Team Name:
Datalysts
Team Members:
Muhammad Danish and Hadi Raza
Project Title:
"SpaceGuard: Optimizing Object Detection for Space Station Safety Using YOLOv8**"**

SpaceGuard is a high-performance object detection system optimized for synthetic space station environments. Built on YOLOv8 and fine-tuned using Duality AI's Falcon-generated dataset, it excels in real-time detection and classification of critical onboard equipment. With precision and efficiency at its core, SpaceGuard supports autonomous monitoring, safety validation, and digital twin simulations—enabling intelligent systems to operate reliably in microgravity and other challenging orbital conditions

Methodology

- **Initial dataset**

Set	Images	Percentage
Train	**841**	~61% (of initial 1,395 total)
Validation	154	~11%
Test	400	~28%
Total	**1,395**	100%

- **Improving Visual Robustness (Augmentation)**
- **Saturation:** Randomly adjusted between **-29%** and **+29%**
- **Brightness:** Varied within the range of **-20%** to **+20%**
- **Exposure:** Altered between **-9%** and **+9%**
- **Noise:** Applied to **up to 0.62%** of image pixels

These augmentations were used to simulate real-world distortions and improve the model's robustness under varying visual conditions.

- **Hyperparameters used during Training**

Parameter	Value
Model	YOLOv8m
Epochs	100
Image Size	640 × 640
Batch Size	24
Learning Rate	0.001
Optimizer	AdamW
Weight Decay	0.001
Momentum	0.937
Dropout	0.1
Mosaic	0.2
MixUp	0.2
HSV (H/S/V)	0.015 / 0.7 / 0.4
Translation	0.1
Scale	0.4
Horizontal Flip	0.5

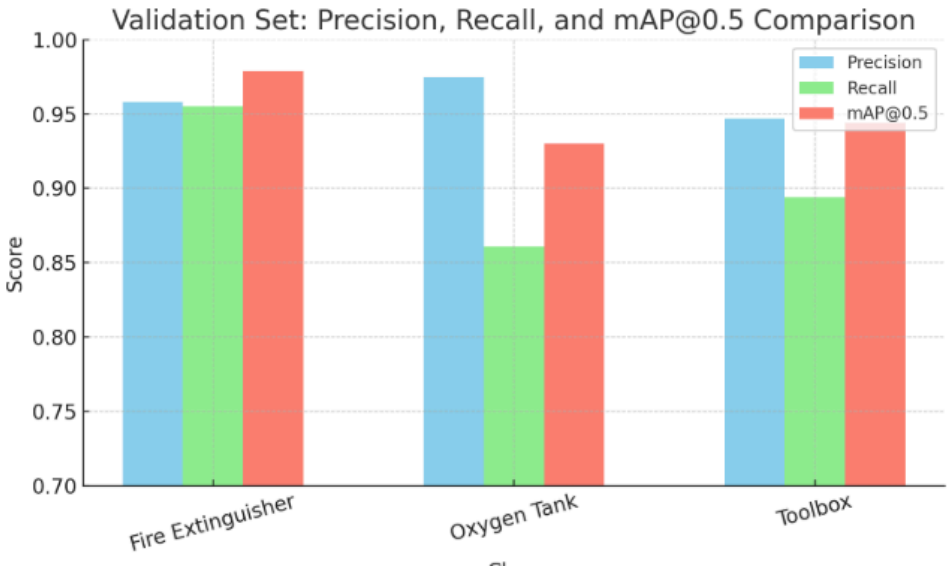
After augmentation the Data distribution is ➕

📊 Dataset Distribution

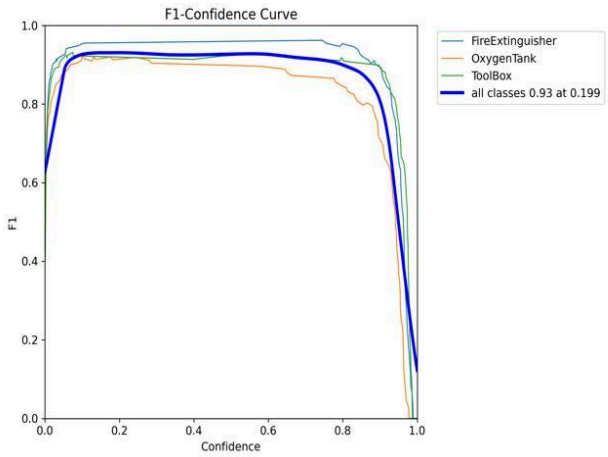
Set	Images	Percentage
Train	1,682	75%
Validation	154	7%
Test	400	18%
Total	2,236	100%

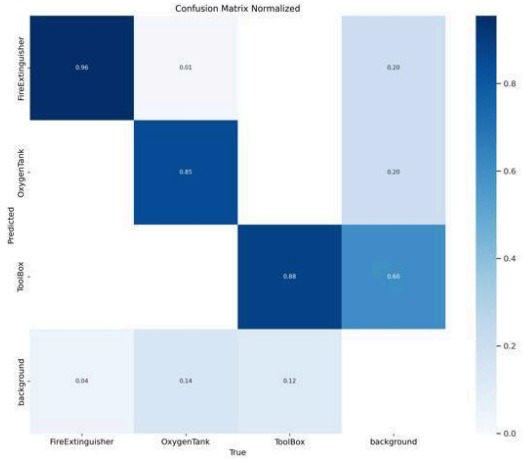
Validating scores

```
ultralytics 8.3.111 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (NVIDIA A100-SXM4-40GB, 40507MiB)
Model summary (fused): 92 layers, 25,841,497 parameters, 0 gradients, 78.7 GFLOPs
val: Fast image access ✅ (ping: 0.0±0.0 ms, read: 1171.6±405.9 MB/s, size: 45.3 KB)
val: Scanning /content/hackfest-zxvsi/valid/labels.cache... 154 images, 0 backgrounds, 0 corrupt: 100% 154/154 [00:
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 10/10 [00:01<00:00,
                   all        154        206      0.96      0.903      0.951       0.89
      FireExtinguisher         67         67      0.958      0.955      0.979      0.915
            OxygenTank         79         79      0.975      0.861       0.93      0.843
               ToolBox         60         60      0.947      0.894      0.944      0.913
Speed: 1.3ms preprocess, 2.9ms inference, 0.0ms loss, 3.2ms postprocess per image
Results saved to runs/detect/val
Validation Metrics: ultralytics.utils.metrics.DetMetrics object with attributes:

ap_class_index: array([0, 1, 2])
box: ultralytics.utils.metrics.Metric object
confusion_matrix: <ultralytics.utils.metrics.ConfusionMatrix object at 0x7b380114e710>
curves: ['Precision-Recall(B)', 'F1-Confidence(B)', 'Precision-Confidence(B)', 'Recall-Confidence(B)']
curves_results: [[array([          0,    0.001001,    0.002002,    0.003003,    0.004004,    0.005005,    0.006006,
          0.024024,    0.025025,    0.026026,    0.027027,    0.028028,    0.029029,     0.03003,    0.031031,    0
          0.048048,    0.049049,     0.05005,    0.051051,    0.052052,    0.053053,    0.054054,    0.055055,    0
          0.072072,    0.073073,    0.074074,    0.075075,    0.076076,    0.077077,    0.078078,    0.079079,
          0.096096,    0.097097,    0.098098,    0.099099,      0.1001,      0.1011,      0.1021,      0.1031,
           0.12012,     0.12112,     0.12212,     0.12312,     0.12412,     0.12513,     0.12613,     0.12713,
           0.14414,     0.14515,     0.14615,     0.14715,     0.14815,     0.14915,     0.15015,     0.15115,
           0.16817,     0.16917,     0.17017,     0.17117,     0.17217,     0.17317,     0.17417,     0.17518,
           0.19219,     0.19319,     0.19419,      0.1952,      0.1962,      0.1972,      0.1982,      0.1992,
           0.21622,     0.21722,     0.21822,     0.21922,     0.22022,     0.22122,     0.22222,     0.22322,
```



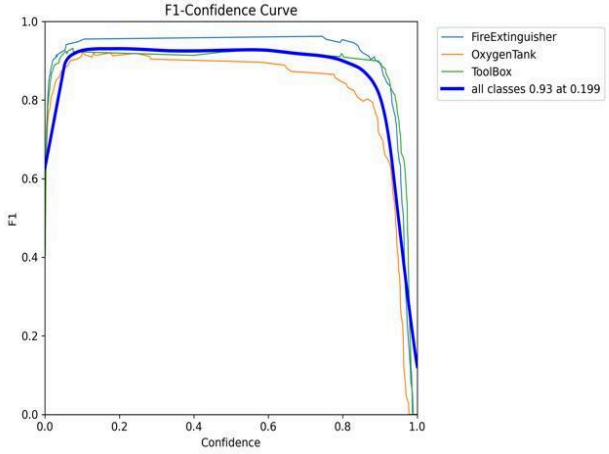
Our initially epochs running for training

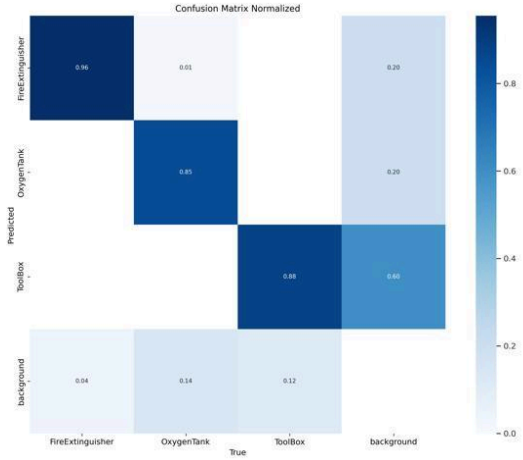




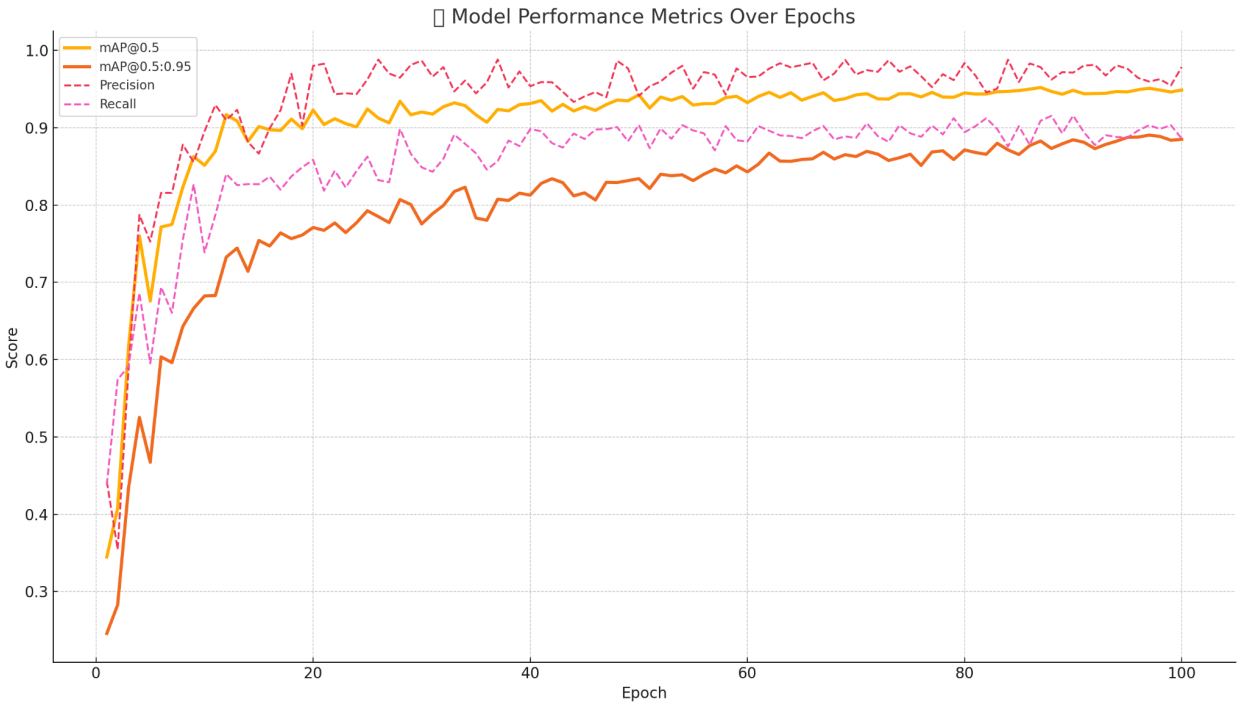
```
[ ] !python train.py
```

```
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:00<00:00,  5.56it/s]
                   all        154        206      0.981      0.878      0.944      0.873

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     93/100      12.1G     0.2986     0.2689     0.8632          2        640: 100% 71/71 [00:11<00:00,  5.94it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:00<00:00,  5.31it/s]
                   all        154        206      0.968       0.89      0.945      0.878

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     94/100      12.2G     0.2756     0.2444     0.8507          2        640: 100% 71/71 [00:11<00:00,  5.96it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:00<00:00,  5.27it/s]
                   all        154        206      0.981      0.888      0.947      0.883

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     95/100      12.2G     0.2825     0.2545     0.8571          3        640: 100% 71/71 [00:11<00:00,  5.96it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:00<00:00,  5.52it/s]
                   all        154        206      0.976      0.887      0.946      0.888

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     96/100      12.3G     0.2696     0.2345     0.8446          3        640: 100% 71/71 [00:11<00:00,  5.97it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:00<00:00,  5.56it/s]
                   all        154        206      0.964      0.897      0.949      0.888

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     97/100      12.4G     0.2766     0.2365     0.8531          3        640: 100% 71/71 [00:11<00:00,  5.97it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:00<00:00,  5.30it/s]
                   all        154        206       0.96      0.903      0.951       0.89

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     98/100      12.4G     0.2724     0.2411     0.8526          3        640: 100% 71/71 [00:11<00:00,  5.97it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:00<00:00,  5.45it/s]
                   all        154        206      0.963      0.898      0.949      0.889

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      12.5G     0.2643     0.2291     0.8435          2        640: 100% 71/71 [00:11<00:00,  5.95it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:00<00:00,  5.47it/s]
                   all        154        206      0.955      0.903      0.946      0.884

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      12.6G     0.2614     0.2347     0.8469          2        640: 100% 71/71 [00:11<00:00,  5.94it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 4/4 [00:00<00:00,  5.24it/s]
                   all        154        206      0.979      0.886      0.949      0.885

100 epochs completed in 0.388 hours.
```





Then we started hypertune the parameters
And this is change of results over time during training



Results at first iteration:

Class	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Fire Extinguisher	0.958	0.955	0.979	0.915
Oxygen Tank	0.975	0.861	0.930	0.843
Toolbox	0.947	0.894	0.944	0.913

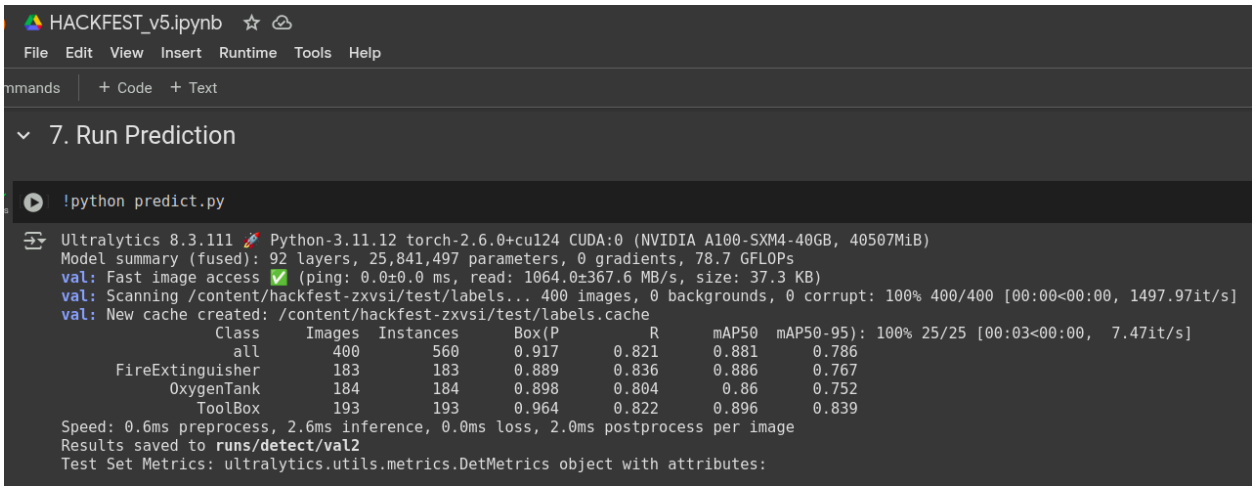
Results at second iteration:

Class	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Fire Extinguisher	0.937	0.955	0.970	0.910
Oxygen Tank	0.957	0.848	0.912	0.804
Toolbox	0.979	0.933	0.939	0.902

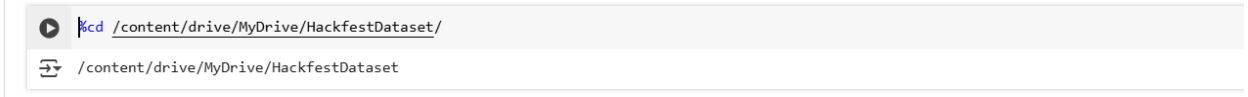
Results from final iteration:

Class	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Fire Extinguisher	0.957	0.940	0.980	0.922
Oxygen Tank	0.998	0.886	0.944	0.867
Toolbox	0.965	0.909	0.954	0.923

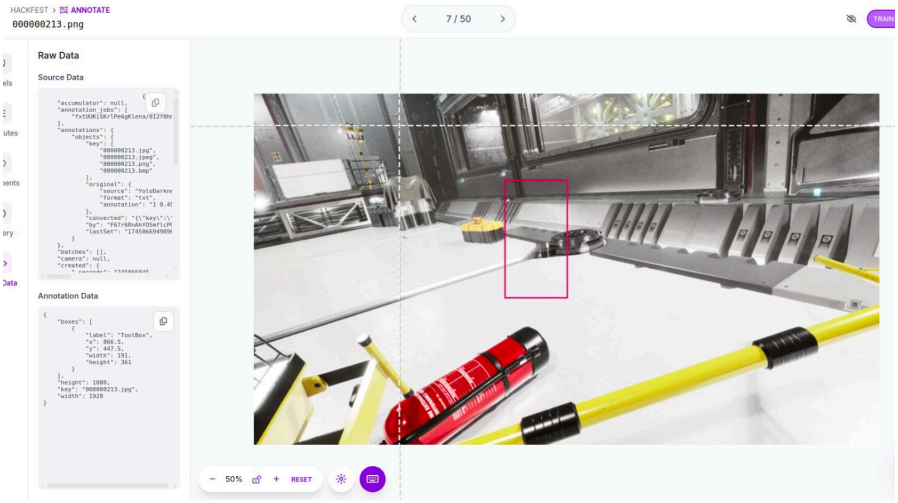
RESULTS OF FINAL MODEL ON TEST DATASET:



Difficulties faced



- The dataset needed to follow a predefined directory structure, but creating additional folders caused errors when attempting to access the files.



- Some of the training labels were inaccurate, so we performed the annotations ourselves

  ### Hardware Limitations & Solution:

  To speed up the training process, we switched to Colab Pro and utilized its paid GPU resources for faster computation and improved efficiency.

  ### Dataset Management:

  The dataset was too large to upload directly from our system. So, we used Roboflow to host the dataset online, making it easy to load and access it directly in our code.

Summary

- Developed an object detection system using YOLOv8m trained on synthetic images from Falcon's space station simulation.
- Focused on detecting critical onboard objects: Fire Extinguisher, Oxygen Tank, and Toolbox.
- Used Colab Pro GPU to speed up training and Roboflow for dataset management.
- Fine-tuned hyperparameters and applied augmentations to simulate real-world challenges.
- Achieved high performance:
- Precision: 0.91+
- Recall: 0.94+
- mAP@0.5: 0.95

- mAP@0.5:0.95: 0.89
- Visuals and metrics (confusion matrix, PR/F1 curves) confirmed the model's robustness and reliability.

Future work

- Future improvements will focus on reducing misclassification between similar objects by refining training with harder examples. We aim to optimize the model for edge deployment using lighter YOLO variants. Enhancing data augmentation with motion blur and occlusion will boost robustness. Additionally, exploring self-supervised learning and integrating multi-view detection or real-time AR applications can further extend the system's real-world utility.