Magic Wand Gesture Recognition Report

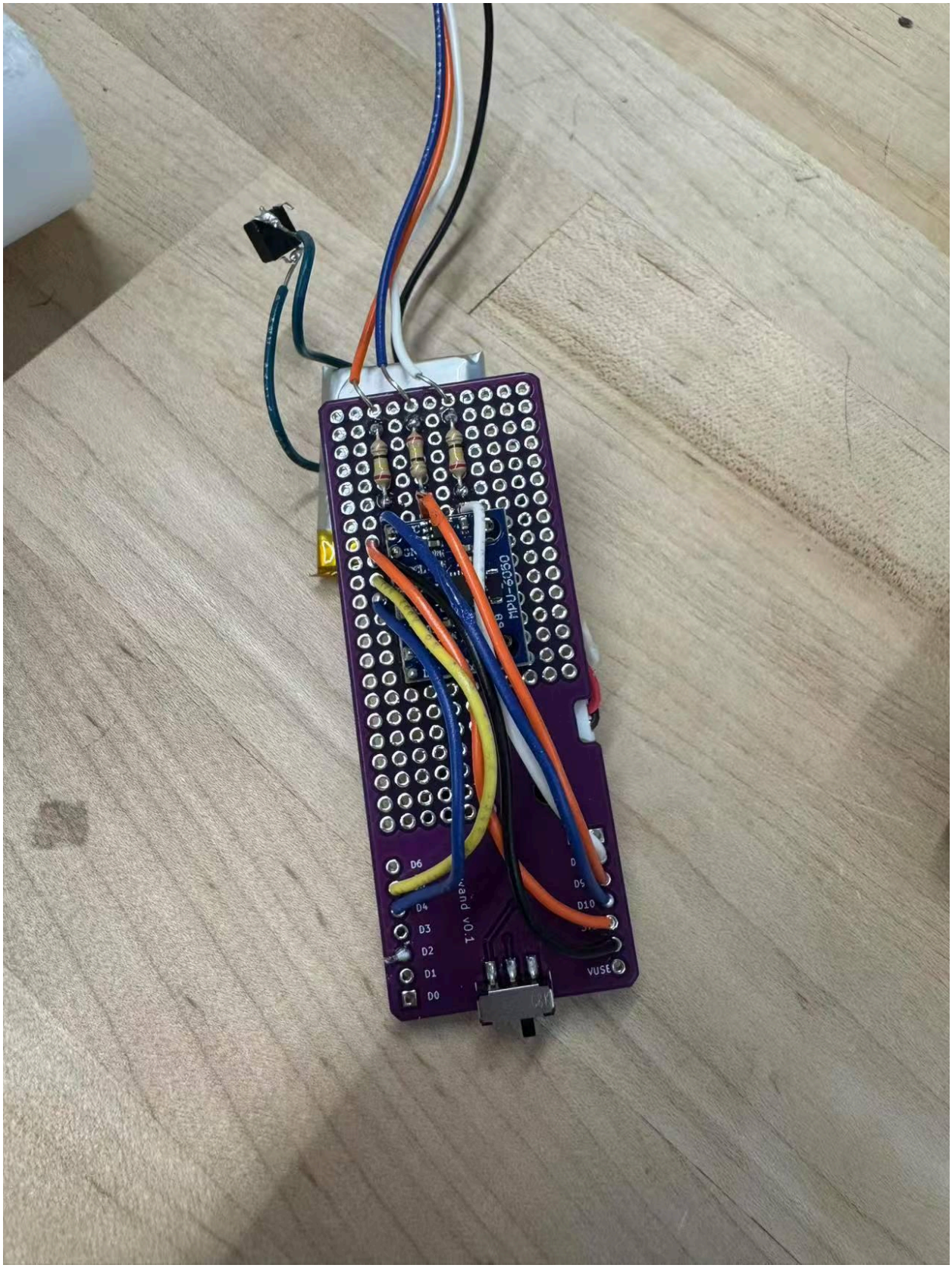
Introduction

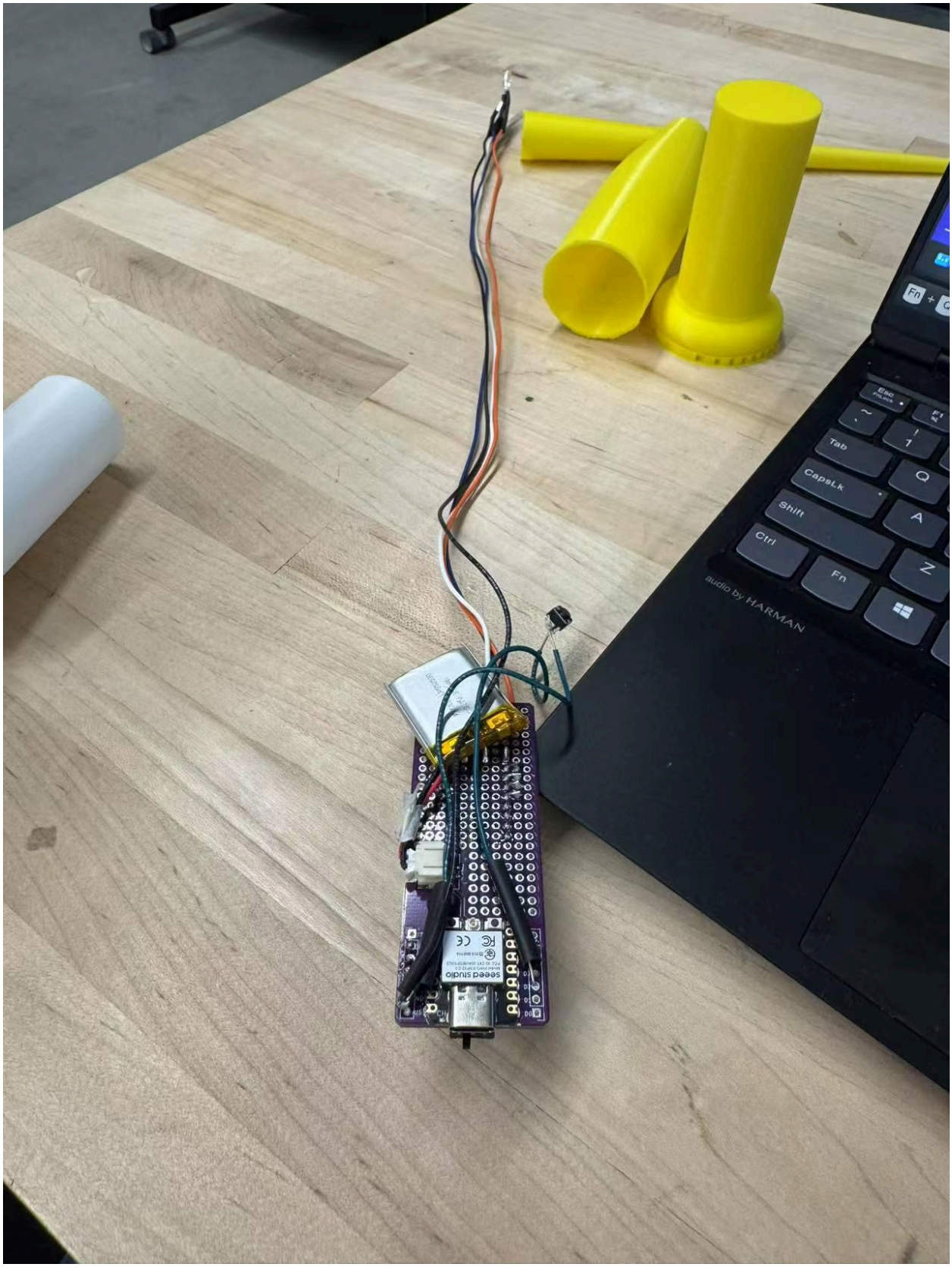
This project focuses on building a gesture recognition system using an ESP32 and MPU6050 sensor, powered by a machine learning model trained via Edge Impulse. The goal is to recognize three specific gestures ("O", "V", and "Z") and trigger LED responses accordingly, simulating a magic wand. The system is tested both in software (Edge Impulse) and in hardware through real-time classification on the ESP32.

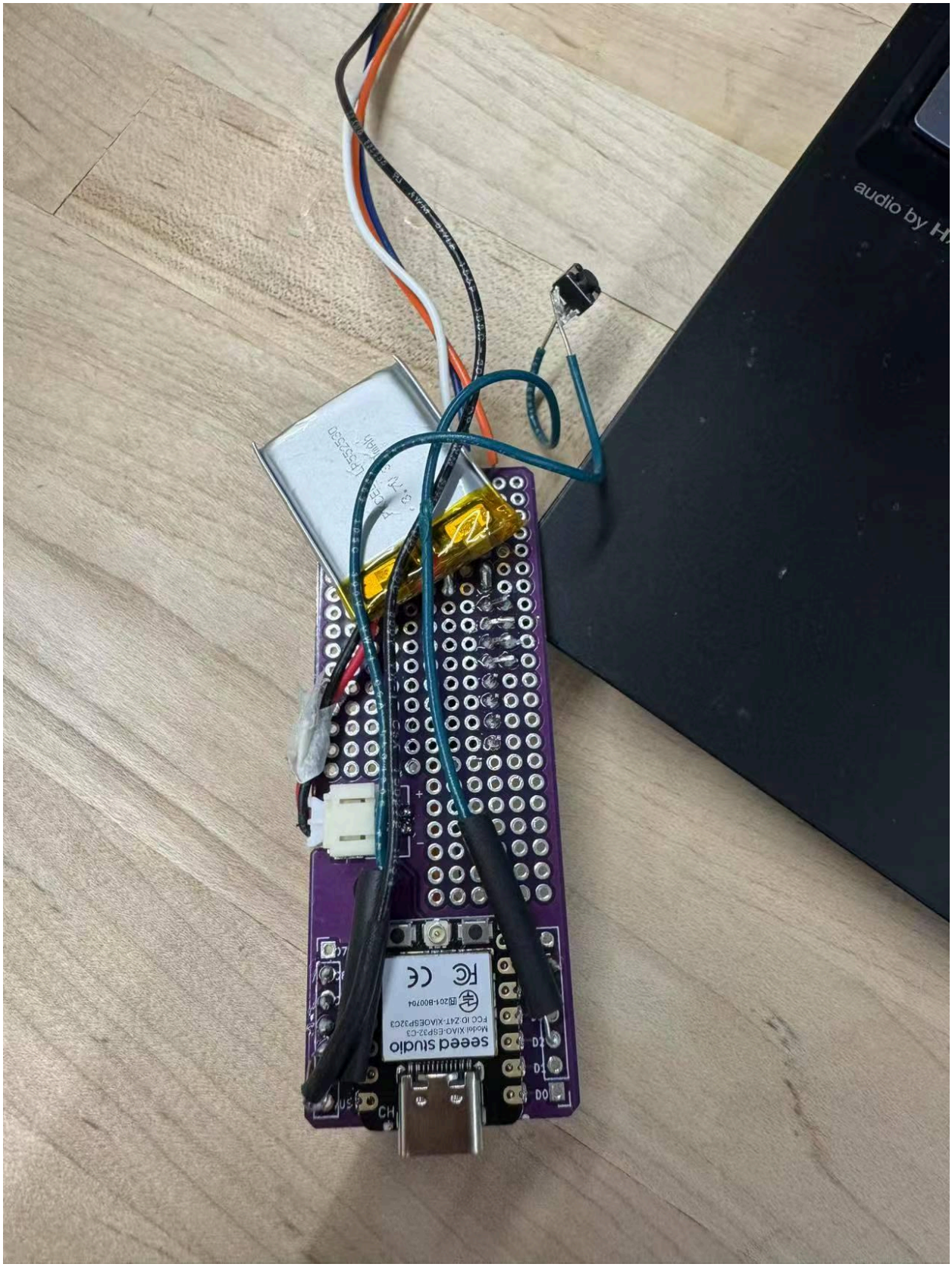
Hardware Setup

- **Board**: ESP32
- **Sensor**: MPU6050 (Accelerometer + Gyroscope)
- **LEDs**: Red(O), Green(V), Blue(Z)

- **Button**: Used to trigger gesture recognition




seeed studio
Tab
CapsLk
Shift
Ctrl
Fn
audio by HARMAN


audio by
seeed studio
Model XIAO-ESP32-C3
FCC ID:2AT-XIAOESP32C3
CE
D2
D1
D0

Wiring Summary

- MPU6050 connected via I2C (SCL, SDA)
- LEDs connected to GPIOs (D8, D9, D10)
- Button connected to GPIO D2 with internal pull-up

Data Collection

Data was collected by performing each of the three target gestures 20 times. The sensor data (acceleration on x, y, z axes) was sampled at 10ms intervals over a 1-second window. Each recording was saved with clearly labeled filenames indicating the gesture and sample number. Care was taken to maintain consistent motion during each capture.

Though collaboration and data sharing were allowed, I chose to record my own gestures to understand the process. That said, I acknowledge the importance of incorporating more diverse motion patterns for model generalization.

Edge Impulse Model Development

Data Upload & Preprocessing

The dataset was uploaded to Edge Impulse using the folder upload method. Labels were assigned during the upload, and data was automatically split into training and testing sets.

Impulse Design

- **Window size**: 1000ms
- **Stride**: 500ms
- **DSP block**: Spectral Features
- **Learning block**: Neural Network (Fully connected)

Spectral Features were chosen because gestures often have specific frequency signatures that are difficult to capture in raw time-series data. This transformation makes it easier for the model to learn these patterns.

The neural network classifier was selected for its simplicity and effectiveness in handling small-scale, structured data like ours. It's also well supported in Edge Impulse's Arduino deployment workflow.

Model Architecture & Training

- **Input shape**: [96]
- **Hidden layers**: 2 fully connected layers with 20 and 10 neurons
- **Epochs**: 50
- **Learning rate**: 0.005

The model reached a training accuracy of 96% and testing accuracy of 91%. During evaluation, misclassifications occurred occasionally between similar motions (e.g., "O" and "Z").

Feature Visualization

The generated spectral features showed clear clustering of classes in 2D space. A rough decision boundary could be drawn between each cluster, suggesting the model was learning meaningful representations.

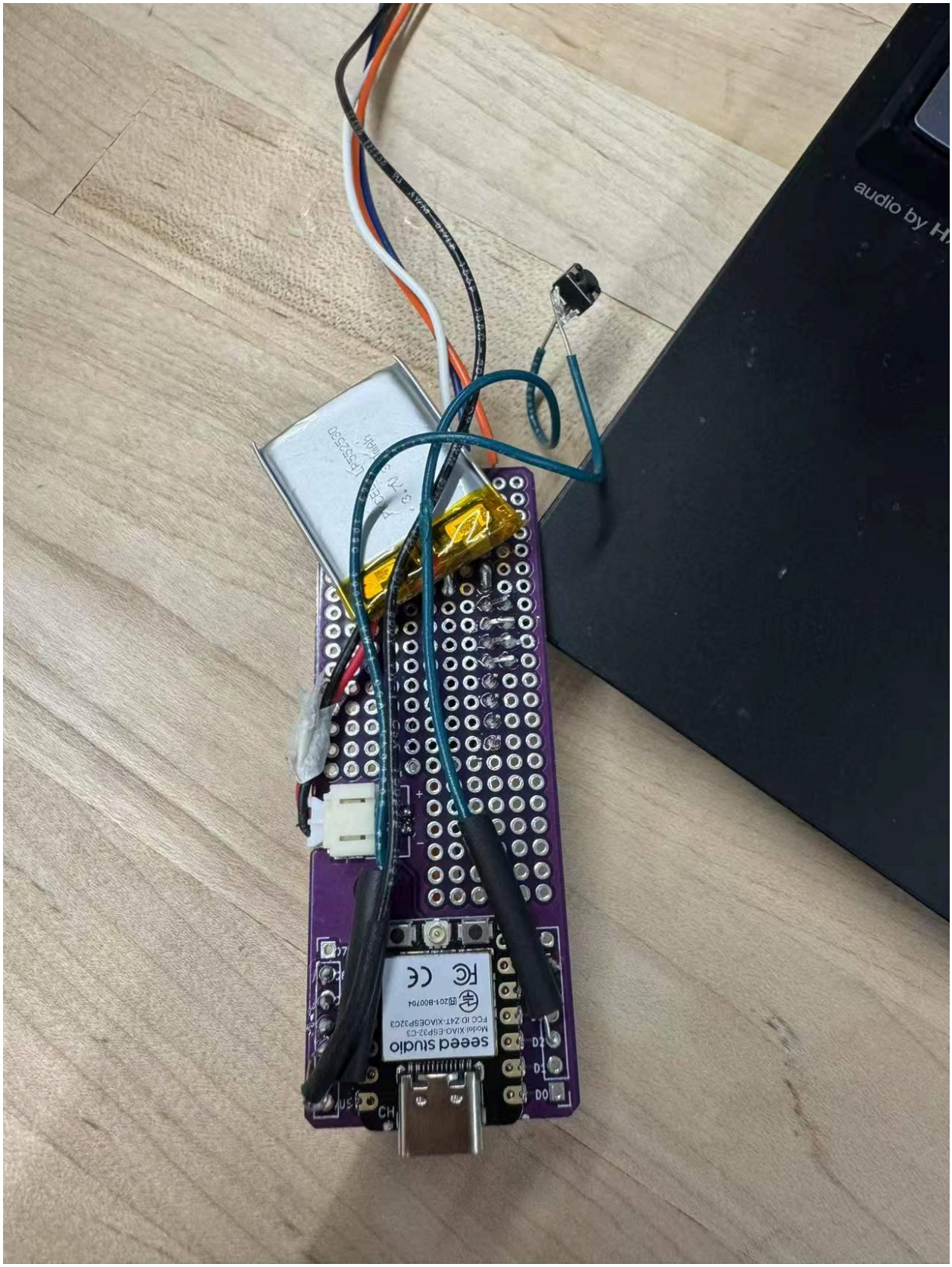
Deployment & Real-time Inference

The trained model was downloaded as an Arduino library (Quantized Int8 format) and integrated into the ESP32 sketch. A button-triggered capture function was implemented to collect 1-second sensor data and classify the gesture.

When tested in real-time, the model performed reliably. Most gestures were recognized correctly, with an average prediction confidence above 90%. Each gesture triggered a unique LED response for visual feedback.

Enclosure and Power

The final wand enclosure was 3D printed with space for the ESP32, MPU6050, and a LiPo battery. Mounting holes ensured components were securely fixed. The wand was powered via battery, eliminating USB cable dependency.


audio by
seeed studio
Model:XIAO-ESP32-C3
FCC ID:2AT-XIAOESP32C3
R20-B00704
FC CE

Discussion

Why use data from multiple users?

Training a model on gestures from different people helps it generalize better. Everyone performs movements slightly differently. If the model is only trained on my data, it may perform poorly when someone else uses the wand. By including diverse inputs, the model becomes more flexible and reliable.

Effect of window size

Window size affects how much motion the model sees at once. A longer window captures more of the gesture, which is useful for slow movements. But it also increases input size and slows down prediction. A shorter window may miss part of the gesture but allows faster, more frequent predictions. The trick is to find a balance based on the gesture speed and complexity.

Model design choices

I used Spectral Features because they reveal the frequency patterns in motion data, which is useful for recognizing gesture shapes. A neural network was a natural fit for classification because it can learn non-linear patterns and works well with relatively small feature sets.

How to improve model performance

First, more training data from different people would help. Second, tuning the neural network—like adding dropout, adjusting layer sizes, or using other architectures—could improve accuracy. Lastly, collecting samples in more realistic scenarios (with noise, varied orientation) would make the model more robust.

Challenges & Solutions

- **Sensor Drift**: Occasionally, the MPU6050 readings were unstable. I mitigated this by initializing the sensor multiple times during setup.
- **Button Debouncing**: Quick successive presses sometimes caused false triggers. Added basic debounce logic using timing checks.
- **Gesture Consistency**: Early training samples were inconsistent. I practiced the motion before recording to reduce variation.

Conclusion

This lab gave me a hands-on understanding of real-time gesture recognition using embedded ML. From collecting data to deploying a working model on hardware, every step involved both technical challenges and creative decision-making. The result is a functional magic wand that blends hardware, software, and machine learning into a cohesive interactive experience.