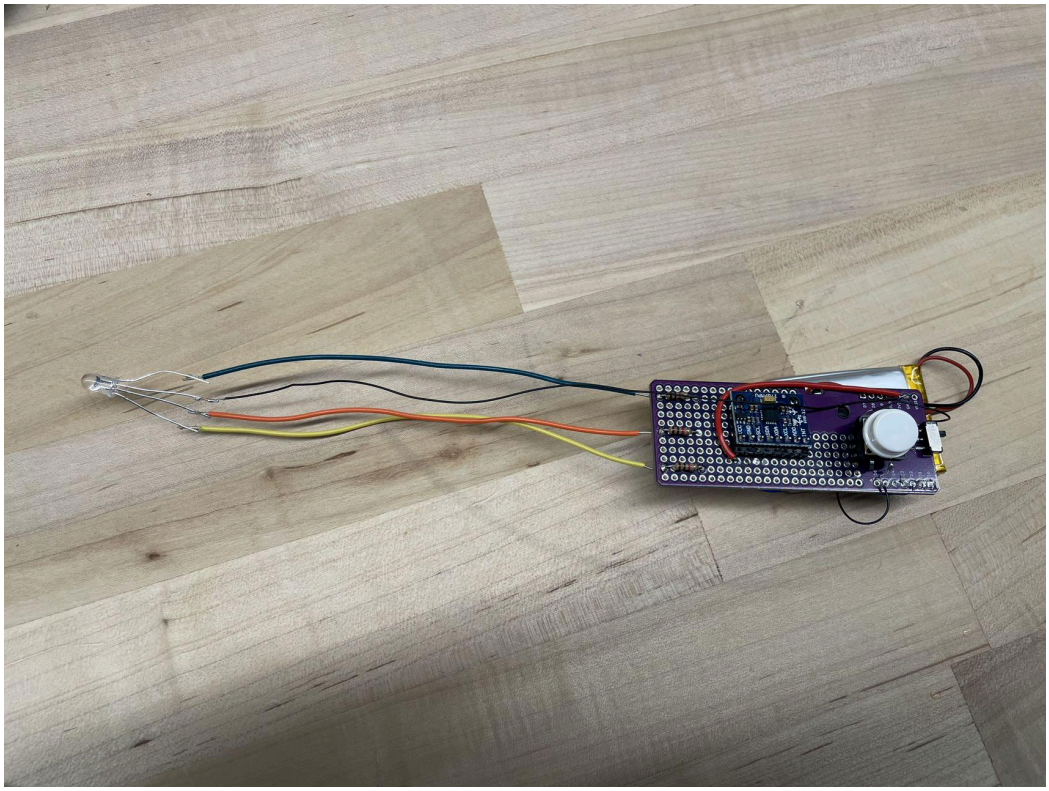
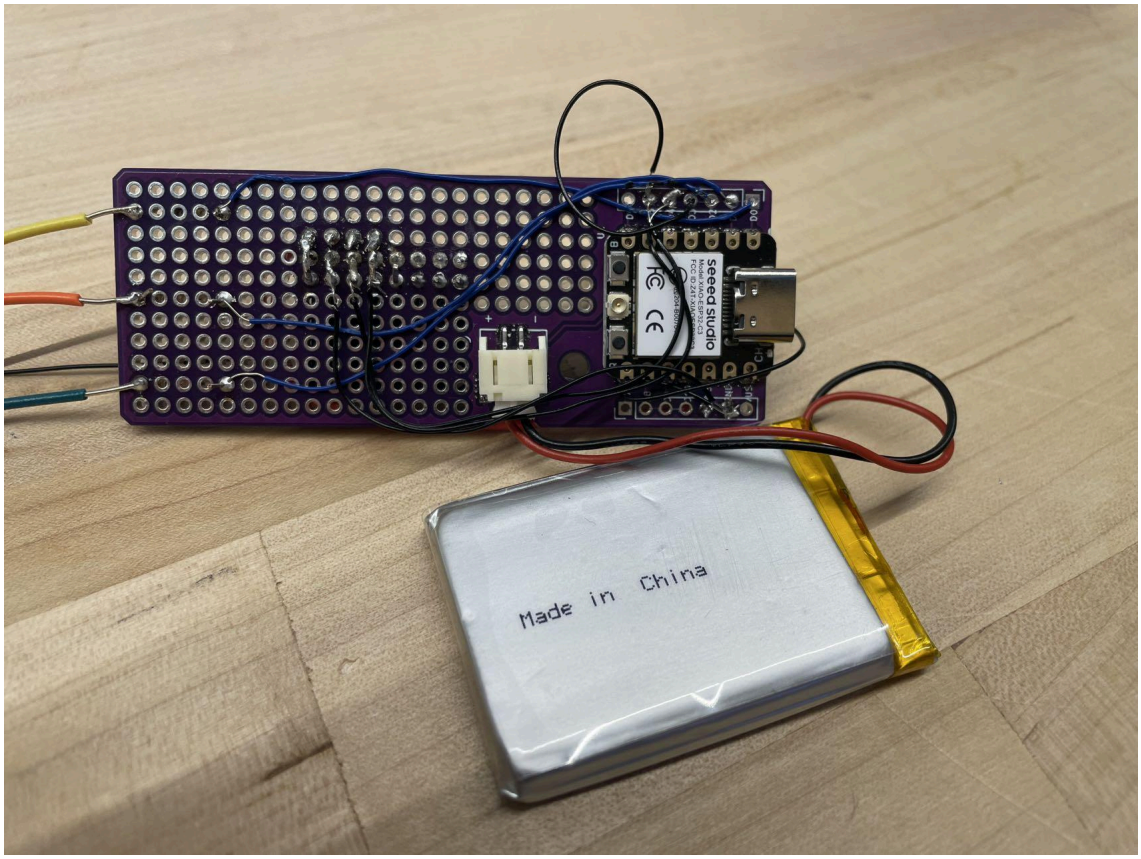
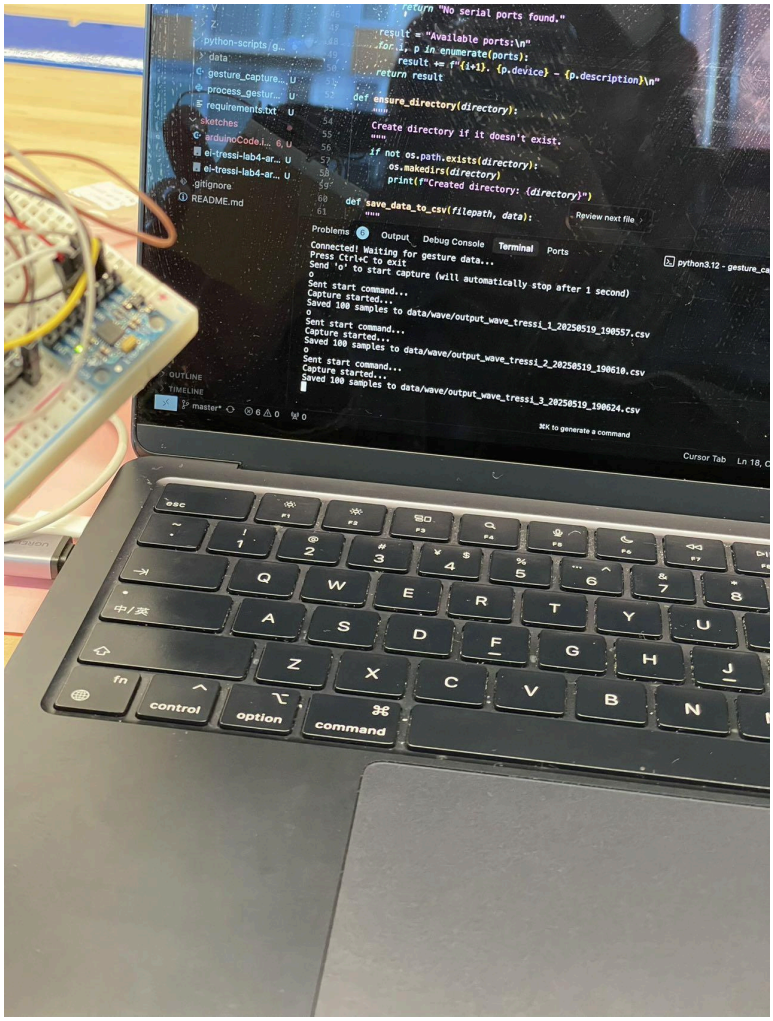
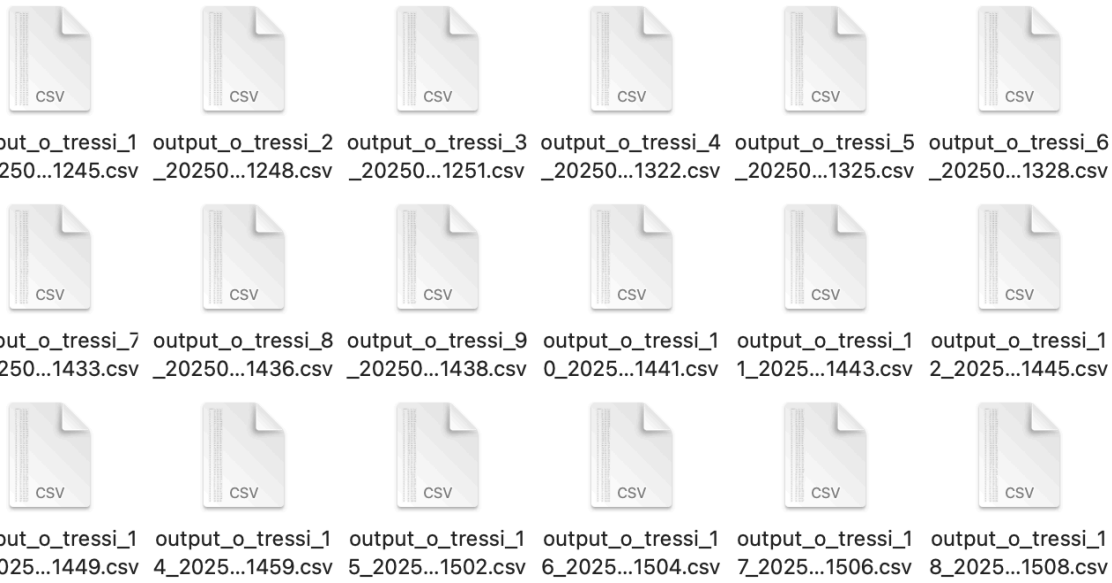
- Pictures of hardware setup and connections





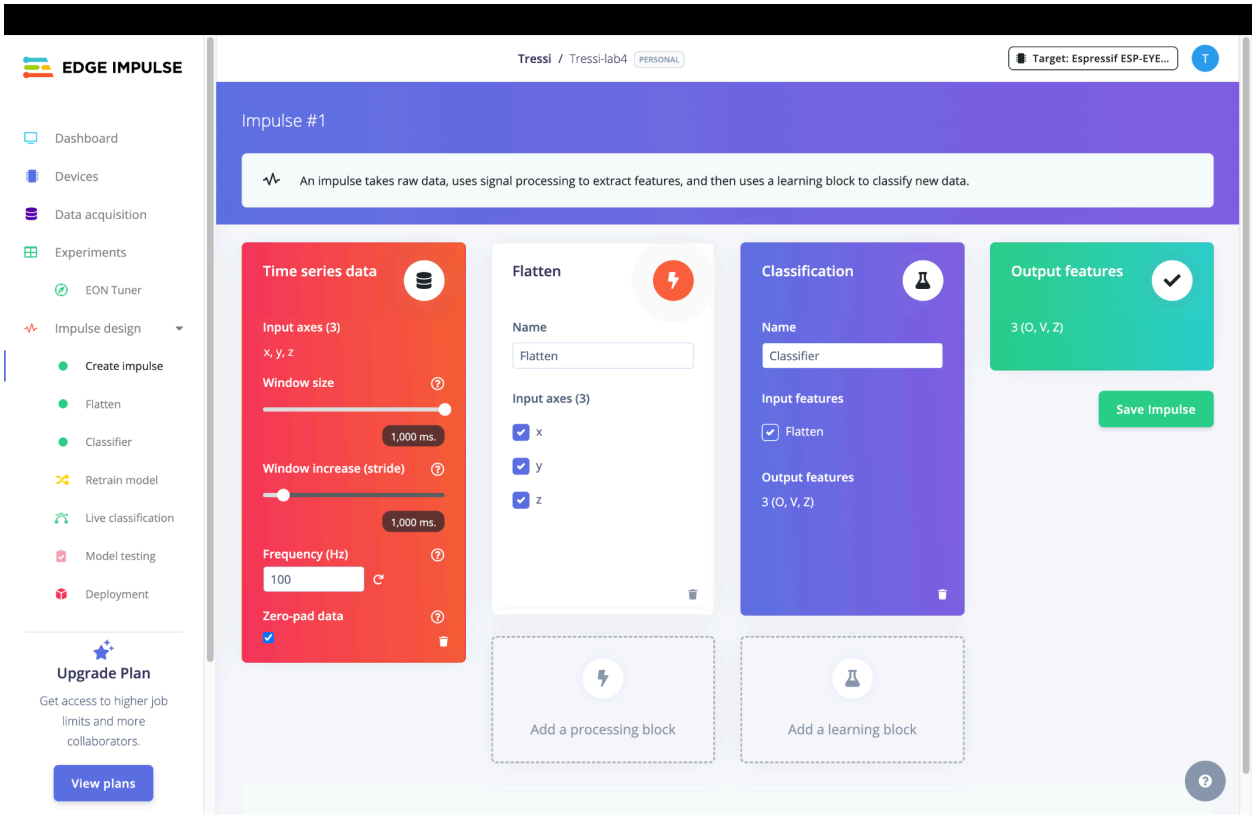
- Data collection process and results





https://drive.google.com/drive/folders/1P98yoxQT7I1G8xl0mnAG1U_o1Ekes4mA?usp=drive_link

- Edge Impulse model architecture and optimization



Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY
91.8%

LOSS
0.56

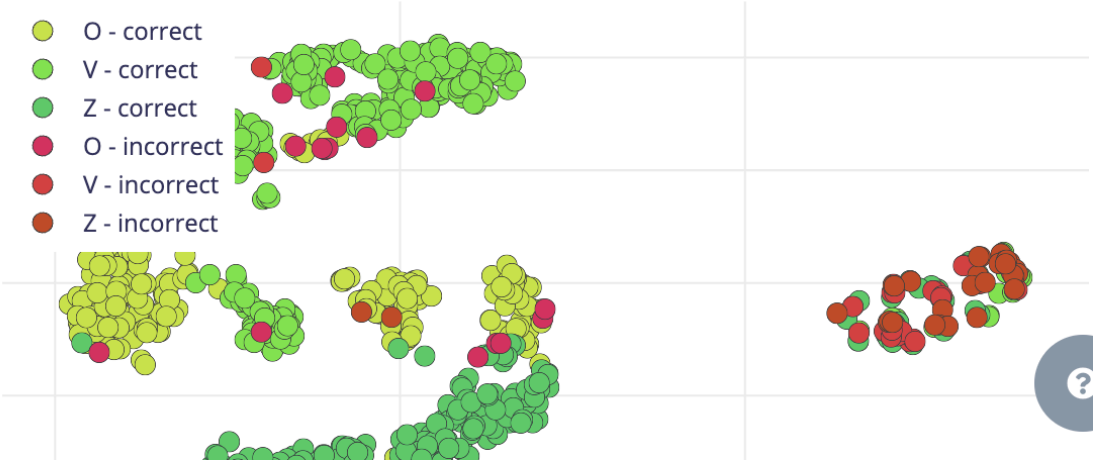
Confusion matrix (validation set)

	O	V	Z
O	93.4%	3.8%	2.8%
V	2.0%	90.9%	7.1%
Z	3.0%	6.1%	90.9%
F1 SCORE	0.94	0.90	0.90

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve	0.98
Weighted average Precision	0.92
Weighted average Recall	0.92
Weighted average F1 score	0.92

Data explorer (full training set)



- Performance analysis and metrics

Results

Model version: Unoptimized (float32)

ACCURACY
88.77%

Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	0.98
Weighted average Precision	0.90
Weighted average Recall	0.90
Weighted average F1 score	0.90

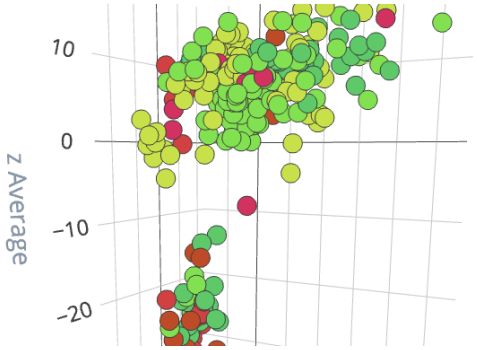
Confusion matrix

	O	V	Z	UNCERTAIN
O	91.3%	2.4%	2.4%	4.0%
V	4.3%	82.9%	10.3%	2.6%
Z	0.8%	7.4%	91.8%	0%
F1 SCORE	0.93	0.86	0.90	

Feature explorer

x Average | y Average | z Average

- O - correct
- V - correct
- Z - correct
- O - incorrect
- V - incorrect
- Z - incorrect



- Answers to questions and your choices to all design options with justifications

  Smaller stride means more overlapping windows and more samples.Longer window means more raw values and larger input size. Larger window sizes can capture full motion in slow gestures. Shorter ones may truncate the pattern.1000ms window + 1000ms stride captures slow gestures better, but increases training size.

  I chose the Flatten DSP block because my input consists of time-series accelerometer data (x, y, z) captured over short windows. Flattening the data preserves the raw temporal structure while keeping the input simple and suitable for lightweight classification. This is particularly useful when gestures have distinctive shapes in the time domain, and the model needs to learn from the full raw motion pattern without abstraction or transformation. It also allows faster experimentation and easier debugging during early development.

- Demo video link
  https://drive.google.com/drive/folders/1IczNkL0uYPg4GFtSkJ3pL_1LNlhnxFjm?usp=sharing

- Challenges faced and solutions

Challenge:

1. MPU6050 Not Detected by ESP32

- **Cause:** I2C pins were not explicitly defined, and ESP32's default I2C pins didn't match my wiring.

- **Solution:** I manually configured the I2C interface using Wire.begin(4, 5) to match the physical connections (SDA on GPIO4, SCL on GPIO5). This resolved the device detection issue.

2. RGB LED Only Displayed One Color

- **Cause:** The pin-to-color mapping was incorrect, and common cathode (共阴) LED control logic wasn't properly configured.

- **Solution:** I tested each GPIO pin manually and identified the correct color mapping:

  - D0 → Blue

  - D1 → Green

- D2 → Red

I also ensured that digitalWrite(pin, HIGH) was used to light up the LEDs, since it was a common cathode RGB module.

3. Unsure Which Model Type to Choose (KNN vs Neural Network)

- Cause: I was uncertain whether to use a simple traditional model like K-Nearest Neighbors (KNN) or a more flexible neural network. I didn't know how to balance accuracy, training time, and performance on the ESP32.

- Solution: I explored the available learning blocks in Edge Impulse and read the documentation. I started with KNN because it trains quickly, works well on small datasets, and is easy to debug. After testing its performance, I compared it with a small neural network to understand the trade-offs. This hands-on comparison helped me build confidence in choosing the right model for both performance and deployment constraints.