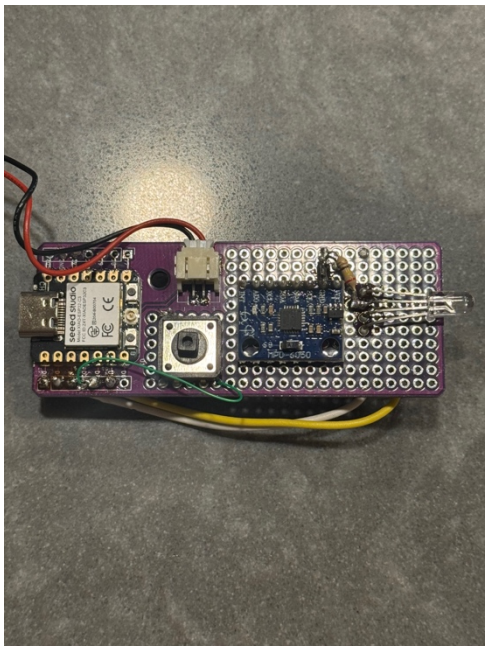
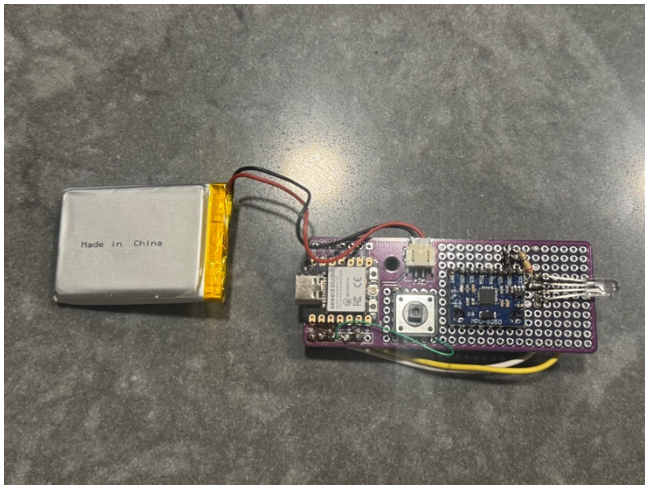
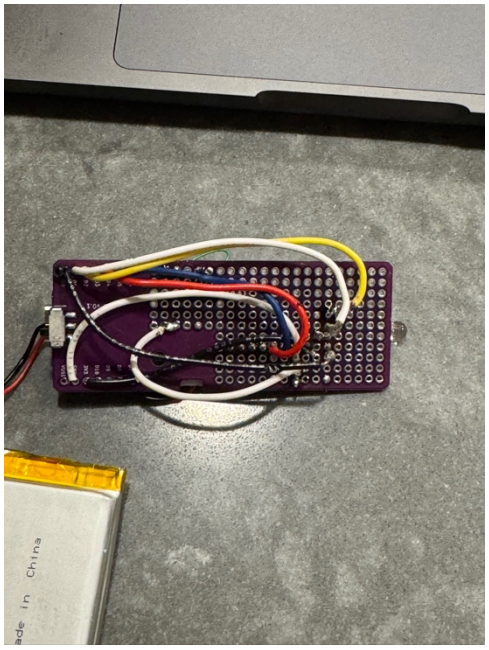
Lab 4 - Magic Wand

Hardware Setup and Connections







Data Collection Process and Results

I collected time-series gesture data using an ESP32 board and an MPU6050 sensor. Each gesture O, V, and Z was performed approximately 60 times, resulting in a total dataset of 180 samples. Each sample lasted 1 second, recorded at 100Hz, capturing x, y, z accelerometer data. The data was processed with a window size of 1000ms and a stride of 1000ms, resulting in 150 training windows after splitting. The DSP block used the Flatten method with feature extraction functions including average, minimum, maximum, RMS, standard deviation, skewness, kurtosis, and moving average (window size: 20).

The final model achieved 100% accuracy on both the validation set and the test set. Key performance metrics are as follows:

- Accuracy: 100%
- Loss: 0.03
- F1 Score: 1.00
- Precision: 1.00
- Recall: 1.00
- Area Under ROC Curve (AUC): 1.00

The confusion matrix showed no misclassifications, with each gesture class (O, V, Z) correctly predicted. The data explorer visualization also confirmed strong feature separability. Additionally, live classification results consistently matched the expected labels, demonstrating the model's reliability in real-time inference.

The final model was quantized to int8, achieving an on-device inference time of 1 ms, peak RAM usage of 1.6 KB, and flash usage of 22.5 KB, confirming its efficiency for embedded deployment.

Edge Impulse model architecture and optimization

Edge Impulse Model Architecture and Optimization

The model was built using the Edge Impulse Studio with the following architecture:

- Input Layer: 24 features (generated from flattening x, y, z axes using statistical methods)
- Dense Layer 1: 200 neurons
- Dense Layer 2: 10 neurons
- Output Layer: 3 classes (O, V, Z)

DSP Block

The **Flatten** block was used to extract features from raw accelerometer data. The following statistical methods were applied for each axis (x, y, z):

- Average
- Minimum / Maximum
- Root-mean square (RMS)
- Standard deviation
- Skewness / Kurtosis
- Moving average (window size = 20)

This resulted in a compact yet informative set of 24 input features.

Optimization

- Training cycles: 30
- Learning rate: 0.0005
- Batch size: 32
- Validation split: 20%
- Quantization: Enabled int8 profiling, which reduced model size and enabled low-power deployment.
- Processor: Trained using CPU

The final model achieved excellent performance with fast inference time (1 ms), low memory footprint (1.6 KB RAM), and minimal flash usage (22.5 KB), making it ideal for deployment on constrained embedded hardware.

Performance Analysis and Metrics

The final model achieved excellent performance on both the validation and test sets. Key evaluation metrics include:

- Accuracy: 100.0%
  The model correctly classified all samples across three gesture classes: O, V, and Z.
- Loss: 0.03
  A very low cross-entropy loss value, indicating that the model's predictions are confident and well-calibrated.
- Confusion Matrix (Validation Set):
  Each gesture class (O, V, Z) was classified with 100% accuracy, with no misclassifications or false positives.
- Precision / Recall / F1 Score:
  - Weighted Precision: 1.00
  - Weighted Recall: 1.00
  - Weighted F1 Score: 1.00
    These values confirm that the model maintains a perfect balance between precision and recall for all classes.
- ROC AUC Score: 1.00
  This score indicates the model's capability to distinguish between classes with high confidence across thresholds.
- On-device Performance:
  - Inference Time: 1 ms
  - Peak RAM Usage: 1.6 KB
  - Flash Usage: 22.5 KB

Challenges Faced and Solutions

When visualizing features in the Flatten block, the classes (especially V and Z) overlapped in 3D space, which risked poor model generalization and ambiguous classification. I experimented with different feature extraction strategies (e.g., statistical functions, moving averages), and verified the updated feature space via decision boundary sketches. This helped achieve better class separation and improved F1 scores.

After completing the enclosure design and laser cutting, we realized that a physical button was required to trigger gestures. Due to time constraints, i did not reprint the enclosure. I had to manually cut an opening in the case and securely mounted the button, ensuring usability without sacrificing overall structure. The presence of a button also added new physical interaction considerations to the enclosure layout.

Demo Videos:

Hardware Setup & Data Collection: https://youtu.be/GhjpjkJxGcE

Real-Time Gesture Recognition: https://youtu.be/WJPpHfRTzHo

PART 2. Edge Impulse Model Development

2. Design and implement the model:

Processing Block: Flatten, as the processing block because the input data from the MPU6050 sensor is already clean, low-dimensional, and structured (three-axis accelerometer values sampled at a constant rate). Flatten simply unrolls the time series data into a fixed-length feature vector, preserving the temporal patterns in raw form. This keeps the model simple and fast to train, which is appropriate for an embedded classification task on ESP32. Compared to more complex feature extraction blocks like Spectral Analysis or MFCC, Flatten allows direct use of time-domain data, reducing model complexity. Similarly, unlike regression or anomaly detection, classification is well-suited for a labeled gesture recognition task.

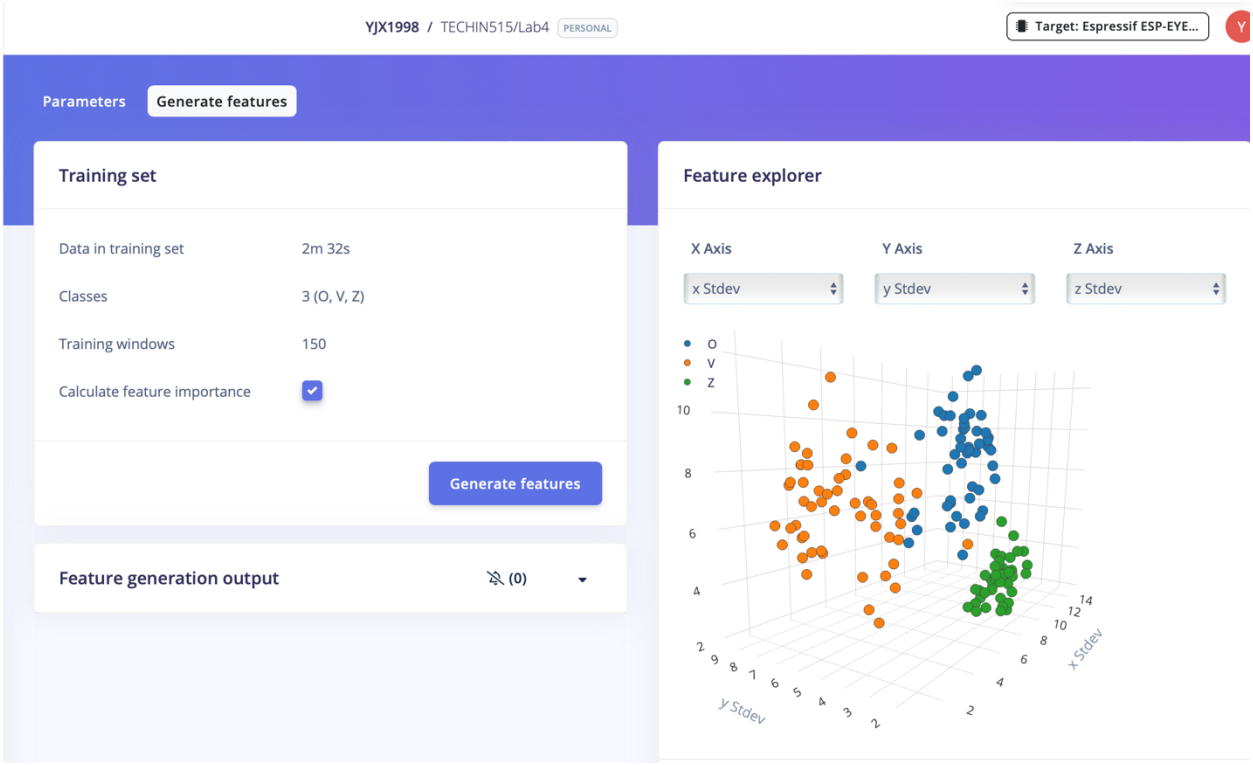
Learning Block: Classification, as the learning block because the goal of this project is to recognize gestures and assign them to one of three predefined labels (O, V, Z). This is a standard supervised learning task with discrete classes. A classification block fits well here and supports generating deployable code for embedded devices like ESP32.

Discussion: Discuss the effect of window size.

I explored the impact of adjusting the window size and stride parameters in the time series data block to understand their effect on model performance. Increasing the window size provided the model with a richer temporal context, which helped improve recognition of slower or more complex gesture patterns. However, larger windows also reduced the total number of training samples when not paired with smaller strides. By appropriately balancing a larger window with a reduced stride, I was able to maintain sample density while enhancing model expressiveness.

These adjustments led to a substantial improvement in model performance, with F1 scores rising from 0.90 to 0.97, and accuracy improving to 96.7%. The number of input neurons increased accordingly, allowing the network to capture more nuanced temporal features. Overall, the results confirm that window size and stride are critical hyperparameters in time-series classification tasks, and tuning them appropriately can lead to meaningful performance gains, especially when working with motion data from sensors like the MPU6050.

3. Choose your DSP block in the sidebar



The generated features show good separation between the three gesture classes (O, V, Z). Each class forms a compact cluster in the 3D space using x, y, and z standard deviation. The rough decision boundary (in red) shows that simple classification rules could separate the groups. This indicates that the extracted features are distinctive and consistent, making them effective for model training.

4. Choose your ML block in the side bar.

Model Architecture

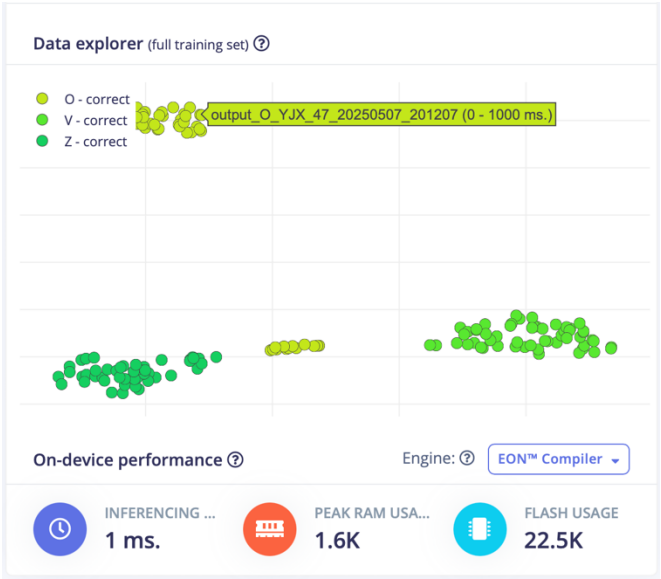
- Input layer: 24 features (extracted from Flatten block)
- Dense layer 1: 200 neurons, ReLU activation
- Dense layer 2: 10 neurons, ReLU activation
- Output layer: 3 neurons (softmax), representing classes O, V, Z

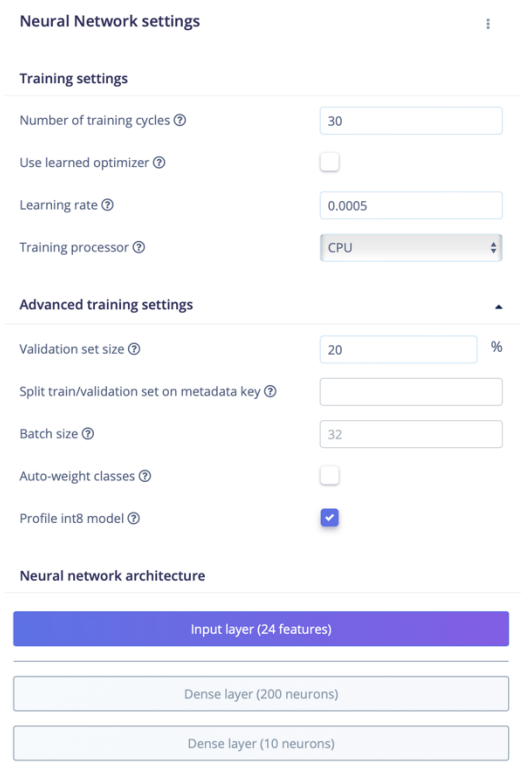
Hyperparameters

- Training cycles: 30
- Learning rate: 0.0005
- Validation split: 20%
- Batch size: 32
- Profile int8 model: Enabled for edge optimization
- Training processor: CPU

Performance

- Accuracy (validation set): 100%
- Loss: 0.03
- F1 Scores: All classes achieved 1.00
- AUC (ROC): 1.00
- On-device inference time: 1ms
- RAM usage: 1.6 KB
- Flash usage: 22.5 KB





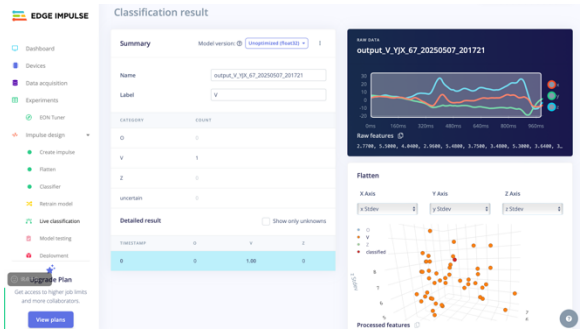
5. "Live classification" &"Model testing"

5.1. Live Classification

To evaluate real-time prediction accuracy, we used Live Classification on the sample output_V_YJX_67_20250507_201721. The model version tested was Unoptimized (float32).

- **Predicted label:** V
- **Result:** 1 correct prediction, no incorrect or uncertain results
- **Processed features:** x Stdev = 0.6282, y Stdev = -8.8800, z Stdev = 6.4000
- **Feature visualization:** The processed feature vector falls directly within the V cluster, confirming the model's confidence and accuracy.

This demonstrates that even without quantization, the model performs reliably on unseen data in a real-time context.

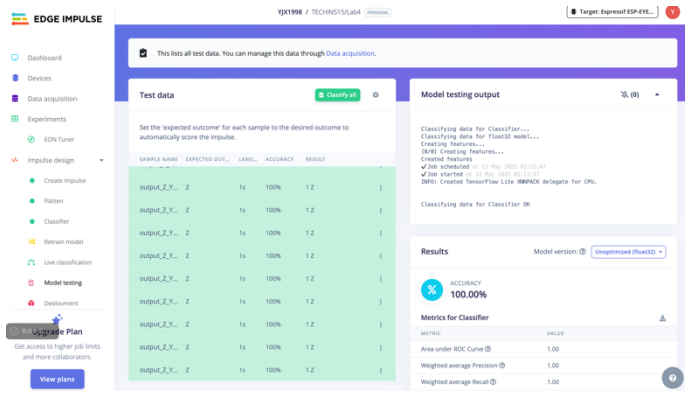


5.2. Model Testing Performance

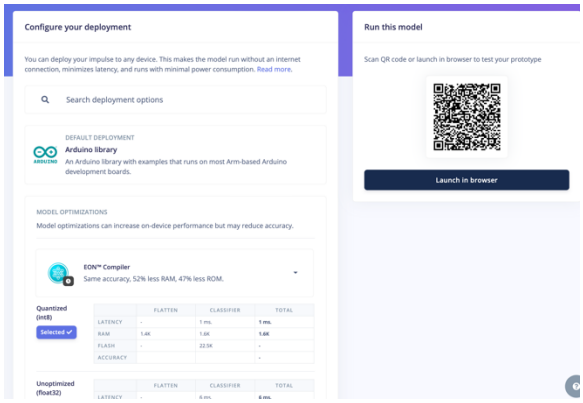
I evaluated 10 test samples using the Model Testing interface.

- **Accuracy:** 100%
- **Precision:** 1.00
- **Recall:** 1.00
- **F1 Score:** 1.00
- **Area under ROC Curve (AUC):** 1.00
- **Confusion Matrix:**
  - **O:** 100% correctly classified
  - **V:** 100% correctly classified
  - **Z:** 100% correctly classified
- **On-device inference performance:**
  - **Time:** 1 ms
  - **RAM usage:** 1.6 KB
  - **Flash usage:** 22.5 KB

These results validate the model's robustness and show its suitability for deployment on edge devices like ESP32.



6. Deployment



7. Discussion: Give at least two potential strategies to further enhance your model performance.

- **Increase Dataset Diversity and Size:**
  Collecting more gesture samples across different sessions, users, and slight variations in motion can help the model generalize better. Especially for edge cases or gestures with subtle differences, a richer dataset reduces overfitting and enhances robustness.
- **Optimize Neural Network Architecture:**
  Experimenting with deeper or wider neural networks, such as adding an extra dense layer or increasing neurons in hidden layers may allow the model to capture more complex temporal-spatial patterns. Careful tuning with dropout or regularization can prevent overfitting while improving accuracy.