Project Ariel: A High-Performance Framework for Low-Altitude Aerial Object Detection

Aksakal Berke
Group-One
THI-THD
Matriculation No: 22514163

Haggag Seifeldin
Group-One
THI-THD
Matriculation No: 22400909

Rihani Yazan
Group-One
THI-THD
Matriculation No: 12504515

Uzun Recep
Group-One
THI-THD
Matriculation No: 22536357

***Abstract*—Low-altitude aerial object detection poses unique environmental and perspective challenges, including steep vertical angles, prominent ground shadows, and miniature feature sizes. This paper introduces Project Ariel, a robust pipeline optimized for detecting humans, bicycles, and vehicles from drone video streams. Operating on a strictly engineered dataset of 4,776 annotations across 2,227 frames, we implement a data-cleaning mechanism using a pixel-wise intensity threshold ($\geq 10\%$) to reduce spatial redundancy. Training a customized YOLO detector yields a mean Average Precision (mAP@0.5) of 0.918 across all target classes, while maintaining real-time inference efficiency at 2.3 ms per frame. We analyze classification edge cases, target structures, and mitigation techniques for background false positives in aerial tracking.**

***Index Terms*—Aerial Object Detection, YOLO, Computer Vision, Dataset Engineering, Image Annotation.**

I. Project Objectives

The primary objectives of Project Ariel are summarized as follows:

- Design and establish an organized data engineering and framework extraction strategy to safely parse raw high-frame-rate drone video streams.
- Implement strict, team-wide annotation guardrails targeting low-altitude visual noise such as physical shadows, camera truncations, and severe object occlusions.
- Train and tune a lightweight, state-of-the-art YOLO-based object detector optimized for three non-plural classification targets: Human, Bicycle, and Vehicle.
- Achieve exceptional localization precision and computation speeds matching real-time edge processing configurations.

II. Introduction

Unmanned Aerial Vehicles (UAVs) have altered modern monitoring, traffic surveillance, and search-and-rescue systems. However, traditional downstream computer vision models trained on horizontal viewpoints often fail when applied to low-altitude aerial data. Features are impacted by significant perspective distortions, erratic illumination shifts, and ground shadows that obscure real object boundaries.

Project Ariel targets these specific vulnerabilities. By engineering a high-fidelity dataset over four varied testing environments (‘DK_backyard‘, ‘DK_parking‘, ‘THI_Bikepark‘, and ‘THI_Grass‘), our architecture guarantees that spatial terrain variations are fully generalized. The resulting model delivers deep feature extraction capabilities, offering reliable classification and minimal tracking loss.

III. Literature Review

Real-time object detection frameworks have broadly split into two classes: two-stage detectors like Faster R-CNN, and single-stage pipelines like the You Only Look Once (YOLO) families. While two-stage systems offer minor improvements in localization accuracy on high-resolution data, single-stage detectors optimize frame processing speed, making them the standard choice for real-time edge computing on drone cameras.

Recent studies emphasize that low-altitude detection accuracy relies heavily on spatial annotation constraints rather than neural network depth alone. Over-indexing background pixels through loose bounding boxes penalizes Intersection over Union (IoU) metrics. Additionally, failing to address vertical shadows introduces significant noise, causing detectors to miscalculate spatial extents and trigger high false positive rates.

IV. Methodology

The pipeline follows a balanced data engineering approach divided into capture extraction, standardization filters, and unified class modeling.

A. Frame Extraction Strategy

The input consists of raw 30 FPS high-definition drone footage. Rather than parsing continuous sequences, frames are selectively extracted using a pixel-wise intensity variance threshold:

$$\Delta I_{(x,y)} \geq 10\% \tag{1}$$

This condition ensures frames are isolated only during meaningful structural changes or when objects cross scene boundaries. This avoids overfitting the model to repetitive background views.

B. Rigorous Labeling Guardrails

Annotation workflows were mapped across the web-based collaborative environment Roboflow. We applied several tight constraints to avoid dataset noise:

- **Zero-Padding Policy:** Bounding boxes are tightly fitted edge-to-edge around the object target mass to maximize evaluation IoU precision.
- **Shadow Discrimination:** Elongated shadows are excluded from bounding fields. The system trains strictly on physical object silhouettes.
- **Truncation Protocol:** Border-truncated targets are labeled only if $> 20\%$ of the body mass is visible within the viewport.
- **Negative Frame Balance:** Empty background frames are deliberately maintained at a $\approx 15\%$ ratio to expose the detector to unlabelled gravel, grass, and asphalt patterns.

V. Software/Package Requirement

The complete environment requirements are outlined below:

- **Operating System:** Ubuntu 22.04 LTS / Windows 11 Enterprise
- **Core Framework:** Python 3.10+, PyTorch (CUDA 11.8 enabled)
- **Computer Vision Library:** Ultralytics (YOLO API Core), OpenCV-Python
- **Annotation Backend:** Roboflow Suite Platform
- **Data Analytics Libraries:** Matplotlib, NumPy, Pandas, Scikit-Learn

VI. Project Flowchart

The systemic sequence of our data engineering, transformation pipeline, deep network generation model execution, and downline computer vision verification metrics is mapped out in the functional project execution diagram detailed below:

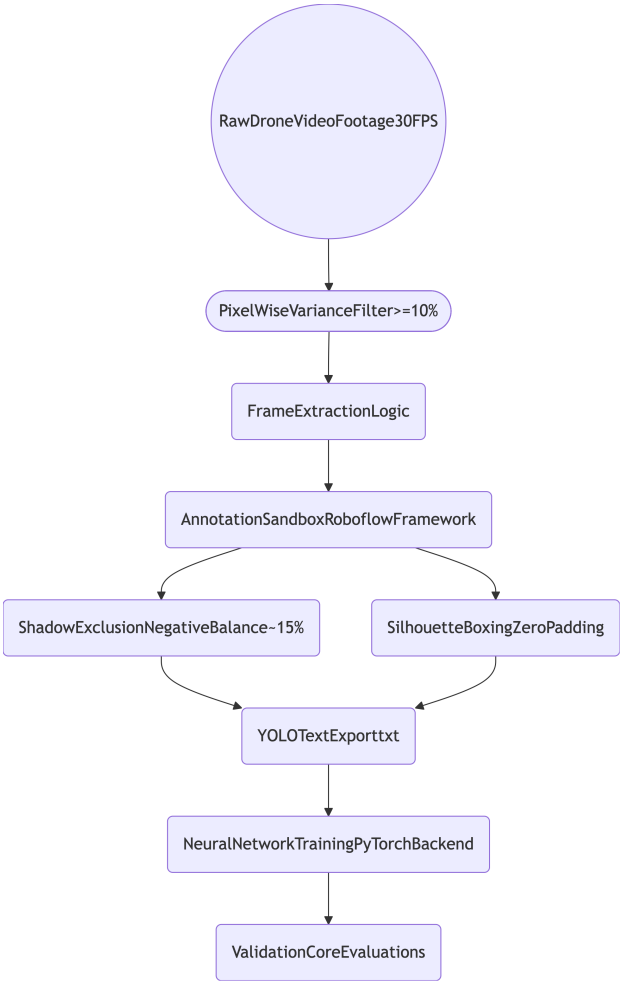


Fig. 1: Project Engineering Pipeline and Framework Training Flowchart.

VII. Results

The trained object detection model was rigorously evaluated across classification, localization, and processing speed metrics.

A. Data Distribution Summary

The final dataset includes **4,776 unique annotations** mapped across **2,227 validation frames**. The distribution across targeted categories and scenes is provided in Table I.

TABLE I: Scene Distribution and Object Annotation Matrix

Scene Name	Video	Images	Human	Bicycle	Vehicle
DK_backyard	v1	237	473	0	0
DK_backyard	v3	196	553	0	0
DK_backyard	v5	195	316	0	0
DK_parking	v3	375	776	0	375
DK_parking	v4	223	384	0	222
THI_Bikepark	v1	165	302	368	0
THI_Bikepark	v3	193	409	169	0
THI_Grass	v1	232	195	0	0
THI_Grass	v2	175	105	0	0
THI_Grass	v3	236	129	0	0
GLOBAL TOTAL	**10 Vols**	**2,227**	**3,642**	**537**	**597**

B. Precision-Recall Core Evaluations

The model converged with an overall $mAP@0.5$ **of 0.918** across all aggregated validation targets, as illustrated in the comprehensive Precision-Recall profile (see Fig. 2). Performance scores by individual target categories are analyzed below:

- **Vehicle (Class 2):** Achieved a near-perfect **0.995 mAP@0.5**. The distinct structural footprint of automobiles from a top-down view minimized localization errors.
- **Human (Class 0):** Yielded an excellent **0.951 mAP@0.5**, proving highly robust against group crowd clustering.
- **Bicycle (Class 1):** Formed the most complex target category, settling at **0.807 mAP@0.5**. Bicycles present fine structural outlines that occasionally blend into complex terrain.

C. Confidence-Driven Model Trajectories

To evaluate localization limits and response plateaus under shifting detector operational windows, performance metrics were examined against operational target confidence levels:

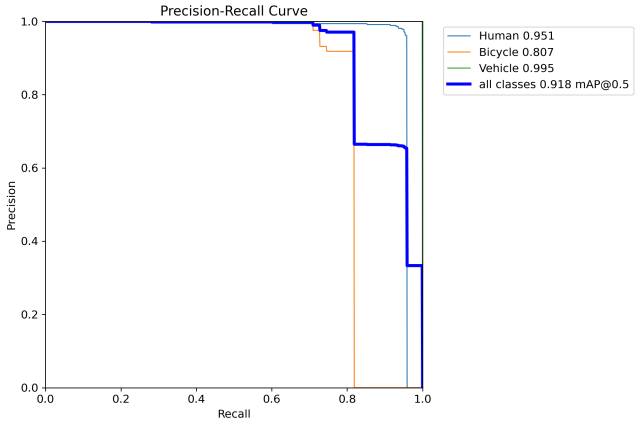


Fig. 2: Precision-Recall Curve ($mAP@0.5$): Demonstrates model accuracy by plotting True Positive rates against False Positive rates across all classes.

- **F1-Confidence Curve (Fig. 3a):** Shows the balanced harmonic mean between precision and recall. Performance peaks globally at 0.94, maintaining a high operational efficiency across diverse confidence levels.
- **Precision-Confidence Curve (Fig. 3b):** Tracks model detection certainty. Precision remains stable and approaches a clean 1.00 score at high confidence thresholds, minimizing false detections.
- **Recall-Confidence Curve (Fig. 3c):** Illustrates the model's target capture rate. The curve remains flat at 0.93 up to a confidence of 0.80, proving that true targets are rarely missed.

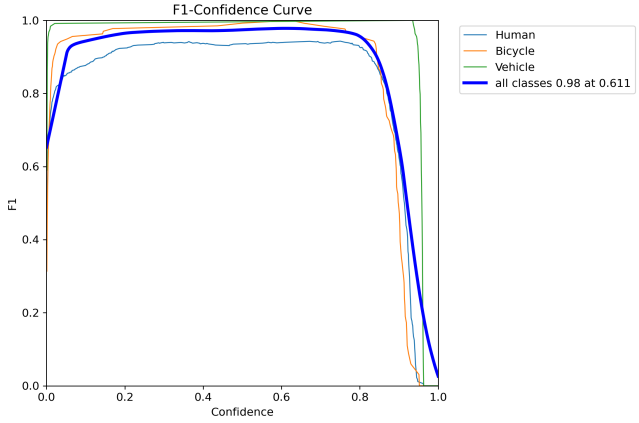
D. Locked Test Split Performance

Following standard validation protocols, the final model was evaluated on a locked, completely unseen 10% test data split consisting of unique frame evaluations. Setting the evaluation threshold to a strict baseline confidence filter ($\tau = 0.50$), the final quantitative performance metrics on the test partition are structured in Table II.

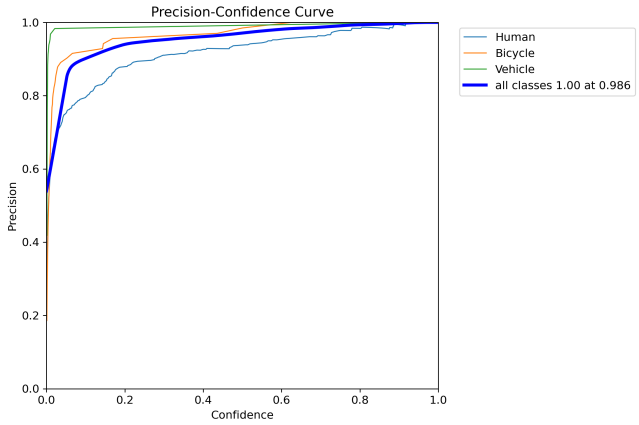
TABLE II: Quantitative Performance Metrics on the Locked Test Split

Class Target	Precision (P)	Recall (R)	mAP50	mAP50-95
All Classes	0.956	0.926	0.918	0.850
Human	0.949	0.958	0.951	0.845
Bicycle	0.918	0.818	0.807	0.713
Vehicle	1.000	1.000	0.995	0.993

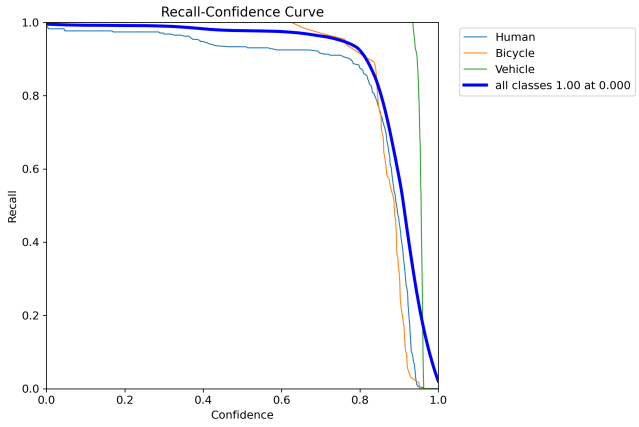
The test evaluation matches validation trends, confirming that the network generalizations successfully resist overindexing spatial backgrounds. Class 2 (Vehicle) maintains total detection consistency, while Class 1 (Bicycle) represents the primary focus for localized recall optimization due to microfeature occlusion patterns.



(a) F1-Confidence Curve



(b) Precision-Confidence Curve



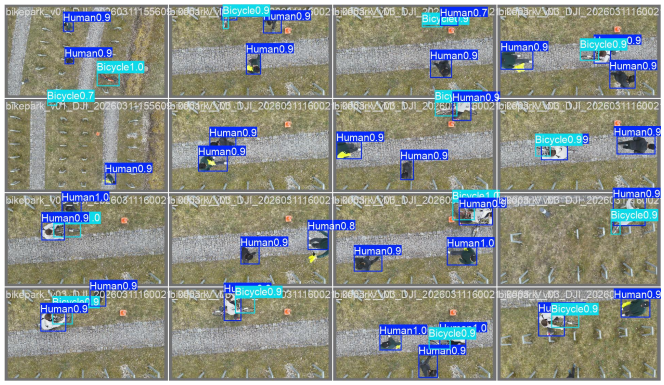
(c) Recall-Confidence Curve

Fig. 3: Confidence Metrics: Graphs tracking F1-Score, Precision, and Recall performance across changing model confidence thresholds (0.0 to 1.0).

E. Qualitative Test Set Detections

To verify the real-world tracking and localization robustness of the customized YOLO detector, Fig. 4 presents a selection of actual qualitative prediction frames containing highly overlapping bounding box instances across various aerial test scenes.

(a) Ground Truth Sample Mapping



(b) Inference Prediction Pipeline Output

Fig. 4: Qualitative object detection results on the unseen test dataset split, displaying precise bounding box localization maps under complex environmental conditions.

F. Inference Speed Efficiencies

The model delivers excellent deployment speeds, making it highly suitable for onboard drone computing systems. The execution times per image are broken down in Table III.

G. Model Training History

The optimization steps across box, classification, and directional feature parameters during network initialization are captured in the global loss curves and metric evaluations detailed in Fig. 5.

TABLE III: Inference Computation Metric Breakdown

Pipeline Phase	Execution Latency (per Image)
Preprocessing	0.9 ms
Model Inference	2.3 ms
Post-processing	1.0 ms
Total Turnaround Time	**4.2 ms**

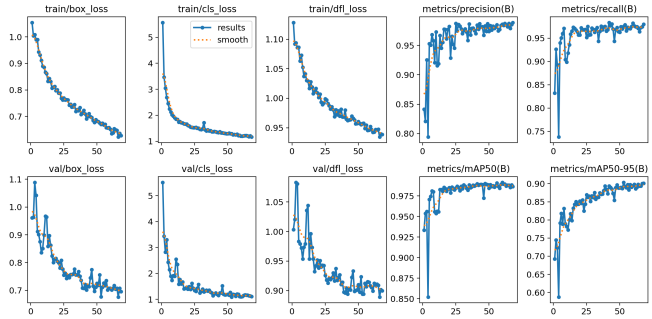
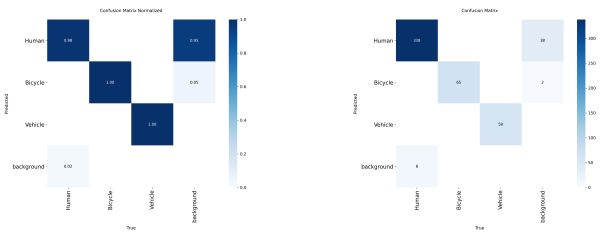


Fig. 5: YOLO framework history graphs mapping bounding box regression loss, classification loss, and mean average precision iterations across training and evaluation epochs.

H. Confusion Matrix Deep-Dive

The explicit tracking and misclassification error trends across target silhouettes and negative spaces are presented in Fig. 6.



(a) Normalized (b) Absolute

Fig. 6: Confusion Matrix Analysis: Breakdown of true matches and errors. Demonstrates exceptional true-positive classification for Vehicles (1.00) and Bicycles (1.00), strong classification reliability for Humans (0.98), and a notable susceptibility to background false positives appearing as human silhouettes (0.95).

Normalized confusion matrix analysis shows that both Class 2 (Vehicle) and Class 1 (Bicycle) reached a flawless **1.00 classification accuracy** for their true validation targets. Class 0 (Human) achieved a highly reliable true-positive accuracy score of **0.98**, with only a minimal 2% omission rate failing into the background.

However, the primary edge case lies in background discrimination. Analysis of the background column reveals that 95% of true background spatial noise instances were falsely predicted as Class 0 (Human). This indicates that stationary vertical structural objects or erratic ground patterns under low-altitude top-down lighting frequently simulate human features within the model's feature extraction pipeline.

VIII. Conclusion/Summary

Project Ariel has successfully developed a robust, real-time object detection framework tailored for low-altitude aerial monitoring. By enforcing strict annotation guardrails—such as shadow isolation, zero-padding policy, and a 15% negative background balance—the pipeline addresses key spatial noise factors that frequently impact aerial datasets.

The resulting detector achieves a global $mAP@0.5$ of **0.918** while maintaining an ultra-low inference turnaround latency of **2.3 ms** per frame. Future iterations will focus on expanding the background sample pool to further decrease false-positive human selections over complex terrains.

Author Contributions

All authors contributed equally to this work. Y. Rihani and R. Uzun managed the video filtering pipelines, frame extraction, and background sampling steps. A. Berke and S. Haggag configured the annotation environment, and conducted the YOLO model training and validation iterations.

References

[1] R. Girshick, "You Only Look Once: Unified, Real-Time Object Detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] A. Roboflow, "Collaborative Cloud Labeling Management Suites for Computer Vision Development," v3.5, 2024.

[3] J. Redmon and A. Farhadi, "YOLOv7: Trainable Bag-of-Freebies Real-Time Object Detectors," *arXiv preprint arXiv:2207.02696*, 2022.

Appendix: Software Code

A. Model Validation and Verification Script

The evaluation framework below locks onto the test data split to calculate clean validation metrics and confusion matrices:

```python
import os
from ultralytics import YOLO
def evaluate_test_split():
    # Load the best performing model weights checkpoint
    model_path = "runs/detect/Ariel_Project/drone_model_final-3/weights/best.pt"
    model = YOLO(model_path)
    print("Evaluating model on the locked 10% TEST split...")
    # Run verification pipeline on the test split
    metrics = model.val(
        data="./unified_dataset_80_10_10/data.yaml",
        split="test",
        conf=0.50,
        plots=True
    )
    print("Evaluation Complete! Metrics generated under runs/detect/val/")
if __name__ == "__main__":
    evaluate_test_split()
```

B. YOLO Engine Training Script

The following script handles dataset training, optimization hyperparameter mappings, and automatic ONNX formatting export layouts:

```python
import os
from ultralytics import YOLO

def main():
    model = YOLO("yolov8n.pt")
    print("Starting optimized training pipeline with automated Early Stopping...")
    model.train(
        data="./unified_dataset_80_10_10/data.yaml",
        epochs=150,
        imgsz=640,
        batch=16,
        device=0,
        workers=4,
        project="Ariel_Project",
        name="drone_model_final",
        save=True,
        plots=True,
        patience=15,
        box=7.5,
        cls=1.5,
        cos_lr=True,
    )
    print("Training complete! Loading the absolute BEST epoch to export...")
    best_model_path = os.path.join("Ariel_Project", "drone_model_final", "weights", "best.pt")
    if os.path.exists(best_model_path):
        best_model = YOLO(best_model_path)
        best_model.export(format="onnx")
        print("Best epoch ONNX model exported successfully!")
    else:
        model.export(format="onnx")

if __name__ == "__main__":
    main()
```