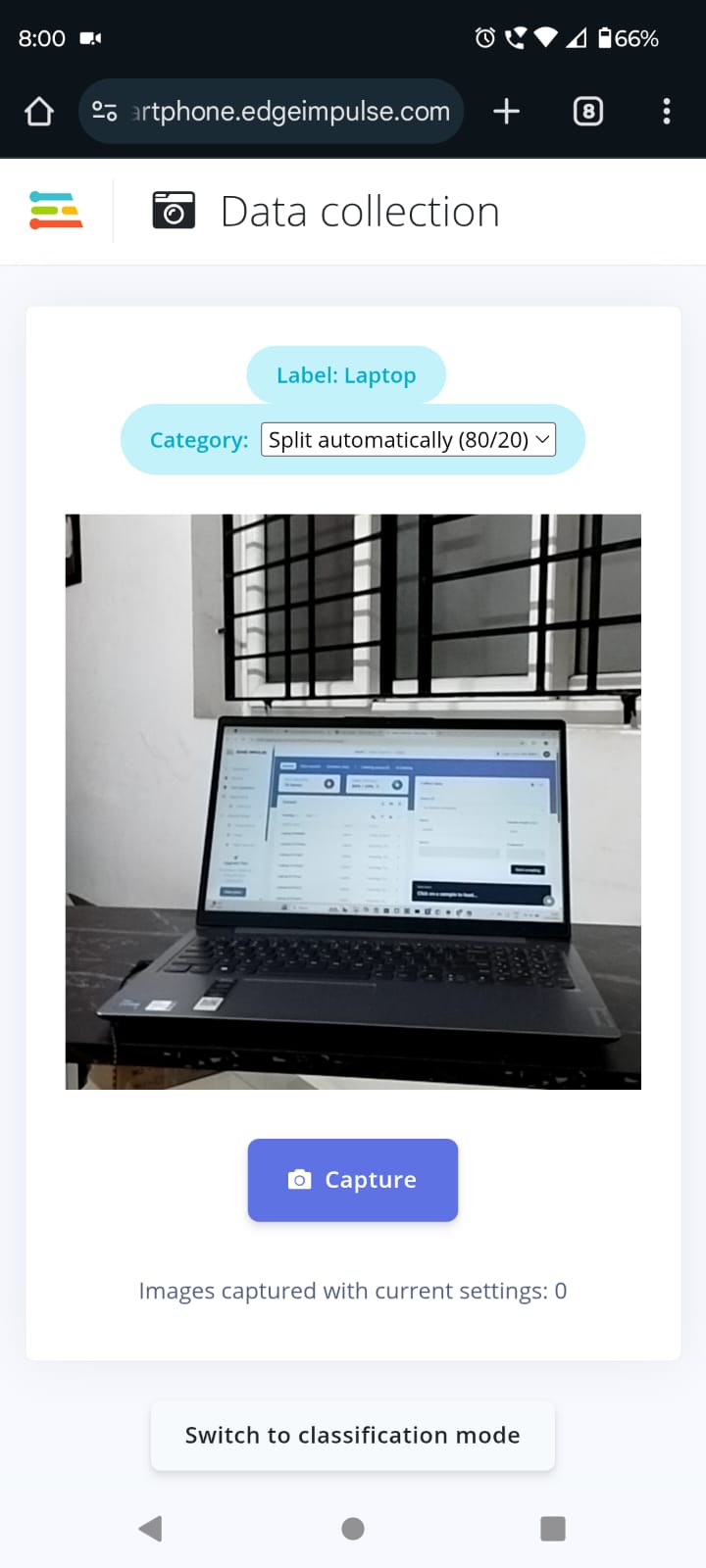
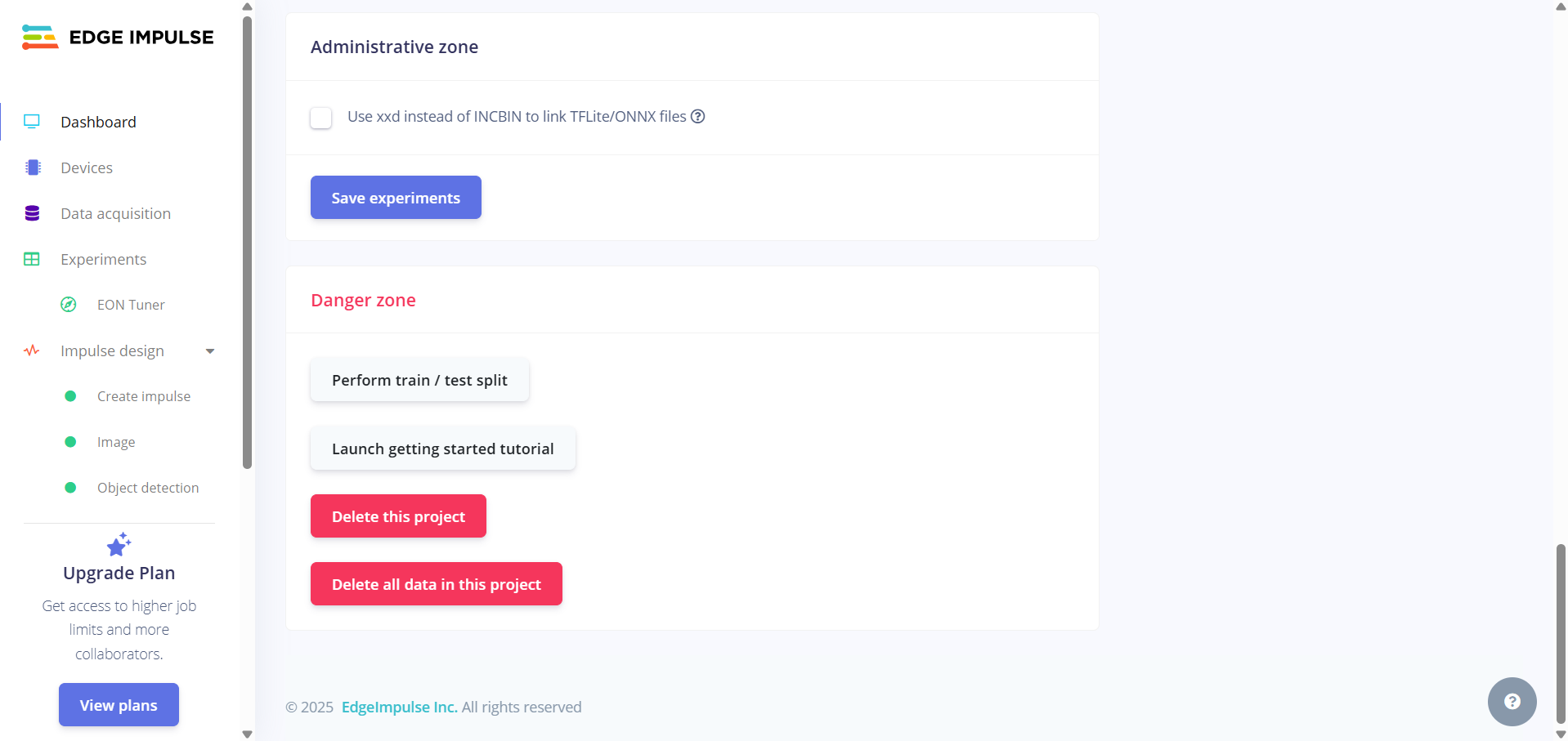
Creating Object Detection Model Using Edge Impulse

# 1. Dataset Preparation

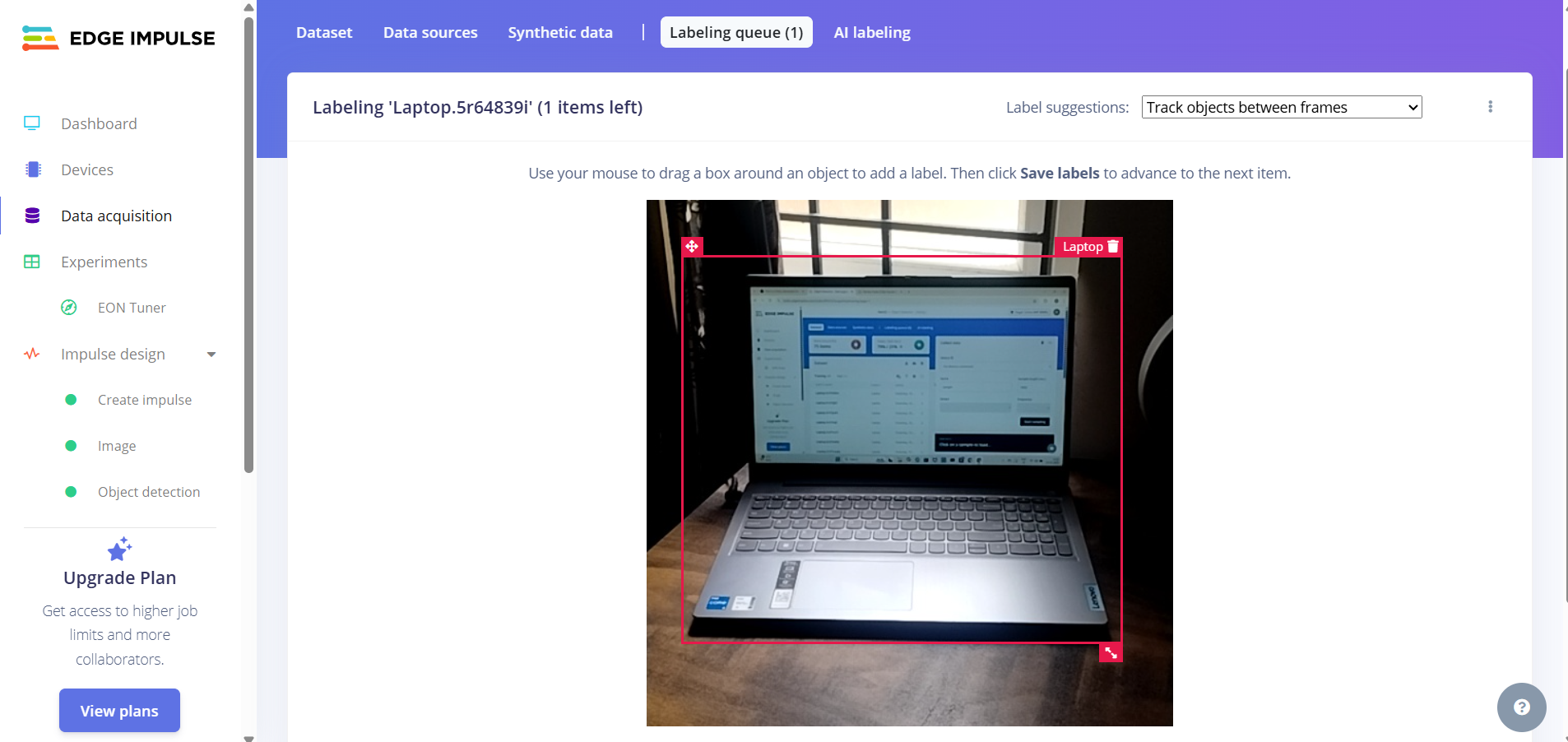
* To begin building the object detection model, I created a dataset by capturing images of a water bottle and a laptop using my mobile phone. You can also collect images using a laptop or a development board.
* Make sure to check if your dataset is diversified enough to train an effective AI model. For object detection tasks, it's important to vary the lighting conditions, angles, and object positions in the dataset. This improves the model's generalization and robustness when deployed.
* When collecting images using a mobile phone through Edge Impulse’s web interface or mobile app, you have the option to assign each image directly to the training set, testing set, or let Edge Impulse automatically split the dataset for you.



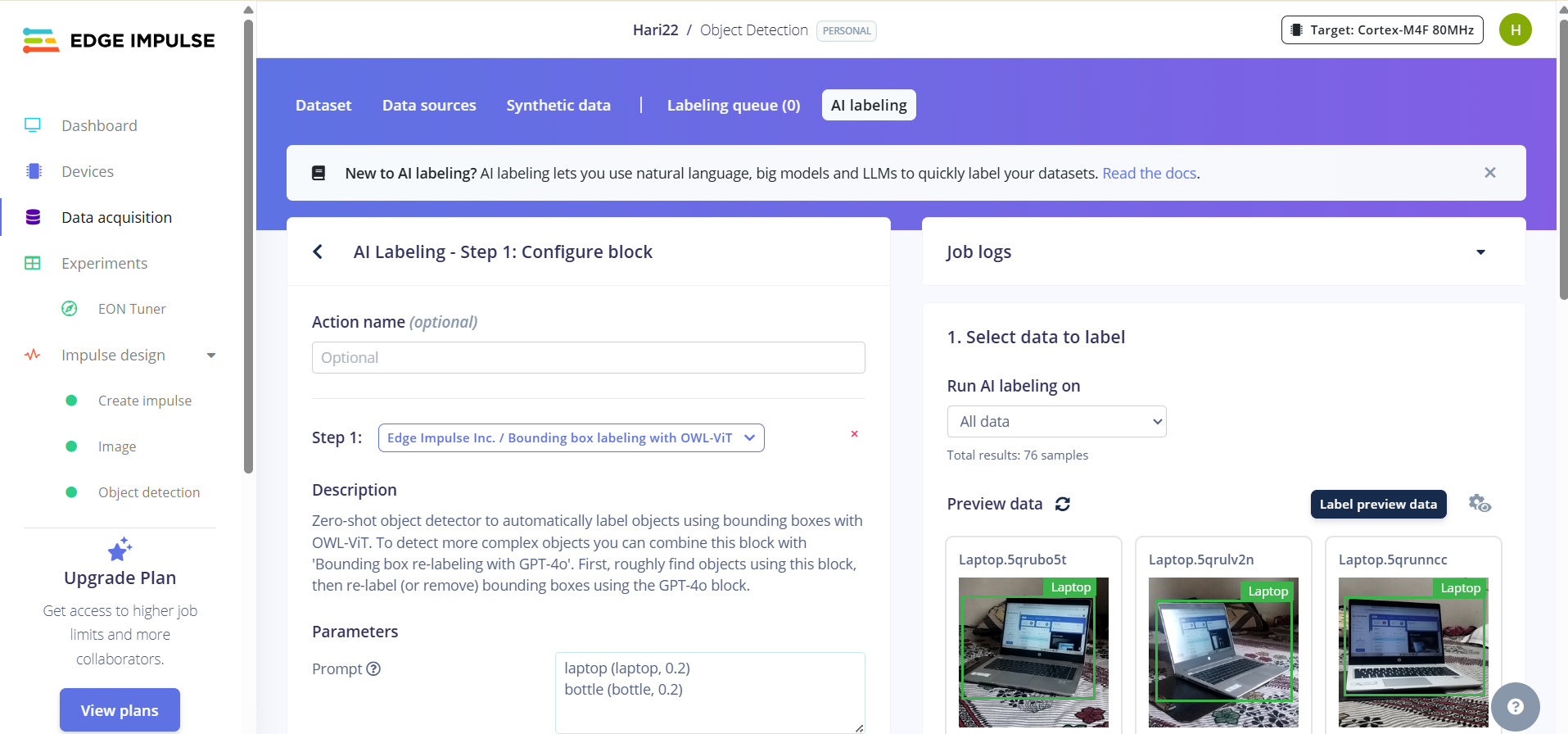
* However, when collecting images using an edge device (such as a Raspberry Pi or microcontroller with the Edge Impulse CLI), all the images are initially added to the training set by default. In this case, you need to manually balance the dataset. To do this, go to the Dashboard of your Edge Impulse project, scroll down, and click “Perform train/test split” to create a well-balanced dataset suitable for training and evaluation.



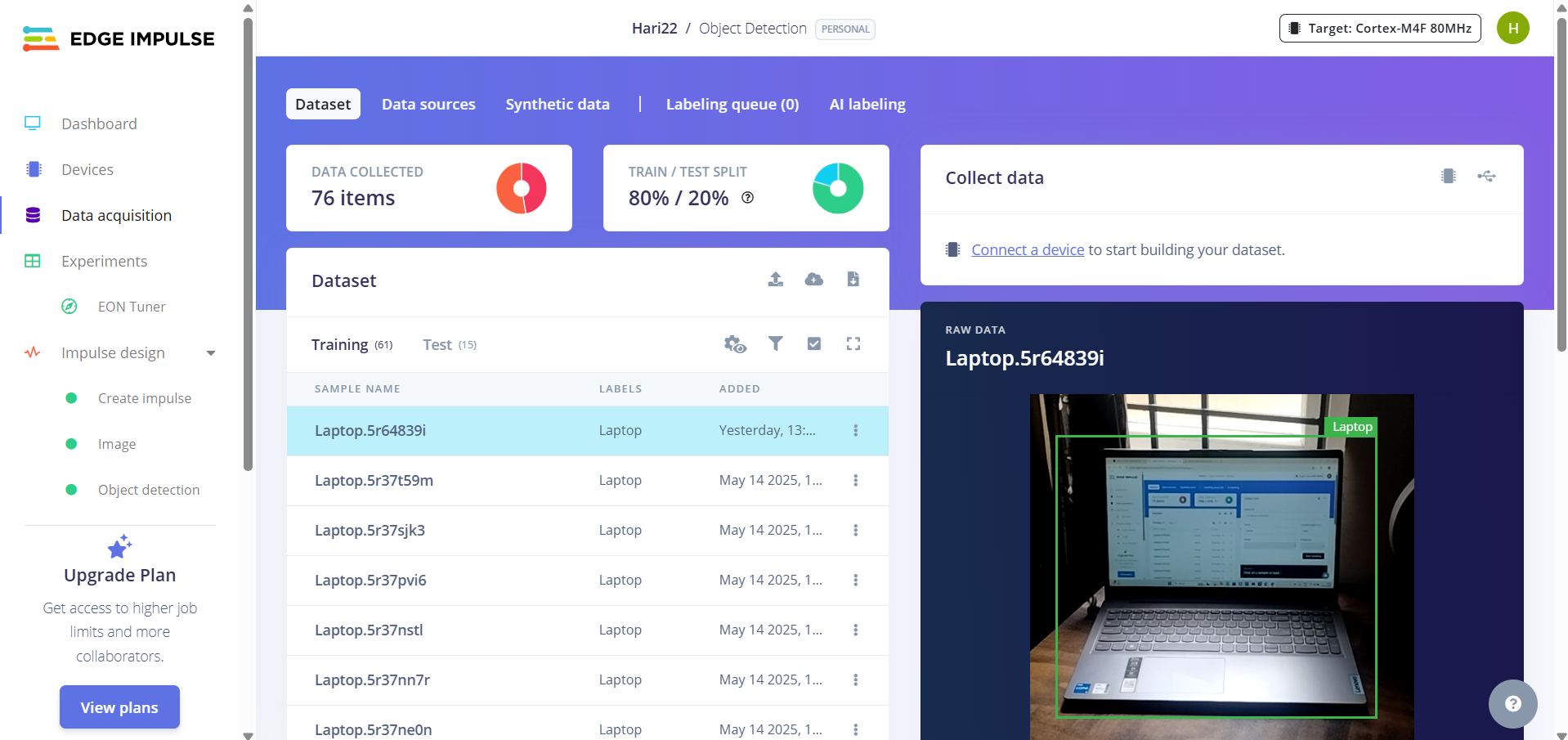
* If you want to upload the dataset, Edge Impulse provides the ability to upload datasets along with labeled annotations.
* Each image must be labeled by manually drawing bounding boxes around each object class in the image. Ensure that the object is tightly bounded to train the model. To do that, go to Labeling queue and draw boxes for each class in the image.



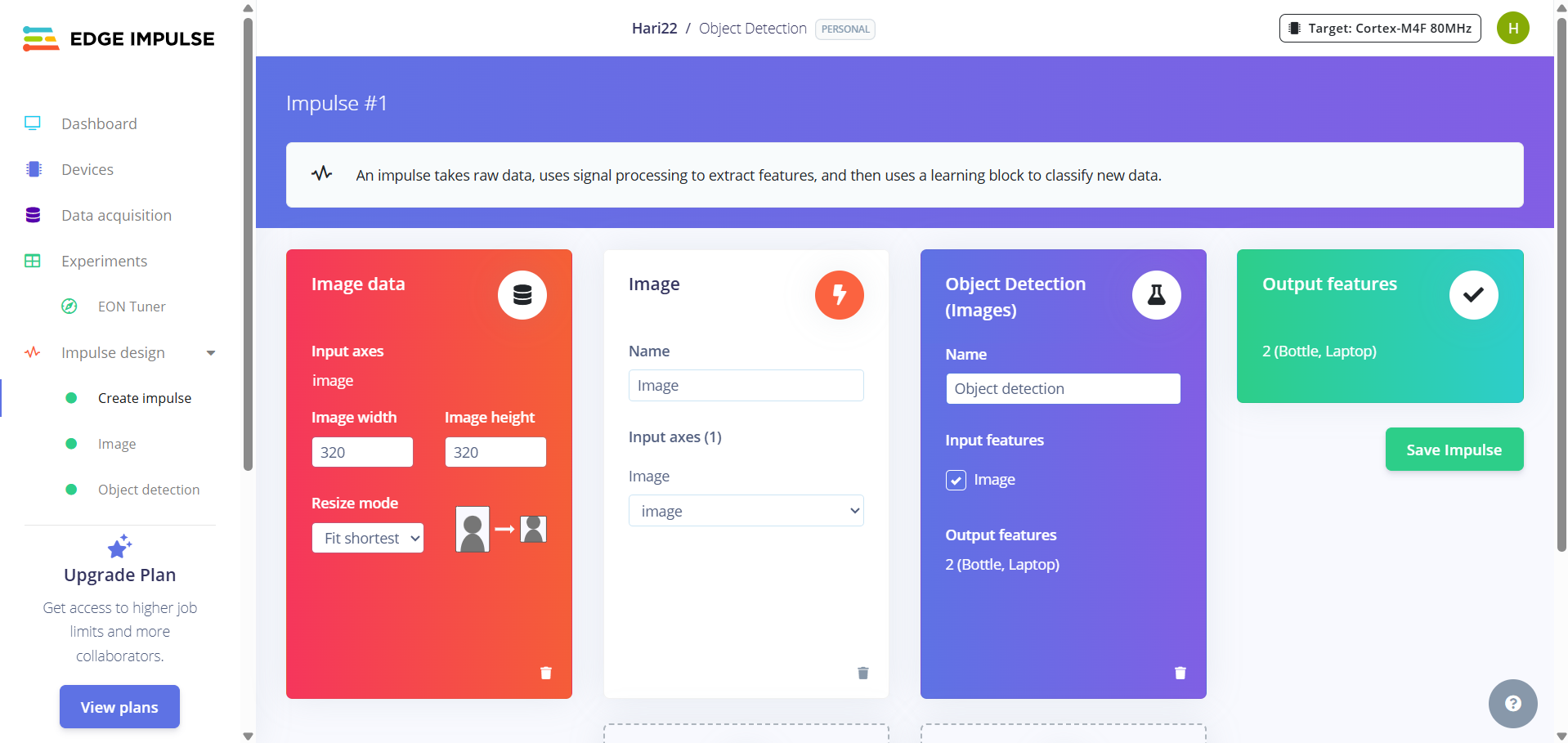
* If your dataset is large, Edge Impulse offers an AI-assisted labeling feature that can help automate this process. You can later manually adjust or fine-tune these labels if needed.
* To label the images using AI, go to AI Labeling in data acquisition.



* Give the action name if you want, then choose the pre trained model which is recommended by the edge impulse. Then specify the objects with their minimum confidence rating (It tells you how sure the model is about its prediction — higher confidence means the model is more certain that it has correctly identified the object.) in the prompts.
* You can also set a threshold percentage to ignore objects below a certain size, which helps reduce false detections from small or irrelevant objects.
* Then select the dropdown based on your requirements and click the Label preview data to display the output of the preview data. If you got the expected output, then scroll down to click Label all data.
* Else, Fine tune the threshold percentage or manually label the incorrect labels in the image.
* Ensure that all data is **properly labeled** and **split into training and testing sets** before training the model.

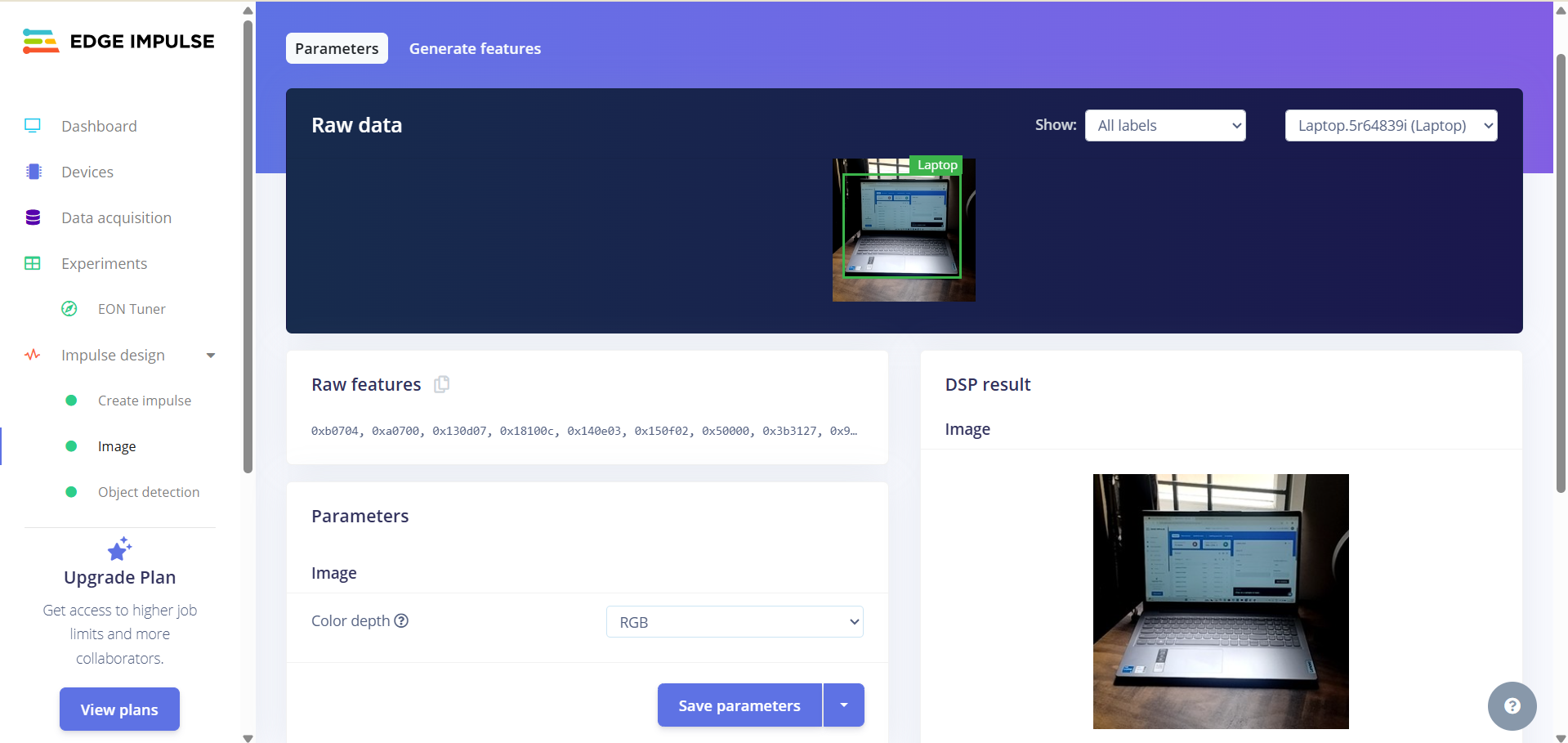


# 2. Creating an Impulse

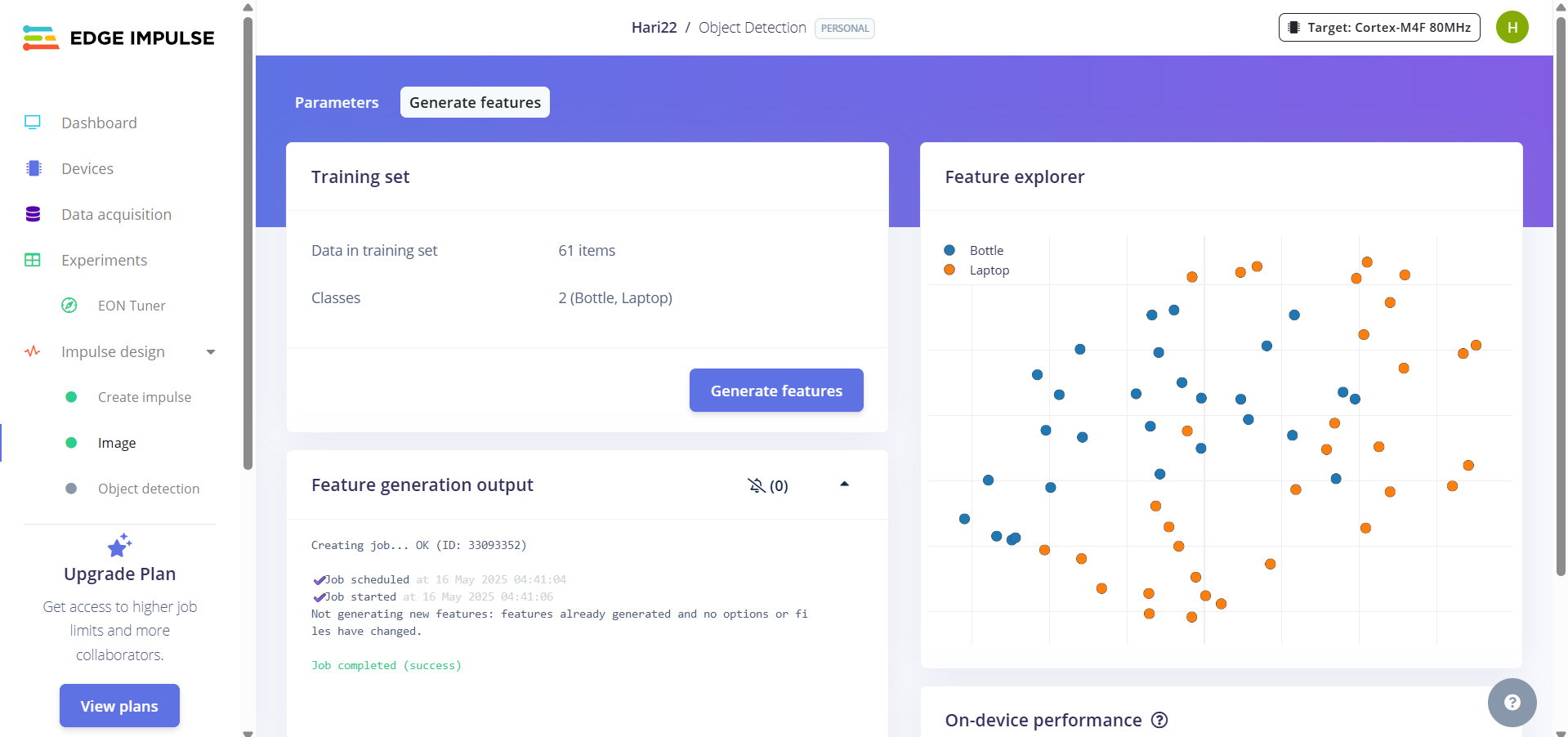
* An Impulse in Edge Impulse defines the steps from raw data to model output. It consists of signal processing and learning blocks that together form the training pipeline.
* To start, choose 'Image Data' as the input data. Set the image width and height based on the camera resolution of your target edge device. This ensures compatibility and efficiency during inference.
* For the processing block, choose 'Image', and for the learning block, choose 'Object Detection'. The output features (object classes) are automatically inferred from your labeled dataset. Click 'Save Impulse' to continue.

# 3. Image (Processing Block)

* You can choose to process the images in either RGB or grayscale format. RGB preserves full color information and is useful when color is important for classification.
* Grayscale reduces each pixel to a single intensity value, making it a more lightweight and faster format, especially for constrained devices. After choosing, click **Save Parameters.**

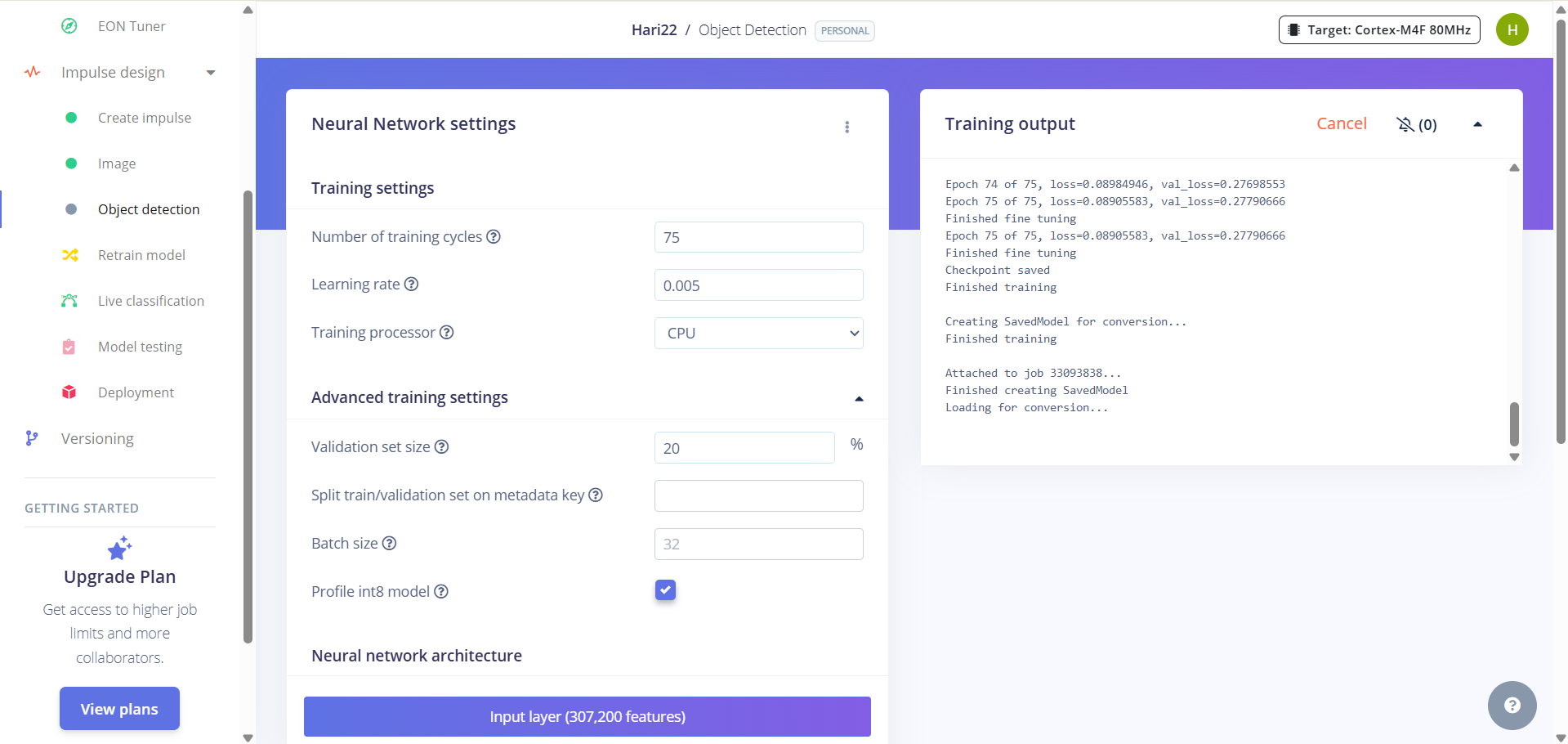
****

* The image processing block converts image data into raw features. You can visualize these features by clicking the 'Generate Features' button. This helps you understand how your data is clustered and whether it's suitable for training.

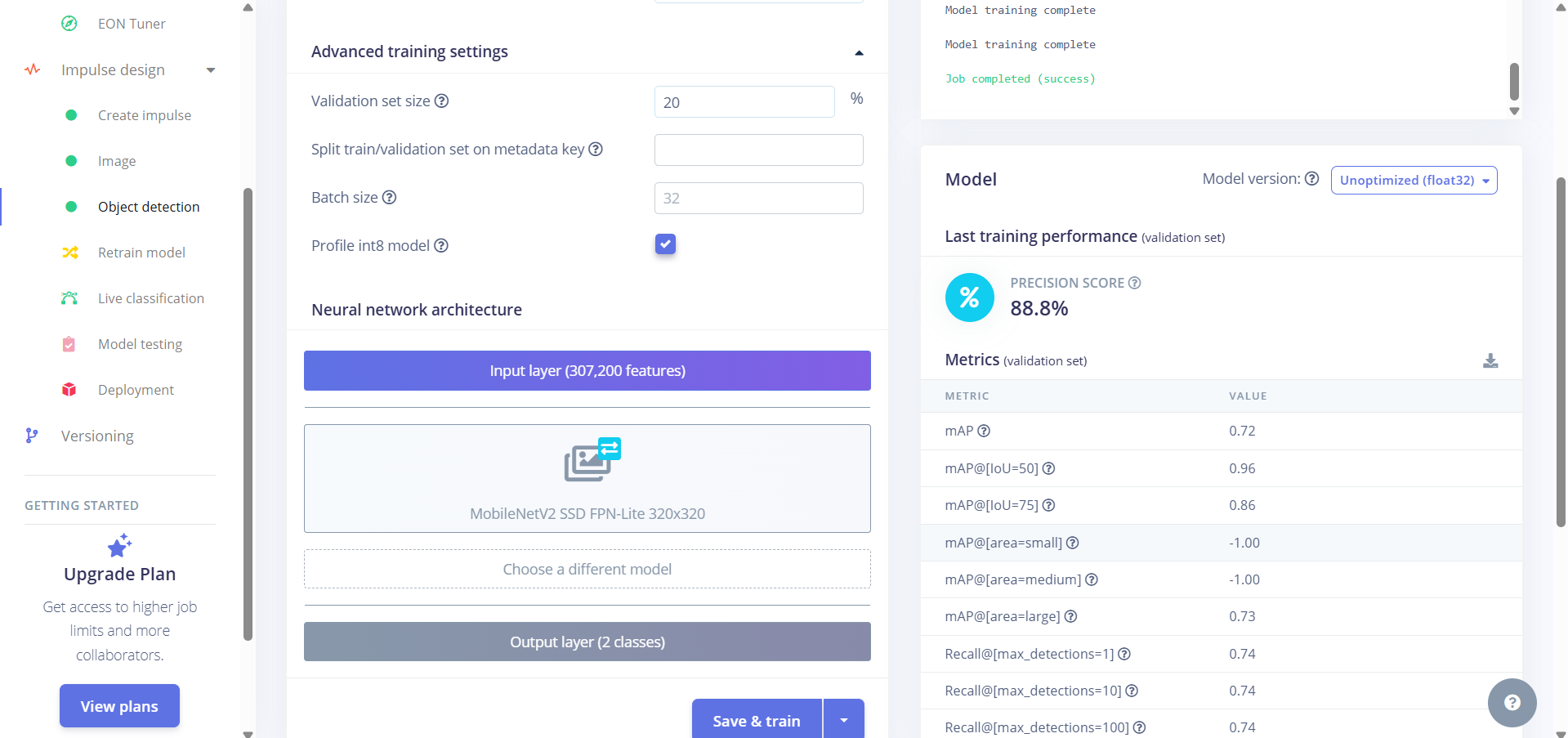


# 4. Object Detection (Learning Block)

* Edge Impulse supports MobileNet and FOMO as object detection models. For this project, MobileNet was chosen because it provides higher accuracy and performs well on general-purpose edge devices.
* The object detection block takes in the features extracted from the images and trains a model to detect and classify objects in real-time. You can configure training parameters which will be discussed below.

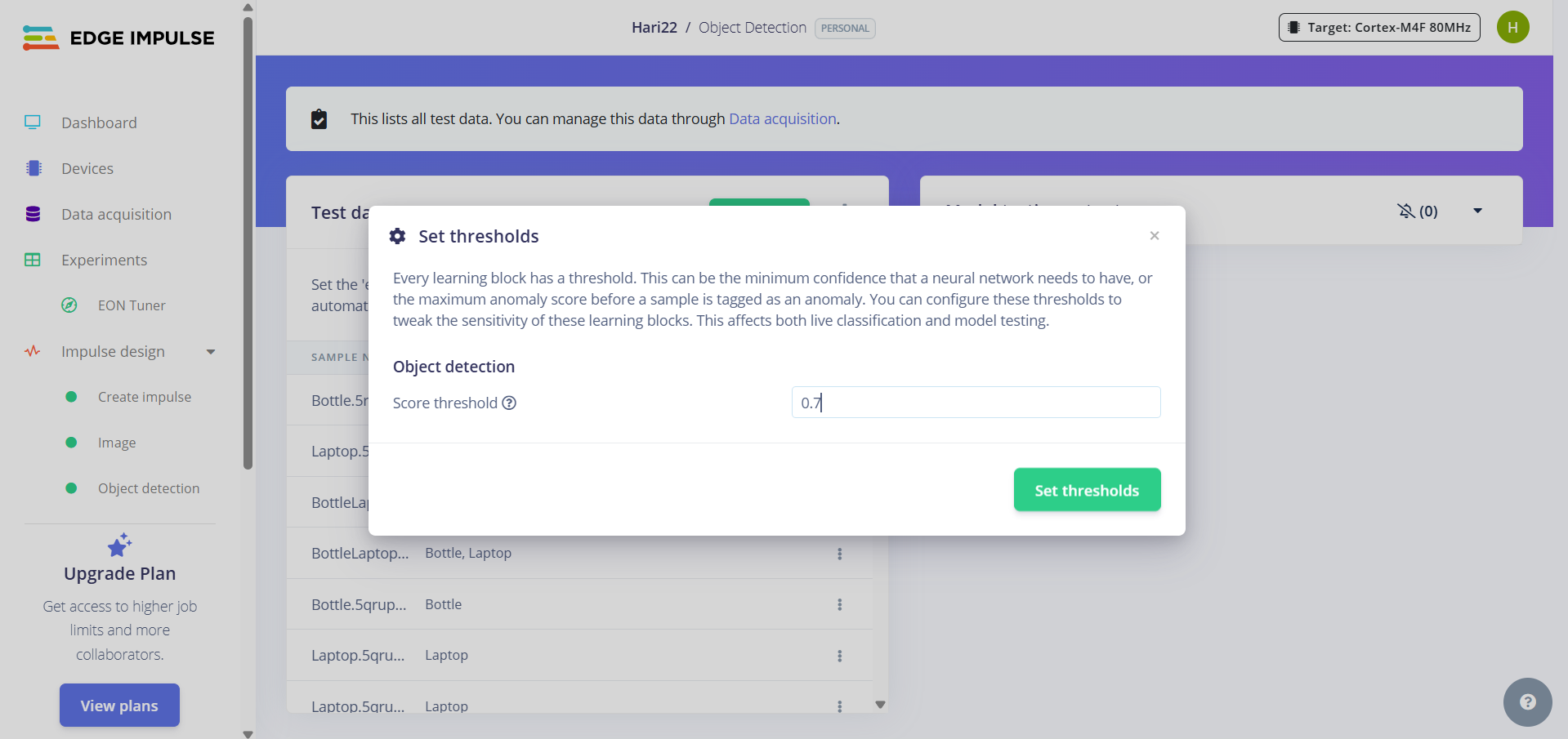


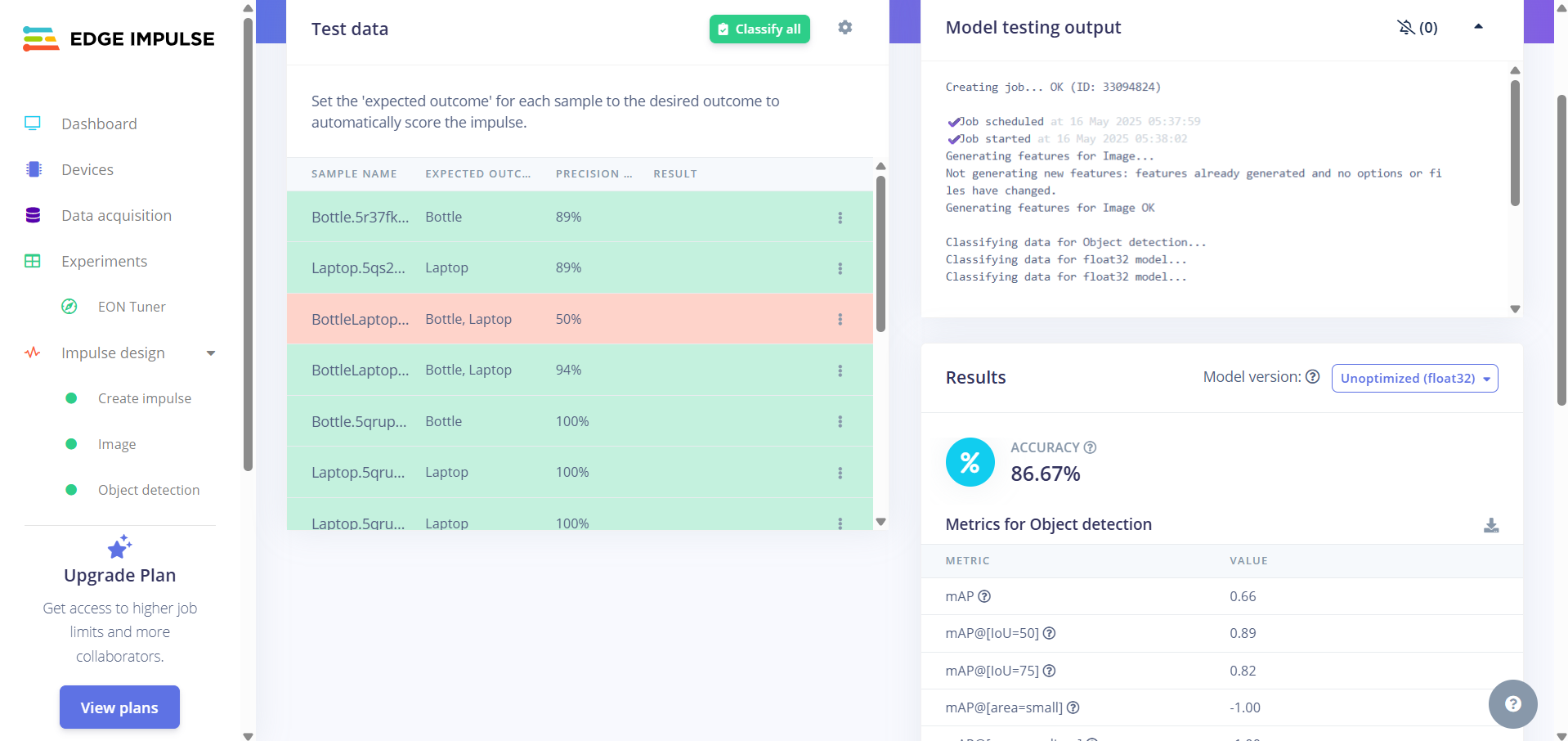
* **Number of Training Cycles:** This means the model will go through the entire training dataset for number of training cycles to learn patterns. More cycles can improve accuracy, but too many may lead to overfitting.
* **Learning Rate:** This controls how fast the model learns during training. If it's too high, the model might miss the best solution. If it's too low, learning may be very slow.
* **Training Processor:** You can select CPU or GPU on some setups which helps to train faster.
* **Validation set size:** Determines what portion of the training data is used for validation during training. Typically set to 20%.
* **Batch size:** Number of training samples processed before the model is updated. Common values range between 16 and 64.
* After setting the parameters, scroll down to click **Save & Train** to train the model. After the training it shows the precision score which indicates the performance of your model in validation set.



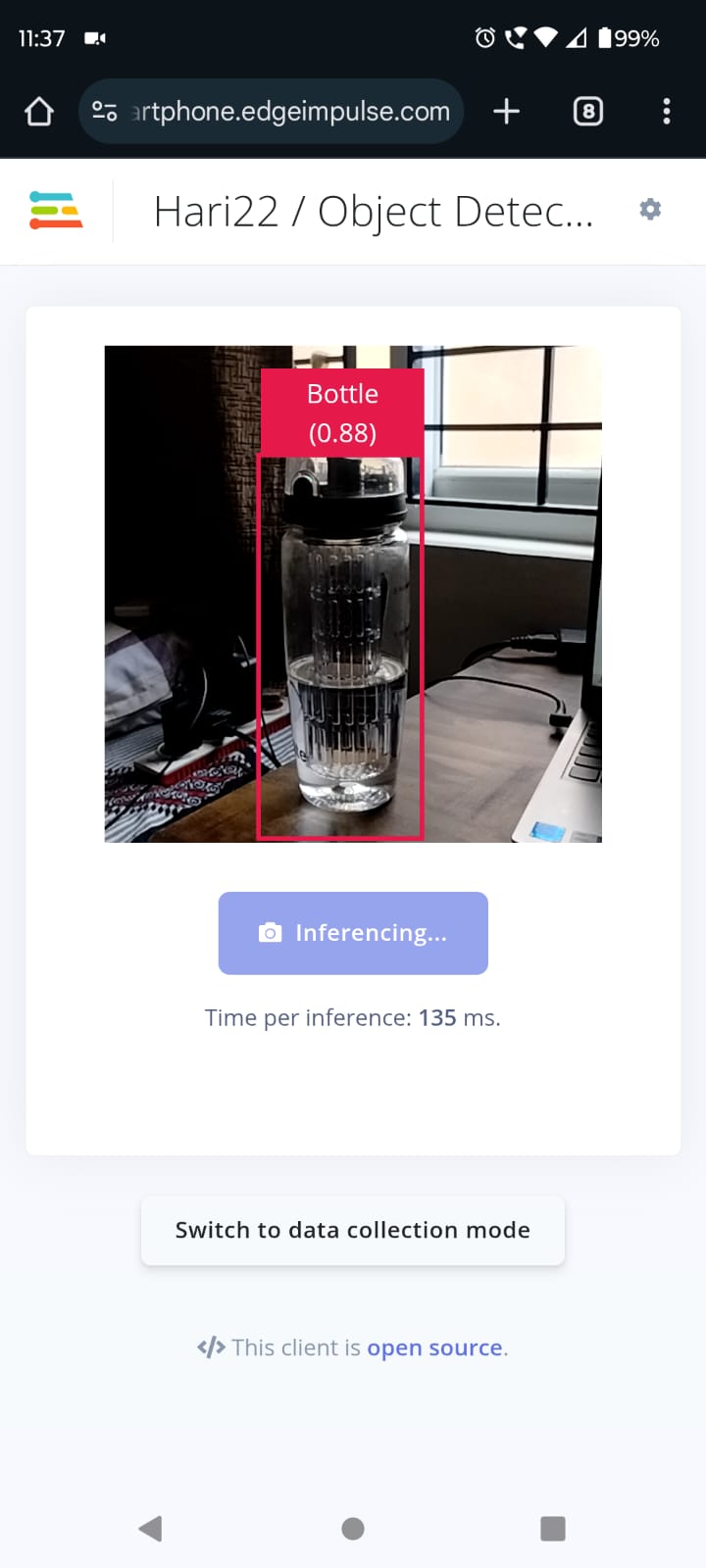
# 5. Model Testing

* Model testing in Edge Impulse involves evaluating the trained model using the test dataset - the part of your data the model hasn’t seen during training. This helps verify how well the model can perform on new, unseen data.
* You can adjust the confidence score to control when the model shows predictions.
  + Click the settings in the Model testing tab
  + Choose the set threshold
  + Enter a value like 0.6 or 0.7



* After setting the threshold, click **classify all** to predict the test dataset.
* After testing, Edge Impulse shows the **accuracy** of the model on the test set.  
  If the accuracy is **100%**, it may be a sign of **overfitting**, meaning the model has memorized the training data instead of learning patterns that generalize well to real-world situations.
* To avoid or fix overfitting, you can:
  + Add more **diverse data** (change lighting, angles, and backgrounds).
  + **Data augmentation** to simulate variation.
  + **Reduce the number of training cycles**.
  + Use a **simpler model** or reduce model complexity.

# 6. Testing the prototype

* Edge Impulse allows you to deploy your trained model to various platforms including **edge devices** like Raspberry Pi, Arduino, STM32, ESP32, and others.
* There are multiple deployment formats available, such as:
  + .eim files (Edge Impulse model files for Linux-based devices)
  + **C++ libraries**
  + **WebAssembly (WASM)** for browser-based inference
  + **TensorFlow Lite (TFLite)** for mobile and embedded systems
* In this project, I did not deploy the model to a physical edge device. Instead, I used the **Live Classification** feature on Edge Impulse through my **mobile phone camera** to test the model in real time.
* This method allows you to:
  + Stream camera input from your mobile device
  + Run the model in the browser
  + View real-time predictions and bounding boxes on detected objects

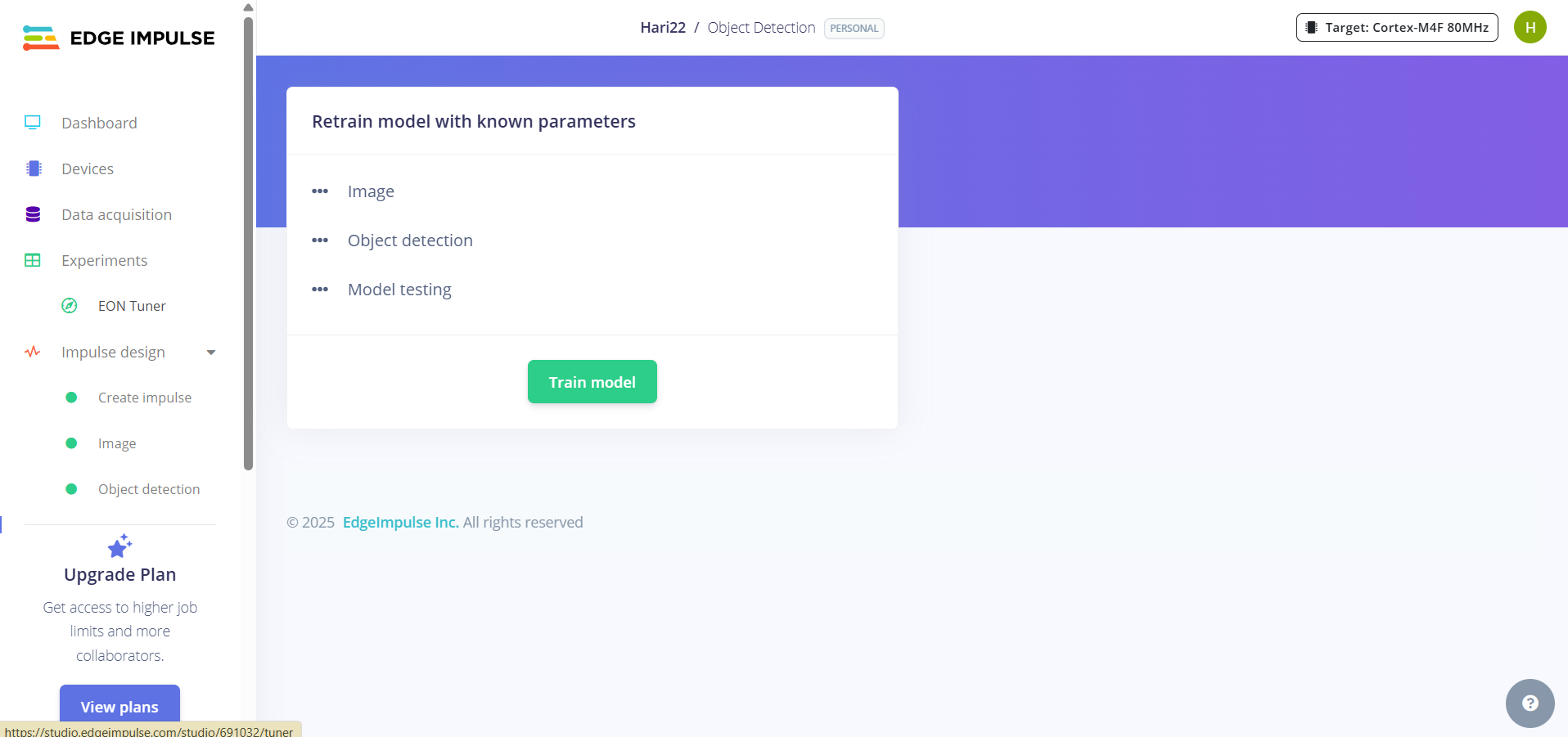
# 7. Retrain the model

After testing your model, you may find that it needs improvement. This could be due to:

* Low accuracy
* Overfitting
* Poor generalization to new data
* Addition of new objects or classes

In these cases, Edge Impulse makes it easy to **retrain your model** without starting from scratch.

To retrain the model:

1. Go to the **Retrain Model** tab on the left sidebar.
2. Review your dataset, training parameters, and impulse settings.
3. Click the **Train Model** button again. 
4. You can retrain your model:

* With new or more diverse data
* By adjusting training parameters (like learning rate, batch size, or number of epochs)
* After fixing incorrectly labeled images
* After improving the train/test data split

Retraining helps refine the model and improve its performance without rebuilding the pipeline.