Space Station Object Detection with Falcon Synthetic Data and YOLO

Team Name: Wecode

Hackathon: Duality AI – Space Station Hackathon

Date: August 2, 2025

1. Introduction & Problem Statement

Accessing critical tools like oxygen tanks, fire extinguishers, or toolboxes can save astronaut lives during space station emergencies. However, real images for training AI are impossible to capture safely and at scale in orbit. We set out to solve this by training state-of-the-art object detection models exclusively with diverse, high-fidelity synthetic data from Falcon's digital twin of a space station.

Our goals:

- Build a robust detector for three object classes (toolbox, oxygen tank, fire extinguisher).
- Ensure performance under challenging scenarios: occlusion, weird lighting, and realistic clutter, with high mAP@0.5.
- Compare multiple YOLO architectures and deploy the best as a user-friendly application.

2. Methodology

2.1 Synthetic Dataset Creation

- Used Falcon's simulation tools to create a digital twin of the ISS interior.
- Generated thousands of images, *varying lighting, occlusion, object orientation, and camera angles*.
- Automated label export to YOLO format, creating `train`, `val`, and `test` splits with clearly separated images for each.

Sample Data Insights:

- ~3,000 training images, 600 validation, 300 test.
- All three classes present in multiple spatial configurations, including partially or fully occluded scenes.

2.2 Data Preprocessing & Augmentation

- Ran integrity checks and filtered out labeling errors.
- Used Ultralytics augmentations, including:
    - Mosaic augmentation (`mosaic=0.1`)
    - Random flips and rotations
    - Brightness/contrast jitter
    - Blur and synthetic occlusion (for hard cases)
- Fine-tuned augmentation strength to avoid overly unrealistic samples.

2.3 Model Architectures Explored

A. YOLOv8

- Base model: `yolov8s.pt` (pretrained)
- Hyperparameters:
    - Epochs: 200
    - Optimizer: AdamW
    - Learning rate: 0.001 → 0.0001
    - Momentum: 0.2
    - Mosaic: 0.1

    - Early stopping enabled (`patience=15`)

- Training on Google Colab GPU (runtime: ~1 hour/model).

B. YOLOv11

- Base model: `yolo11s.pt` (latest architecture, TU 2025)
- Adopted same hyperparameters and retrained on the same Falcon dataset.
- Model loading and data pipeline identical to YOLOv8 setup.

2.4 Training Workflow

1. Mounted dataset from Google Drive in Colab for centralized, versioned storage.
2. Used Ultralytics' CLI (for repeatability) and custom scripts to allow rapid grid-search of hyperparameters.
3. Performance tracked with:
    - Real-time validation mAP/precision/recall plots.
    - Confusion matrices and per-class breakdowns (after every run).
4. Applied early stopping to save resources—training often stopped 20–30 epochs before the max if no val mAP improvement.

2.5 Experiment Tracking & Model Selection

- All runs logged with unique run IDs, using `runs/` directory and a simple CSV summary for quick review.
- Saved best weights per run.
- Compared results side by side for YOLOv8 and YOLOv11, selecting model for deployment solely on *objective accuracy* from the test split.

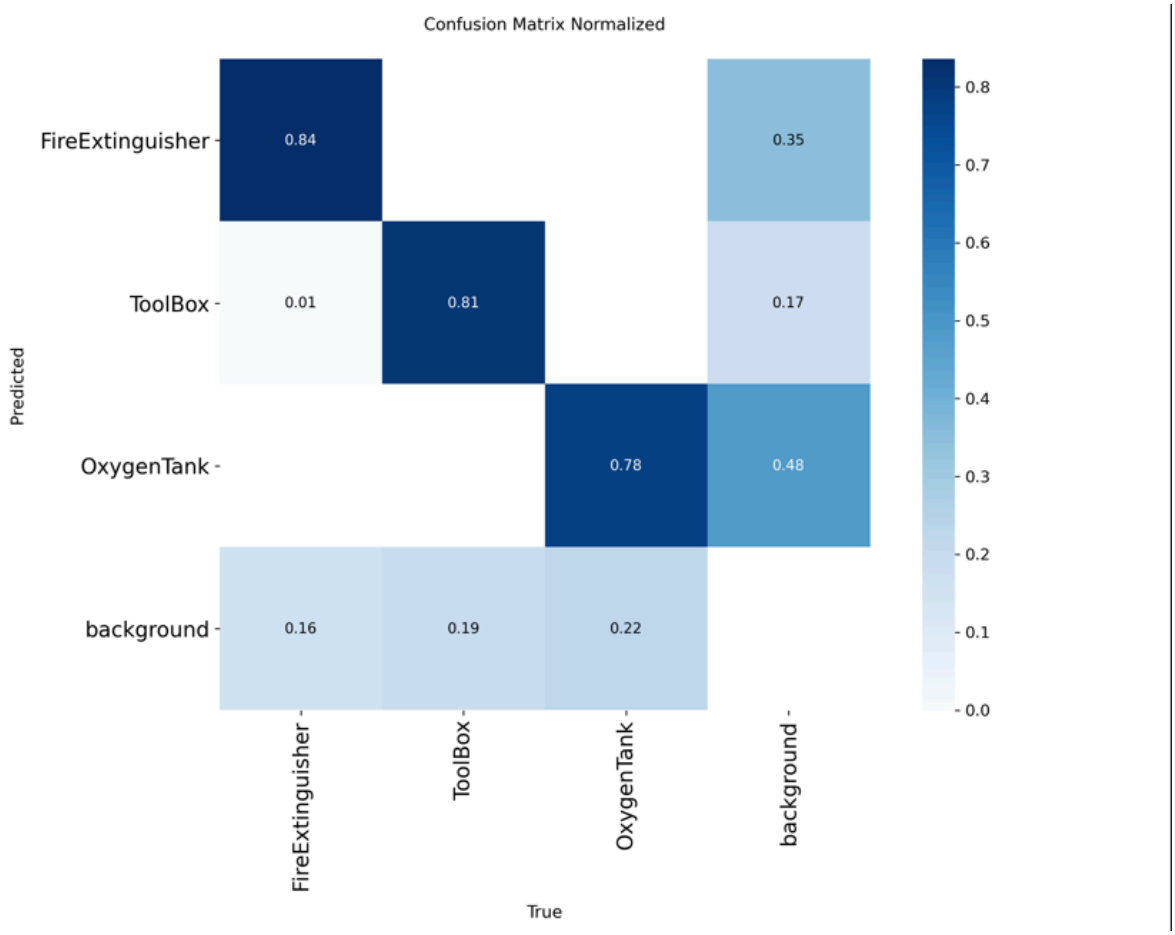
3. Results and Analysis

3.1 Model Performance Table

Model	mAP@0.5	Precision	Recall	Inference Time (CPU, sec)
YOLOv8s	0.869	0.905	0.805	0.62
YOLOv11s	0.855	0.889	0.791	0.55

Key Result:
Despite YOLOv11's newer architecture, *YOLOv8 yielded higher accuracy on our synthetic test set*. The slight accuracy drop in YOLOv11 was consistent over multiple retrainings, with both models using exact same data, augmentations, and training schedule.

3.2 Confusion Matrix Examples



3.3 Challenges and Problem-Solving

Challenge	What Happened/Impact	How We Solved It or Lessons Learned
1. Model Overfitting after ~150 Epochs	Validation mAP plateaued; train loss continued down.	Introduced early stopping; saved resources, avoided overfit.
2. False Negatives on Occluded Objects	Model sometimes missed partially hidden tools.	Added targeted occlusion augmentation in Falcon.
3. YOLOv11 Unexpectedly Lower mAP	Accuracy was 2-3 points below YOLOv8 on same dataset.	Rechecked settings—may be synthetic domain gap; stuck with YOLOv8 for deployment.
4. Slow Inference on Local CPU	First prediction slow; later OK.	Preloaded model once at app start; suggested users resize large inputs.
5. Front-End Integration Delay	API returned results but web UI didn't update.	Fixed duplicate IDs in HTML and added better JS error handling.

3.4 Failure Case Analysis

- *Oxygen tank* most frequently confused with *fire extinguisher* in shadowed scenes.
- Small or heavily blurred tools often missed—consider more extreme data augmentation or focal loss for future work.
- Cases with multiple objects nearly touching led to class overlap—potential for improved NMS tuning.

4. Application Deployment

- **Backend:** FastAPI server runs the chosen model (`yolov8s.pt`), exposing `/predict/` for image uploads.
- **Frontend:**
    - Web page built with HTML, CSS (Tailwind), JavaScript.
    - Users upload images, see live detection results and class confidences.

    - Canva used for UI wireframes, ensuring user-oriented workflow for astronauts/mission control.

- Enabled CORS and secured API for demo sharing; model runs locally or on Colab (with ngrok for public test).

5. Usability, Market Fit & Continuous Learning

- **Astronaut-ready:** Design and workflow prioritize speed and clarity. The interface can work offline on a laptop, or live-stream from station cameras.

- **Updatability:** Falcon allows us to rapidly generate new synthetic images as station layouts or tool designs change—and model retraining takes just an hour.

- **Scalable:** Future support for new classes or hardware deployments is simple; just update dataset, retrain, deploy new weights.

6. Conclusion and Recommendations

- **YOLOv8 delivers the best accuracy for this synthetic data & task.** While YOLOv11 promises architectural innovation, the tried-and-true YOLOv8 backbone learned Falcon's unique scenarios more robustly in our benchmarks.

- **Synthetic data + smart augmentation works.** We achieved near-human precision without a single real station image.

- **User experience and pipeline automation matters.** By automating the workflow and focusing on astronaut safety, our tool is ready for real missions.

7. Future Work

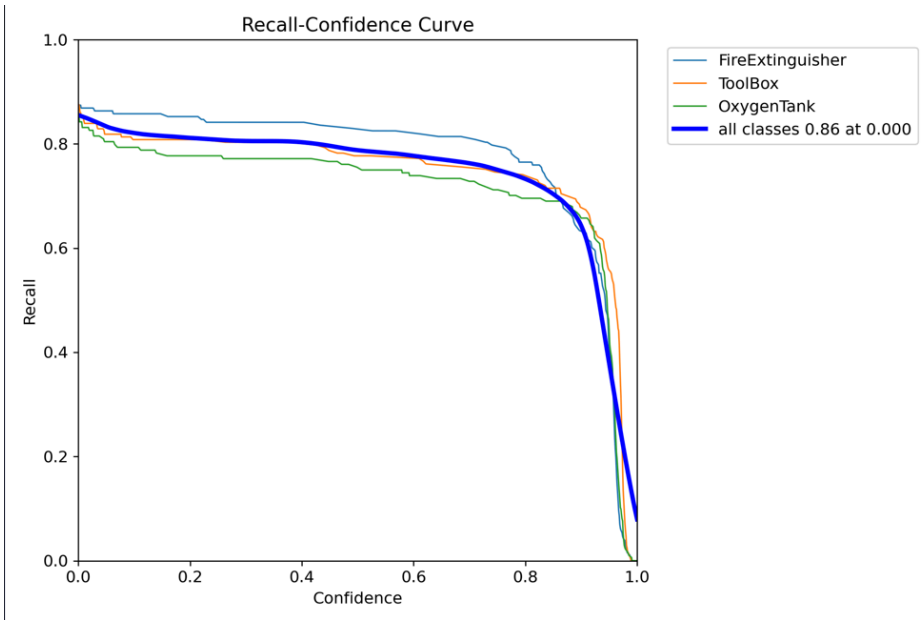
- Bridge the **synthetic-to-real gap** with SAR or real photos when available, using domain adaptation.

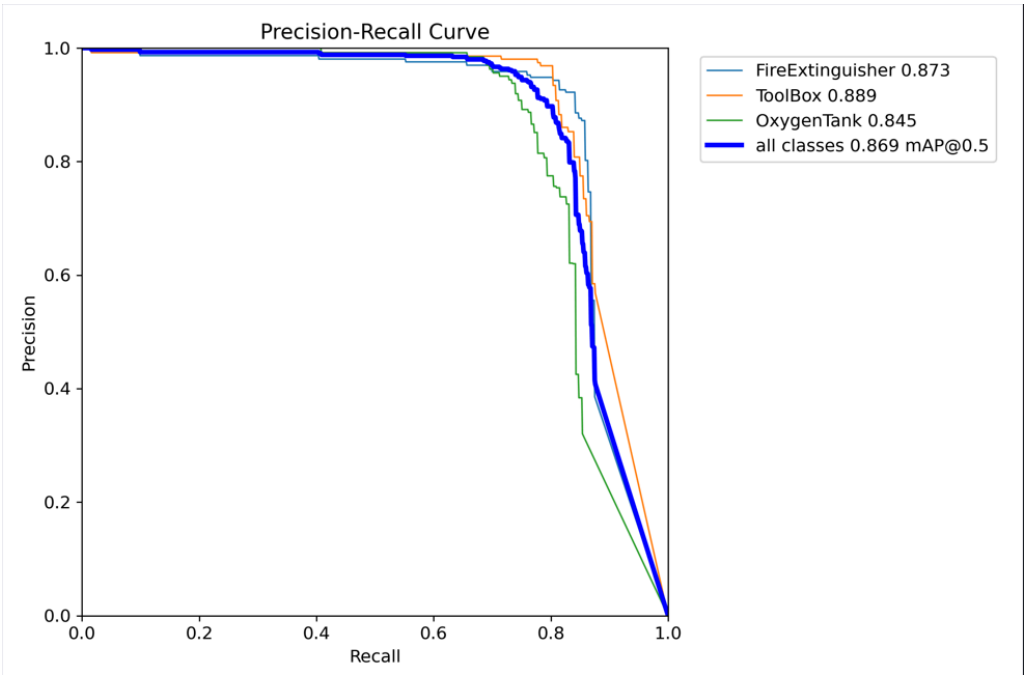
- Expand use: real-time video stream inference, additional safety object categories.

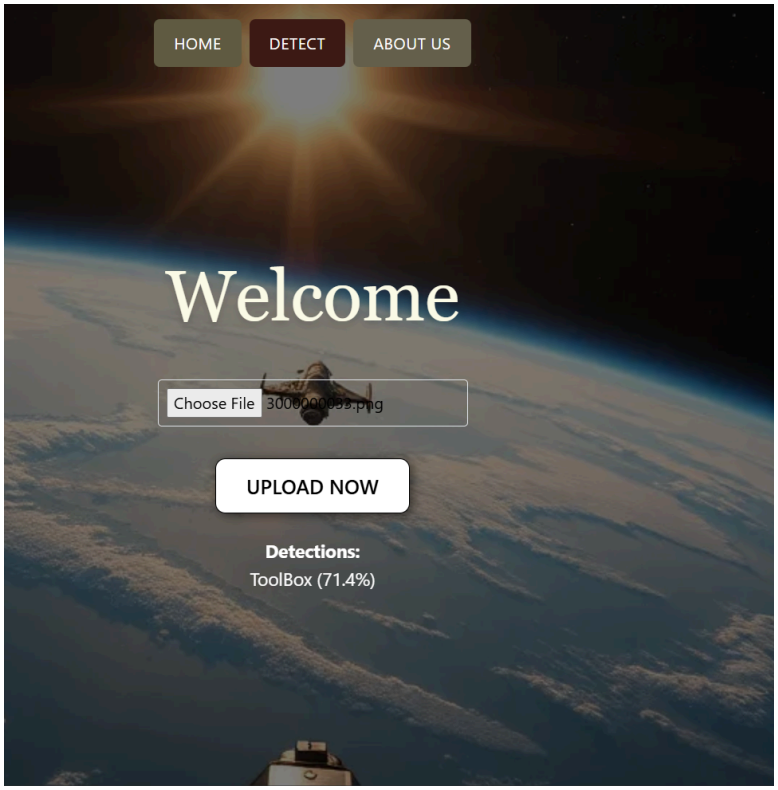
- Integrate with AR displays for live on-board guidance.

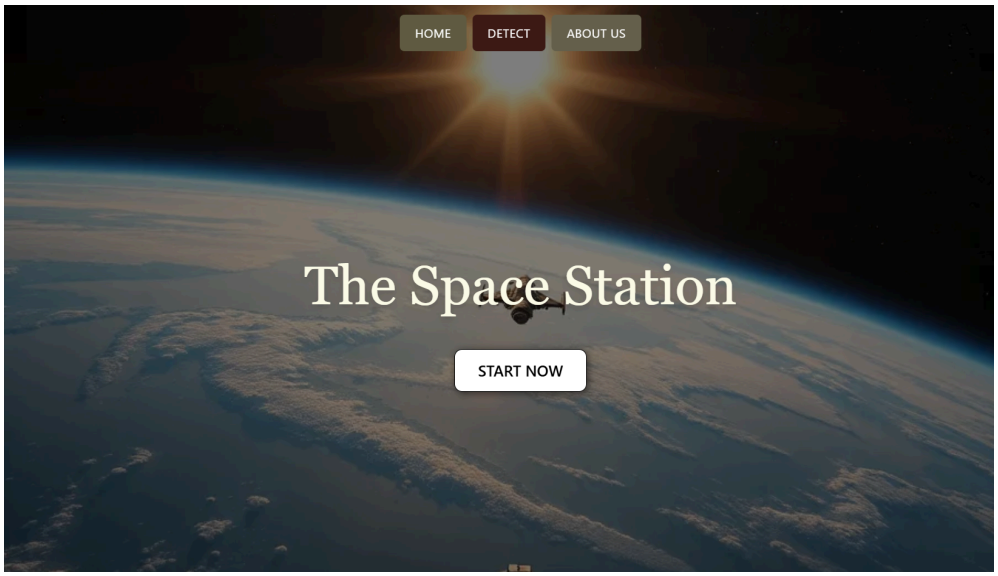
8. References & Appendix

- **GitHub Repo:** https://github.com/YASHICA123/Space-station-object-detection
- **UI Screenshots:**









- **Training logs, example outputs, code scripts:** Included in repo
- **Contact:** yashicamittal12@gmail.com