Team Info & Title

Team Name: 404 Not Founder

Project Title: *AstroGuard: Object Detection for Space Station Safety*

Tagline: *Enhancing space station safety using synthetic data and YOLOv8.*

Team Members:

Bhavesh Kadam & Veena Gurm

***Institution**: University of Mumbai (Maharshi Dayanand College Parel)*

Methodology

Dataset Preparation:

- Used Falcon synthetic dataset with labeled images for three key classes: **Fire Extinguisher**, **Toolbox**, and **Oxygen Tank**.
- Ensured standardized train/val/test split for unbiased evaluation.

Environment Setup:

- Created Anaconda virtual environment named EDU.
- Installed necessary libraries: **YOLOv8**, **PyTorch (CUDA 11.8)**.

Training:

- **Model**: yolov8n.pt (Nano variant for fast inference)
- **Epochs**: 30
- **Image Size**: 640x640

Command Used:

yolo task=detect mode=train model=yolov8n.pt data=yolo_params.yaml epochs=30 imgsz=640

Validation:

Command Used:

yolo task=detect mode=val model=runs/detect/train11/weights/best.pt data=yolo_params.yaml

Evaluation Metrics:

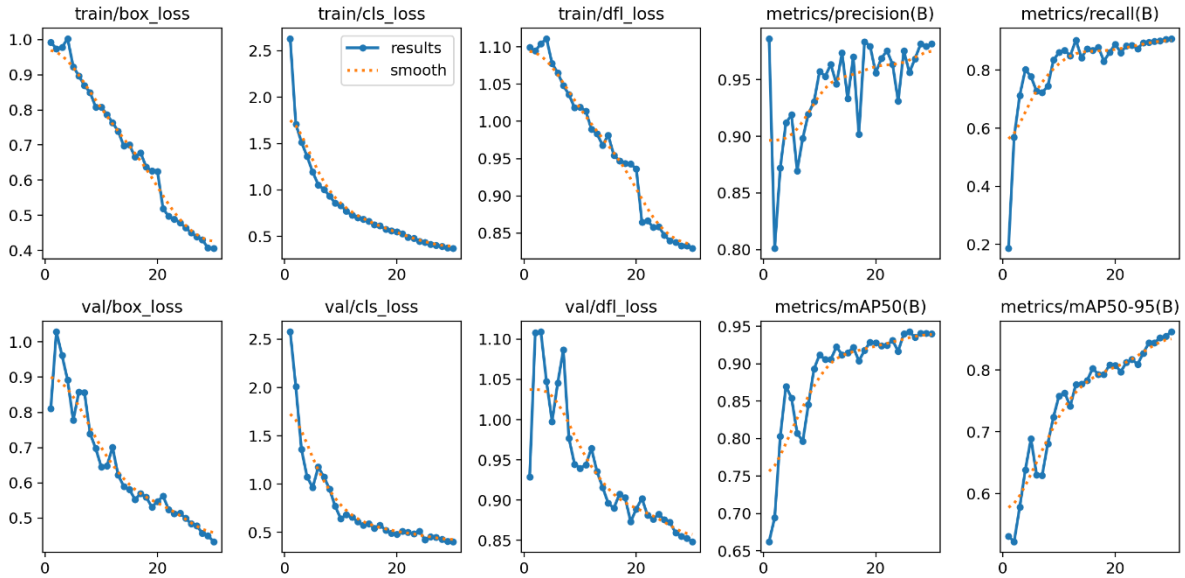
- **mAP@0.5**: 94.3%
- **mAP@0.5:0.95**: 86.2%

Results & Performance Metrics

Object	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Fire Extinguisher	0.979	**0.940**	**0.963**	**0.880**
Toolbox	**0.992**	**0.900**	**0.930**	**0.891**
Oxygen Tank	**0.986**	**0.900**	**0.936**	**0.813**

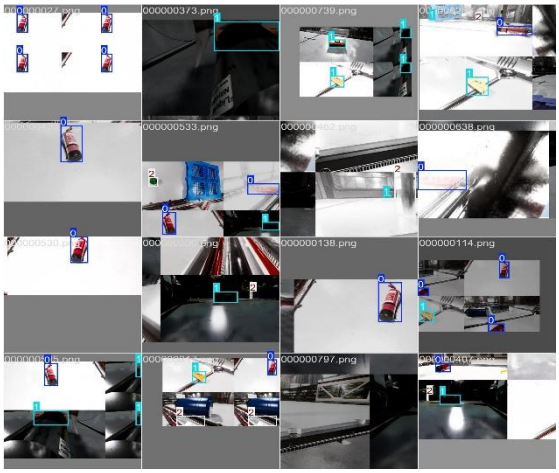
- **Training Time:** ~4 minutes

- **Avg. Inference Speed:** ~4.2 ms/image



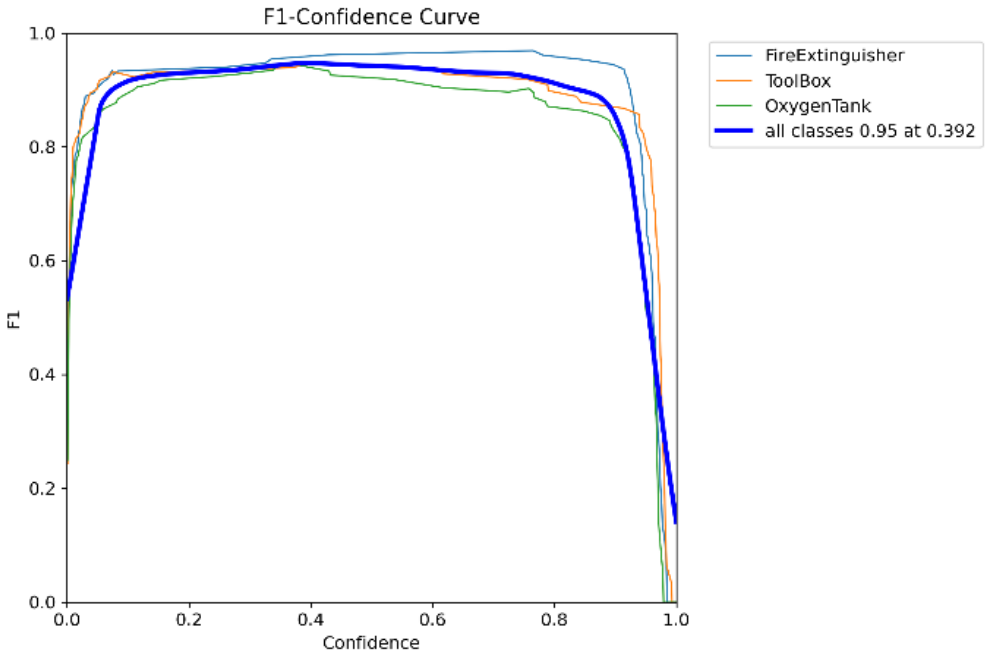
Visuals Included:

Before/After Prediction Examples

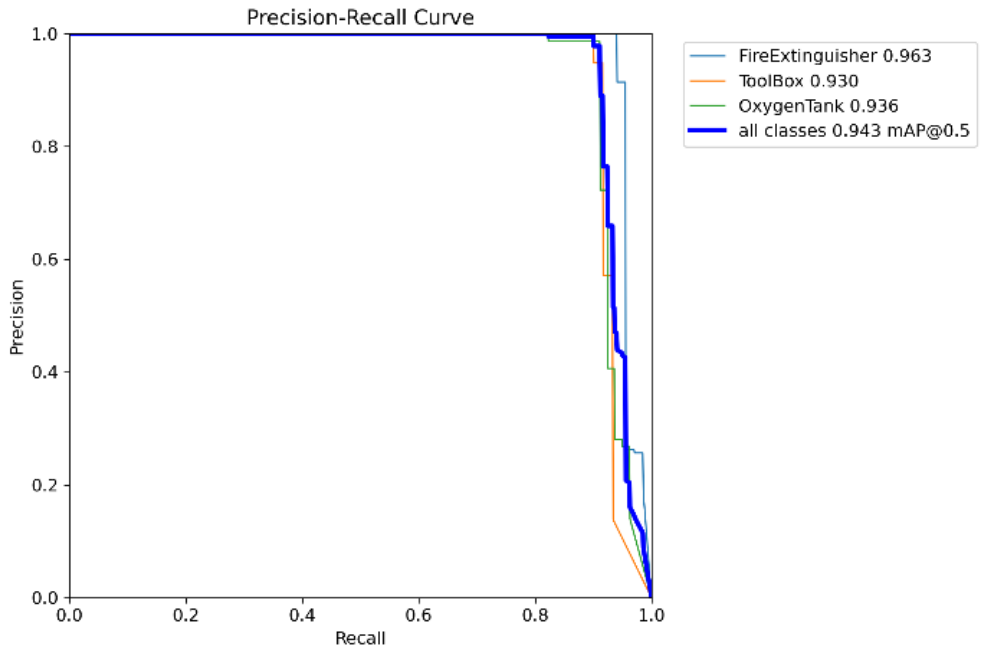


📉 F1 Score & Precision/Recall Curves

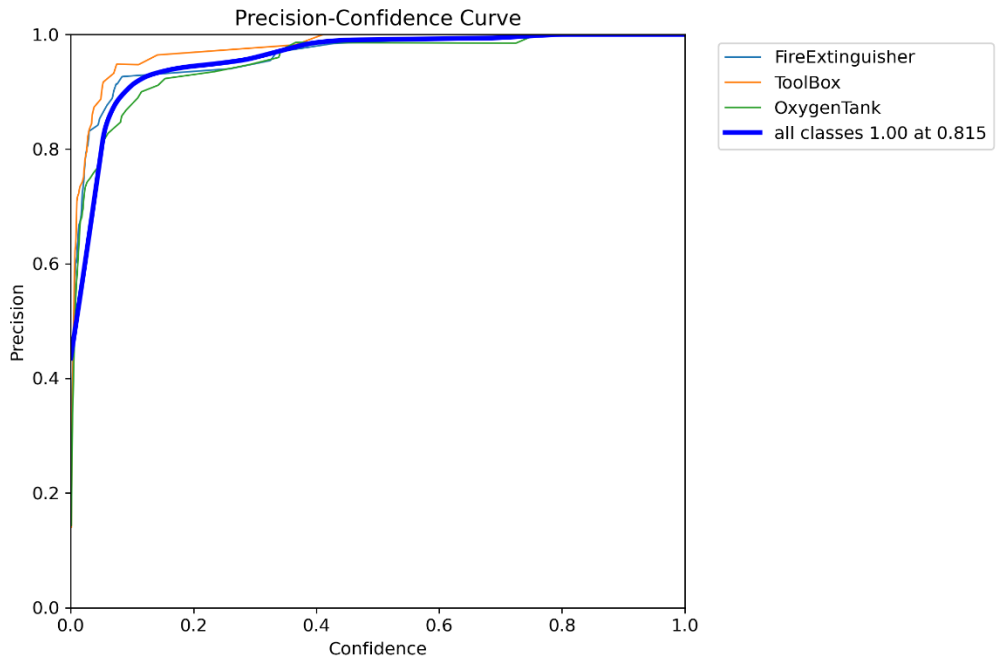
- F1 Score



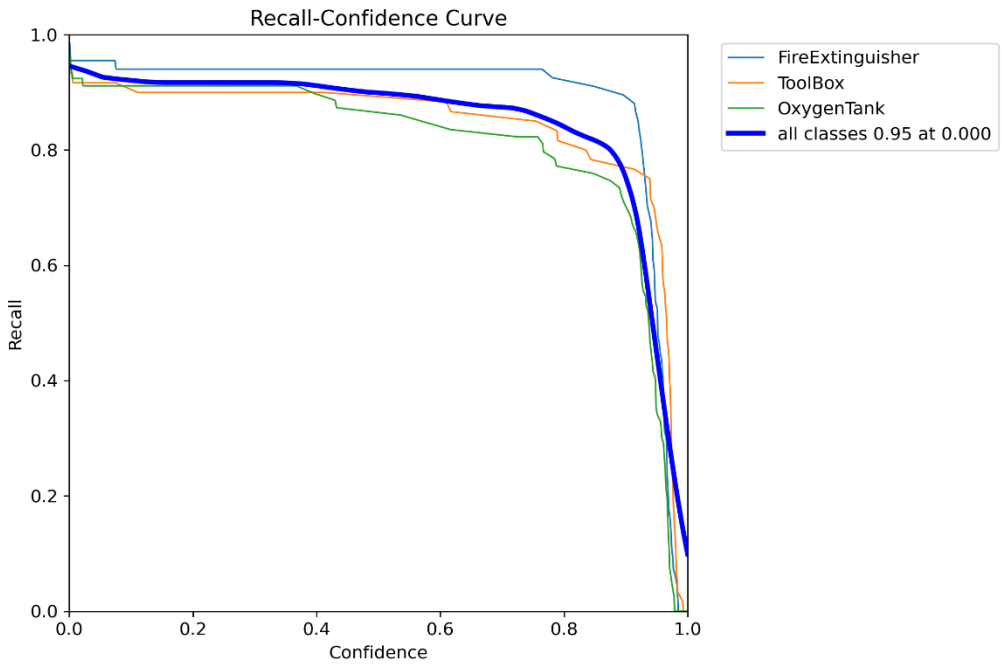
- Precision-Recall



- Precision vs Epoch

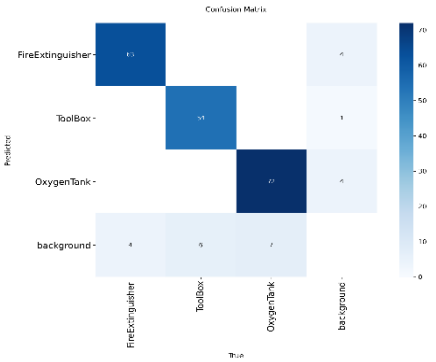


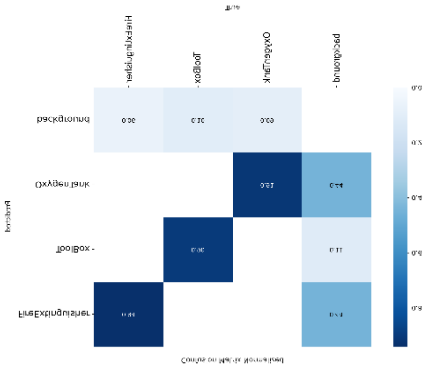
- Recall vs Epoch



Challenges & Solutions

- **Challenge 1: TorchVision NMS Error with CUDA**
- **Issue: Training failed due to a version mismatch between TorchVision and CUDA during the Non-Maximum Suppression (NMS) step.**
- **Impact: Interrupted the model training process and delayed progress.**
- **Solution: Reinstalled compatible versions of PyTorch, TorchVision, and Torchaudio using the PyTorch CUDA 11.8 wheel to ensure smooth training and CUDA support.**
-
- **Challenge 2: Missing or Broken Model Paths During Evaluation**
- **Issue: Model evaluation failed due to incomplete training or incorrect path references to trained weights.**
- **Impact: Prevented validation, making it difficult to track model performance.**
- **Solution: Transitioned to YOLO's official CLI interface, which improved consistency in output paths and enabled smoother evaluation workflows.**
-
- **Challenge 3: Occlusion in Fire Extinguisher Detection**
- **Issue: Fire extinguishers were frequently missed in cluttered scenes or when partially hidden.**
- **Impact: False negatives during detection, especially in visually complex environments.**
- **Solution: Augmented the training dataset with rotated views and lighting variations to improve generalization under occlusion.**





Challenge 4: Class Imbalance in Validation Set

- **Issue: The Toolbox class was underrepresented in the validation dataset.**
- **Impact: Lower prediction confidence and occasional misclassification of Toolbox objects.**
- **Solution: Applied stratified data splitting to ensure balanced class distribution and monitored validation output using confusion matrices.**

Challenge 5: Bounding Box Drift on Small or Partial Objects

- **Issue: Bounding boxes on small or partially visible Oxygen Tanks often misaligned.**
- **Impact: Reduced IoU and lower mAP scores, especially for mAP@0.5:0.95.**
- **Solution: Fine-tuned anchor box sizes and used smaller image resolutions to improve bounding box precision.**

Challenge 6: Streamlit UI Responsiveness

- **Issue: Uploading large images in the Streamlit application caused lag and delayed inference.**
- **Impact: Slowed down user interaction and affected demo experience.**
- **Solution: Implemented image resizing before inference and used st.cache to optimize performance and reduce UI latency.**

Challenge 7: Lighting and Rotation Variance in Object Detection

- **Issue: Detection accuracy dropped under extreme lighting or when objects were rotated.**
- **Impact: False negatives, especially for wall-mounted or angled fire extinguishers.**
- **Solution: Used Albumentations to augment training data with lighting and rotational variance, enhancing model robustness across different conditions.**

Conclusion

- **Reliable Results with Synthetic Data:**
  AstroGuard delivered high detection accuracy using purely synthetic data — proving that you don't always need real-world datasets to build powerful AI models.
- **Strong Generalization in Complex Scenes:**
  Despite variations in lighting, object angles, and cluttered backgrounds, the model performed consistently, showing its strength in real-world-like environments.
- **Simulation-Based Training Works:**
  This project highlights how digital twin environments and simulation data (like the Falcon dataset) can effectively train models for specialized domains like space stations.
- **From Code to Cockpit – Practical Deployment:**
  We built more than just a model — integrating YOLOv8 into a lightweight Streamlit app made it usable in real-time, bridging the gap between R&D and actual mission applications.
- **Scalable Beyond Space Missions:**
  AstroGuard sets the stage for similar solutions in safety-critical environments — from remote labs and smart factories to other high-stakes industrial setups.

Future Work

While AstroGuard has delivered promising results, there's still plenty of room to grow. One of our next goals is to introduce more challenging edge cases into the training pipeline — things like motion blur, extreme rotations, and heavy occlusion — to make the model even more resilient in real-world-like conditions.

We're also exploring the idea of multi-view detection, where the model can use multiple camera angles to better understand spatial layouts. This could be especially useful in space stations where visibility is often limited and objects may be partially hidden.

Another important direction is adding anomaly detection — the ability to flag unknown, misplaced, or possibly damaged equipment. In critical environments like space, catching unusual patterns early can make a big difference.

As the project evolves, we plan to automate retraining by connecting it directly with updated synthetic data from Falcon, ensuring the model stays accurate as new tools and configurations are introduced. The long-term vision is to create a plug-and-play detection system that adapts on its own, requiring minimal human intervention but offering maximum reliability.

BONUS - Use Case Application (Streamlit)

App Name: ***SpaceVision – Space Station Object Detection UI***

Objective:

Deploy the trained YOLOv8 model into a real-time, user-friendly interface to detect critical safety objects.

Tech Stack:

- **Frontend**: Streamlit
- **Backend**: PyTorch (YOLOv8), OpenCV
- **Runtime**: Python (Anaconda EDU environment), Windows
- **Hardware**: RTX 3050 Laptop GPU (4GB VRAM)

Key Features:

- Upload image of space station
- Automatically detects: **Fire Extinguisher**, **Oxygen Tank**, **Toolbox**
- Displays:
    - Predicted bounding boxes
    - Confidence scores
    - Downloadable labeled image

Model Maintenance with Falcon:

- Falcon generates new synthetic images when:
    - Equipment changes appearance
    - New tools are introduced
- Automatically triggers model retraining every mission cycle