P2: gesture-interaction

@Flora(weierx@andrew.cmu.edu)

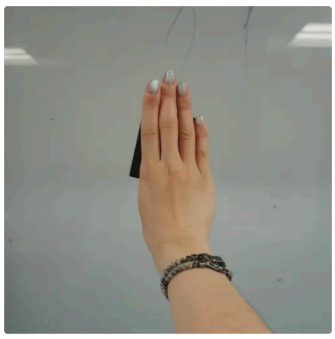
Project Overview

The goal of this project is to collect the user's gesture action signals in real time through sensors and perform gesture classification and recognition based on machine learning models in order to control the square on the screen to move to the specified target position. The project continuously optimizes the interaction method through several iterations in order to improve the user experience and interaction efficiency.

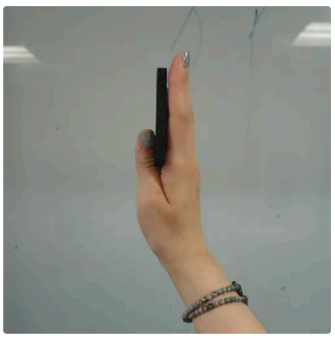
Version Iteration

All versions of the following gestures are designed to follow the following two rules:

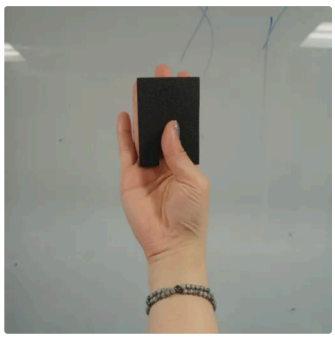
- **Hand position of the device**: always hold the device so that the X-plane of the device is parallel to the palm of the hand, with the palm facing in the positive direction of the Z axis



Front



Side



Back

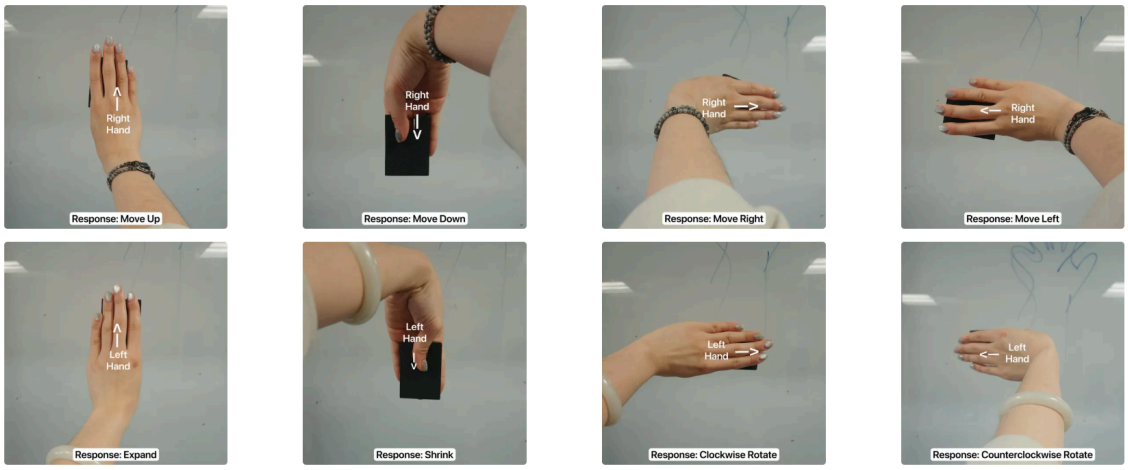
- **Static/Stop moving**: Z-axis vertically and horizontally facing down (simplest handling: just lay the device flat on the table)

Version-1: Statistic Mode Gesture(Baseline)

1. **Design Concept**

   In the first iteration, the system recognizes static hand poses (distinct hand orientations or postures). Each distinct pose corresponds to a specific directional command (up, down, left, right). Both left and right hands employ identical static gestures to control different degrees of freedom (DoFs).

2. **Gesture Schematic**



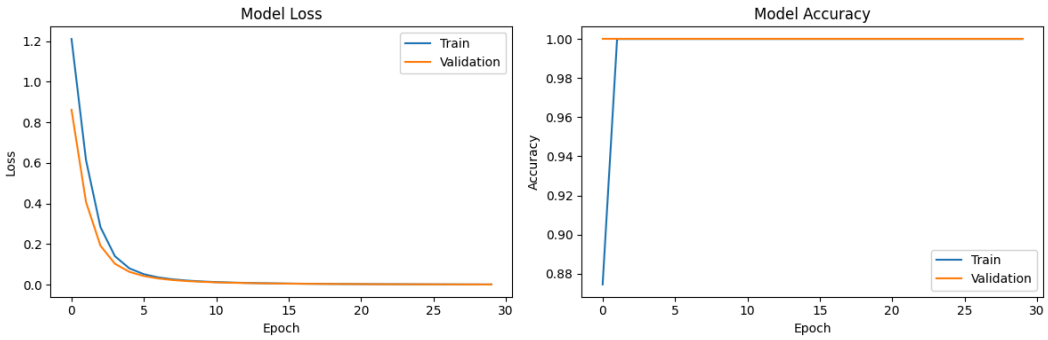
3. Model Design

Model Selection

Train Model	Gesture Type	Model Architecture	Data Scale	Gesture Detect (Label)	Test Size	Learning Rate	Epochs	Early Stopping
Keras	Static	Simple NN	2686	5	0.2	3e-4	30	patience=2

Layer(type)	Output Shape	Param
Layer_1(Dense)	(None, 64)	19,264
Layer_2(Dense)	(None, 5)	325

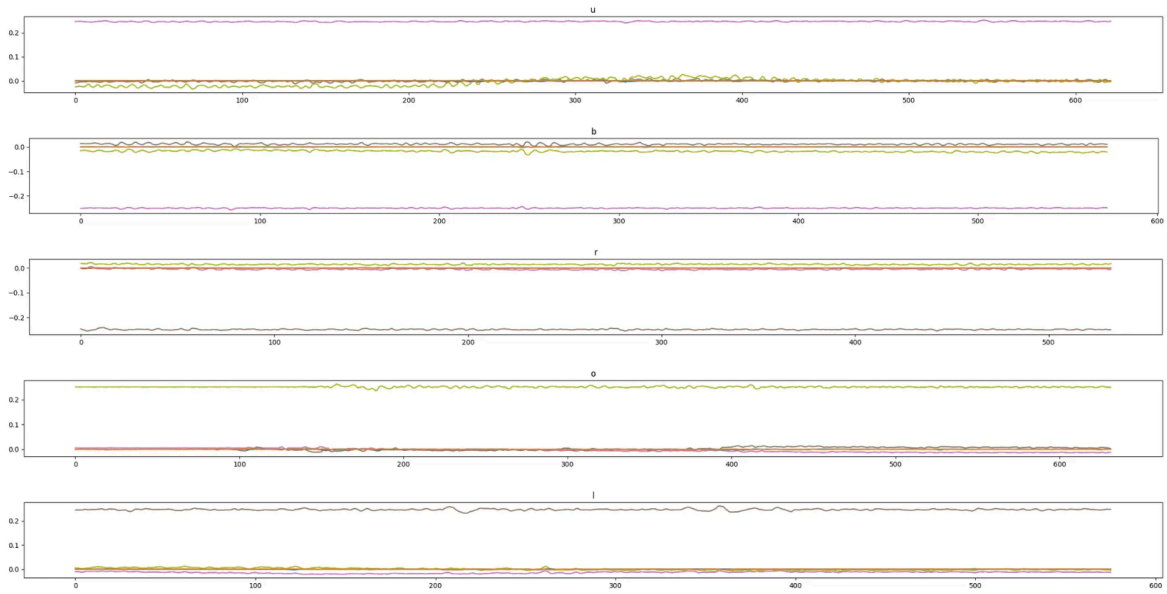
Result



Test Loss: 0.0015, Test Acc: 1.0000

4. Analysis of User Test

User IMU Data For Training (Complete test data of four users can be found in `./data/UserData/`)



Result For User Interaction

User	Time(sec/destination)	Error
1	15.220394	0/5
2	13.840933	0/5
3	17.334022	0/5
4	19.033405	0/5
Average	16.36	0/5

Strengths:

- Simplicity of gesture recognition due to static poses being relatively easy to classify.
- Reliable and predictable interaction, ideal for initial validation and robust proof-of-concept.

Weaknesses:

- Users reported a lack of engagement, describing the interaction as repetitive and tedious.
- Symmetrical behaviors for both hands led to boredom and a lack of motivational feedback, reducing long-term engagement.

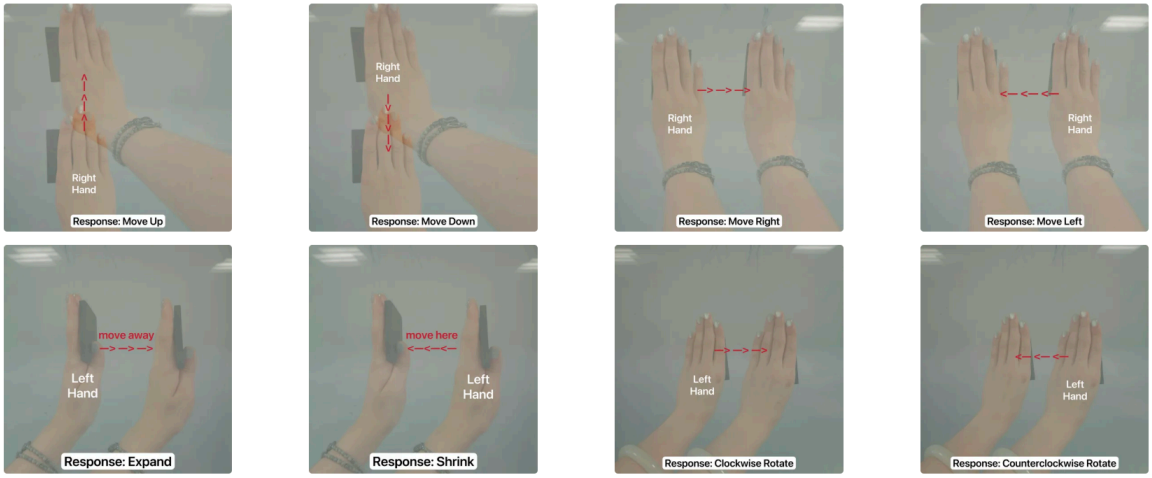
Version-2: Dynamic Mode Gesture(Acceleration-based)

1. **Design Concept**

   To enhance interactivity, the second iteration integrated dynamic gestures based on acceleration signals. This meant gestures were identified by quick movements of the hands in different directions. Importantly, distinct gestures were allocated to each hand:

   - **Right hand:** Dynamic acceleration gestures to control translation (up, down, left, right).
   - **Left hand:** Separate dynamic acceleration gestures to control rotation and scaling.

2. **Gesture Schematic**



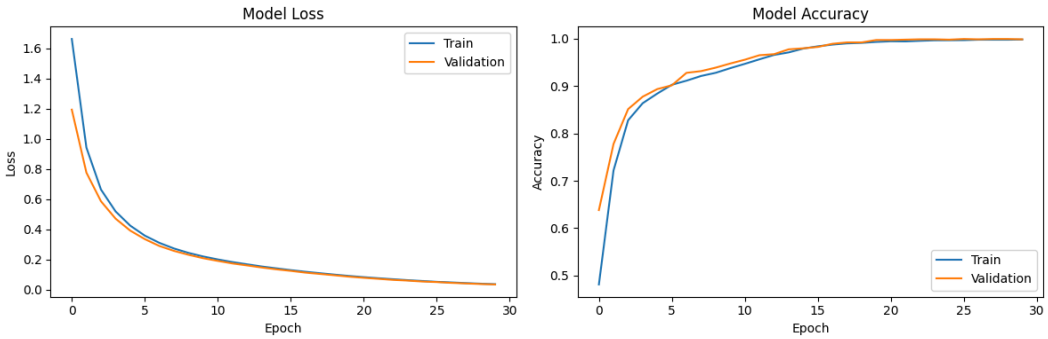
3. **Model Design**

   Model Selection

Train Model	Gesture Type	Model Architecture	Data Scale	Gesture Detect (Label)	Test Size	Learning Rate	Epochs	Early Stopping
Keras	Dynamic	Simple NN	11418	9	0.2	3e-4	30	patience=2

Layer(type)	Output Shape	Param
Layer_1(Dense)	(None, 64)	19,264
Layer_2(Dense)	(None, 9)	585

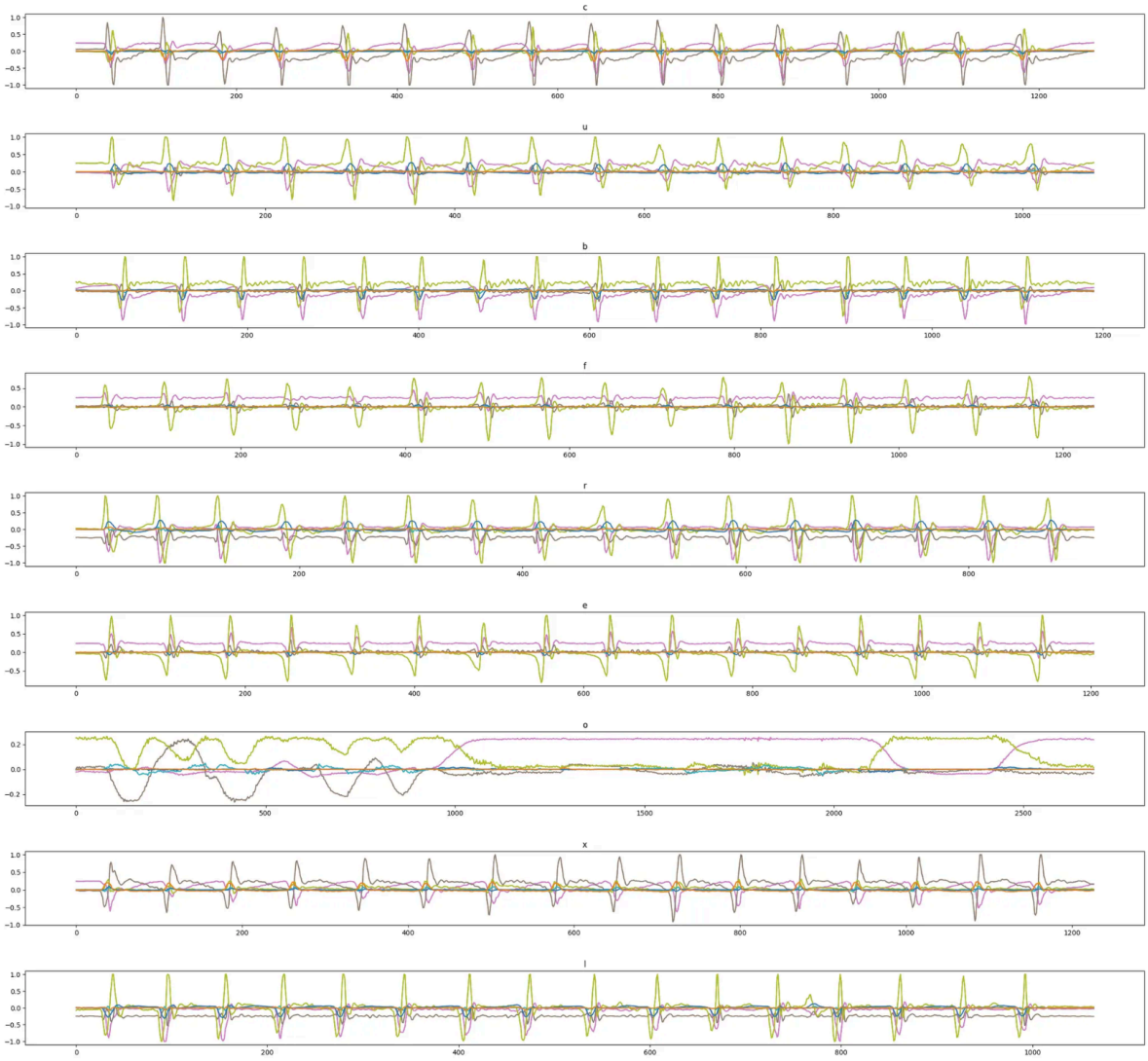
Result



Test Loss: 0.0386, Test Acc: 0.9965

4. Analysis of User Test

User IMU Data For Training (Complete test data of four users can be found in `./data/UserData/`)



Result For User Interaction

User	Time(sec/destination)	Error
1	30.874629	0/5
2	29.312200	0/5
3	38.498009	1/5
4	36.219093	1/5
Average	33.73	0.5/5

Strengths:

- Improved engagement due to dynamic interactions being more playful and physically active, increasing user interest.
- Clear separation of tasks per hand, enhancing interaction clarity.

Weaknesses:

- Increased cognitive and physical demand, since dynamic movements required precise control and coordination, which could lead to user fatigue.
- Cognitive load was high due to requiring bilateral coordination of entirely different dynamic gestures, complicating user experience and posing left/right hemisphere coordination challenges.

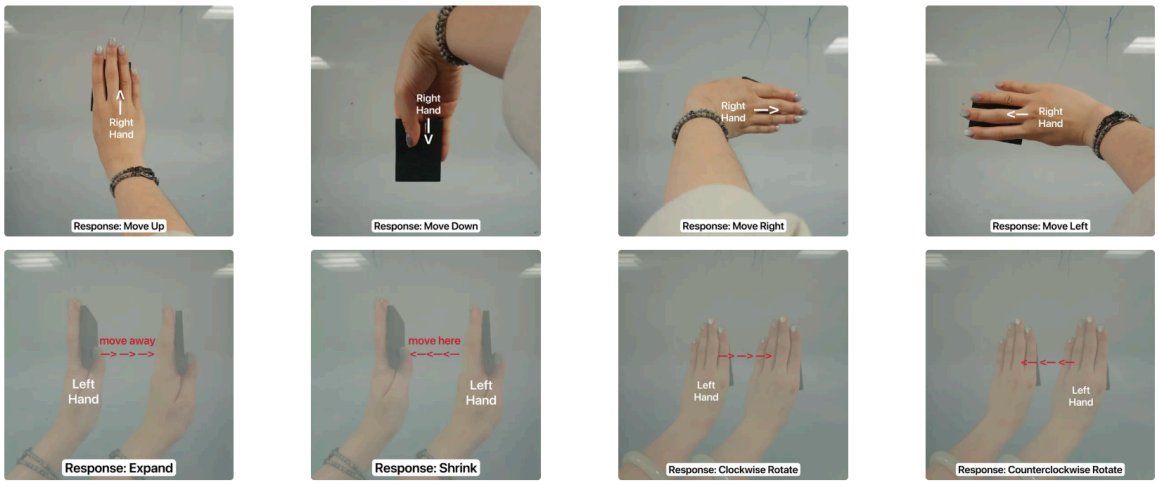
Version-3: Blending Static and Dynamic Gestures(Hybrid Design)

1. **Design Concept**

   To resolve the issues encountered in the previous iterations, the third iteration adopted a hybrid design. This involved combining static and dynamic gestures for a more balanced cognitive load:

   - **Right hand:** Reverted to static gestures (from Iteration 1) to control translation (up/down/left/right).
   - **Left hand:** Retained dynamic gestures from iteration 2 for rotation and scaling.

2. **Gesture Schematic**



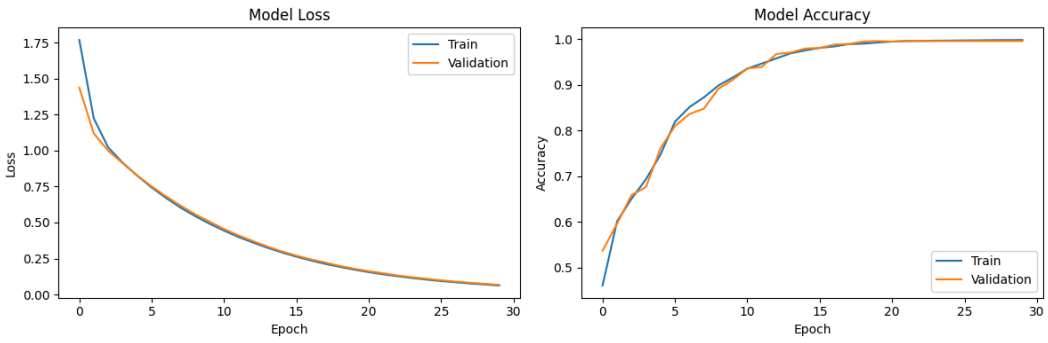
3. Model Design

Model Selection

Train Model	Gesture Type	Model Architecture	Data Scale	Gesture Detect (Label)	Test Size	Learning Rate	Epochs	Early Stopping
Keras	Hybrid	Simple NN	7659	9	0.2	3e-4	30	patience=2

Layer(type)	Output Shape	Param
Layer_1(Dense)	(None, 64)	19,264
Layer_2(Dense)	(None, 9)	585

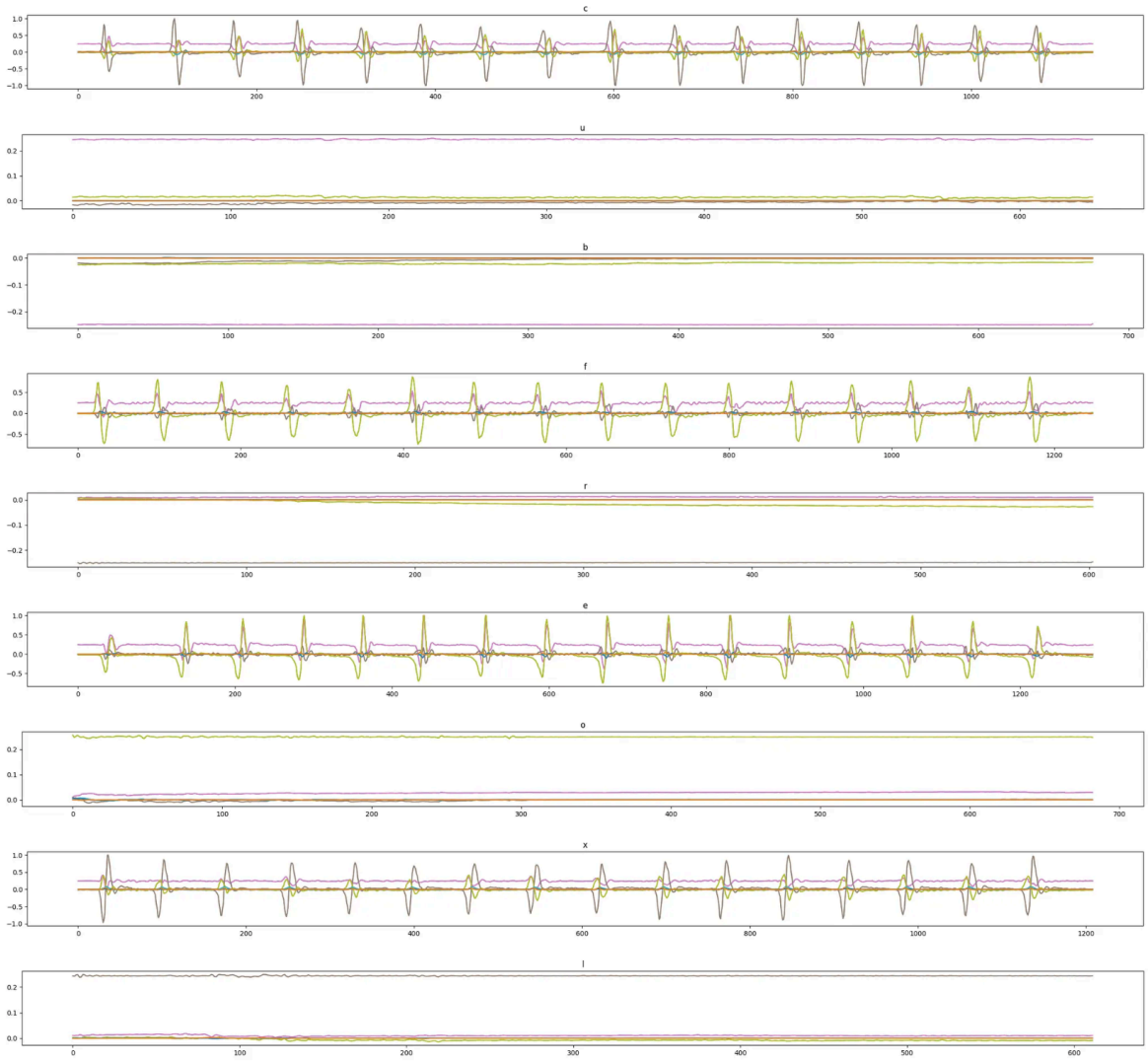
Result



Test Loss: 0.0607, Test Acc: 0.9974

4. Analysis of User Test

User IMU Data For Training (Complete test data of four users can be found in `./data/UserData/`)



Result For User Interaction

User	Time(sec/destination)	Error
1	25.983720	0/5
2	22.348476	0/5
3	28.936467	1/5
4	30.3876412	0/5
Average	26.90	0.25/5

Strengths:

- Reduced cognitive complexity and fatigue by using simple static gestures for primary movement tasks (translation).
- Preserved engaging and intuitive aspects of dynamic gestures for secondary tasks (rotation and scaling), improving user satisfaction.
- Clearly delineated tasks between hands, significantly decreasing cognitive conflict during interactions.

Version-Final: Hybrid Gesture Interaction with Enhanced Intuitiveness

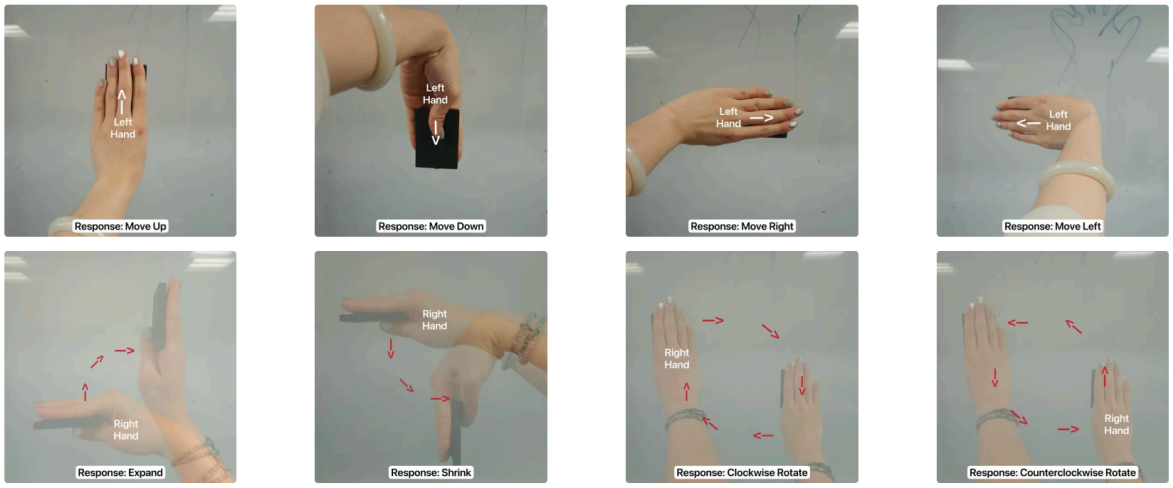
1. **Design Concept**

   The final iteration further improved upon the hybrid approach by carefully selecting gestures based on intuitive cognitive mappings and real-world metaphors. Specifically:

   - **Translation (Static Gestures):** Maintained simple directional static gestures (right hand).
   - **Rotation and Scaling (Dynamic Gestures, left hand):**
     - **Rotation:** Circular dynamic gestures (clockwise and counterclockwise rotations) were introduced, aligning perfectly with users' mental models of rotation.
     - **Scaling:** Implemented intuitive vertical gestures—moving upwards to enlarge, downward movements to shrink—thus aligning gesture semantics with perceptual intuition.

   Additionally, this iteration incorporated adaptability considerations by training both IMU devices on all nine gestures, enabling the system to flexibly adapt to the user's dominant hand. Users could freely choose their preferred dominant hand without retraining or recalibration, dramatically increasing accessibility and comfort for various users.

2. **Gesture Schematic**



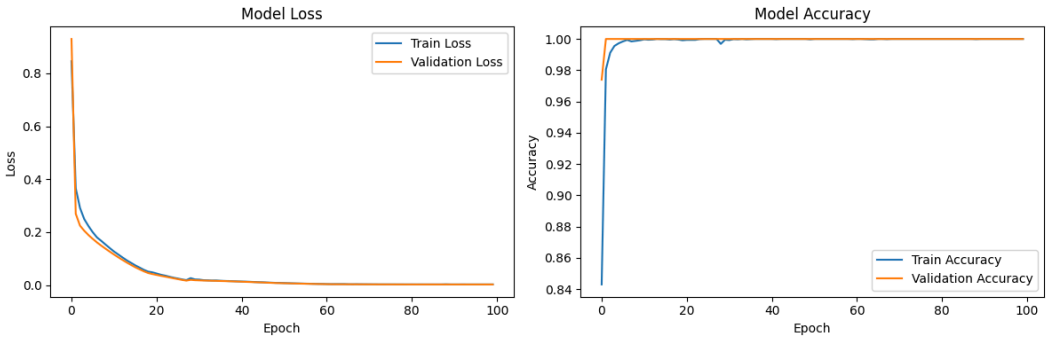
3. Model Design

Model Compare

Version	Train Model	Gesture Type	Model Architecture	Data Scale	Gesture Detect (Label)	Regularization	Learning Rate	Epochs	Early Stopping	Reduce LR on Plateau
Final	Keras	Hybrid	Enhanced Dense	12154	9	L2, BatchNorm, Dropout	3e-4 (adaptive reduction)	100	patience=5 (restore best weights)	Applied, factor=0.5, patience=2

Layer(type)	Output Shape	Param
Layer_1(Dense)	(None, 128)	38,528
Layer_2(BatchNormalization)	(None, 128)	512
(Dropout)	(None, 128)	0
(Dense)	(None, 64)	8,256
(BatchNormalization)	(None, 64)	256
(Dropout)	(None, 64)	0
(Dense)	(None, 9)	585

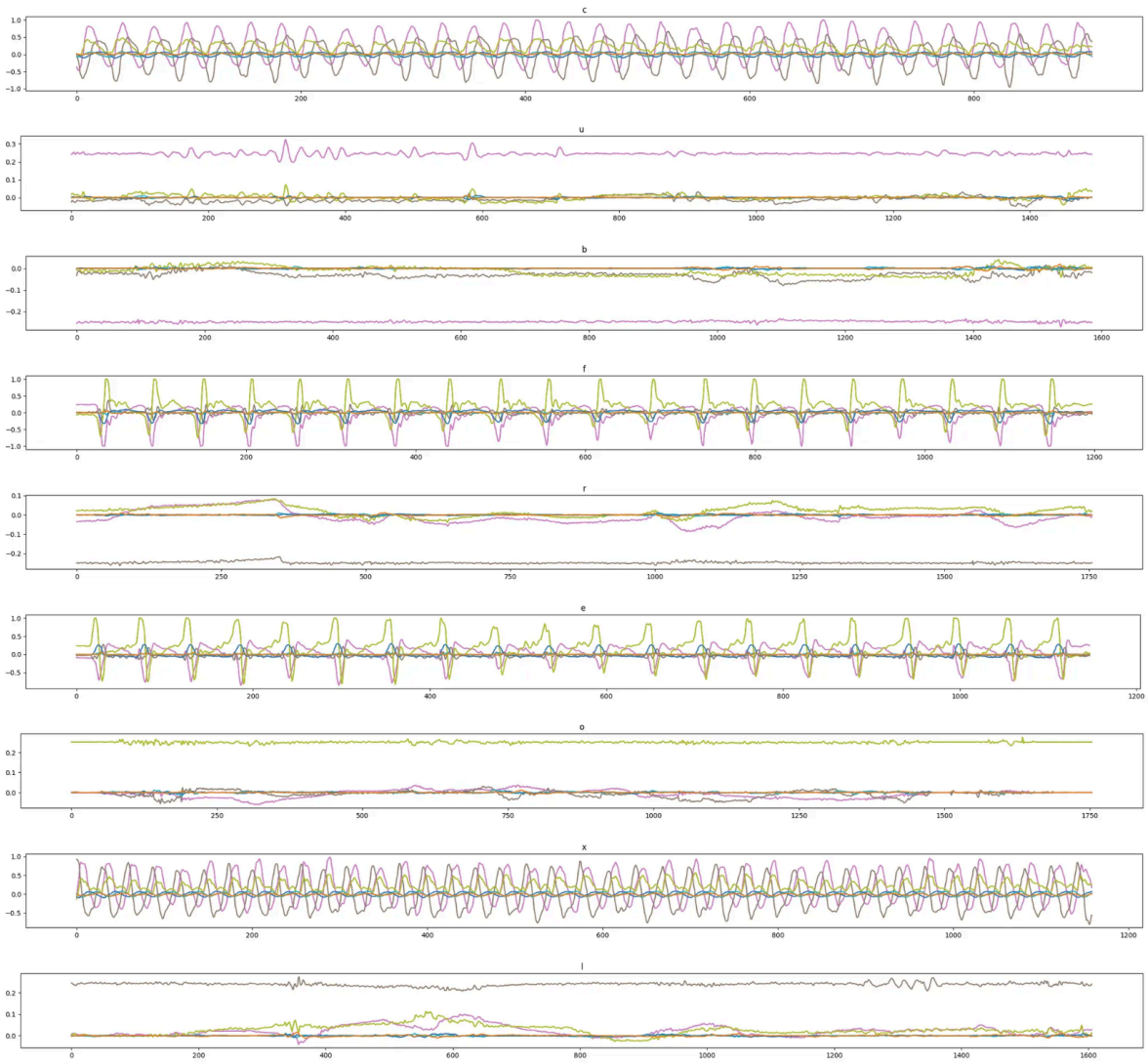
Result



Test Loss: 0.0028, Test Acc: 1.0000

4. Analysis of User Test

User IMU Data For Training (Complete test data of four users can be found in `./data/UserData/`)



Result For User Interaction

User	Time(sec/destination)	Error
1	25.937461	0/5
2	19.837290	0/5
3	23.847294	0/5
4	20.927364	0/5
Average	21.47	0/5

Strengths:

- Intuitive and natural gesture mappings significantly enhanced user learning speed and engagement.
- Improved accessibility: suitable for left-handed, right-handed, and disabled users alike, meeting inclusivity design goals.
- Substantially lowered cognitive load by clearly aligning gestures with intuitive physical metaphors, enabling quicker memorization and lessening fatigue.

Optimization

Model Optimization

Version	Train Model	Gesture Type	Model Architecture	Data Scale	Gesture Detect (Label)	Regularization	Learning Rate	Epochs	Early Stopping	Reduce LR on Plateau
Base	Keras	Hybrid	Simple NN: Dense(64)->Dense(output)	2686	5	None	3e-4	30	patience=2	None
Final	Keras	Hybrid	Enhanced Dense (128)-> BatchNorm -> Dropout(0.3)-> Dense	12154	9	L2 (0.001), BatchNorm, Dropout(0.3)	3e-4 (adaptive reduction)	100	patience=5 (restore best weights)	Applied, factor=0.5, patience=2

Throughout the project, Keras was consistently chosen as the training framework, ensuring development stability, ease of model comparison, and streamlined debugging. This consistency allowed clearer evaluation of iterative enhancements without the complexity and overhead associated with switching between frameworks.

From the base iteration, which employed a simple two-layer neural network, the model architecture evolved significantly. The optimized iteration introduced increased complexity by incorporating additional neurons, Batch Normalization, Dropout layers, and L2 regularization. These changes notably improved model accuracy, reduced overfitting, and enhanced the robustness of gesture classification. Additionally, a substantial expansion of the dataset—from approximately 2,681 samples initially to over 12,154 samples in the optimized

version—enabled the model to capture a wider variety of gesture nuances, resulting in improved prediction stability and generalization to diverse user behaviors.

Besides, for the pre-procedure of training data, I added small bias for statistic gestures in order to increase stability and fault tolerance of action recognition.

Before

After

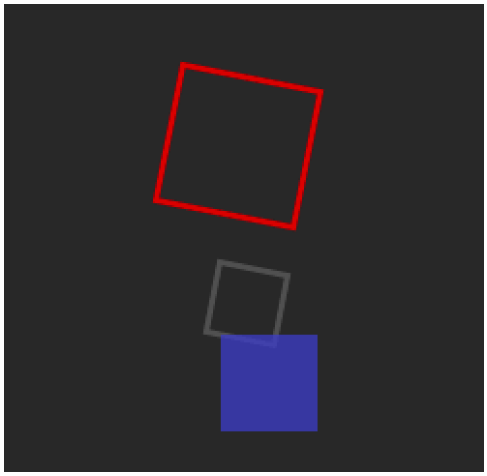
Furthermore, training procedures were refined by extending the training epochs from 30 to 100, alongside adjustments in early stopping criteria and adaptive learning rate scheduling. These modifications effectively addressed convergence issues, preventing premature termination of training and reducing overfitting risks.

Collectively, these iterative enhancements resulted in a gesture interaction system that achieves significantly better performance, stability, and user-centric intuitiveness.

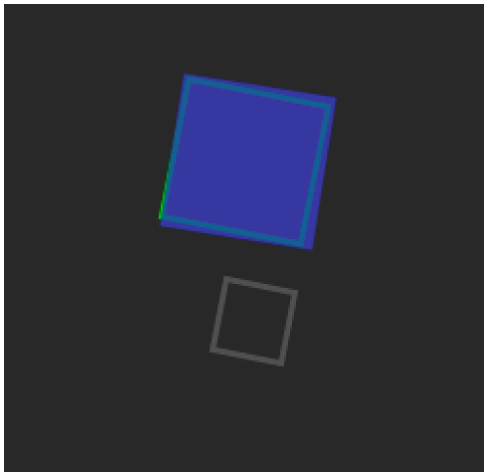
UI Optimization

Vision

For better visual effect, I optimized `UI.pde` by adding an extra procession which is dynamically changing the target object's color based on alignment accuracy significantly enhances user experience by providing clear and immediate visual feedback. This intuitive color-coding (green for correct alignment and red otherwise) reduces cognitive load, allowing users to quickly assess their progress without distracting textual or numeric cues, thus improving interaction efficiency and user confidence.



Before



After

Optimization effect

Version(Average Result)	Time(sec/destination)	Error
Version 1	16.36	0/5
Version 2	33.73	0.5/5
Version 3	26.90	0.25/5
Final Version(After Model Optimization)	21.47	0/5
Final Version(After UI Optimization)	15.98	0/5

*Bonus: TinyML

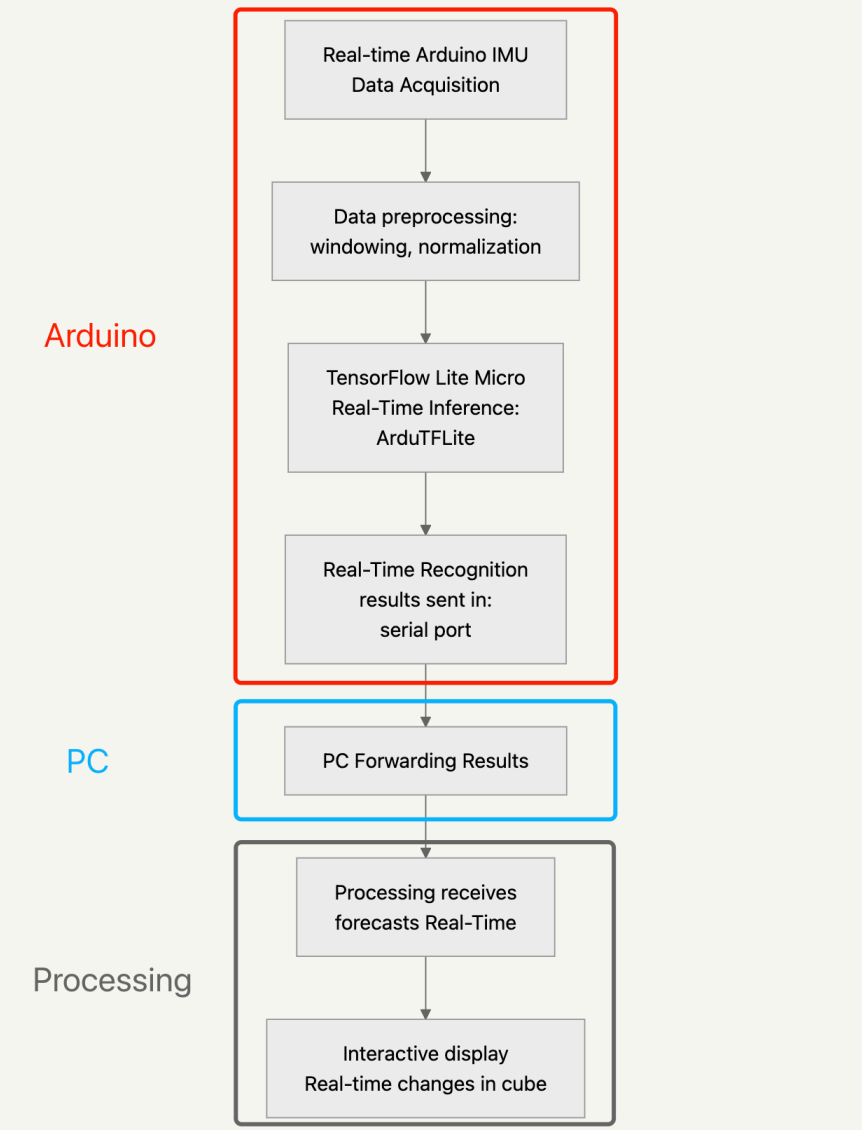
The original project was realized as:

1. Arduino side Capture raw data from IMU sensors (acceleration, gyroscope).
2. The data is transferred to the computer via USB cable.
3. The data is transferred to the computer via USB cable. On the computer (Python side), a pre-trained Keras model is used to predict gesture categories using real-time inference.
4. The Python program then sends the prediction results to Processing via UDP to control the UI interaction.

Migrating gesture prediction to the Arduino side was implemented in the following steps：

1. **Model training and preparation (Python side)**
    - Use the existing Keras model and make necessary optimizations and trimmings to make it suitable for Arduino resources.
    - Convert a Keras model to a TensorFlow Lite model (.tflite) that is usable by TinyML .
2. **Model conversion (TFLite):** Use TensorFlow Lite Converter to `.keras.tflite` convert the model to format:
3. **Model formatted as Arduino C header file (.h):** Convert `.tflite` the model file into a C language array so that Arduino can call it directly.
4. **Model deployment and inference on Arduino:** Use `ArduTFLite` the library to load the model and receive IMU data in real time to complete gesture recognition:
5. **Output prediction results in real time:** The recognized gesture category ID is sent to the computer for subsequent interaction design control.

After migrating the system flow is shown below.

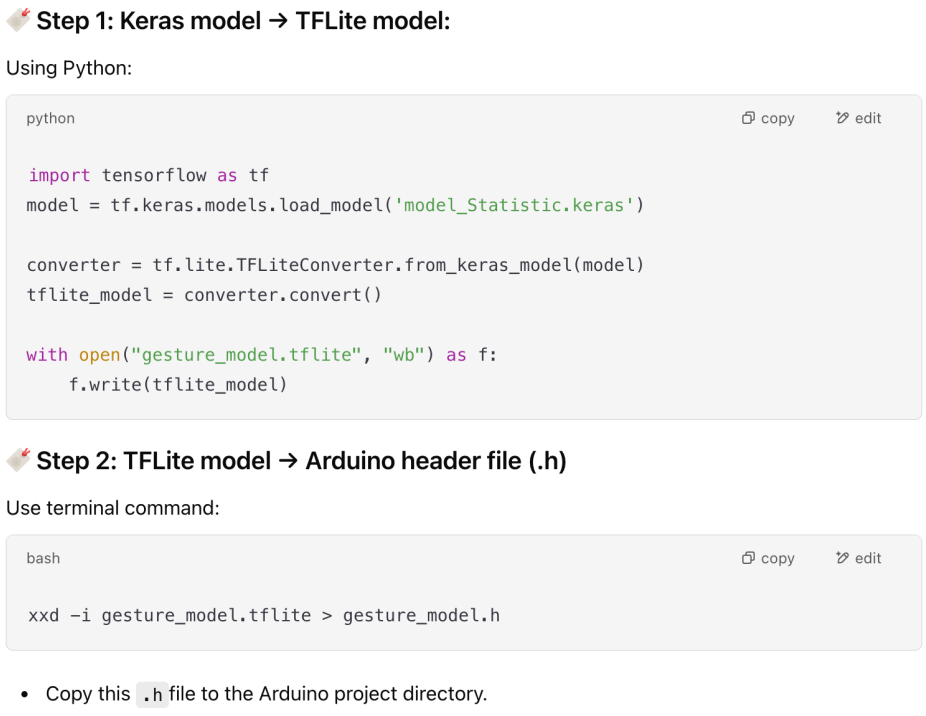



```
graph TD;
    A[Real-time Arduino IMU
    Data Acquisition] --> B[Data preprocessing:
    windowing, normalization ];
    B --> C[TensorFlow Lite Micro
    Real-Time Inference:
    ArduTFLite];
    C -->D[Real-Time Recognition
    results sent in:
    serial port];
```

```
D -->E[PC Forwarding Results];
E -->F[Processing receives
forecasts Real-Time];
F -->G[Interactive display
Real-time changes in cube];
```

Reference

- https://github.com/arduino-libraries/ArduinoBLE/tree/master
- https://docs.arduino.cc/programming/
- AI using (gpt-4o)
  - Prompt: “Briefly describe the main steps about how to use the ArduTFLite library to classify gestures based on IMU sensor data.”
  - Reply which was used in *convert_to_Arduino.py* and terminal to generate *gesture_model.tflite* and *gesture_model.h*



Step 1: Keras model → TFLite model:

Using Python:

python copy edit

```python
import tensorflow as tf
model = tf.keras.models.load_model('model_Statistic.keras')

converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open("gesture_model.tflite", "wb") as f:
    f.write(tflite_model)
```

Step 2: TFLite model → Arduino header file (.h)

Use terminal command:

bash copy edit

```bash
xxd -i gesture_model.tflite > gesture_model.h
```

- Copy this `.h` file to the Arduino project directory.