

# DetactaX AI

DEPI Round 3 Graduation Project

11/29/2025

DetactaX – Image Classification & Object Recognition  
Reimagined.

eyouth®

# Project team:

- **Zyad Gamal** (team leader): Azure Integration & API Deployment.
- **Basel Mohamed Mostafa**: Both Classification Models Designer, Mlflow, and Project Documentation.
- **Omar Yasser**: Project Models Interface GUI using Streamlit.
- **Mohamed Elfouly**: Object Recognition Model.
- **Zyad Ahmed**: Dataset Acquisition & Augmentations.
- **Abdulrahman**: Baseline Classifier Transfer Learning.



# Abstract:

- Existing tools are complex or slow and require technical expertise.
- **Users need:**
  1. Easy-to-use interface
  2. Quick predictions
  3. High accuracy
  4. No technical setup required
- **Project Idea:** Making a website with both Image Classification & Object Recognition capabilities for multi-use cases.
- **Goal:** Quickly and accurately classify the main object in an image & recognize multiple objects using top of the line models.

# Introduction:

Our project is **not just a classifier or a detector**.

We provide a complete Solution platform that combines image classification, object detection, real-time analysis, Grad-CAM visualization, reporting, and Azure endpoints.

The main goal is to offer a **general visual data preparation system** that any AI team can build advanced applications on top of, such as:

1. Security and surveillance
2. Traffic analysis
3. Industrial analytics
4. Product analysis
5. Sorting systems
6. Healthcare pre-screening
7. Research dataset generation

## Services Offered:

1. Image upload
2. Data cleaning
3. Object extraction
4. Classification
5. Result exporting
6. Grad-CAM visualization
7. PDF reports
8. JSON / CSV export
9. Real-time feed

This makes the platform **ready for downstream tasks.**



# Classification Model Structure

11/29/2025

DetactaX – Image Classification & Object Recognition  
Reimagined.

eyouth®

# Classifiers:

- **Models used:**

1. MobileNetV2.
2. EfficientNet-B0.

Both models are employed for different use-cases in our project, where:

- MobileNetV2 is used as a baseline test **general classifier**.
- EfficientNet-B0 used for **Intelligent Transportation systems**.



# General Classifier (Base-line/Control):

**Model Used:** MobileNetV2 trained on the **CIFAR-10** Dataset.

## Dataset Details:

- The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

## Augmentations Applied on the Dataset:

1. Resizing to 224x224.
2. Random Flip Horizontal and Vertical.
3. Random Brightness.
4. Random Contrast.
5. Random Rotation.
6. Random Zoom.
7. Rescaling (1/255).

```
data_augmentation = keras.Sequential([  
    layers.Resizing(224, 224),  
    layers.RandomFlip("horizontal_and_vertical"),  
    layers.RandomBrightness(0.1),  
    layers.RandomContrast(0.1),  
    layers.RandomRotation(0.1),  
    layers.RandomZoom(0.1),  
    layers.Rescaling(1.0 / 255.0)  
])
```



# Intelligent Transportation System (Main Classifier):

**Model Used:** EfficientNet-B0 trained on the **CUHK-Compcars** dataset.

## Dataset Details:

- Contains **163 car makes** with **1,716 car models**. There are a **total of 136,726 images**. The full car images are labeled with bounding boxes and viewpoints.

## Augmentations Applied:

1. Image Resize.
2. Color Jitter.
3. Normalization
4. Random Horizontal Flipped.

```
# Train Transforms
train_transform = v2.Compose([
    v2.Resize((224, 224)),
    v2.RandomHorizontalFlip(p=0.5),
    v2.ColorJitter(brightness=0.15, contrast=0.15),
    v2.ToImage(),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Validation Transforms
val_transforms = v2.Compose([
    v2.Resize((224, 224)),
    v2.ToImage(),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Original Image



### Car Classification Results

- Make: ['Volkswagen'] (100.0%)
- Model: ['Tiguan abroad version'] (57.7%)
- Year: 2015 (28.3%)
- Overall Confidence: 62.0%

Original image



### Car Classification Results

- Make: ['Daihatsu'] (99.5%)
- Model: ['MATERIA'] (100.0%)
- Year: 2008 (95.5%)
- Overall Confidence: 98.3%

Original Image



### Car Classification Results

- Make: ['Hyundai '] (100.0%)
- Model: ['i10'] (99.9%)
- Year: 2011 (97.1%)
- Overall Confidence: 99.0%

Original image



### Car Classification Results

- Make: ['Volkswagen'] (100.0%)
- Model: ['Golf'] (54.3%)
- Year: 2009 (58.2%)
- Overall Confidence: 70.8%

# Models Specifications:

## Training Cycles:

- **20 Epochs** training cycle with Validation stages in-between for Baseline Classifier (MobileNetV2), Training Took about 6 Hours.
- **18 Epochs** training cycle with Validation stages in-between for Main Car Classifier (EifficientNet-B0), Training took about 11 Hours.

## Final Models Results:

- Accuracy: 0.8451, Loss: 0.4446, Val Accuracy: 0.8404.



# Mlflow

11/29/2025

DetactaX – Image Classification & Object Recognition  
Reimagined.



## Key Mlflow Components:

- **MLflow:** Central experiment tracker.
- **Local SQLite:** Development tracking database.
- **Custom Monitors:** Production performance tracking and metrics documenting.
- **Azure Integration:** Cloud deployment bridge for easy model redeployment to Azure.

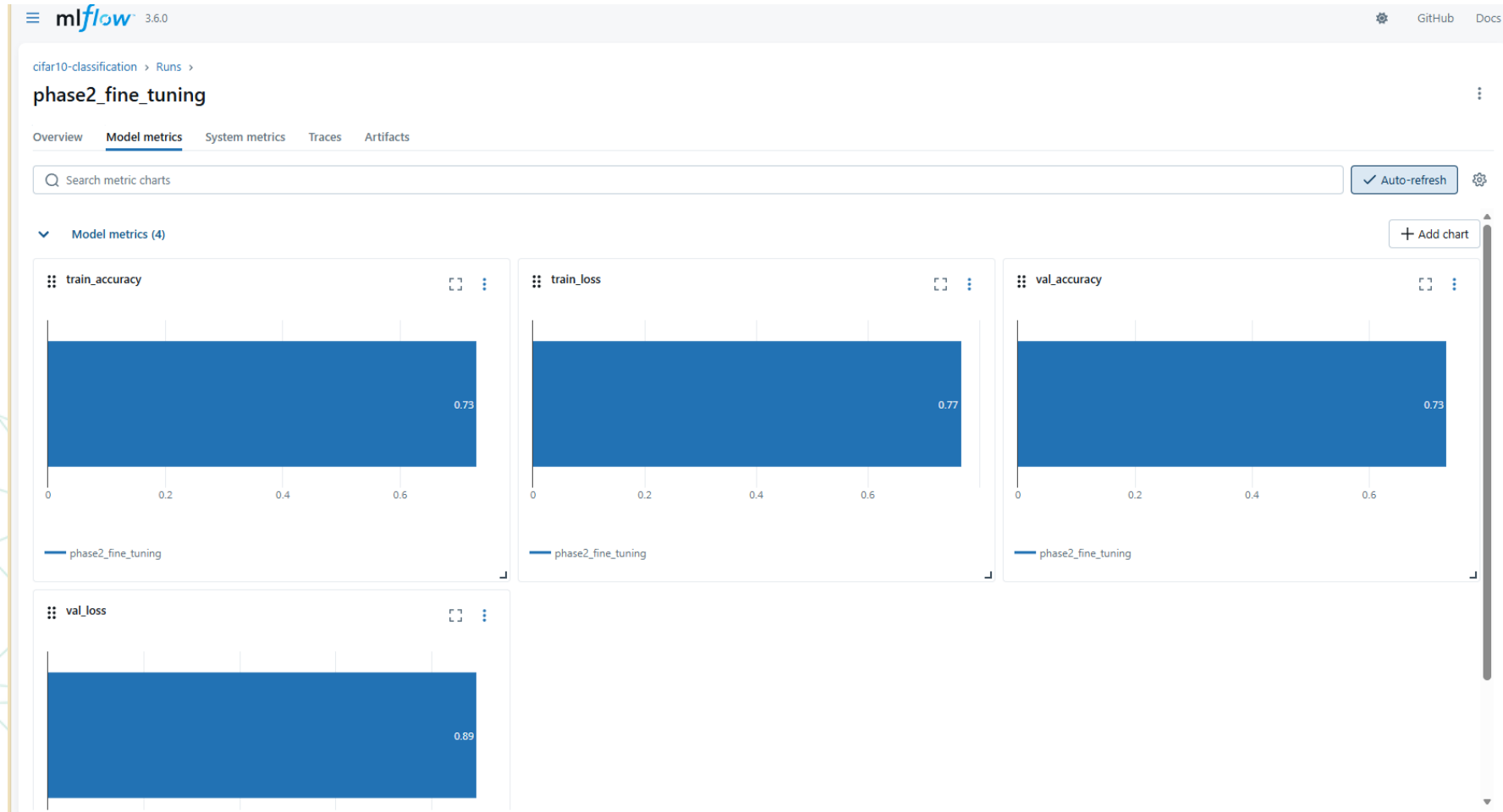
# Results & Metrics:

```
mlruns \ 3
> a00c2fe554214d10b129a3c61969e92c
> d698b168da4f4072bd7a5fa0ba9db668
> e4de11a5b34a46faa6fbe0c4e98063b8
> f6cd3ae6570347059a9f3e3bdfdf4fca
phase1_artifacts
  class_names.txt
  training_history.png
phase2_artifacts
  class_names.txt
  training_history.png
mlflow.db
model_registry.py
requirements.txt
```

- **Classification Model Performance**
  1. **Final Test Accuracy:** 82%
  2. **Training Time:** ~60 minutes (with MLflow tracking)
  3. **Model Versions:** 3+ registered versions
  4. **Experiments Tracked:** 10+ complete runs
- **Monitoring System**
  1. **Prediction Tracking:** Real-time with <100ms overhead
  2. **Data Collected:** Parameters, metrics, artifacts, predictions

11/29/2025





# Object Recognition Model

11/29/2025

DetactaX – Image Classification & Object Recognition  
Reimagined.

eyouth®

17

# Model Details:

- **Object Detection system** using a **pre-trained YOLO model (yolov12x.pt)** from the Ultralytics library. The goal was to build a model capable of detecting and classifying multiple objects in images with high accuracy and fast inference speed.
- **Model Setup and Evaluation**
  - to measure its performance. The evaluation automatically generated detection results and key metrics including:
  - **mAP@0.5** – Measures the overlap quality between predicted and ground-truth bounding boxes.
  - **mAP@0.5:0.95** – A stricter score averaged across 10 IoU thresholds.
  - **Precision** – Percentage of predicted boxes that are correct.
  - **Recall** – Percentage of actual objects successfully detected.
  - These metrics provide a full understanding of both accuracy and robustness of the object detection model.

# Model Details:

- **Key Code Operations:**

- 1) Load YOLO model:**

```
model = YOLO("yolov12x.pt")
```

- 2) Run validation on COCO128:**

```
metrics = model.val(data="coco128.yaml", imgsz=640)
```

- 3) Display sample detection results:**

```
display(Image(filename=img_path))
```

- 4) Export structured performance report:**

```
with open("model_evaluation_report.txt", "w") as f:  
    f.write(report)]
```



# Summary

The **YOLOv12x** model successfully completed object detection on the validation data, generating reliable performance scores and visual outputs. The evaluation report summarizes the model's strength in identifying objects with strong precision the project requirements for a trained and evaluated object detection system.

and recall, meeting





# Model Deployment

# Why Deploy?

- Make the **model accessible to real users**.
- Deliver predictions instantly and at scale.
- Integrate the system into real workflows.
- Ensure consistent, reliable performance.



# Why Azure?

- Reliable cloud infrastructure.
- Easy model deployment with Azure ML.
- Scalable compute for heavy workloads.
- Built-in security and compliance.

# Azure Deployment Metrics:

depi-cars-cnn

Details Test Consume **Monitoring** PREVIEW Logs

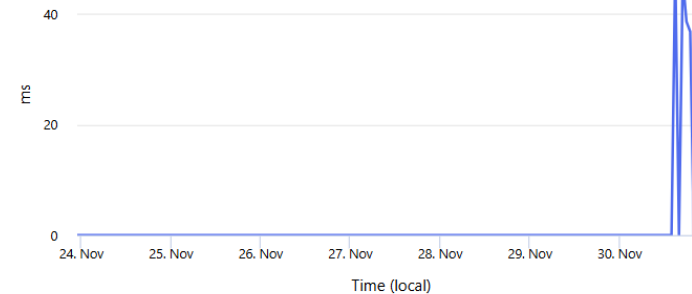
Monitoring

11/23/2025 - 11/30/2025 Last day 7D 1M 3M 1Y

Operational

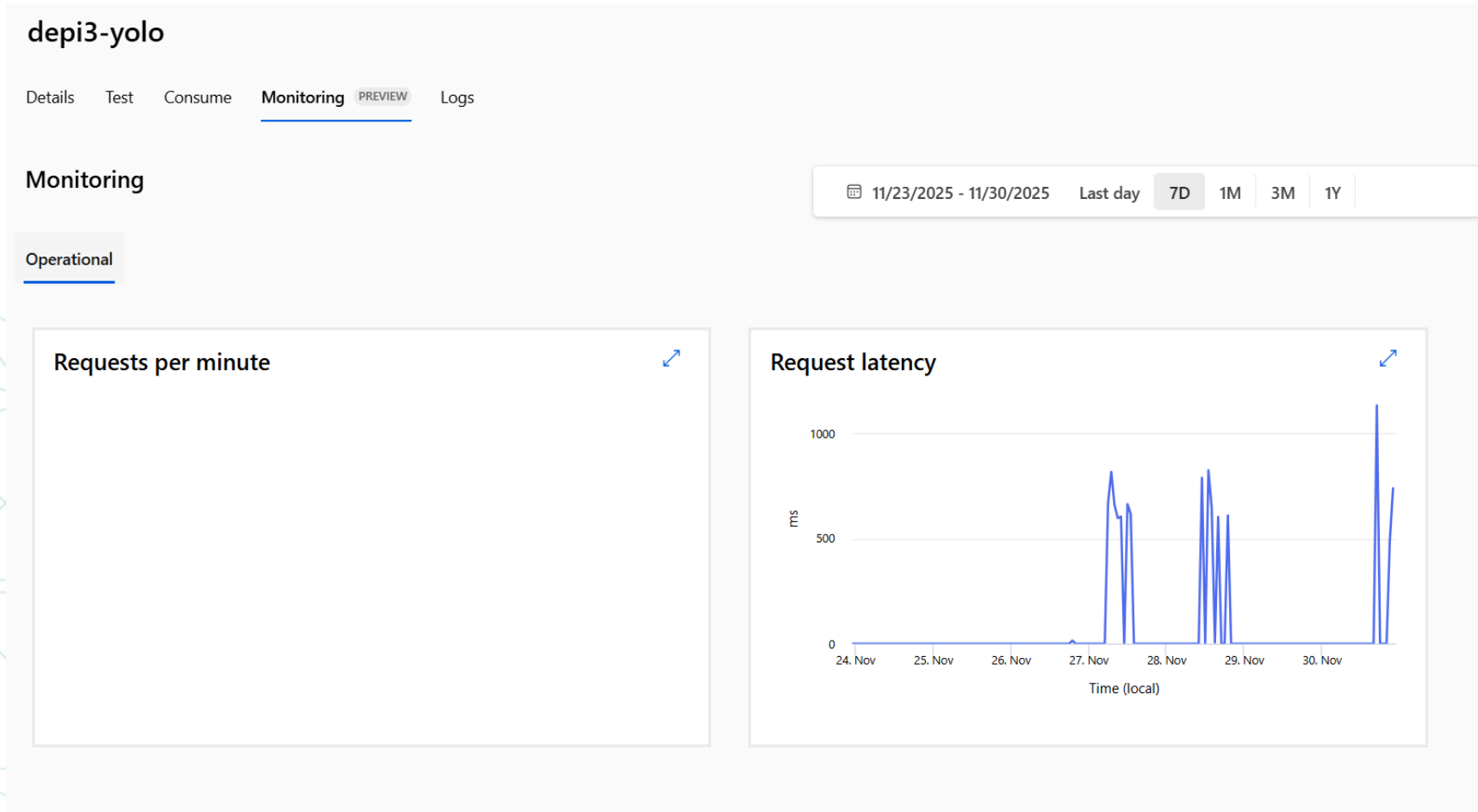
Requests per minute

Request latency



11/29/2025

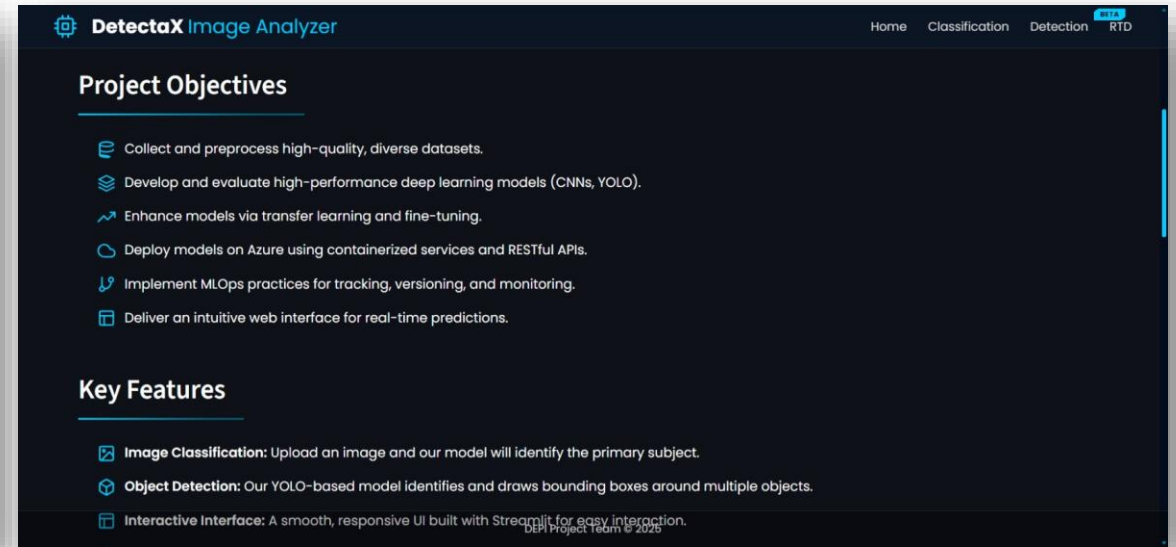
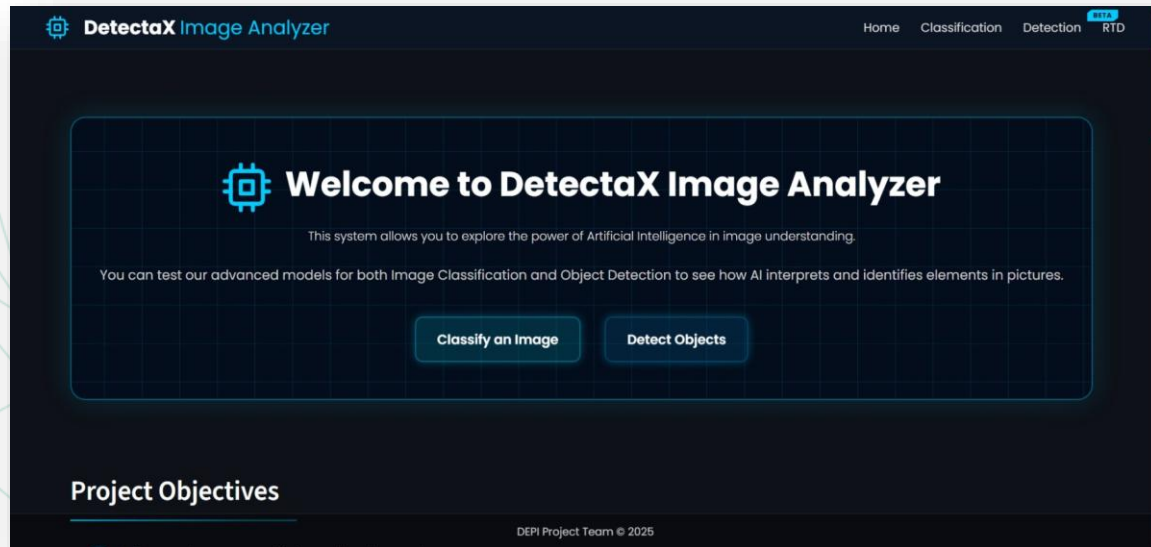
# Azure Deployment Metrics:



11/29/2025

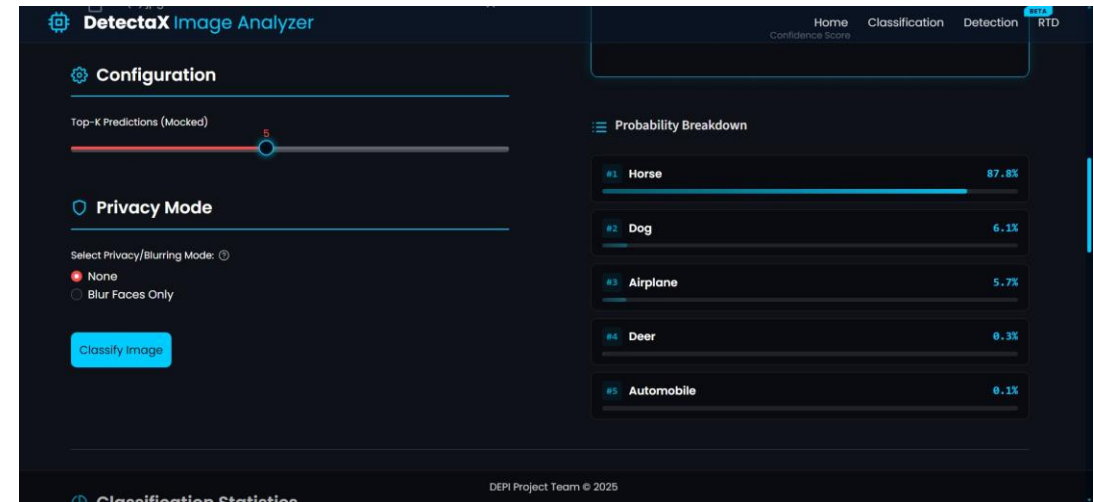
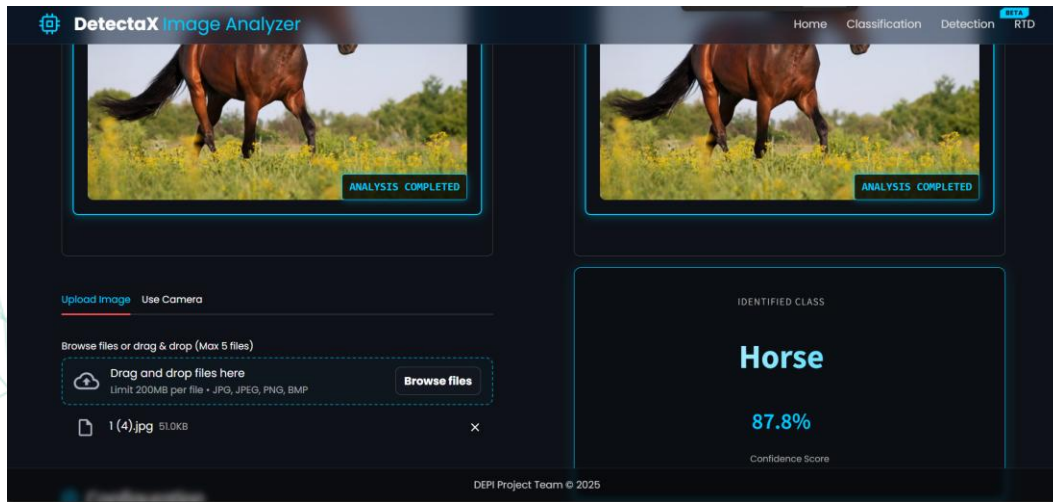
# Web Interface (Frontend & API Integration)

# Web Interface Overview



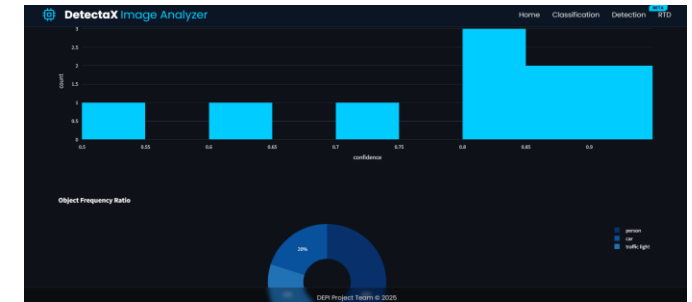
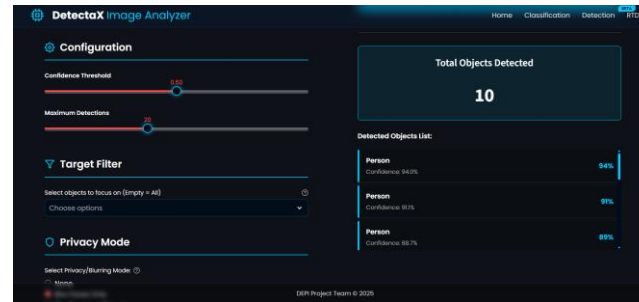
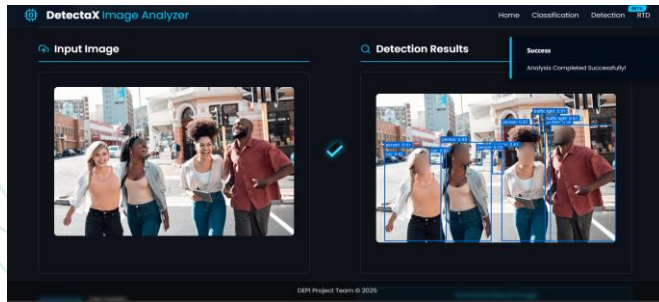
- Full interactive web interface built with **Streamlit**.
- Supports image upload, batch processing, camera streaming, and real-time detection.
- Unified dark **glassmorphism** theme with custom CSS.
- Smooth navigation and consistent UI design across all modules.

# Image Classification Module



- Upload single or multiple images (**Batch Processing**).
- Automatic preprocessing and validation of all images.
- Sends images to Azure **CNN** classification endpoint.
- Displays predicted class, confidence score, and analytical charts.
- Session history maintained for all processed images.
- Stable fallback logic for handling API downtime.

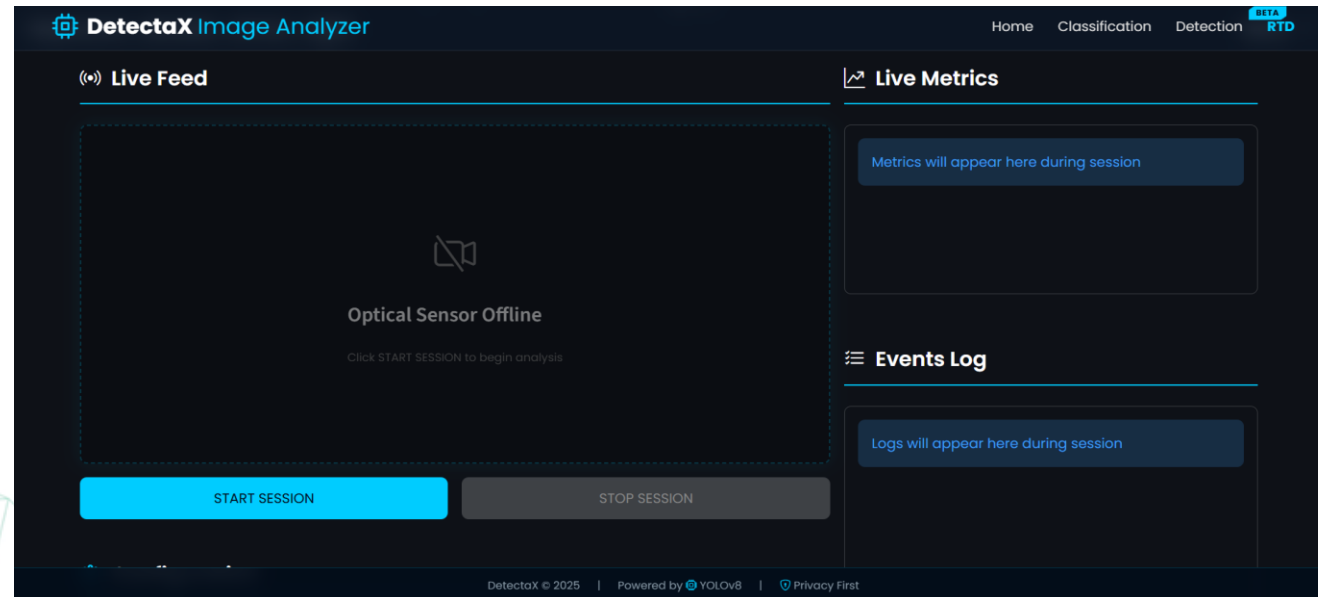
# Object Detection Module



- Supports single-image and multi-image detection (**Batch Processing**).
- Performs detection using **YOLOv12x** via Azure cloud endpoint.
- Adjustable confidence threshold and class filtering.
- Custom bounding-box visualization with class-based colors.
- Object **cropping**, ZIP export, PDF reporting, and detection analytics.
- Clean, guided workflow for processing multiple images sequentially.

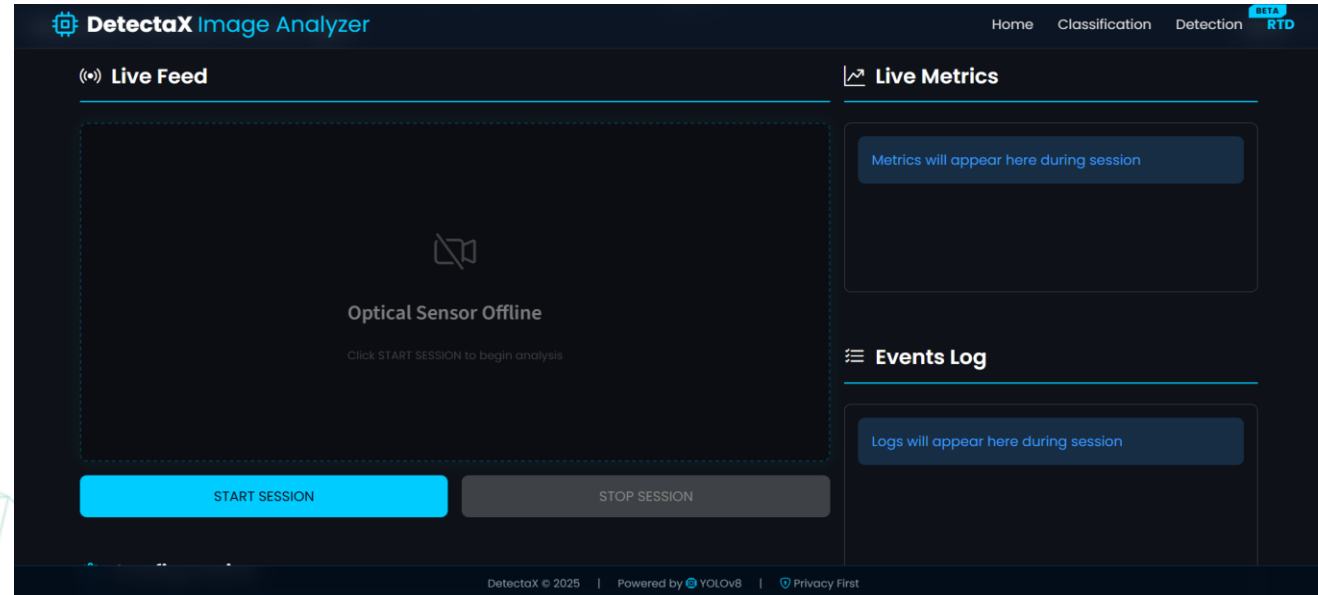


# Real-Time Detection (RTD)



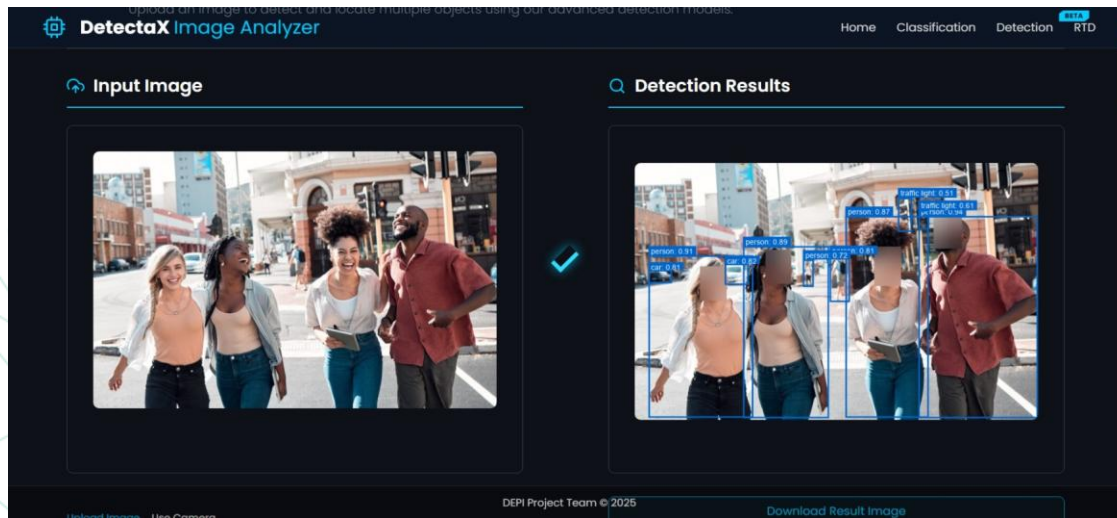
- Real-time camera feed processing using **OpenCV**.
- Continuous YOLOv12x inference per frame with **FPS** calculation.
- Auto-snapshot capturing during the session.
- Live statistics and dynamically updated charts.

# Real-Time Detection (RTD)

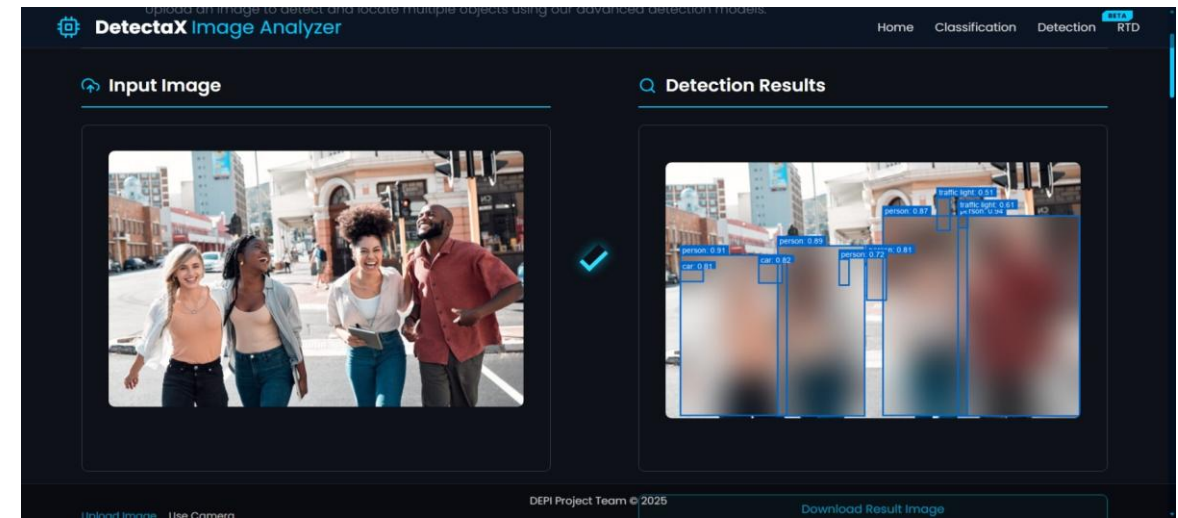


- Real-time camera feed processing using **OpenCV**.
- Continuous YOLOv12x inference per frame with **FPS** calculation.
- Auto-snapshot capturing during the session.
- Live statistics and dynamically updated charts.

# Smart Privacy System



(Blur Faces Only)



(Blur Detected Objects)

- Optional privacy modes: **face blur** and full-person masking.
- Utilizes **YOLO-Face** for accurate face detection.
- Selective blurring without affecting full frame.
- Available in both static image detection and real-time mode.

# Technology Stack & Versions

- **Python 3.12**
- **Streamlit 1.28.0**
- **OpenCV 4.8.1.78**
- **Pillow 10.0.1**
- **NumPy 1.24.3**
- **Requests 2.31.0**
- **Matplotlib 3.7.2**
- **Plotly 5.15.0**
- **Kaleido 0.2.1**
- **Ultralytics 8.1.0**
- **Torch 2.1.0**

# Thank You!