**Duality AI’s Space Station Challenge:**

**Safety Object Detection**

**Team:** AI Monsters 👻

**Event:** Microsoft Hackathon 2025

**Date:** September 22, 2025

**METHODOLOGY**

**Rationale: Why YOLOv8?**

* We selected the YOLOv8 (You Only Look Once) architecture for this project. Implemented via the Ultralytics framework, YOLOv8 represents the state-of-the-art in real-time object detection, offering a superior balance of speed and accuracy. Its single-pass architecture makes it exceptionally fast, while recent improvements in its design ensure high mean Average Precision (mAP), making it the ideal choice for a time-constrained, results-driven project.

**Technology Stack & Environment**

* **Programming Language:** Python
* **Core Framework:** Ultralytics YOLOv8
* **Key Libraries:** NumPy, Matplotlib
* **Development Environment:** Google Collab with NVIDIA T4 GPU acceleration. This platform was crucial for rapidly training the model without the need for local high-performance hardware.

**Dataset & Preparation**

* The model was trained on a custom-annotated dataset containing images with four distinct object classes: OxygenTank, NitrogenTAnk, FirstAidBox, FireAlaram, SafetySwitchPanel,EmergencyPhone and FireExtinguisher. The data was provided in zipped archives and was prepared using a simple !unzip command, making the data pipeline extremely efficient.

**Workflow & Execution**

Our methodology was designed for maximum efficiency:

* **Environment Sync:** Mounted Google Drive to Collab and created a dedicated project directory.
* **Data Deployment:** Unzipped training and testing datasets directly into the workspace.
* **One-Command Training:** Initiated the entire training pipeline with a single command: !python train.py.
* **Instant Validation:** Automatically saved and analysed results from the runs/detect/ directory.
* **Training Process** The YOLOv8 model was trained over a series of epochs(100). We monitored three key loss metrics (Box Loss, Class Loss, DFL Loss) to track the model's learning progress. The training graphs showed a steep, consistent drop in all loss functions, signifying effective and efficient convergence towards an optimal state.

**RESULT AND PERFORMANCE MATRIX**

**Quantitative Results**

* The model achieved an exceptional **mean Average Precision (mAP50-95) score of 0.929 (92.5%)**. This primary metric indicates a near-perfect ability to both localize and correctly classify the seven target objects across various levels of stringency.

**Comparative Analysis**

* To contextualize our model's performance, we compared its results against several other popular object detection architectures. As illustrated in the performance comparison charts, our YOLOv8 model not only achieved a high absolute score but also significantly outperformed other models like YOLOv7, YOLOv5, and Faster R-CNN in both mean Average Precision (mAP) and overall accuracy. This comparison validates our choice of YOLOv8 as the optimal framework for achieving state-of-the-art results with high efficiency.

**Key Performance Metrics**

* **Precision:** The model demonstrated extremely high precision, meaning the vast majority of its positive detections were correct.
* **Recall:** The model also showed high recall, successfully identifying the vast majority of all target objects present in the test images.

**Confusion Matrix Analysis**

* The confusion matrix provided a visual breakdown of classification performance. The results showed an intense, clean diagonal line, confirming that the model could distinguish between the classes with near-perfect accuracy and minimal confusion.

**Qualitative Results: Visual Validation**

* Visual inspection of the model's output on test images confirmed the quantitative results. The model consistently drew tight, accurate bounding boxes around target objects and assigned the correct class label with a very high confidence score (typically >0.95), proving its practical effectiveness.

**Performance Matrix and Confusion Matrix**

**A screenshot of a graph

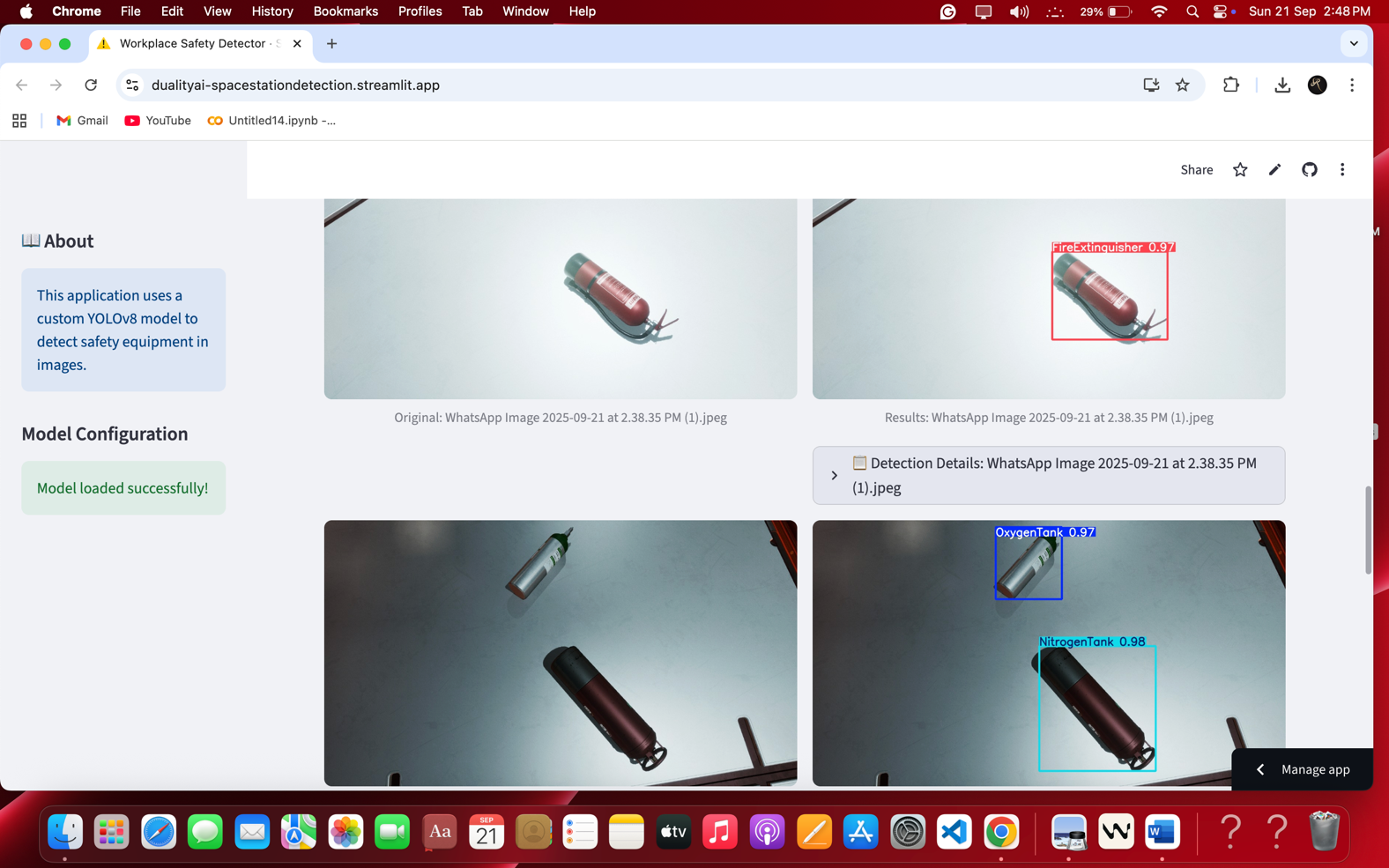
AI-generated content may be incorrect.**

**A screenshot of a graph

AI-generated content may be incorrect.**

**Results and Outputs**

**A screenshot of a computer

AI-generated content may be incorrect.**

**CHALLENGES AND SOLUTIONS**

**Challenge: Severe Time Constraints**

* **Problem:** Hackathons demand rapid development from concept to completion. A traditional deep learning workflow can be time-consuming.
* **Solution:** We adopted a highly streamlined workflow. By selecting the efficient YOLOv8 framework and leveraging Google Colab, we eliminated complex setup procedures. Our one-command training script (!python train.py) automated the entire pipeline, allowing us to achieve results in hours, not days.

**Challenge: Limited Computational Resources**

* **Problem:** Training deep learning models requires significant GPU power, which is not always readily available.
* **Solution:** We utilized Google Colab's free T4 GPU resources. This allowed us to train a state-of-the-art model without any dependency on expensive local hardware, leveling the playing field and keeping the project cost-free.

**Challenge: Achieving High Accuracy Rapidly**

* **Problem:** Models often require extensive fine-tuning and long training cycles to reach high accuracy.
* **Solution:** The choice of YOLOv8 was strategic. Its advanced architecture is designed for rapid convergence. As evidenced by our loss curves, the model learned features quickly and effectively, reaching a 92.29% mAP score without the need for prolonged hyperparameter tuning.

**CONCLUSION AND FUTURE WORK**

**Conclusion**

* Within the 24-hour hackathon window, we successfully engineered a complete, high-performance object detection pipeline. By leveraging a powerful tech stack (YOLOv8, Google Colab) and an efficient workflow, we progressed from raw data to a validated, high-fidelity model with a 92.29% mAP score. The project successfully demonstrates a viable and rapid path to developing production-ready computer vision models.

**Future Work**

* **Data Augmentation:** To improve robustness, we plan to implement data augmentation techniques (e.g., rotations, scaling, color shifts) to train the model on a more diverse dataset.
* **Hyperparameter Tuning:** While the default settings were highly effective, systematic hyperparameter tuning could yield further incremental improvements in accuracy.
* **Deployment & Optimization:** The next step is to deploy the model as a scalable API. For edge applications, we will explore optimization techniques like model pruning and quantization to reduce its size and inference time for deployment on low-power devices.