Final Report

Project: TinyML-Based Project on FPGA Board with RISC-V Core

Name: Aman Chauhan
Branch: Btech. Computer Science and Engineering (CSE Core)
Roll Number: 22BCE0476
College: Vellore Institute of Technology, Vellore
Mentor: Dr. Sudip Roy

Week: *Final Presentation*
Date: *30th June 2025*

Internship Overview

- Designed, simulated, and deployed TinyML models on a **RISC-V FPGA** (simulated) and embedded boards (**ESP32, Arduino**).
- Used simulation tools (**RVfpga, Verilator, GTKWave, Whisper**) for RISC-V core development—no physical FPGA required.
- Built and optimized ML models (**digit recognition, anomaly detection**) with Edge Impulse and TensorFlow Lite Micro.
- Deployed quantized models on ESP32/Arduino for real-time edge AI inference.
- Demonstrated a complete workflow: **data collection, training, quantization, deployment, and testing** on embedded systems.

Abstract

- Designed and simulated a TinyML system on an FPGA board with a RISC-V core using software tools—**no physical hardware needed**.
- Developed and **deployed lightweight ML models** (digit recognition, anomaly detection) on ESP32 and Arduino for real-time edge AI.
- Demonstrated a complete workflow from **data collection to model training, quantization, and embedded deployment**.

Table of Contents

Introduction to TinyML, RISC-V, FPGA

- **TinyML**: Brings machine learning to microcontrollers and edge devices, enabling real-time AI with minimal power and memory.
- **RISC-V**: Open-source, modular CPU architecture ideal for custom, low-power embedded systems and hardware-software co-design.
- **FPGA**: Hardware platform for prototyping and deploying custom digital logic, allowing flexible integration of RISC-V cores and AI accelerators.
- **Project Relevance**: Integrates RISC-V simulation (no physical FPGA needed) with TinyML model deployment, demonstrating practical edge AI on resource-constrained devices

Project Motivation

- **Real-time, low-power AI** needed on edge devices
- RISC-V + FPGA: **flexible, open-source, customizable hardware**
- **TinyML**: ML on microcontrollers for intelligent edge
- **Combined**: Enables cost-effective, complete edge AI workflow (no physical FPGA needed)
- **Outcome**: Advanced AI on low-cost, resource-limited devices

Project Objectives

- Understand and simulate **RISC-V architecture using FPGA-based tools**
- Program and debug **RISC-V systems in C and assembly** (no physical hardware needed)
- **Develop and train TinyML models** (digit recognition, anomaly detection)
- **Optimize and convert models to TensorFlow Lite for embedded deployment**
- **Deploy and test models on ESP32 and Arduino** for real-time inference
- Try to **Integrate RISC-V and TinyML workflows** for practical edge AI

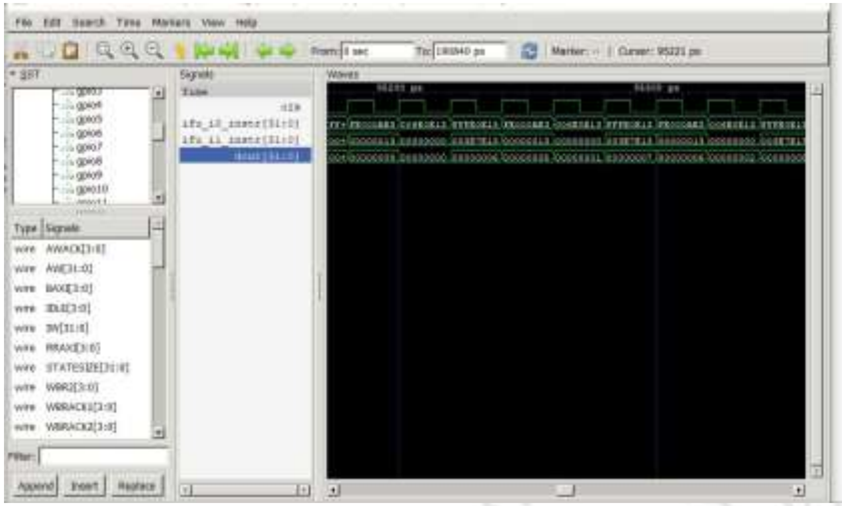
Tools & Technologies

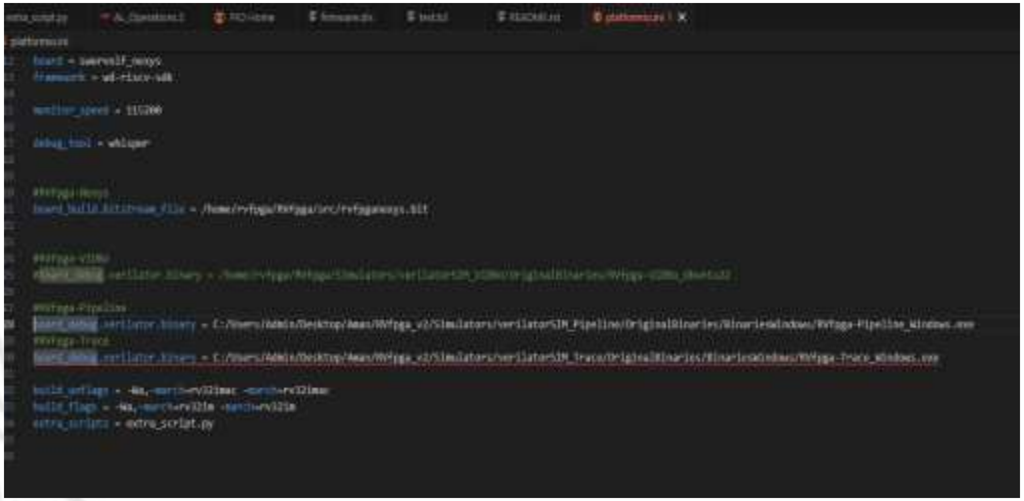
- **Hardware:**
  - Nexys A7 FPGA (simulated)
  - ESP32, Arduino boards
  - DHT11 sensor
- **Software:**
  - Vivado, Verilator, GTKWave, Whisper
  - Edge Impulse, TensorFlow Lite Micro
  - Arduino IDE, PlatformIO, Python
- **Programming:**
  - C, C++, Python, RISC-V Assembly, Verilog
- **Libraries/Resources:**
  - Edge Impulse docs, TensorFlow Lite Micro, RVfpga HarvardX

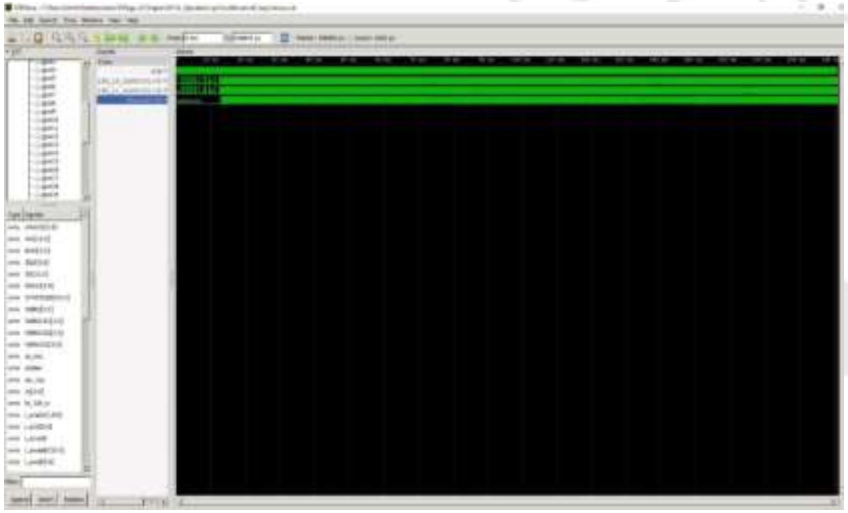
Simulation-Driven Development

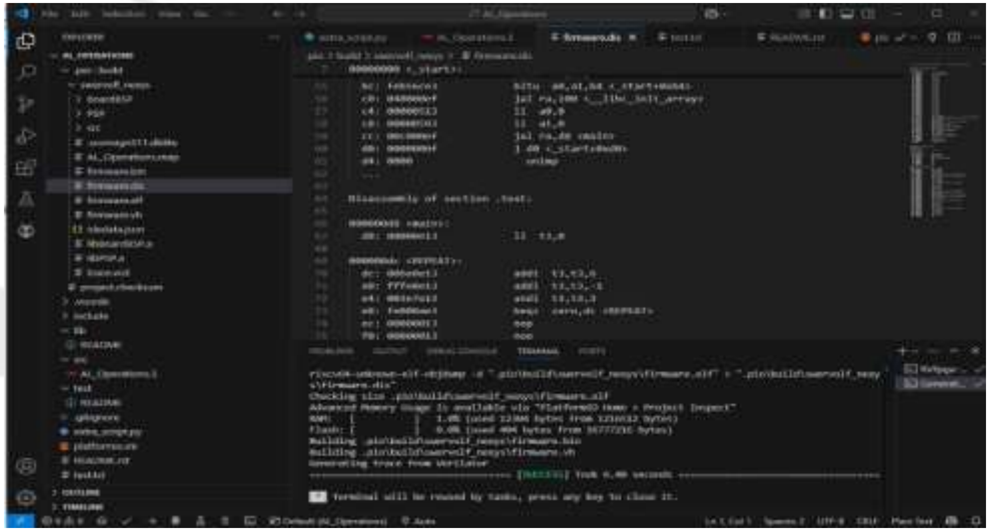
- Used **RVfpga and SweRVolf SoC** on a virtual Nexys A7 board for hardware design and simulation.
- **Verilator:** Cycle-accurate simulation and VCD waveform capture.
- **GTKWave**: Deep signal and pipeline stage analysis for debugging.
- **Whisper**: Instruction-level RISC-V simulation, interactive debugging, and golden model verification—no physical hardware needed.
- Enabled step-by-step validation of C/assembly programs and SoC design before deployment.

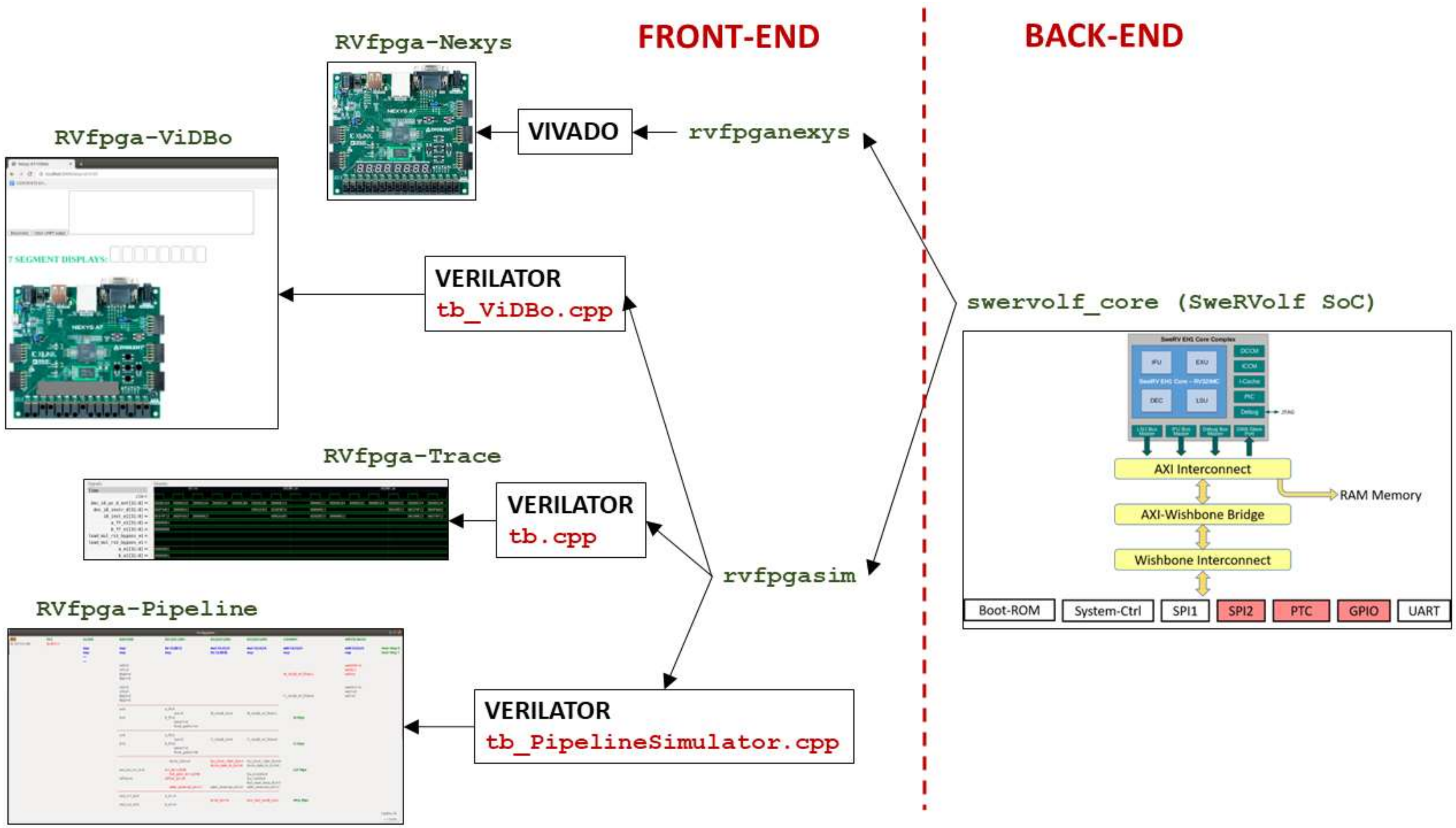
- Used **Verilator for cycle-accurate simulation** and VCD waveform capture; analyzed signals and pipeline stages with GTKWave for deep debugging.
- Employed Whisper simulator for instruction-level simulation, interactive debugging, and as a "**golden model**" for verifying C/assembly code correctness.
- Explored **SoC peripherals** (GPIO, UART, memory controllers) and performed hardware/software co-design tasks.
- Implemented and tested image processing (**RGB to Grayscale**) using both C and RISC-V assembly on the simulated core.
- Followed the **RVfpga HarvardX edX** course and labs for structured, step-by-step learning and practical assignments.
- Documented the entire workflow, enabling **full RISC-V and FPGA learning without requiring real hardware.**










FRONT-END
BACK-END
RVfpga-Nexys
VIVADO
rvfpganexys
RVfpga-ViDBo
7 SEGMENT DISPLAYS:
VERILATOR
tb_ViDBo.cpp
swervolf_core (SweRVolf SoC)
AXI Interconnect
RAM Memory
AXI-Wishbone Bridge
Wishbone Interconnect
Boot-ROM
System-Ctrl
SPI1
SPI2
PTC
GPIO
UART
RVfpga-Trace
VERILATOR
tb.cpp
rvfpgasim
RVfpga-Pipeline
VERILATOR
tb_PipelineSimulator.cpp

Quantization

•Process of **converting high-precision floating-point models (FP32) to lower-precision formats (INT8, FP16)** for **efficient deployment on embedded devices**.

•Reduces model size and computational requirements while maintaining **acceptable accuracy** for edge AI applications.

Core Concept:

•**Formula: quantized_value = (float_value / scale) + zero_point**

•**Reverse: float_value = (quantized_value - zero_point) × scale**

•Maps continuous floating-point values to discrete integer representations.

Why Quantization?

•Default TensorFlow models use 32-bit floating-point weights **(~4 bytes per parameter**)

•Quantization reduces this to 8-bit integers **(~1 byte per parameter)** = 75% size reduction

Types of TFLite Quantization

- **Post-Training Quantization (Most Common):**
- Applied after model training is complete
- Dynamic Range Quantization: Weights quantized, activations remain FP32
- Full Integer Quantization: Both weights and activations quantized to INT8
- Float16 Quantization: Reduces precision to 16-bit floats
- **Quantization-Aware Training (QAT):**
- Simulates quantization effects during training
- Better accuracy preservation but requires model retraining
- Precision Options:
- INT8: 4x smaller models, 3x+ speedup, works on microcontrollers

Applying Quantizer to Weights → Generate Codebook → Quantize the Weights with Codebook → Retrain Codebook

-0.2	1	0.4
0.1	-0.5	-0.7
1.4	0.6	-0.1

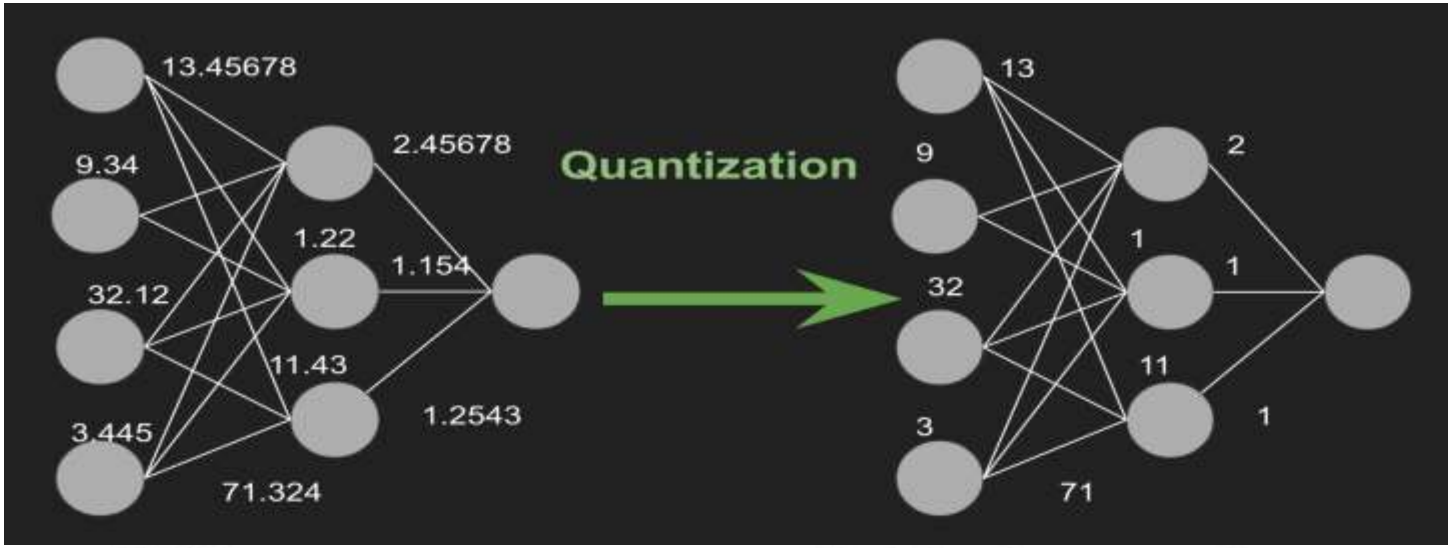
Convolutional Filter

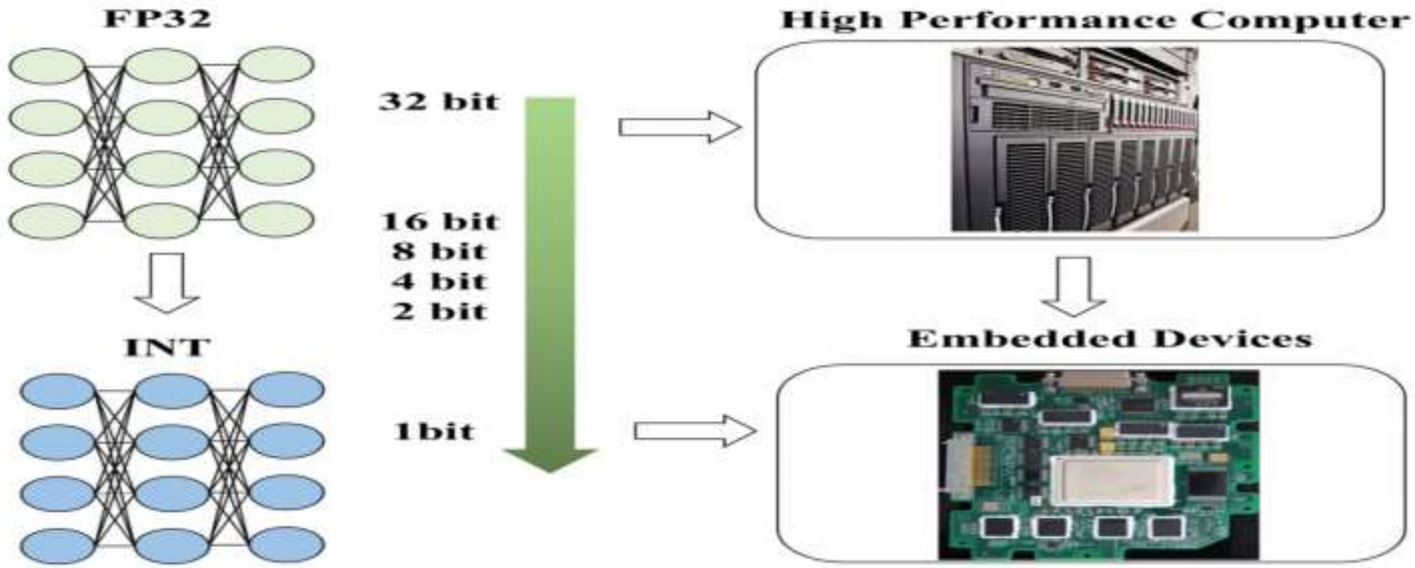
index	[in bits]	Value
0	[00]	-0.6
1	[01]	-0.1
2	[10]	0.5
3	[11]	1.2

Codebook

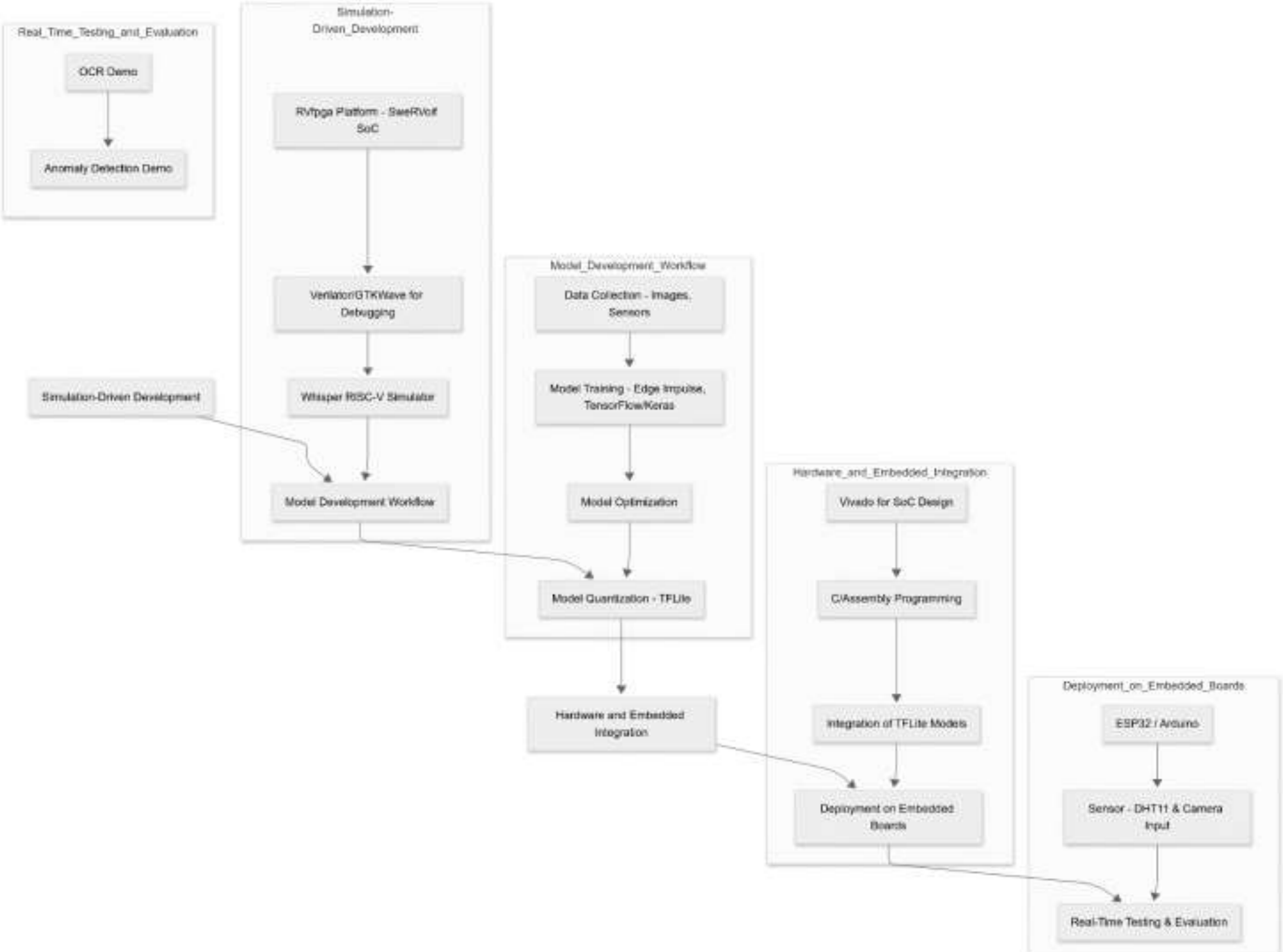
-0.1	1.2	0.5
-0.1	-0.6	-0.6
1.2	0.5	-0.1

Quantized Filter

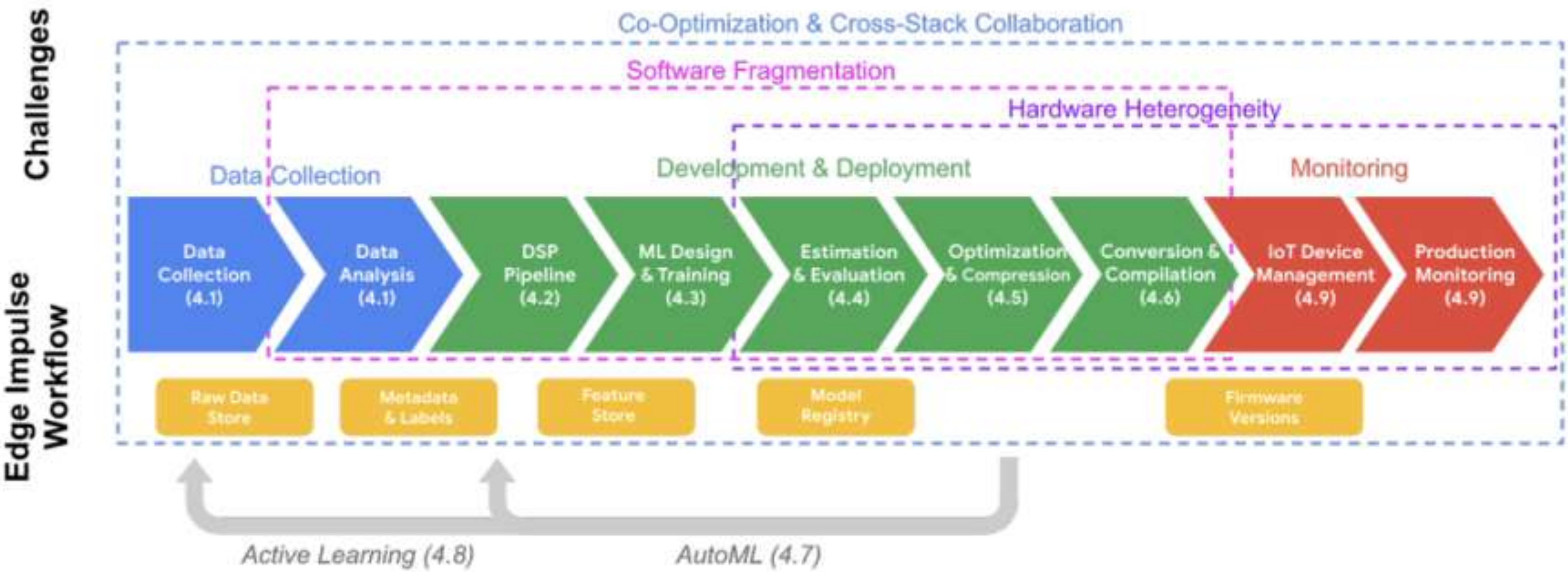

13.45678
9.34
32.12
3.445
2.45678
1.22
1.154
11.43
1.2543
71.324
Quantization
13
9
32
3
2
1
1
11
1
71


FP32
INT
32 bit
16 bit
8 bit
4 bit
2 bit
1bit
High Performance Computer
Embedded Devices

System Architecture



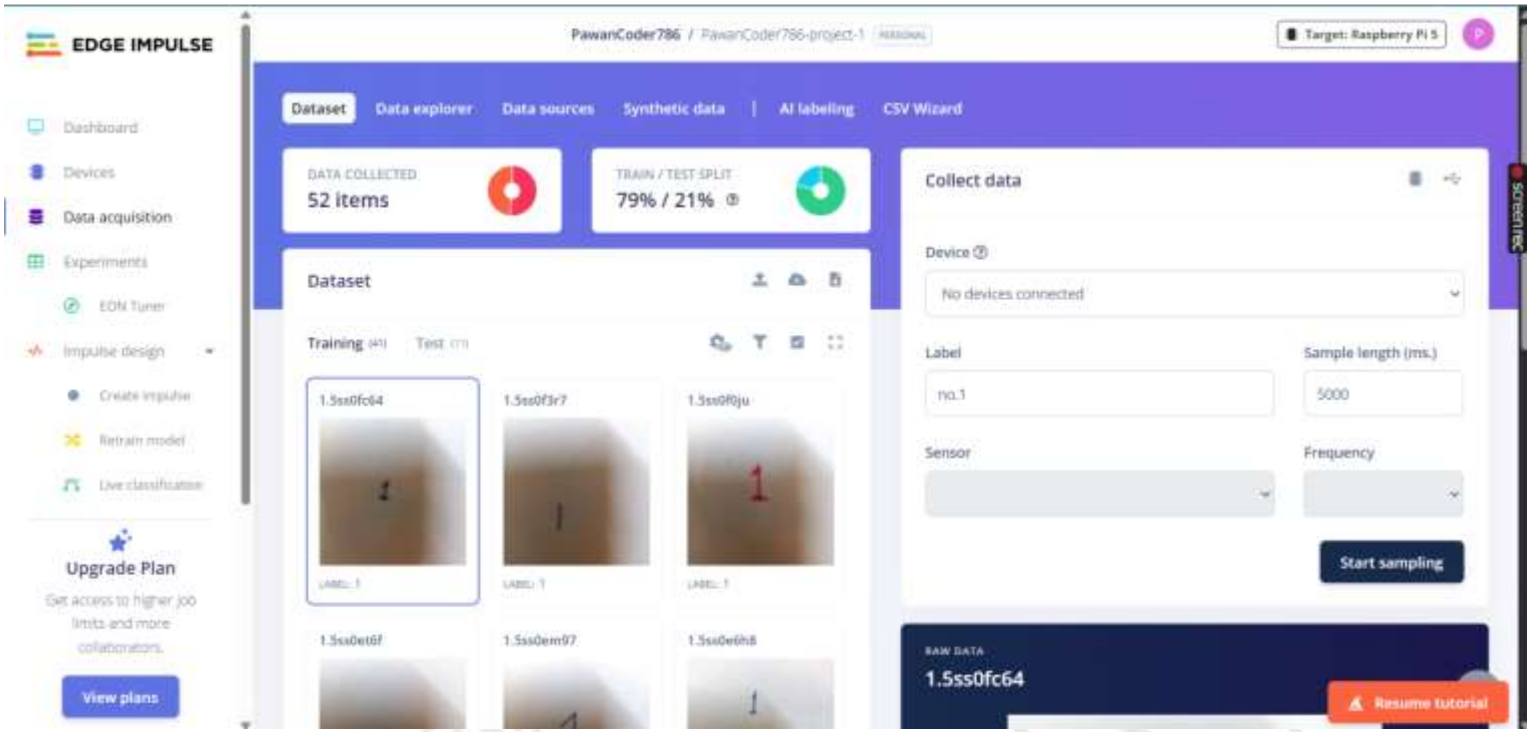
ESP32 Model Creation & Deployment Demo Section



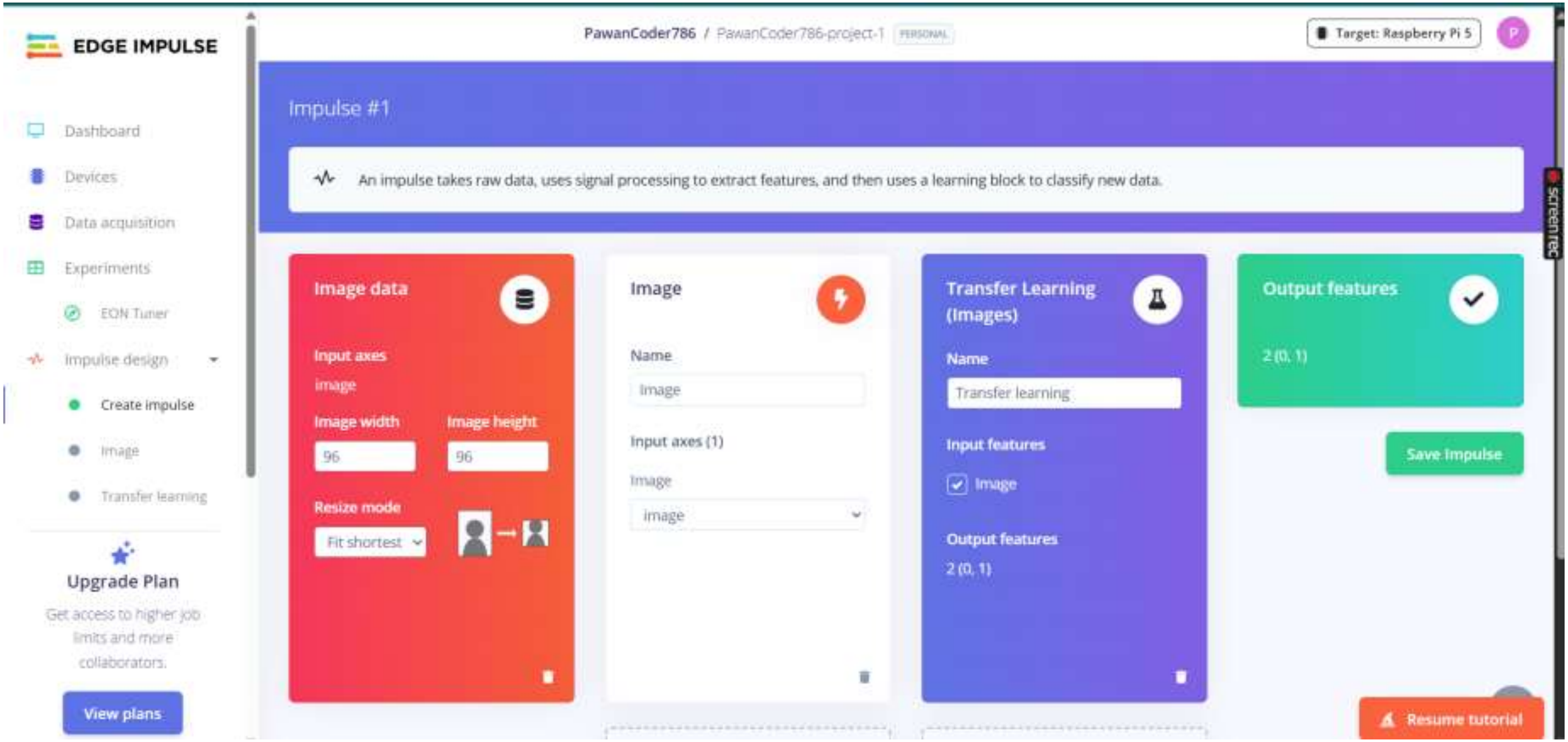
Model Development Workflow

- **Data Collection**:
  - Gathered images (for OCR) and sensor data (for anomaly detection) using Edge Impulse and hardware sensors.
- **Model Training**:
  - Used MobileNetV2 for image classification and K-means for anomaly detection; trained and validated models in Edge Impulse and TensorFlow Lite Micro.
- **Quantization**:
  - Converted trained models to lightweight TFLite format for efficient deployment on embedded devices.
- **Deployment**:
  - Exported models as Arduino libraries; integrated and tested on ESP32/Arduino for real-time inference.

Data Acquisition for OCR Model



Edge Impulse Impulse Creation



Training & Quantization

'raining output

```
 1.0000 - 499ms/epoch - 499ms/step
poch 10/10
/1 - 0s - loss: 0.0805 - accuracy: 1.0000 - val_lo
 1.0000 - 480ms/epoch - 480ms/step
inished training

aving best performing model... (based on validati
till saving model...
aving best performing model OK

onverting TensorFlow Lite float32 model...
ttached to job 34057611...
onverting TensorFlow Lite int8 quantized model...
```

Neural Network settings

Training settings

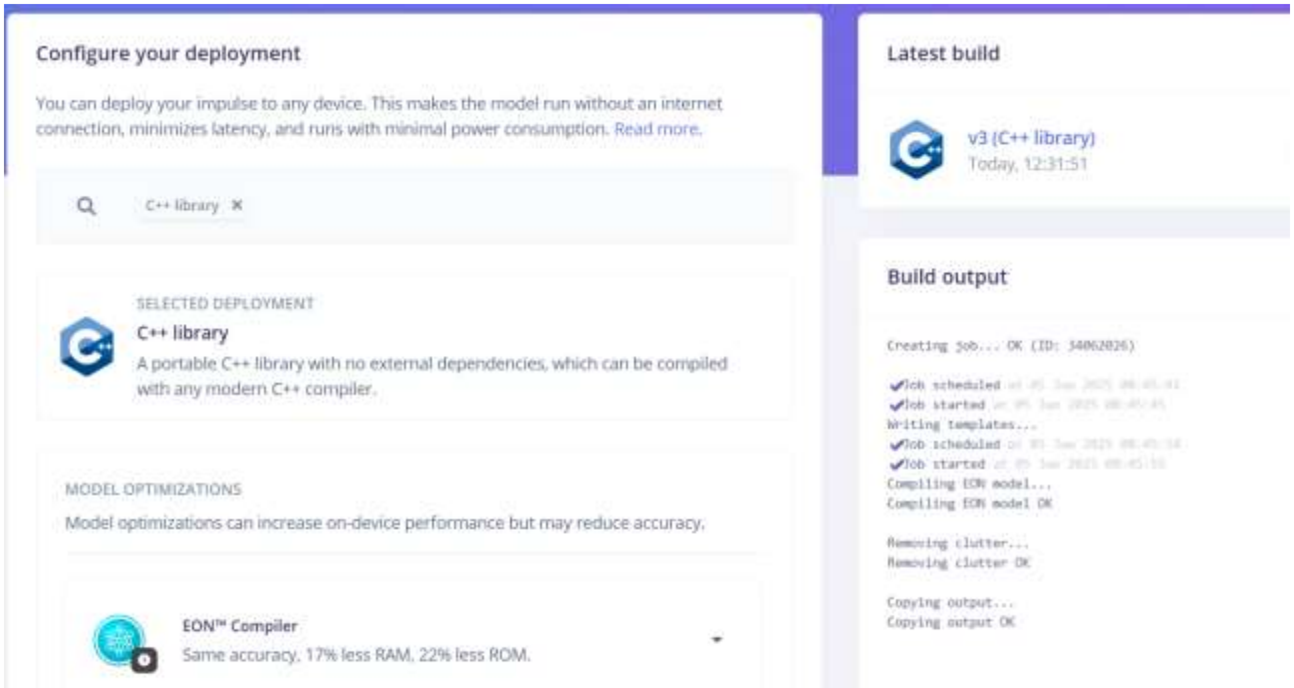
Training processor ⓘ CPU

Advanced training settings

Neural network architecture Save

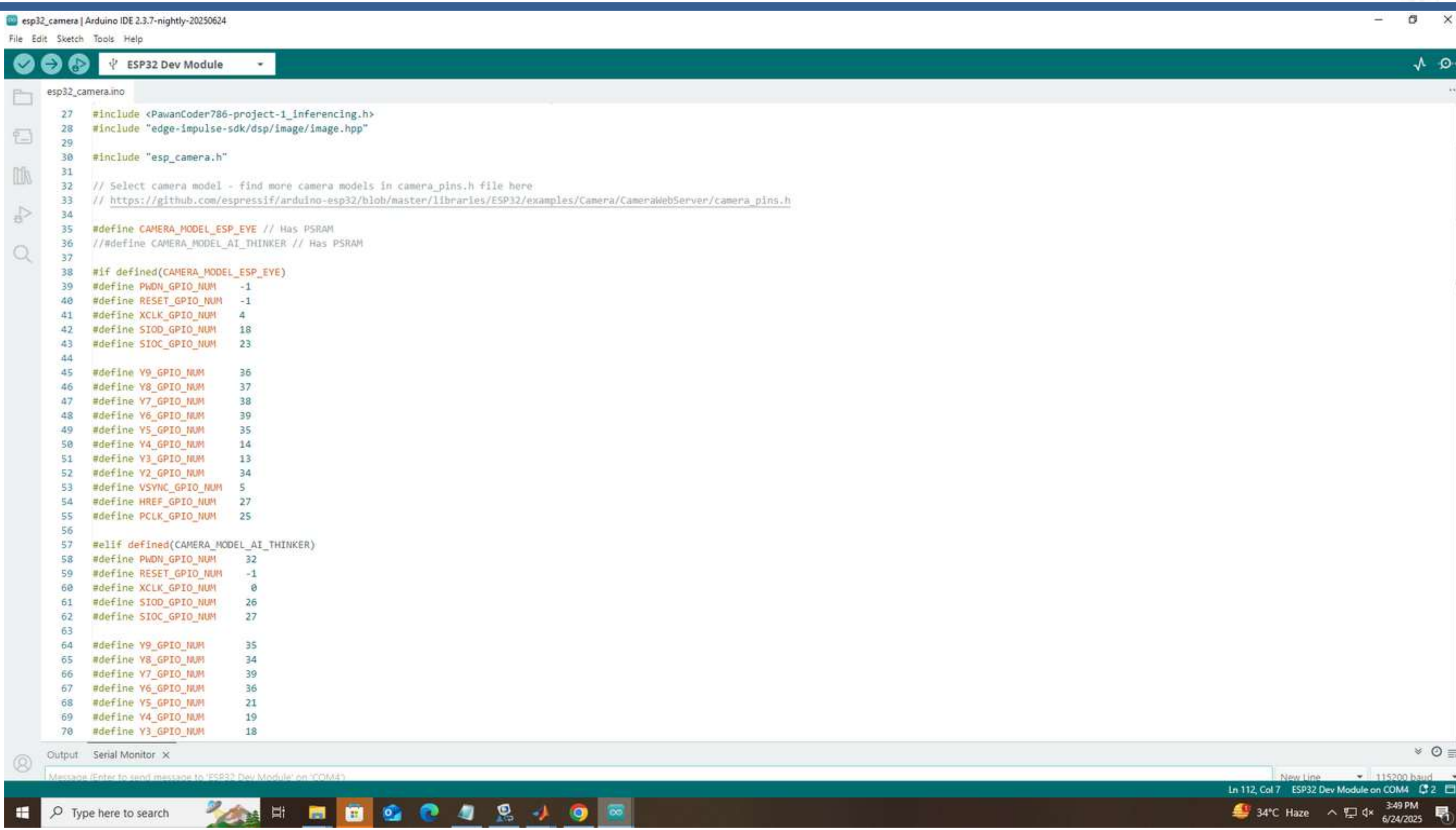
```python

import math, requests
from pathlib import Path
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Dense, InputLayer, Dropout, Conv1D, Flatten, Reshape, MaxPooling1D
        , BatchNormalization,
    Conv2D, GlobalMaxPooling2D, Lambda, GlobalAveragePooling2D)
from tensorflow.keras.optimizers.legacy import Adam, Adadelta
from tensorflow.keras.losses import categorical_crossentropy


sys.path.append('./resources/libraries')
import ei_tensorflow.training

WEIGHTS_PATH = './transfer-learning-weights/keras
    /mobilenet_v2_weights_tf_dim_ordering_tf_kernels_0.35_96.h5'

# Download the model weights
root_url = 'https://cdn.edgeimpulse.com/'
p = Path(WEIGHTS_PATH)
if not p.exists():
```

Exporting & Integrating Model with Arduino IDE



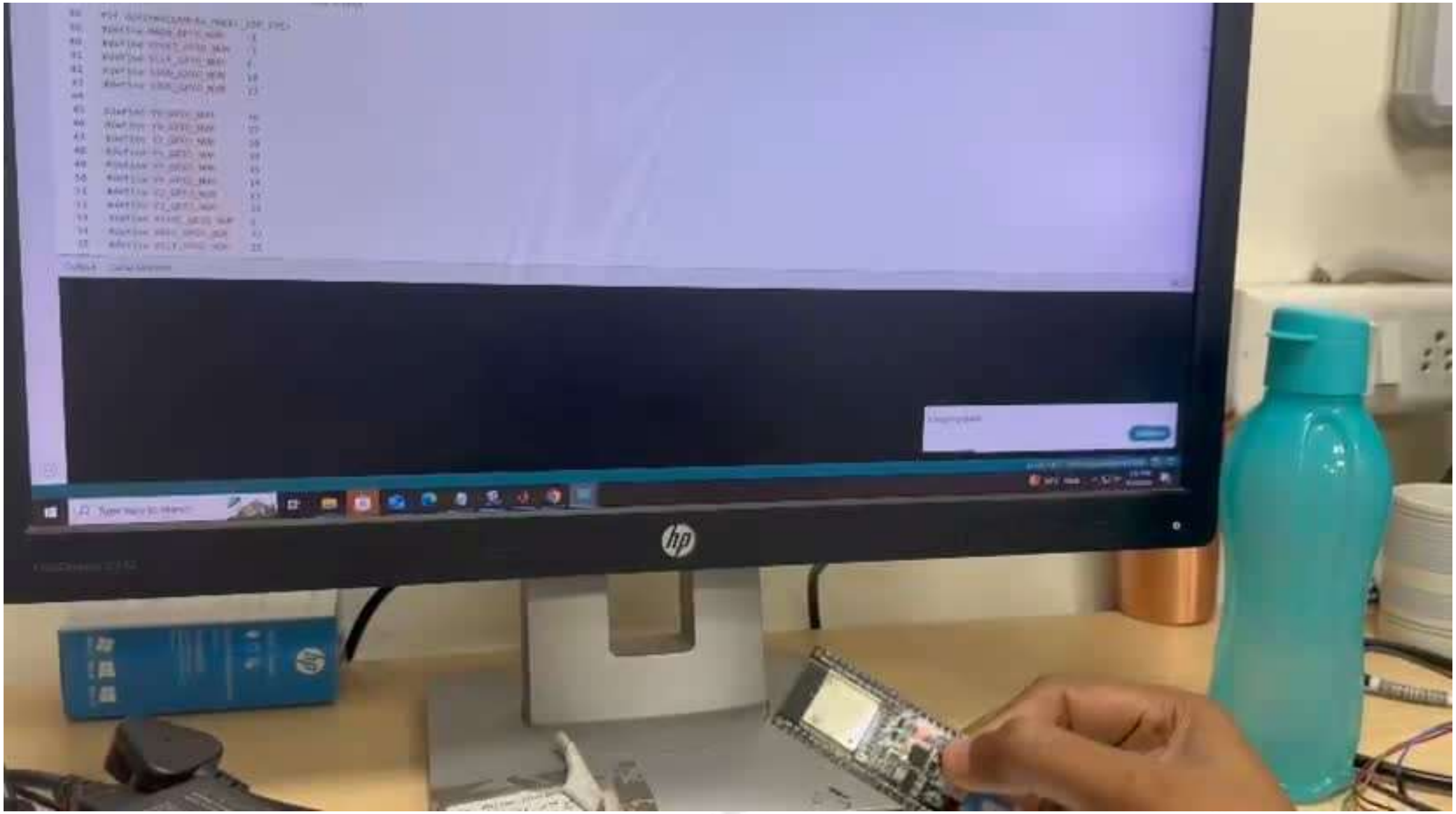
Name	Size	Date Modified
README.txt	1.2 KB	2025-06-05 08:46:20
CMakeLists.txt	1 KB	2025-06-05 08:46:20
model-parameters		2025-06-05 14:16:45
tflite-model		2025-06-05 14:16:45
edge-impulse-sdk		2025-06-05 14:16:45

ESP32 Code Integration



- Adding the Model in the form of a zip add library

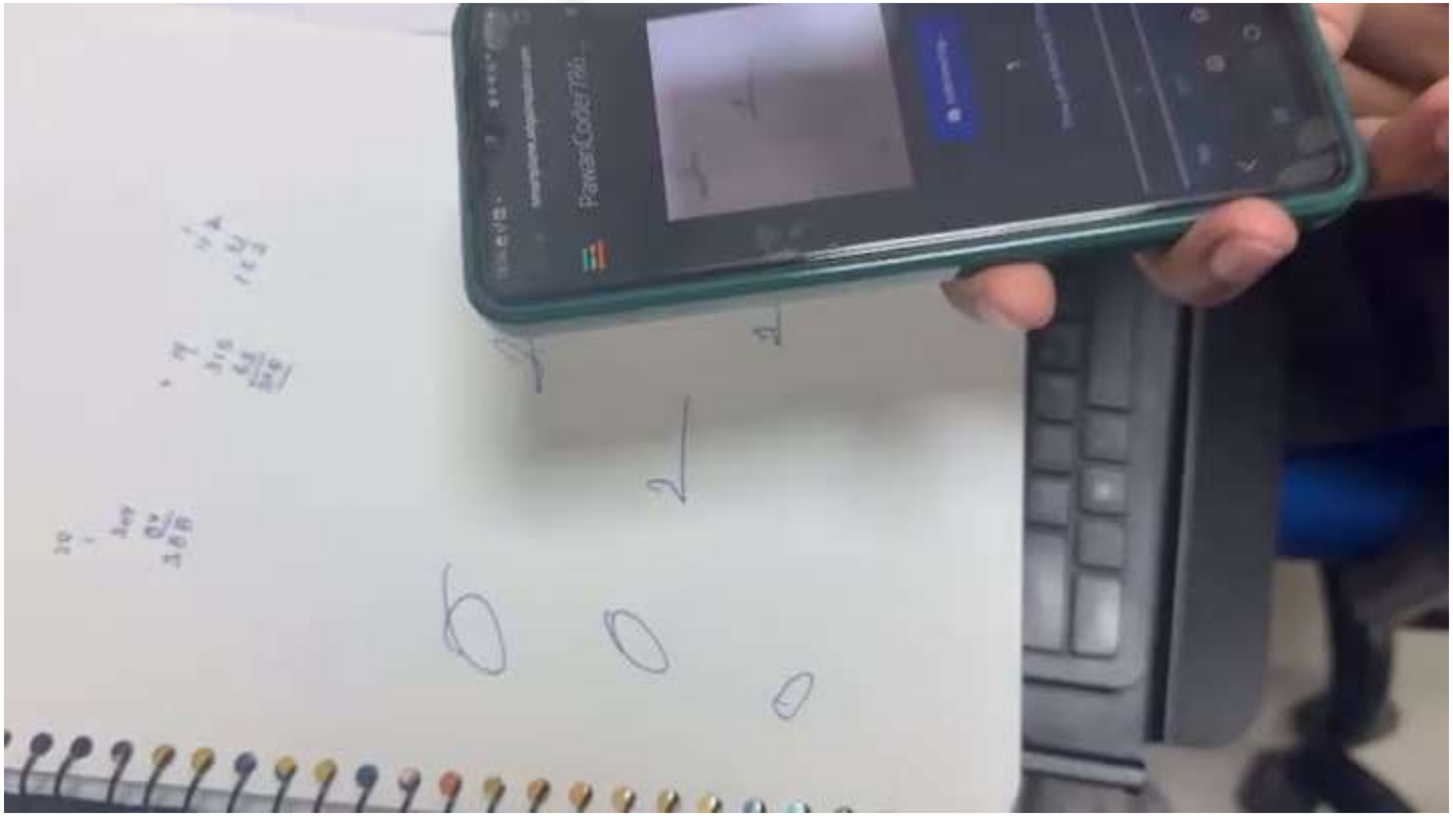
Compiling & Uploading to ESP32



Not able to compile b/c of no camera module and less RAM so can't take feature vector as input

Live Demo: OCR Model Inference

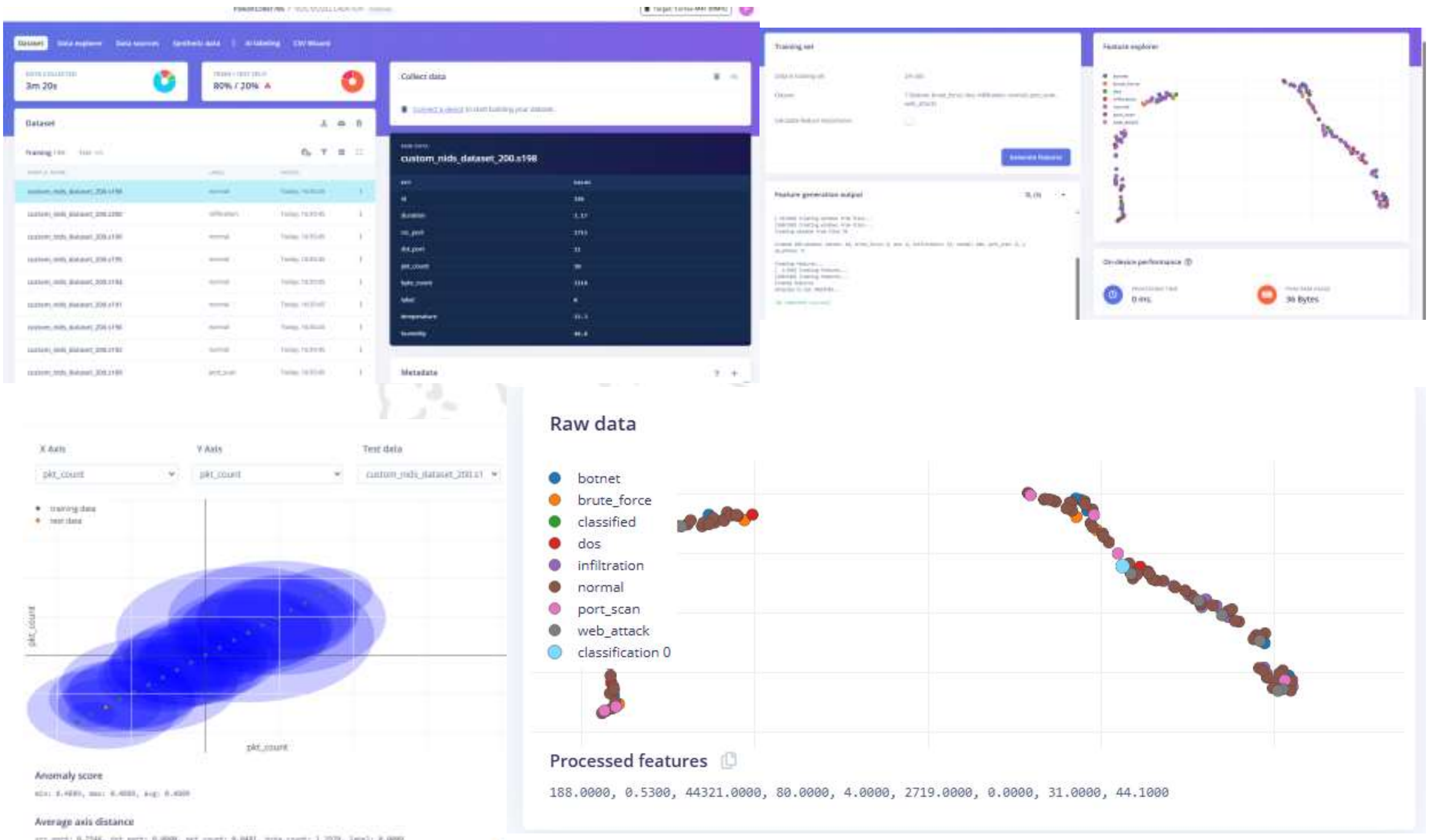
- Using Android Device for Live Classification
- Project Link:-https://studio.edgeimpulse.com/public/725817/live



Anomaly Detection Model Demo Section

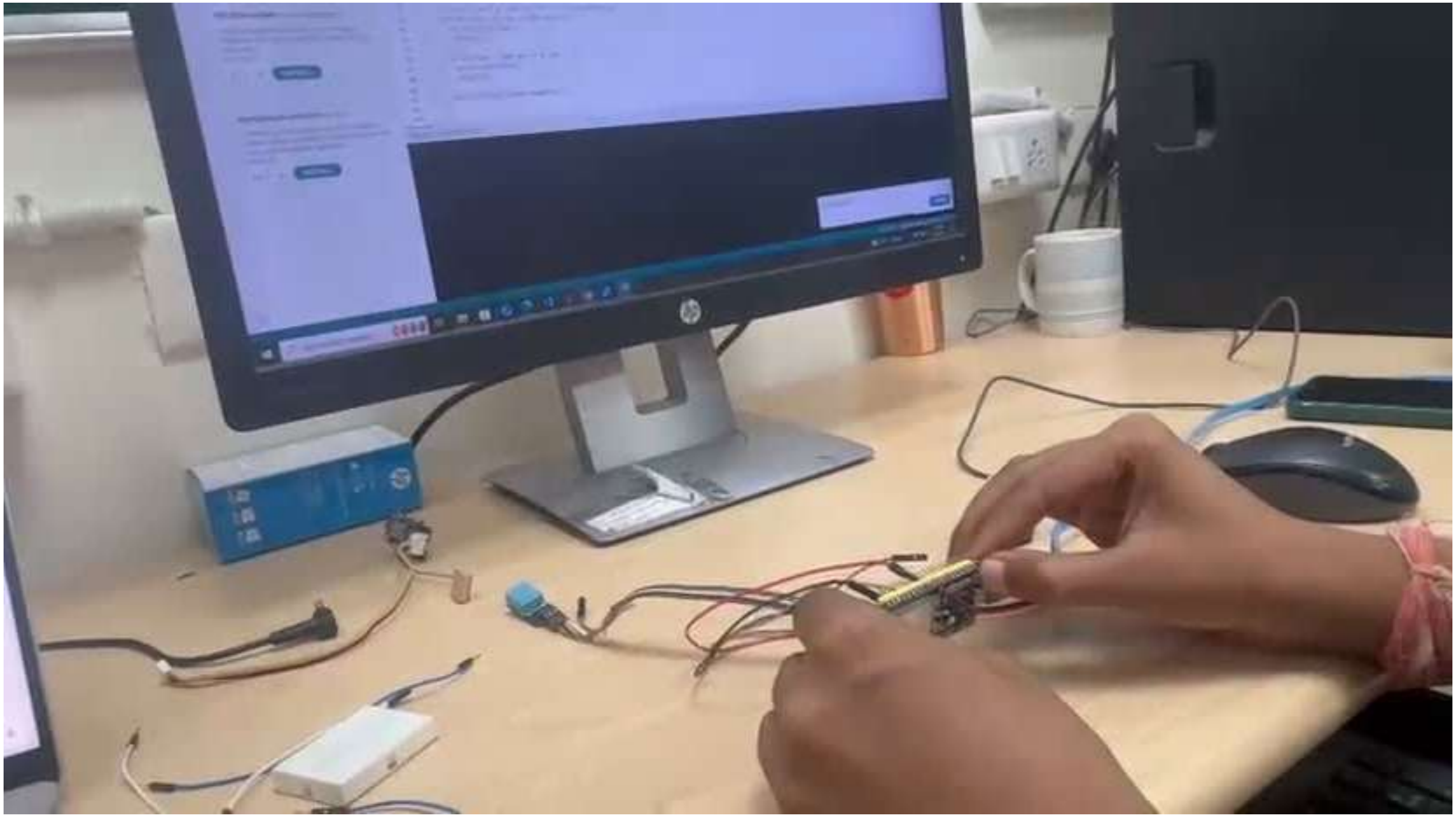
- Collected **real-time temperature and humidity data using DHT11 sensor on ESP32/Arduino.**
- Trained a **K-means clustering model** in Edge Impulse to detect abnormal sensor readings (anomalies) in both temperature and humidity.
- Deployed the model for live anomaly detection; flagged unusual values and triggered servo control for demonstration.
- module on ESP32, monitoring network traffic patterns**Proposed and partially implemented a network intrusion detection system (NIDS)** alongside sensor data for comprehensive anomaly detection.
- Demonstrated **robust, real-time environmental and network anomaly detection** on embedded edge devices.

Edge Impulse Workflow for Anomaly Detection



Live Demo: Real-Time Anomaly Detection

- Project Link(to see trained model):-https://studio.edgeimpulse.com/public/730895/live



Multi-Classification Model

- Developed a **multi-class classification model** using TensorFlow/Keras for embedded deployment.
- Steps included **environment setup, data normalization, and configuring a data loader** (batch size 30).
- Trained the model using cross-entropy loss; evaluated performance on validation and test datasets.
- **Quantized the trained model to TFLite format** for deployment on ESP32/Arduino, following the same workflow as other models.
- Successfully tested real-time predictions on embedded hardware, confirming multi-class inference capability on resource-constrained devices

Steps Involved:-

Libraries Installation:-

Oceans cover two-thirds of the planet. In this assignment, you will build a classifier to tell several types of creatures apart.

```
import os

from collections import Counter

import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from PIL import Image
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from torch.utils.data import DataLoader, random_split
from torchinfo import summary
from torchvision import datasets, transforms
from tqdm.notebook import tqdm

torch.backends.cudnn.deterministic = True
```

```python
if torch.cuda.is_available():
    device = "cuda"
```

```python
os.listdir("sea_creatures")
```

```
['test', 'train']
```

```python
train_dir = "sea_creatures/train"

# Get the list of class names (each folder is a class)
classes = os.listdir(train_dir)

# Print the class names
print(classes)
```

```
['Puffers', 'Sea Urchins', 'Turtle_Tortoise', 'Whale', 'Jelly Fish', 'Sharks', 'Octopus', 'Sea Rays', 'Dolphin']
```

Transform Pipeline:-

```python
height = 224
width = 224
class ConvertToRGB:
    def __call__(self, img):
        if img.mode != "RGB":
            img = img.convert("RGB")
        return img
transform = transforms.Compose([
    ConvertToRGB(),
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
print(transform)
```

```
Compose(
    <__main__.ConvertToRGB object at 0x7fa102cbbb10>
    Resize(size=(224, 224), interpolation=bilinear, max_size=None, antialias=True)
    ToTensor()
)
```

```python
sample_file = "sea_creatures/train/Dolphin/10004986625_0f786ab86b_b.jpg"

image = Image.open(sample_file)

transformed_image = transform(image)
print(transformed_image.shape)
```

```
torch.Size([3, 224, 224])
```

```python
dataset = ImageFolder("sea_creatures/train", transform=transform)
print("Image size", dataset[0][0].shape)
print("Label", dataset[0][1])
```

```
Image size torch.Size([3, 224, 224])
Label 0
```

Data loader(batch size-30)

```python
batch_size = 32
dataset_loader = DataLoader(dataset, batch_size=batch_size)
# Get one batch
first_batch = next(iter(dataset_loader))
print(f"Shape of one batch: {first_batch[0].shape}")
print(f"Shape of labels: {first_batch[1].shape}")
```

```
Shape of one batch: torch.Size([32, 3, 224, 224])
Shape of labels: torch.Size([32])
```

Transform Normalize:-

```
transform_norm =transforms.Compose(
    [
        ConvertToRGB(),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=mean, std=std),
    ]
)
print(transform_norm)
```

```
Compose(
    <__main__.ConvertToRGB object at 0x7fa100bbda50>
    Resize(size=(224, 224), interpolation=bilinear, max_size=None, antial
    ToTensor()
    Normalize(mean=tensor([0.2992, 0.4125, 0.4588]), std=tensor([0.2697,
)
```

```
norm_dataset = datasets.ImageFolder(root=train_dir,transform=transform_no
print("Image size", norm_dataset[0][0].shape)
print("Label", norm_dataset[0][1])
```

```
Image size torch.Size([3, 224, 224])
Label 0
```

Set up data loaders for both the training and validation data sets. Use the same batch size as before. Remember to set `shuffle=True` on the training loader.

```
train_loader = DataLoader(train_dataset,batch_size=32,shuffle=True)

val_loader = DataLoader(val_dataset,batch_size=batch_size)
```

- Build Model:-

```
model.append(torch.nn.Dropout(p=0.5))
model.append(torch.nn.Linear(in_features=576, out_features=500))
model.append(torch.nn.ReLU())
model.append(torch.nn.Dropout())
model.append(torch.nn.Linear(500, 9))  # 9 output classes

summary(model, input_size=(batch_size, 3, height, width))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Sequential                               [32, 9]                   --
├─Conv2d: 1-1                            [32, 16, 224, 224]        448
├─ReLU: 1-2                              [32, 16, 224, 224]        --
├─MaxPool2d: 1-3                         [32, 16, 56, 56]          --
├─Conv2d: 1-4                            [32, 32, 56, 56]          4,640
├─ReLU: 1-5                              [32, 32, 56, 56]          --
├─MaxPool2d: 1-6                         [32, 32, 14, 14]          --
├─Conv2d: 1-7                            [32, 64, 14, 14]          18,496
├─ReLU: 1-8                              [32, 64, 14, 14]          --
├─MaxPool2d: 1-9                         [32, 64, 3, 3]            --
├─Flatten: 1-10                          [32, 576]                 --
├─Dropout: 1-11                          [32, 576]                 --
├─Linear: 1-12                           [32, 500]                 288,500
├─ReLU: 1-13                             [32, 500]                 --
├─Dropout: 1-14                          [32, 500]                 --
├─Linear: 1-15                           [32, 9]                   4,509
==========================================================================================
Total params: 316,593
Trainable params: 316,593
Non-trainable params: 0
```

Cross Entropy Loss:-

```
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
model.to(device)
# Send the model to the GPU
```

```
Sequential(
  (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
  (2): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): ReLU()
  (5): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): ReLU()
  (8): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (9): Flatten(start_dim=1, end_dim=-1)
  (10): Dropout(p=0.5, inplace=False)
  (11): Linear(in_features=576, out_features=500, bias=True)
  (12): ReLU()
  (13): Dropout(p=0.5, inplace=False)
  (14): Linear(in_features=500, out_features=9, bias=True)
)
```

Training Of Dataset

```
# Import the train and predict functions from `training.py`, instead of typing them out!
from training import train, predict
epochs = 10
train(model,optimizer,loss_fn,train_loader,val_loader,epochs=10,device=device)
# Train the model for 10 epochs
```

```
Training:    0%|          | 0/155 [00:00<?, ?it/s]
Scoring:    0%|          | 0/39 [00:00<?, ?it/s]
Epoch: 1, Training Loss: 1.69, Validation Loss: 1.50, Validation accuracy = 0.47
Training:    0%|          | 0/155 [00:00<?, ?it/s]
Scoring:    0%|          | 0/39 [00:00<?, ?it/s]
Epoch: 2, Training Loss: 1.46, Validation Loss: 1.40, Validation accuracy = 0.51
Training:    0%|          | 0/155 [00:00<?, ?it/s]
Scoring:    0%|          | 0/39 [00:00<?, ?it/s]
```

Evaluate Model Performance:-

```
# Compute the probabilities for each validation image
probabilities = predict(model,val_loader,device)
# Get the index associated with the largest probability for each
predictions = torch.argmax(probabilities,dim=1)

print("Number of predictions:", predictions.shape)
```

```
Predicting:    0%|          | 0/39 [00:00<?, ?it/s]
Number of predictions: torch.Size([1236])
```

Testing of dataset:-

```
test_dir = "sea_creatures/test"
test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

test_dataset = test_dataset = datasets.ImageFolder(root=test_dir, transform=test_transforms)

print("Number of test images:", len(test_dataset))

test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
Number of test images: 699
```

```
# Predict the probabilities for each test image
test_probabilities = predict(model, test_loader, device=device)

# Get the index associated with the largest probability for each test image
test_predictions = torch.argmax(test_probabilities, dim=1)

print("Number of predictions:", test_predictions.shape)
```

```
Predicting:   0%|          | 0/22 [00:00<?, ?it/s]
Number of predictions: torch.Size([699])
```

Task 1.5.22: Convert the class index to the class name for each test image.

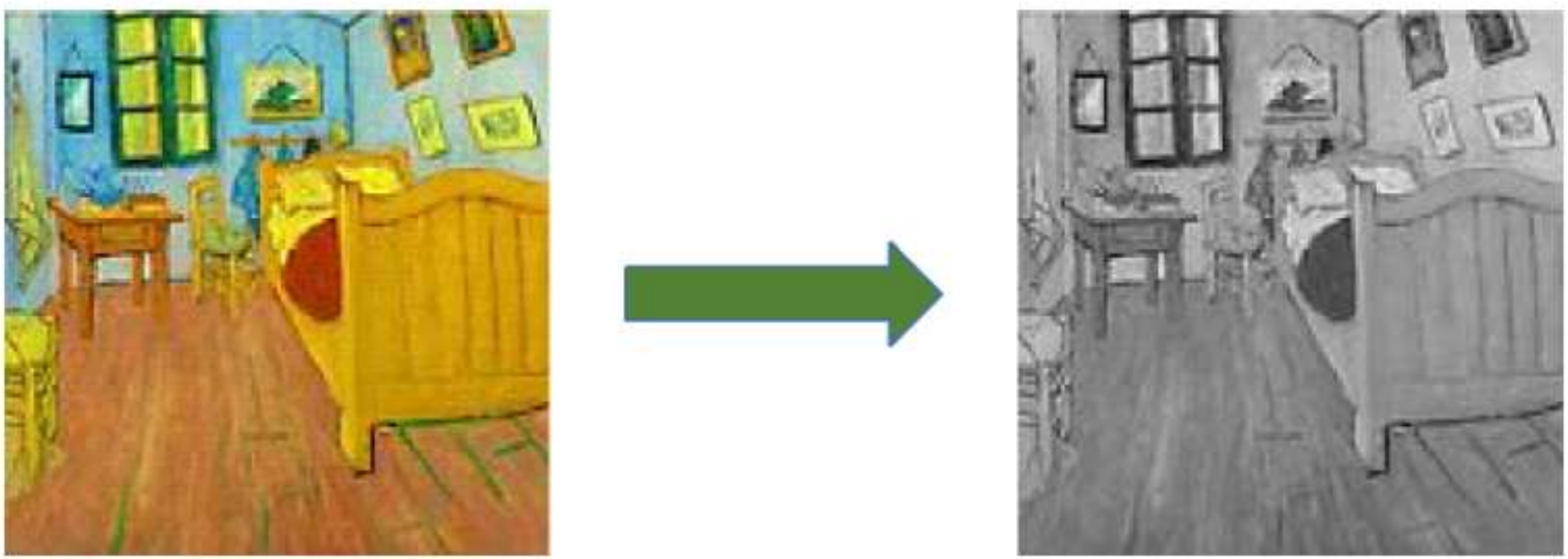
```
test_classes = [classes[i] for i in test_predictions]

print("Number of class predictions:", len(test_classes))
```

Predictions through the validation dataset

```python
import matplotlib.pyplot as plt
import random

# Sample 12 random indices from the test dataset
sample_indices = random.sample(range(len(test_loader.dataset.samples)), 12)

# Create a grid of 4x3 subplots
fig, axes = plt.subplots(4, 3, figsize=(20, 10))

# Iterate over the sampled indices and plot the corresponding images
for ax, idx in zip(axes.flatten(), sample_indices):
    image_path = test_loader.dataset.samples[idx][0]
    img = Image.open(image_path)

    # Display the image on the axis
    ax.imshow(img)
    ax.axis('off')

    # Get the predicted class for this image
    predicted_class = test_classes[idx]

    # Set the title of the subplot to the predicted class
    ax.set_title(f"Predicted: {predicted_class}", fontsize=14)

plt.tight_layout()
```

Image Processing on RISC-V FPGA

Program that processes an RGB image (left side of the image below), and generates a grayscale version of that image (right side of the image below).



Transformation of an RGB Image to a Grayscale Image

Results & Evaluation

- **Real-time, low-power AI** is needed for embedded and IoT devices.
- **RISC-V + FPGA:** Open-source, customizable, flexible hardware for prototyping and acceleration.
- **TinyML:** Brings machine learning to microcontrollers for intelligent edge applications.
- **Combined:** RISC-V simulation with TinyML enables a cost-effective, complete edge AI workflow—no physical FPGA needed.
- **Outcome:** Demonstrates scalable, efficient AI on low-cost, resource-constrained platforms.

Conclusion & Future Work

- Successfully worked on RISC-V architecture using simulation-driven workflows and deployed **TFlite models** on **ESP32/Arduino** for real-time edge AI.
- Demonstrated **practical applications** like digit recognition and anomaly detection, confirming feasibility on resource-constrained hardware.
- **Overcame hardware and dataset limitations through creative use of simulation tools** and embedded deployment.
- **Future work**: Expand datasets, develop more complex TinyML models, and integrate additional peripherals (e.g., camera modules) for richer demonstrations.
- **Plan to deploy and validate the workflow on physical FPGA hardware** and explore hardware acceleration for scalable, energy-efficient edge AI systems.

References

- RVfpga HarvardX edX course and official source code (RISC-V SoC simulation, debugging, and hardware design).
- Whisper RISC-V simulator, Verilator, GTKWave for simulation and instruction-level debugging.
- TinyML foundational courses, TensorFlow Lite Micro documentation, and Edge Impulse guides for model development and deployment.
- WorldQuant University Applied AI Lab content for practical AI workflows.
- MIT Han Lab TinyML GitHub, Efinix TinyML Platform, and community repositories for code and deployment strategies.
- Books, research papers, and presentations on TinyML and RISC-V FPGA integration.
- Internal reports: Final_Report_CoDA_LAb.pdf, week1–6.pptx, week5-1.pptx, and all weekly progress PPTs.
- YouTube FPGA programming playlists and tutorial videos for hardware/software co-design