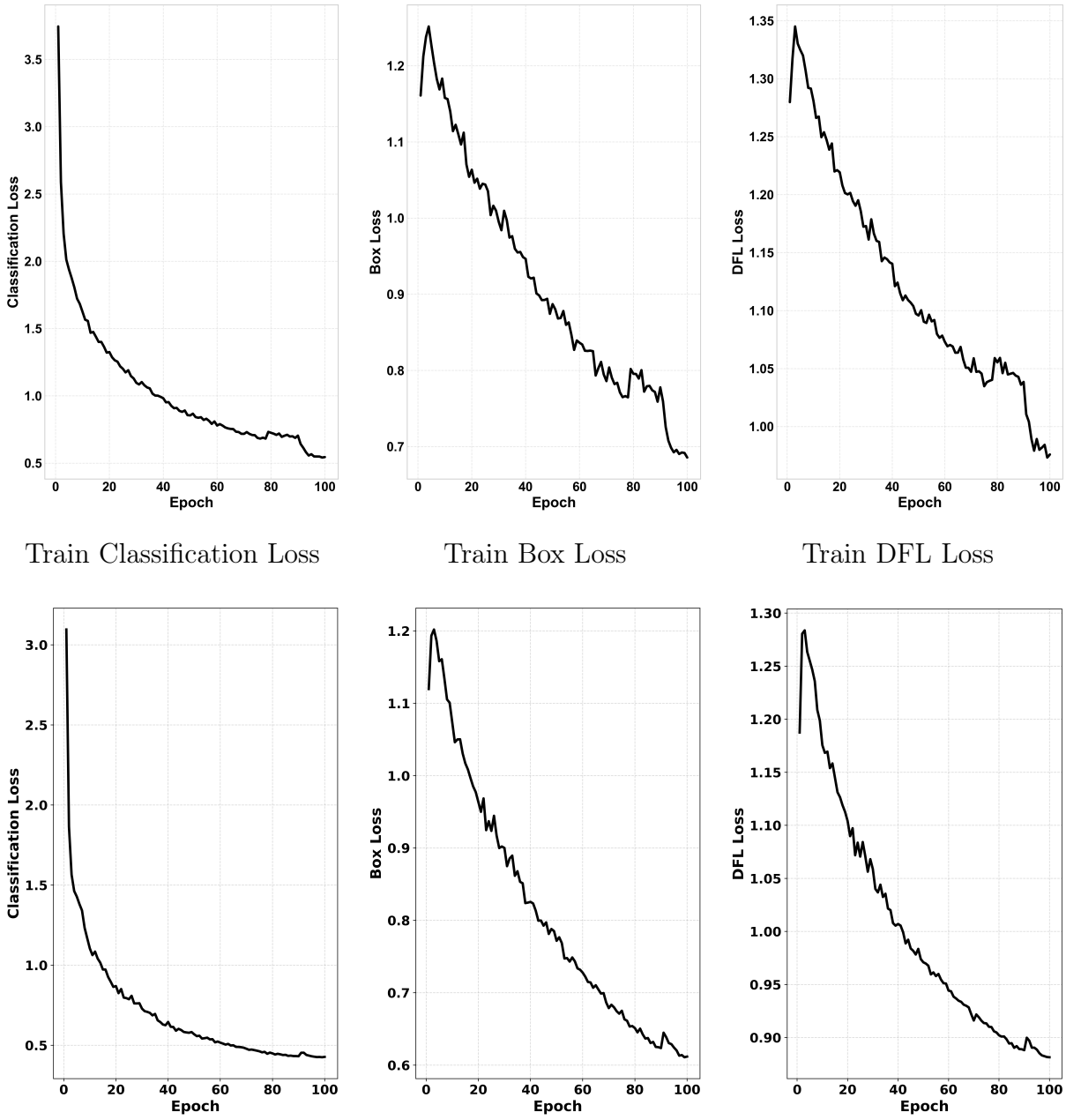


Figure 9: Training and validation losses for classification, box, and DFL components over 100 epochs.

3.1.3 Overfitting Assessment

To evaluate whether the model was overfitting to the dataset, training and validation losses were plotted across all 100 epochs. All three loss components on both training and validation sets followed a downward trend, with no significant divergence between them. For classification loss, the validation value at the final epoch (0.4274) remained close to its lowest point (0.4251 at epoch 98), and was slightly below the final training loss (0.5453). Similar trends were observed for box loss (final val: 0.6118 vs. train: 0.6857) and distribution focal loss (final val: 0.8815 vs. train: 0.9760). A summary of these

values is provided in Table 2. Interestingly, validation losses were consistently lower than training losses, which suggests that the model generalized well to unseen data and did not overfit — a promising sign of robust feature learning.

Table 2: Final training and validation losses, along with best validation values and their corresponding epochs.

Loss Type	Final Train Loss	Final Val Loss	Best Val Loss (Epoch)
Classification	0.5453	0.4274	0.4251 (98)
Box	0.6857	0.6118	0.6109 (98)
DFL	0.9760	0.8815	0.8815 (99)

3.1.4 Training Setup

The model was trained using the YOLOv8n default configuration over 100 epochs. Training was conducted locally on an NVIDIA RTX 4070 GPU, using the Ultralytics implementation of YOLOv8 in Python.

3.1.5 Comparison to Previous Models

Table 3 compares the performance of the Improved and Final models. The Improved model, trained for 50 epochs on the combined dataset, reached a precision of 0.9112, recall of 0.7565, mAP@0.5 of 0.8613, and mAP@0.5:0.95 of 0.5489. In contrast, the Final model, trained for 100 epochs with documentary frames and enhanced dog sampling, significantly outperformed it across all metrics. This confirms that both dataset augmentation and longer training contributed to stronger generalization and robustness in detecting camels. For definitions of these metrics, refer to Section 2.3

Table 3: Performance comparison across model iterations. The Final model (100 epochs, augmented dataset) outperforms the Improved model (50 epochs, combined dataset) on all validation metrics.

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Initial (camel-only)	—	—	—	—
Improved (50 epochs)	0.9112	0.7565	0.8613	0.5489
Final (100 epochs)	0.9781	0.9637	0.9858	0.8558

Notes: The initial camel-only model consistently overfit and failed to generalize; stable validation metrics are omitted. The Improved model used a combined dataset (camel + mini-COCO). The Final model added documentary frames and targeted dog oversampling.

.

3.1.6 Impact of Data Augmentation

To test whether data augmentation could improve robustness under low-visibility conditions (such as sandstorms, low light, or glare), four augmentation techniques were applied

during early experiments. The objective was to expose the model to degraded inputs so it could generalize better to real-world desert highway scenarios.

However, the augmented datasets produced less stable training curves, and the model often failed to converge as smoothly as with the unaugmented dataset. Validation metrics also lagged behind those of the baseline runs. These results suggest that the augmentations applied were not well-suited to the domain, and may have confused the model rather than improving robustness. Consequently, the best-performing models reported in this work were trained on the unaugmented dataset. Future work should explore more carefully tailored augmentations, such as synthetic sand, glare, or low-light conditions, to better simulate real deployment environments.

3.1.7 Interpretation and Limitations

The high validation accuracy and real-time inference performance demonstrate that the trained YOLOv8n model is suitable for edge deployment in safety-critical environments such as desert highways. The model maintained low losses and high precision and recall throughout training, without signs of overfitting. These trends suggest that performance could improve further if training were extended beyond 100 epochs.

However, limitations remain. Despite deliberately oversampling dog images with multiple bounding box annotations to strengthen class separation, the model still occasionally misclassified brown dogs as camels due to their visual similarity. This indicates that while targeted dataset balancing reduced false positives, further improvements are required to fully address edge cases.

Transition. While the training results confirm the accuracy and robustness of the YOLOv8n model, its practical viability also depends on deployment performance. The following section evaluates system-level behavior across multiple hardware environments, including Raspberry Pi and laptop platforms.

3.2 Component-Level Evaluation

To evaluate real-time performance across platforms, the trained YOLOv8n model was deployed on four hardware setups: Raspberry Pi 3, Raspberry Pi 5, and two laptop environments. Each platform was tested under similar conditions to compare latency, throughput, and resource utilization. While YOLOv8n is the most lightweight variant in the YOLOv8 family, it still requires a non-trivial amount of computational resources for real-time inference. As a result, performance on resource-constrained devices like the Raspberry Pi was limited, despite using the optimized nano version. The results below illustrate how hardware specifications significantly impact real-time detection capabilities.

3.2.1 Raspberry Pi 3

The Raspberry Pi 3 presented significant challenges. Initially, the camera module was not detected due to a change in library naming conventions ("libcams" was renamed to "rpicam"). After resolving this, the major bottleneck was rendering. The script launched a separate rendering window with frame rate overlays, which placed a heavy load on the CPU since the device lacks GPU acceleration. As a result, frame rates dropped

severely, and bounding-box rendering further exacerbated latency. No stable results could be obtained, making this platform unsuitable for real-time YOLOv8n inference.

3.2.2 Raspberry Pi 5

On the Raspberry Pi 5, the environment was rebuilt on a new microSD card. Performance was better than the Pi 3 but still below expectations. Using a mobile phone as an RTSP camera, the system achieved ∼300 ms latency with live rendering enabled. Removing rendering and limiting output to terminal logs reduced latency only marginally (to ∼200 ms). This indicates that the bottleneck is not only rendering but also the overall compute limitations of the Pi 5 when handling YOLOv8n workloads.

3.2.3 Laptop Environments

On standard laptop hardware with integrated webcams, the system performed significantly better:

- **Average FPS:** 33.67
- **Average Latency:** 40.62 ms
- **CPU Usage:** ∼1%
- **RAM Usage:** ∼9 GB

These results establish a strong performance baseline and confirm that YOLOv8n is capable of real-time execution under adequate computational resources. However, the RAM usage of approximately 9 GB is non-trivial and may limit deployment on memory-constrained systems.

3.3 Comparison to Benchmarks

A latency of < 100 ms is typically expected for real-time video applications. The laptops met this requirement comfortably (40.62 ms), while both Raspberry Pi devices fell well short (200–300 ms). Similarly, the laptops achieved FPS values above the real-time threshold (30 FPS), whereas the Pi devices could not sustain comparable rates.

3.4 Identified Bottlenecks and Suggested Improvements

- **Raspberry Pi 3:** Lack of GPU support and insufficient CPU resources made YOLOv8n rendering and inference infeasible. Recommendation: discontinue use of Pi 3 for this workload.
- **Raspberry Pi 5:** Latency was dominated by CPU-bound inference. Even with rendering removed, performance remained below real-time benchmarks. External accelerators (e.g., Coral TPU, NPU hats) could provide substantial improvements.
- **Laptops:** System performance was within acceptable ranges. The primary bottleneck observed was high RAM usage (∼9 GB), which may limit portability.

3.5 System-Level Evaluation

When evaluated as a complete system, performance is highly hardware-dependent. On resource-constrained platforms (Raspberry Pi 3/5), the model is unable to achieve real-time inference, making them unsuitable without hardware accelerators or aggressive model optimization. On laptop-class machines, the system consistently outperformed expectations, achieving low latency and high frame rates.

4 Conclusion and Future Work

Our trained YOLOv8n model provides high detection accuracy while maintaining deployability on edge devices such as the Raspberry Pi, enabling the entire process to run locally without requiring internet connectivity. This makes it an important step toward reducing camel–vehicle collisions, particularly in remote desert highways where network infrastructure is limited. In addition to achieving strong accuracy, the model demonstrated robust generalization across validation sets, with no signs of overfitting, confirming its reliability in detecting camels under diverse conditions. Although resource-constrained devices such as the Raspberry Pi 3 and 5 currently struggle to achieve real-time performance due to CPU-bound latency, the system has demonstrated strong feasibility on laptop-class hardware. With future optimizations such as model quantization, pruning, or the integration of hardware accelerators, CAMelFree has the potential to deliver reliable, low-latency detection in the field, thereby saving lives and reducing economic losses from camel-related accidents.

References

[1] Khalid Alnujaidi. *Images of Camels Annotated for Object Detection*. https://www.kaggle.com/datasets/khalidalnujaidi/images-of-camels-annotated-for-object-detection. Accessed: 2025-08-27. 2023.

[2] Abdulaziz Alsaihati. *Camel Detection Dataset (Documentary Frames, Manually Annotated in Roboflow)*. https://app.roboflow.com/object-detection-bg1li/camel-2-l4wja/1. Accessed: 2025-08-27. 2025.

[3] Abdulaziz Alsaihati. *Camel Detection Dataset (Manually Labeled in Roboflow)*. https://app.roboflow.com/object-detection-bg1li/camel-upwsj/1. Accessed: 2025-08-27. 2025.

[4] Ali S. Al-Ghamdi and Saad A. AlGadhi. "Warning signs as countermeasures to camel-vehicle collisions in Saudi Arabia". In: *Accident Analysis Prevention* 36.5 (2004), pp. 749–760. DOI: 10.1016/j.aap.2003.05.006.

[5] suiyi. *VOC-COCO Mini (voc-coco-kgnol) Dataset*. https://universe.roboflow.com/suiyi-cdzew/voc-coco-kgnol. Accessed: 2025-08-27. 2025.

[6] Cosmos Therapy. *World of Camels | A Short Documentary*. https://youtu.be/sq9LwzMLFA8. Accessed: 2025-08-27. 2025.