Lab 4 Magic Wand Report

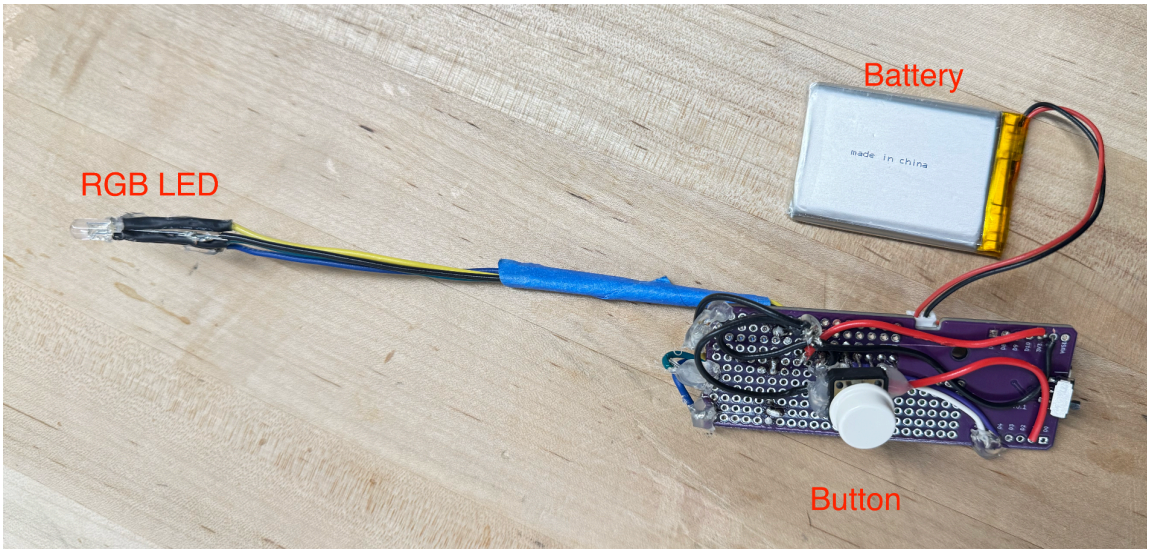
Jazmyn Zhang, TECHIN515, 05/19/2025

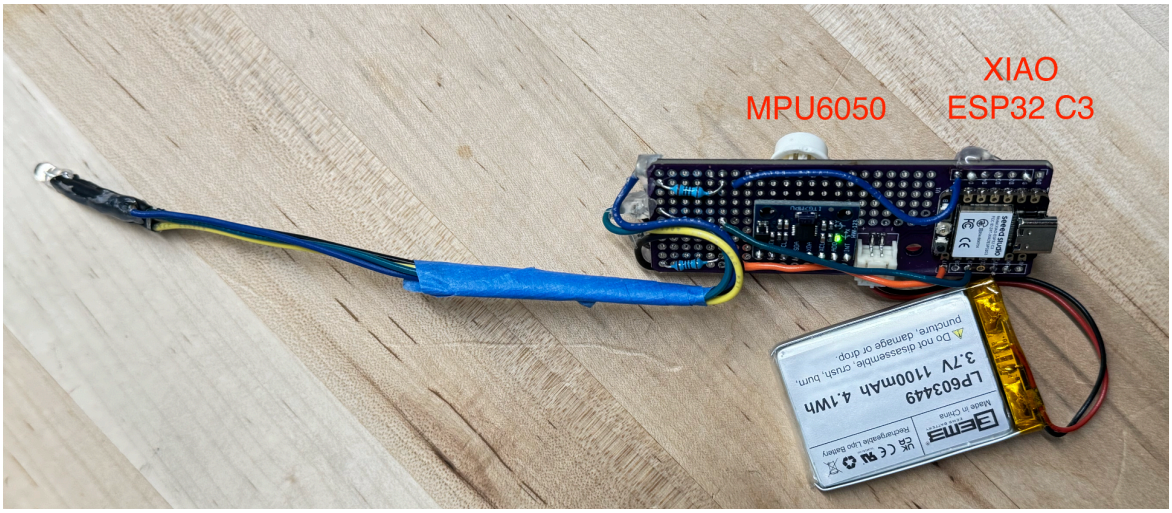
Github Link: https://github.com/Jaazmyn/TECHIN515_Magic_Wand.git

Hardware setup and connections

Hardware Components

- XIAO_ESP32_C3
- MPU6050 Accelerometer
- Push Button
- RGB LED (with separate Red, Green, Blue pins)
- Jumper Wires





Pin Connections

MPU6050 Sensor

- VCC → 3.3V
- GND → GND
- SCL → ESP32 D5
- SDA → ESP32 D4

Push Button

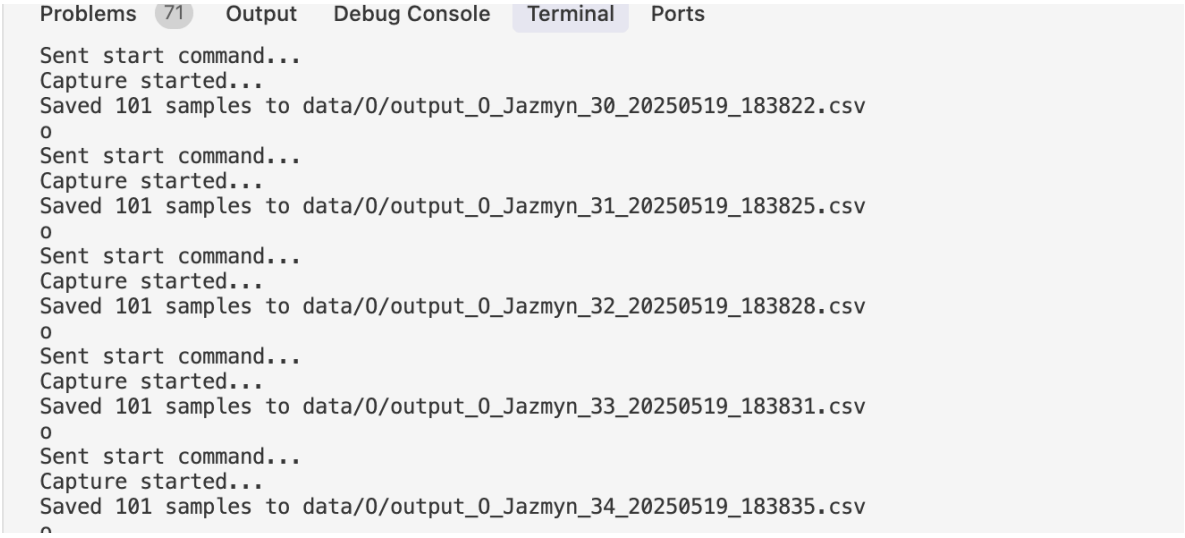
- One side → ESP32 D3
- Other side → GND

RGB LED

Red Pin → ESP32 D7
Green Pin → ESP32 D8
Blue Pin → ESP32 D9 (GPIO 2)
Common Cathode → GND
Current-limiting resistors (100Ω) in series with each color pin.

Data collection process and results

The ESP32 streams sensor data to the computer via USB serial. When I manually input "o" in the Python script, it triggers the ESP32 to collect gesture data, which the script then saves as a CSV file. A total of 326 gesture samples were collected, consisting of 121 samples for the Z gesture, 82 for the O gesture, and 123 for the V gesture.



```
       0
(base) jazmynzzz@JazmyndeMacBook-Pro gesture_capture % cd data
(base) jazmynzzz@JazmyndeMacBook-Pro data % ls Z | wc -l
     132
(base) jazmynzzz@JazmyndeMacBook-Pro data % ls O | wc -l
     160
(base) jazmynzzz@JazmyndeMacBook-Pro data % ls V | wc -l
     128
(base) jazmynzzz@JazmyndeMacBook-Pro data %
```

Edge Impulse model architecture and optimization

The neural network was developed and trained using Edge Impulse's built-in learning block with the following configuration:

Input Features: 21

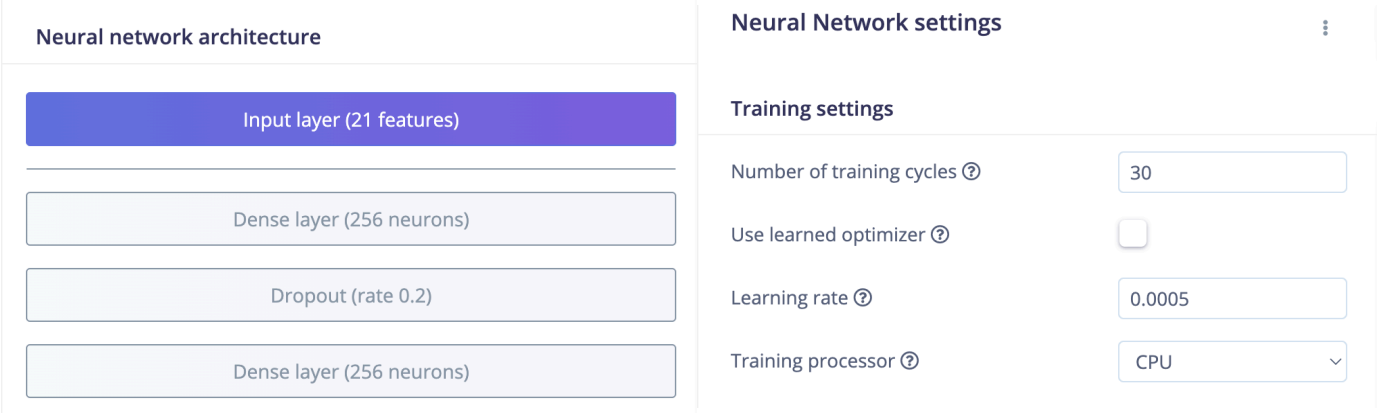
Architecture:

- Input layer with 21 features
- Dense layer with 256 neurons
- Dropout layer with a rate of 0.2
- Dense layer with 256 neurons

Training Settings:

- Number of training cycles: 30
- Learning rate: 0.0005
- Processor: CPU

Model Type: Quantized (int8), optimized for deployment on edge device



Model Optimization

Quantized (int8)

Selected ✓

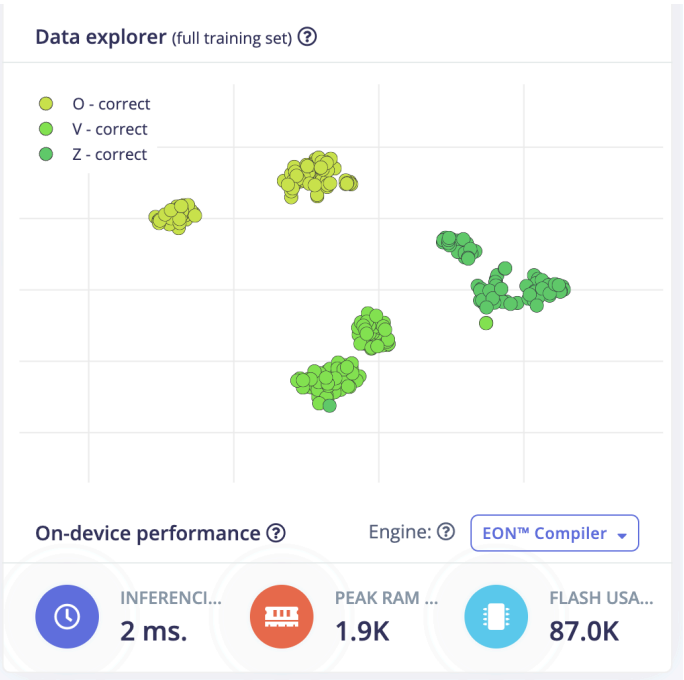
	FLATTEN	CLASSIFIER	TOTAL
LATENCY	-	2 ms.	**2 ms.**
RAM	1.3K	1.9K	**1.9K**
FLASH	-	87.0K	-
ACCURACY			**100.00%**

Unoptimized (float32)

Select

	FLATTEN	CLASSIFIER	TOTAL
LATENCY	-	25 ms.	**25 ms.**
RAM	1.3K	3.3K	**3.3K**
FLASH	-	293.6K	-
ACCURACY			**100.00%**

Performance analysis and metrics



The gesture recognition model was evaluated using a validation set containing samples for three gesture classes: "O", "V", and "Z". The results demonstrate outstanding performance across all key metrics:

Accuracy: The model achieved 100% accuracy, correctly classifying every sample in the validation set. This is further supported by the confusion matrix, which shows no misclassifications for any gesture class.

Loss: The final loss value is 0.01, indicating that the model predictions are highly confident and closely match the true labels.

Precision, Recall, and F1 Score: All weighted averages for precision, recall, and F1 score are 1.00, reflecting perfect performance in distinguishing between the gesture classes.

Area Under ROC Curve (AUC): The AUC is 1.00, confirming that the model can perfectly separate the classes without overlap.

On-device Performance: The model is highly efficient, with an inference time of just 2 ms, peak RAM usage of 1.9 KB, and a flash memory footprint of 87 KB. This makes it well-suited for deployment on resource-constrained devices like the ESP32.

The model demonstrates perfect classification performance on the validation set and is optimized for real-time, embedded applications. These results suggest that the system is robust and reliable for gesture recognition tasks using the current dataset.

```
Initializing MPU6050...
MPU6050 initialized successfully
Send 'o' to start gesture capture
Starting gesture capture...
Capture complete
INFO: Impulse maintains state. Call run_classifier_init() to reset state (e.g. if data stream is interrupted.)
Prediction: O (99.61%)
Starting gesture capture...
Capture complete
Prediction: Z (81.25%)
Starting gesture capture...
Capture complete
Prediction: V (73.44%)
Starting gesture capture...
Capture complete
Prediction: Z (79.69%)
Starting gesture capture...
Capture complete
Prediction: V (71.48%)
Starting gesture capture...
Capture complete
Prediction: V (73.44%)
Starting gesture capture...
Capture complete
Prediction: O (93.75%)
```

Discussions and Justification

1. **Why should you use data collected by multiple people, not just your own?**
   Using data from multiple people increases the diversity of the dataset, which helps the model generalize better to new users. If the model is only trained on one person's gestures, it may overfit to that person's style and perform poorly when used by others. By including data from different people, the model becomes more robust and reliable in real-world scenarios, where users' hand sizes, movement speeds, and gesture styles can vary significantly.

2. **Discussion: Effect of window size (Impulse Design stage)**
   Number of samples generated:
   A smaller window size with a small stride generates more samples, increasing the dataset size. A larger window size produces fewer samples.

   Number of neurons in the input layer: The input layer size is proportional to the window size (window size × number of features per sample). Larger windows mean more input neurons, which can increase model complexity and memory usage.

   Effectiveness for slow-changing patterns: Larger windows are better at capturing slow gestures, as the entire gesture can fit within one window. Small windows may split slow gestures across multiple samples, making it harder for the model to recognize the pattern.

3. **Why did you choose your DSP/feature extraction method?**
   I chose the Flatten block because my gestures are primarily time-based and do not have strong frequency components. Flattening the data preserves the temporal sequence, which is important for distinguishing between different gesture shapes. If my gestures had more periodic or oscillatory characteristics, I might have chosen Spectral Analysis instead.

4. **Why did you choose your learning block (model type)?**
   I selected a neural network (Keras) classifier because it can learn complex, non-linear relationships in the data, which is important for distinguishing subtle differences between gestures. Simpler models like decision trees may not capture these nuances as effectively.

5. **Discussion: Performance metrics**
   I evaluated my model using accuracy, precision, recall, F1 score, and the confusion matrix. Accuracy gives an overall sense of correctness, while precision

and recall help understand how well the model distinguishes between specific gestures. The confusion matrix shows which gestures are most often confused with each other. I also monitored the confidence (probability) output by the model for each prediction, which helps assess how certain the model is about its decisions.

6. **How does the model perform on the device vs. in the cloud?**
   The model generally performs slightly worse on the device compared to the cloud due to hardware limitations, quantization, and real-time noise. However, the performance is still acceptable for practical use. Testing on the device is essential to ensure the model is robust to real-world conditions, such as sensor noise and user variability.
7. **Strategies to further improve model performance**
   Collect more diverse data: Increasing the amount and diversity of training data (different users, environments, gesture speeds) can help the model generalize better.
   Data augmentation: Applying techniques such as adding noise, scaling, or time-warping to the training data can make the model more robust.
   Optimize model architecture: Experimenting with different numbers of layers, neurons, or activation functions may improve accuracy without increasing resource usage too much.

Demo video link

https://drive.google.com/drive/folders/1CzzxtOf-7CFoAXy1HPyOzfV9sO-T55OT?usp=sharing

Challenges faced and solutions

- Soldering on the board
  During assembly, soldering jumper wires onto the board posed a challenge due to the limited space and the potential for weak or unstable connections. To address this, we carefully selected appropriately sized and insulated jumper wires to ensure secure and reliable solder joints.

- ESP32_C3 Not Connecting to the Battery
  At one point, the ESP32-C3 module failed to power on when connected to the battery. To troubleshoot this, we used a digital multimeter (DMM) to verify voltage levels and continuity, which helped identify and resolve issues with the power supply connection.

- LED Not Controlled by ESP32_C3
  Initially, the RGB LED did not respond to commands from the ESP32-C3. This was resolved by carefully reviewing the code and ensuring that the correct GPIO pins were defined and used consistently in the program.

Enclosure and Battery

Final choice of battery is 3.7V 11000mAh.