Part 1: Data Collection

Discussion: Why should you use training data collected by multiple students rather than using your own collected data only? Think about the effectiveness and reliability of your wand.

A: Gestures vary per person—speed, grip, angle. Relying solely on one individual's data risks overfitting. Aggregating data from multiple users boosts generalization, making the wand more reliable in real-world use.

Part 2: Edge Impulse Model Development

2.Discussion: Discuss the effect of window size. Consider

·the number of samples generated

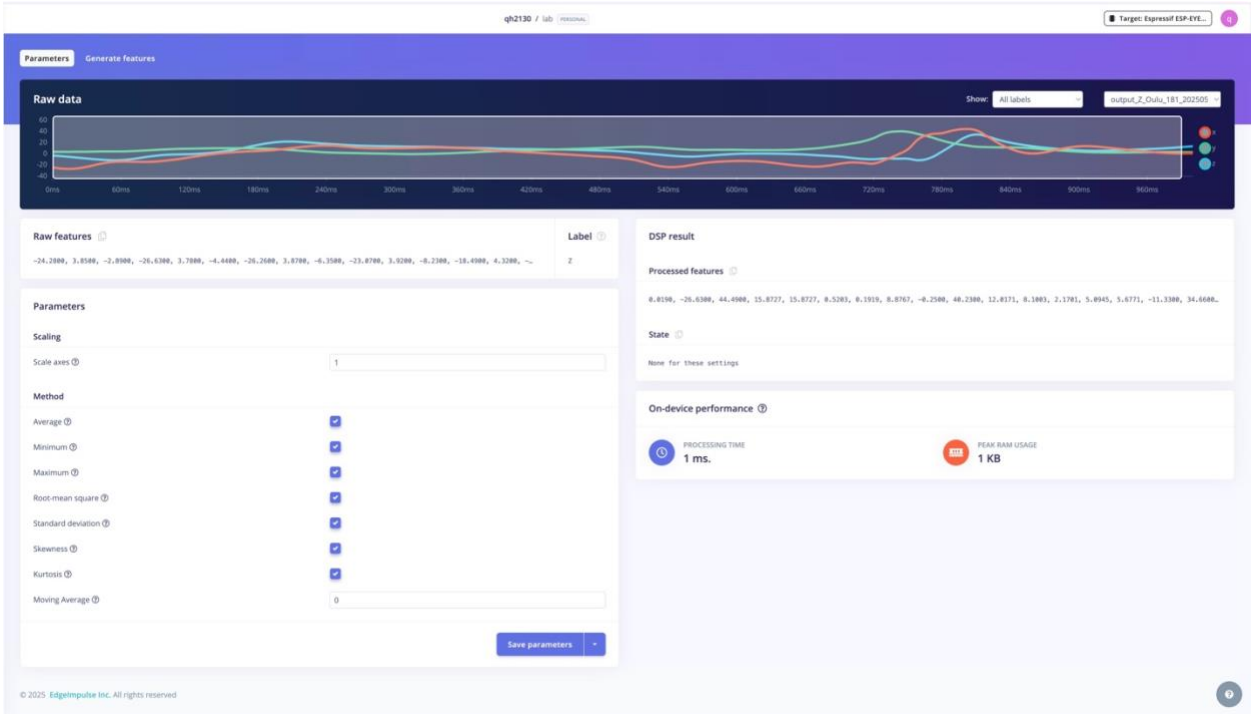
·the number of neurons in your input layer of neural network

·effectiveness when capturing slow-changing patterns

A： Window size controls how much motion is captured per gesture. A longer window (e.g. 1000ms) means more samples per gesture, which helps capture slow movements but increases model input size and memory use. For example, with 21 features extracted from a 1000ms window, the input layer has 21 neurons. If the window were longer, the model would require more neurons and complexity. Shorter windows may miss key parts of gestures, especially slower ones, leading to poor accuracy. Our 1000ms window offers a good balance between completeness, responsiveness, and model efficiency.

3.Choose your DSP block in the sidebar.

·Tune the hyperparameters and visualize the generated features until you are satisfied with the features.



·Take a screenshot of your generated features, and sketch a rough decision boundary between classes. Explain why do you believe the generated features are good enough.

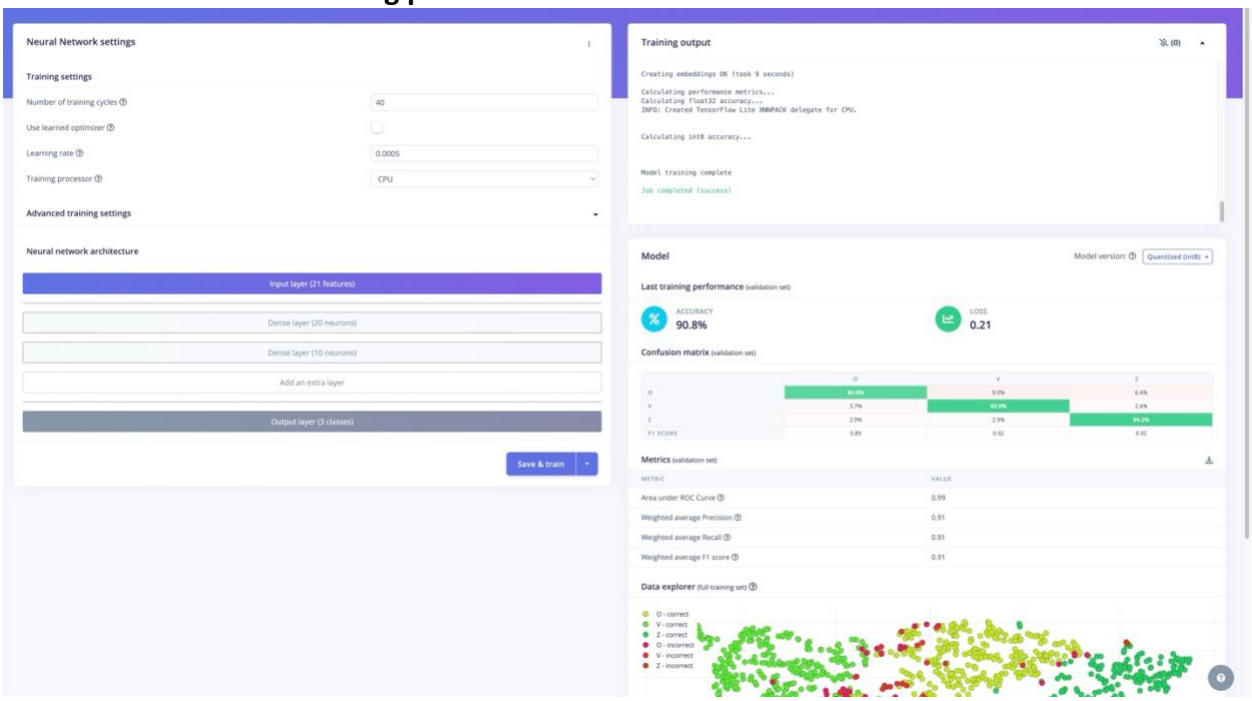
I chose the Flatten block to extract a compact and efficient feature set from the raw accelerometer data. It computes seven statistical metrics—mean, min, max, standard deviation, RMS, skewness, and kurtosis—for each axis, resulting in a total of 21 features.

This default feature set worked well. The Feature Explorer visualized the three gesture classes (O, V, Z) as well-separated clusters, suggesting effective class distinction. A rough decision boundary between them showed that the gestures were mostly linearly separable.

Overall, these features were lightweight yet effective, which made them a good fit for training a simple dense neural network optimized for real-time deployment.

4.Choose your ML block in the side bar.

Tune the number of training epochs, learning rate, and neural network architecture until you are satisfied with the learning performance.



Pic4:Classifier block

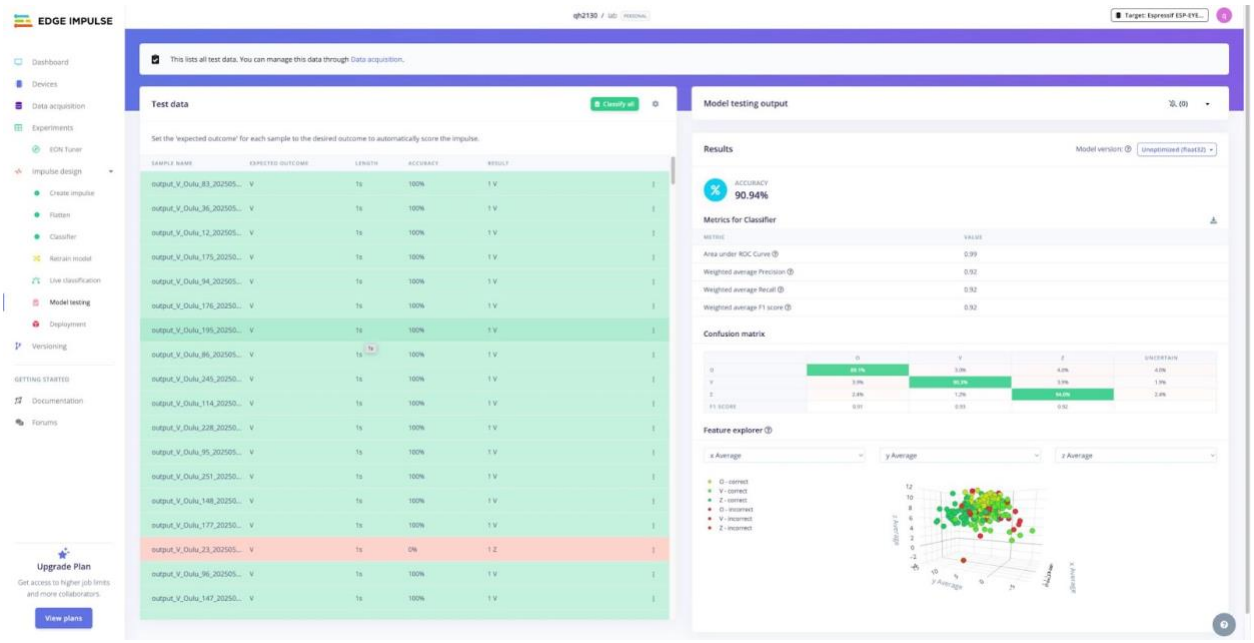
Report the learning performance, your choices of hyper-parameters, and architecture.

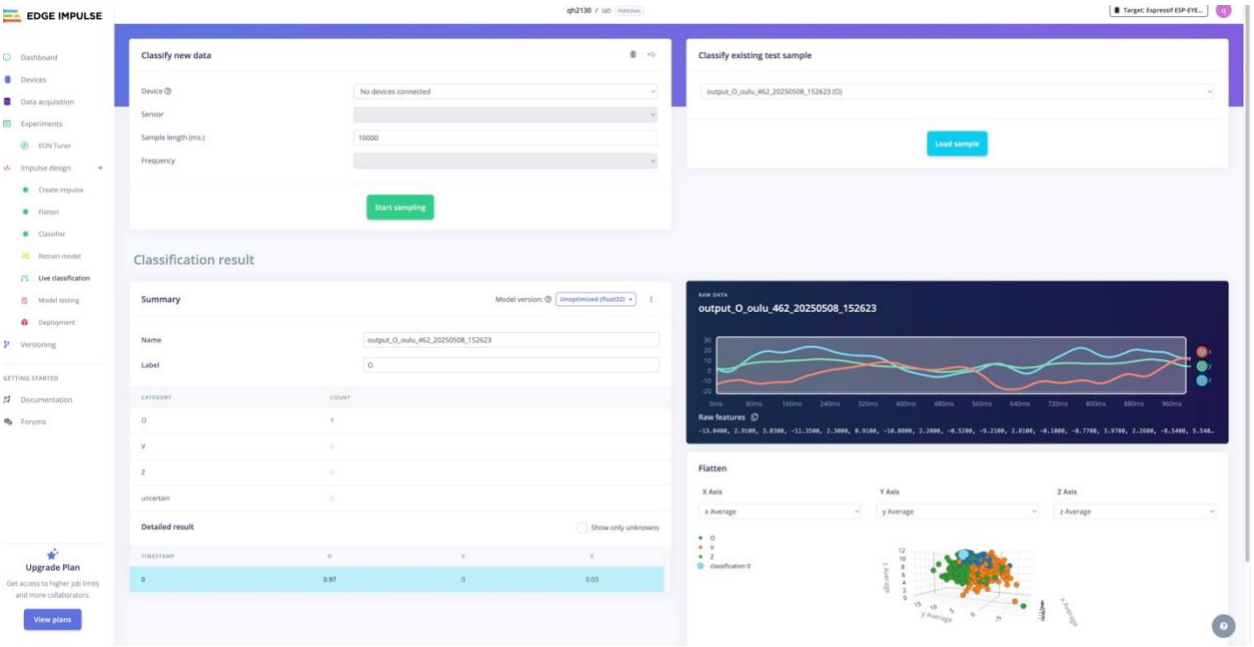
I used the Classifier block to train the model, taking the 21 features from the Flatten block as input. The network architecture included two dense layers with 20 and 10 neurons, followed by a Softmax output layer for the three gesture classes (O、V、Z).

I trained the model for 30 epochs with a learning rate of 0.0005. The final training accuracy reached 90.8%, with a weighted F1 score of 0.91 and ROC AUC of 0.99, indicating a strong classification performance.

The model is compact enough to run on the ESP32, requiring just 1 ms for inference, 1.4 KB of RAM, and 15.8 KB of flash memory. Performance was consistent across different gestures, and there were no signs of overfitting.

5.Use "Live classification" and "Model testing" in sidebar to test your model performance. Please clearly document all metrics being used, e.g., accuracy, TP, FP, F1, etc.





I tested the model using both the “Model testing” and “Live classification” tools in Edge Impulse. Model testing yielded 90.94% accuracy, with weighted precision, recall, and F1 score each around 0.91. The ROC AUC was 0.99, indicating strong overall performance.

According to the confusion matrix, class “O” achieved 88.5% accuracy with some misclassification to classes Z and V. Class “Z” reached 91.3% accuracy, and “V” showed

the strongest performance at 92.7%. Only a small number of samples across all classes were labeled as uncertain.

During live classification, I uploaded a new sample, which was accurately identified as a “V” gesture with 96% confidence, confirming the model's reliability in real-time use.
Overall, the results demonstrated clear class separation, balanced performance, and consistent predictions.

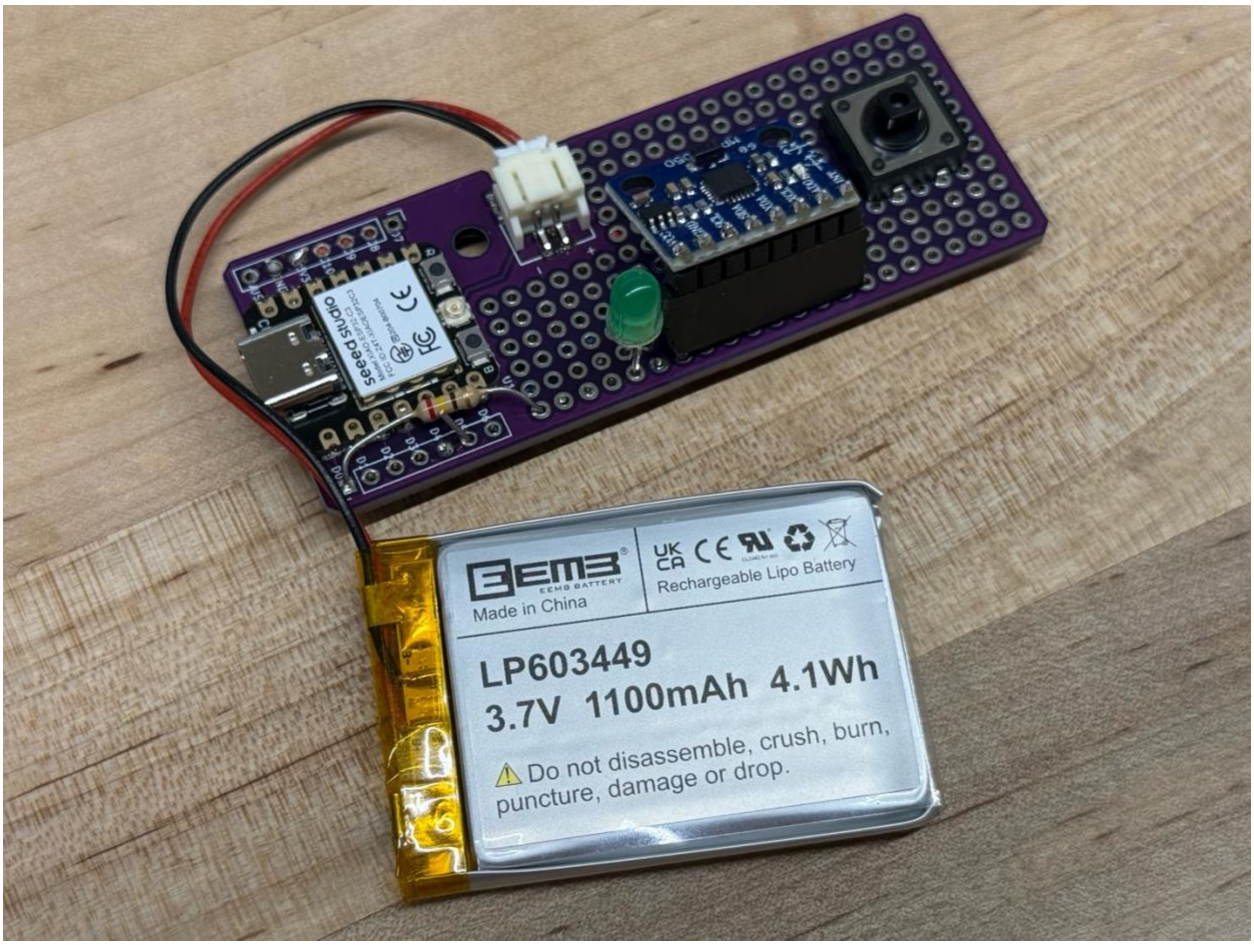
7.Discussion: Give at least two potential strategies to further enhance your model performance.

1. To further improve the model’s performance, I could add gyroscope data alongside the accelerometer readings. This would allow the model to better capture rotational motion, which could be especially useful for distinguishing between gestures like “O” and “Z”.

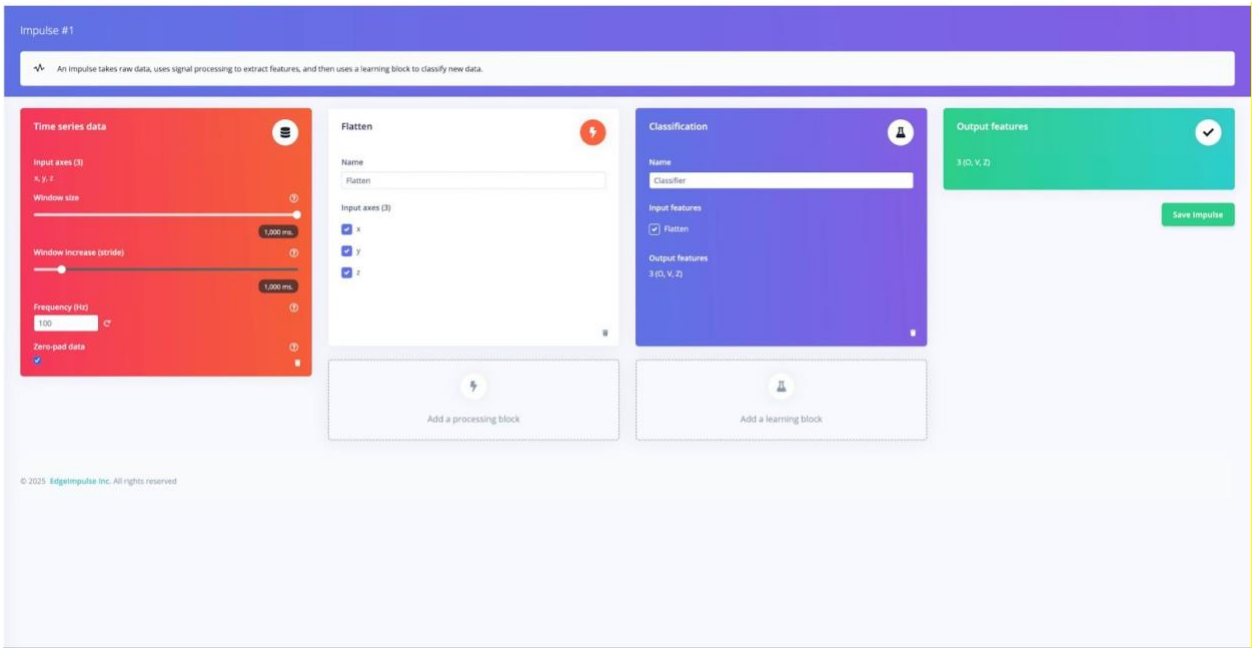
2. Another strategy is to switch to a temporal model, such as an LSTM or TCN, that takes into account the sequence of motion over time rather than relying solely on statistical summaries. This might help the model better recognize gestures that have similar shapes but differ in movement timing.

3. Finally, I could improve generalization by increasing dataset diversity—collecting more samples from different users, at varying speeds, and with more natural variations. This would help reduce overfitting and improve robustness in real-world use.

Part 3: ESP32 Implementation
Modify the provided wand.ino code: Implement a button-triggered inference rather than typing 'o' in seriel monitor: ESP32 predicts the gesture once the button is pressed.
Code in folder
hardware setup



Edge Impulse model architecture and optimization



Challenges Faced and Solutions

One challenge was the confusion between similar gestures like “Z” and “V”, especially when performed at different speeds. To solve this, more data was collected from different users, and slow, consistent gestures were encouraged during recording.
Another issue was keeping the sensor orientation consistent between data collection and real use. This was solved by fixing the PCB inside the 3D-printed enclosure using a custom slot that holds the board in place and prevents rotation.
Memory constraints on the ESP32 were also a concern. Using the Flatten DSP block and quantized int8 model helped keep RAM usage low while maintaining good accuracy.
Lastly, to avoid misclassification due to background motion, a physical button was added to trigger gesture capture intentionally. This improved both user control and model stability.

Part 4: On-device Deployment and Real-time Inference

The trained model was exported using **int8 quantization** to reduce memory and computational footprint. On-device performance on ESP32C3:

- Inference time: **1 ms**
- Peak RAM usage: **1.4KB**
- Flash usage: **15.8KB**

Gesture recognition is triggered via a **physical button**. When pressed, the ESP32 collects 1 second of acceleration data, extracts features, runs inference, and outputs the gesture result.

Gesture-to-LED mapping:

- Gesture "O": 2 quick flashes
- Gesture "Z": 3 medium-speed flashes
- Gesture "V": 1 slow flash

The result is also printed to the Serial Monitor in the format: Gesture Detected: V [V] (92.13%)

Part5: Evaluation and Metrics

Confusion Matrix (Validation Set):

	Predicted O	**Predicted Z**	**Predicted V**
Actual O	88.5%	5.1%	6.4%

Actual Z	2.4%	91.3%	6.3%
Actual V	4.9%	2.9%	92.7%

Metrics:

- Weighted average precision: 0.91
- Weighted average recall: 0.91
- Weighted average F1 score: 0.91
- AUC (ROC): 0.99

Part 6: Challenges and Improvements

One of the primary challenges encountered was distinguishing between visually and dynamically similar gestures, particularly linear ones such as “Z” and “V.” These gestures often shared overlapping directional patterns and comparable acceleration profiles, leading to classification confusion.

To mitigate this issue, several strategies were implemented:

- The sensor’s spatial alignment was stabilized by embedding the circuit board within a tightly-fitted 3D-printed enclosure, ensuring consistent orientation across sessions.
- A physical button was integrated to initiate gesture capture only during intentional movement, reducing noise from idle motion.
- Feature extraction was enhanced by incorporating higher-order statistical metrics—such as skewness and kurtosis—to better capture subtle variations in gesture shape and dynamics.

Potential Enhancements

To further improve model robustness and recognition accuracy, the following improvements are recommended:

- Integrate gyroscope data to capture rotational motion, which could better differentiate gestures with similar translational paths.
- Explore temporal deep learning models (e.g., LSTM or TCN) that can learn time-dependent patterns across gesture sequences.
- Expand the training dataset with recordings from a diverse group of users, capturing variations in speed, hand size, and style, to improve real-world generalization.

Part 7: Hardware Design

The system is housed in a custom-designed 3D-printed wand enclosure that ensures both functionality and ergonomic comfort. A LP603449 3.7V 1100mAh Li-Po battery powers the embedded electronics, offering portable and rechargeable operation.

The internal components, including the Seeed Studio XIAO ESP32C3 board and IMU sensor, are securely mounted on a perfboard, which is firmly slotted into the wand casing. To maintain orientation fidelity between data collection and real-world usage, the PCB is aligned with the same axes used during model training. A dedicated recess in the enclosure ensures tight board placement, preventing internal shifting.

The exterior design includes:

- A **button on the top face** (mapped to GPIO10) used to trigger gesture recognition.
- A **green status LED** (GPIO2) located below the button, which provides visual feedback during interaction.
- An **opening at the bottom** of the wand exposes the USB-C port for easy charging and debugging without disassembly.

The form factor supports single-handed operation, making the wand both user-friendly and robust for real-time gesture interaction. This integration results in a self-contained, responsive, and compact interactive device optimized for embedded ML deployment.