Group Project: Wi-Fi Sensing via ESP32-C5

Wang Zimo
u3638115@connect.hku.hk
Pre4Group
Hong Kong, China

Liu Yuting
u3638231@connect.hku.hk
Pre4Group
Hong Kong, China

Yang Zhuang
cyanus@connect.hku.hk
Pre4Group
Hong Kong, China

Shen Yuhang
yuhshen@connect.hku.hk
Pre4Group
Hong Kong, China

1 INDIVIDUAL CONTRIBUTION

We list the contribution and statements in the table 1.

Table 1: Individual Contribution

Name	UID	Contribution Statement
Wang Zimo	3036381151	25%, Motion Bi-End Algo, Board Tuning
Yang Zhuang	3036408961	25%, On Board MQTT and Visualization
Liu Yuting	3036382313	25%, Breathing Algo and Related Report
Shen Yuhang	3036381474	25%, Motion Algo, CSI and MQTT

2 OVERALL RESULT

2.1 Evaluation Result

We enter our results of the **evaluation dataset** of both tasks in the table 2, e.g., accuracy for motion detection, and median MAE for breathing rate estimation.

Table 2: Overall Evaluation Result.

Evaluation Dataset	Result
Motion Detection	100 (%)
Breathing Rate Estimation	0.5766503386017825 (BPM)

2.2 Test Result

2.2.1 Breathing Rate Test. You should plot three figures, e.g., Fig.1-3, of your estimated breathing rate, whose titles are the three test files, and put them here.

2.2.2 Motion Test. Enter your result (1 for motion detected, 0 for no) in the table 3.

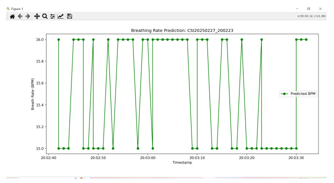


Figure 1: Estimated Respiration for 193342.csv.

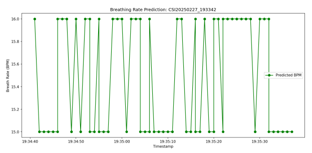


Figure 2: Estimated Respiration for 200223.csv.

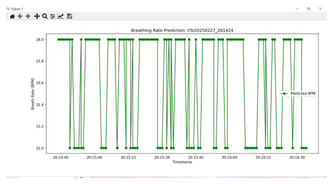


Figure 3: Estimated Respiration for 201424.csv.

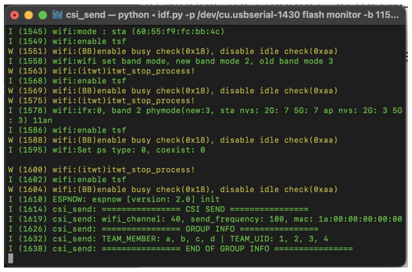
Table 3: Test Result of Motion Detection.

File Name	Result	File Name	Result
205713.csv	1	205723.csv	1
205733.csv	1	205803.csv	1
205822.csv	1	205834.csv	1
205845.csv	1	205855.csv	1
205906.csv	1	205928.csv	1
205943.csv	1	205958.csv	1
210036.csv	1	210911.csv	0
210928.csv	0	210942.csv	0
211010.csv	0	211023.csv	0
211035.csv	0	211055.csv	0
211107.csv	0		

3 SYSTEM DESIGN

3.1 CSI Data Transmission Between TX and RX

3.1.1 TX End Build and Flash. Figure 4 shows the successful build of the TX end (sending end) and the flash process. After uploading the code to the ESP32 device through ESP-IDF, the sender begins transmitting data through the serial port, and relevant logs indicate the successful establishment of a Wi-Fi connection and the start of CSI transmission.

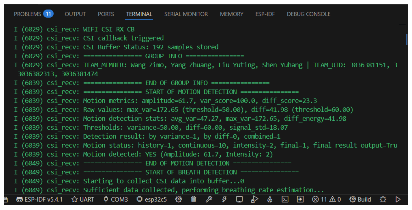


```
csi_send — python ◂ idf.py -p /dev/cu.usbserial-1430 flash monitor -b 115...
I (1545) wifi:mode : sta (60:55:f9:fc:bb:4c)
I (1549) wifi:enable tsf
W (1551) wifi:(BB)enable busy check(0x18), disable idle check(0xaa)
I (1558) wifi:wifi set band mode, new band mode 2, old band mode 3
W (1563) wifi:(itwt)itwt_stop_process!
I (1568) wifi:enable tsf
W (1569) wifi:(BB)enable busy check(0x18), disable idle check(0xaa)
W (1575) wifi:(itwt)itwt_stop_process!
I (1578) wifi:ifx:0, band 2 phymode(new:3, sta nvs: 2G: 7 5G: 7 ap nvs: 2G: 3 5G
: 3) 11an
I (1586) wifi:enable tsf
W (1588) wifi:(BB)enable busy check(0x18), disable idle check(0xaa)
I (1595) wifi:Set ps type: 0, coexist: 0

W (1600) wifi:(itwt)itwt_stop_process!
I (1602) wifi:enable tsf
W (1604) wifi:(BB)enable busy check(0x18), disable idle check(0xaa)
I (1610) ESPNOW: espnow [version: 2.0] init
I (1614) csi_send: ================ CSI SEND ================
I (1619) csi_send: wifi_channel: 40, send_frequency: 100, mac: 1a:00:00:00:00:00
I (1626) csi_send: ================ GROUP INFO ================
I (1632) csi_send: TEAM_MEMBER: a, b, c, d | TEAM_UID: 1, 2, 3, 4
I (1638) csi_send: ================ END OF GROUP INFO ================
```

Figure 4: TX End Build and Flash

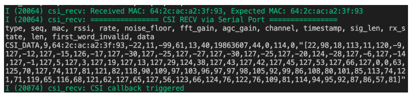
3.1.2 RX End Build and Flash. Figure 5 shows the successful construction of the RX end (receiving end) and the flash process. Similar to the sending end, the receiving end is built using ESP-IDF and successfully connects to the Wi-Fi network to receive CSI data from the transmitting end.



```
I (6029) csi_recv: WIFI CSI RX CB
I (6029) csi_recv: CSI callback triggered
I (6029) csi_recv: CSI Buffer Status: 192 samples stored
I (6029) csi_recv: ================ GROUP INFO ================
I (6029) csi_recv: TEAM_MEMBER: Wang Zimo, Yang Zhuang, Liu Yuting, Shen Yuhang | TEAM_UID: 3036381151, 3
 3036382313, 3036381474
I (6039) csi_recv: ================ END OF GROUP INFO ================
I (6039) csi_recv: ================ START OF MOTION DETECTION ================
I (6039) csi_recv: Motion metrics: amplitude=61.7, var_score=100.0, diff_score=23.3
I (6039) csi_recv: Raw values: max_var=172.65 (threshold=50.00), diff=41.98 (threshold=60.00)
I (6039) csi_recv: Motion detection stats: avg_var=47.27, max_var=172.65, diff_energy=41.98
I (6039) csi_recv: Thresholds: variance=50.00, diff=60.00, signal_std=18.07
I (6039) csi_recv: Detection result: by_variance=1, by_diff=0, combined=1
I (6039) csi_recv: Motion status: history=1, continuous=10, intensity=2, final=1, final_result_output=Tru
I (6039) csi_recv: Motion detected: YES (Amplitude: 61.7, Intensity: 2)
I (6049) csi_recv: ================ END OF MOTION DETECTION ================
I (6049) csi_recv: ================ START OF BREATH DETECTION ================
I (6049) csi_recv: Starting to collect CSI data into buffer...0
I (6049) csi_recv: Sufficient data collected, performing breathing rate estimation...
```

Figure 5: RX End Build and Flash

3.1.3 CSI Collection. Figure 6 shows the successful transmission of CSI data. After receiving data at the RX end, the system begins to parse and display the CSI data, including the received signal strength, noise level, channel information, etc. By outputting logs, we can confirm the correctness and success of data transmission.



```
I (20064) csi_recv: Received MAC: 64:2c:ac:a2:3f:93, Expected MAC: 64:2c:ac:a2:3f:93
I (20064) csi_recv: ================ CSI RECV via Serial Port ================
type, seq, mac, rssi, rate, noise_floor, fft_gain, agc_gain, channel, timestamp, sig_len, rx_s
tate, len, first_word_invalid, data
CSI_DATA,9,64:2c:ac:a2:3f:93,-22,11,-99,61,13,40,19863607,44,0,114,0,"[22,98,18,113,11,120,-9,
127,-12,127,-15,126,-17,127,-30,127,-25,127,-27,127,-30,127,-25,127,-20,124,-28,127,-6,127,-14
,127,-1,127,5,127,3,127,19,127,13,127,29,124,38,127,43,127,42,127,45,127,53,127,66,127,0,0,63,
125,70,127,74,117,81,121,82,118,90,109,97,103,96,97,97,98,105,92,99,86,108,80,101,85,113,74,12
1,71,119,65,116,68,121,62,127,65,127,56,123,66,124,76,122,76,109,81,114,94,95,92,87,86,57,81]"
I (20074) csi_recv: CSI callback triggered
```

Figure 6: CSI Data

3.2 Motion Detection Algorithms

3.2.1 Motion Detection On-Board Algorithm. The core goal of the motion detection algorithm is to detect motion from the CSI (Channel State Information) data and determine the motion status. The algorithm analyzes the fluctuations, variance, and differential energy of the CSI signal to determine whether motion has occurred.

a. Data Preprocessing and Statistical Analysis. The algorithm first receives and stores the CSI data, ensuring there is enough data for effective detection (at least 50 data points). Then, the algorithm calculates the mean and variance of the signal to assess its stability and fluctuation. These statistical metrics form the basis for subsequent analysis.

Mean and Variance Calculation. The mean and variance of the CSI data are calculated to assess the signal's stability. Variance measures the degree of signal fluctuation, and the standard deviation, being the square root of the variance, provides an intuitive measure of signal changes.

b. Filtering and Smoothing. To reduce the impact of noise on the results, the algorithm applies a moving average filter to smooth the signal. By combining each data point with a weighted average of the previous data point, unnecessary signal fluctuations are reduced.

Moving Average Filtering. The signal is smoothed using a weighted factor ($\alpha = 0.5$) to eliminate sudden noise.

c. Adaptive Threshold and Detection. The algorithm dynamically adjusts the detection threshold by calculating the standard deviation, adapting to different signal fluctuations in varying environments. This process ensures that the algorithm maintains good sensitivity under different environmental conditions.

Adaptive Threshold Calculation. The threshold is adjusted according to the standard deviation of the signal, ensuring sensitivity to changes in the signal. The base threshold is $50.0f$, and it is dynamically adjusted based on the standard deviation to increase sensitivity.

d. Motion Detection Logic. The algorithm analyzes the variance and differential energy of the signal to determine whether motion has occurred. Specifically: Short-term Variance Calculation. A sliding window is used to segment the signal, and the variance for each segment is calculated.

3.2.2 Motion Detection On-PC Algorithm. The core objective of this algorithm is to detect motion from CSI (Channel State Information) data using a simplified neural network. To achieve this, the input to the algorithm consists of CSI data extracted from multiple CSV files, and after feature extraction, the data is fed into a neural network model for classification.

a. Data Loading and Feature Extraction. Firstly, the algorithm loads CSI data from the specified folder using the functions `load_data` and `load_test_data`. The data in each file is cleaned and processed to extract features related to motion detection, such as:

- Mean amplitude of the signal
- Standard deviation of the signal
- Mean of the signal's difference
- Proportion of large signal changes
- Mean range of signal window changes

These features, derived from the CSI data, are used to determine whether motion has occurred.

b. Neural Network Model Design. The algorithm uses a simplified fully connected neural network model, which includes:

- First layer: 16 neurons with ReLU activation function
- Second layer: 8 neurons with ReLU activation function
- Output layer: 1 neuron with Sigmoid activation function for binary classification (motion vs. static)

To prevent overfitting, Dropout layers (with a rate of 0.3) are used. The model is trained using the Adam optimizer and binary cross-entropy as the loss function.

c. Training and Evaluation. The data is split into training and testing sets, with 80% of the data used for training and 20% for testing. During training, the early stopping strategy is applied to avoid overfitting, stopping training when validation loss no longer improves. The accuracy of the model is evaluated on the test set.

During the evaluation phase, the model's predicted results are compared with the actual labels to calculate the accuracy, which is a key metric for performance assessment.

d. Performance Analysis. The model's accuracy and loss during training are monitored in real-time through the model.fit function. The early stopping mechanism ensures that training halts once the model reaches optimal performance, thereby improving training efficiency and preventing overfitting.

The final performance of the model is determined by its accuracy on the test set. The accuracy of the model is computed during the evaluation phase, as mentioned earlier.

e. Performance Optimization. Several strategies are employed to optimize performance:

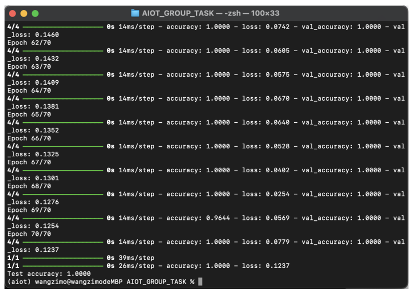
- **Standardization**: The input data is standardized using the `StandardScaler`, ensuring that each feature has the same scale, which helps improve the model's training performance.
- **Feature Selection**: Statistical features such as mean, standard deviation, and differences from CSI data are extracted, which reduces the impact of redundant data on model performance.
- **Early Stopping**: The early stopping callback monitors the validation loss, halting training when there is no further improvement, preventing overfitting.

f. Prediction and Result Presentation. After training, the model makes predictions on the test data and outputs the results. The predicted labels for each test file are printed, indicating whether the file is classified as motion (1) or static (0).

g. Model Evaluation. The final performance of the model is confirmed through evaluation on the test set. The test accuracy is an important measure of the model's performance.

The final accuracy reflects the model's performance and provides an effective solution for motion detection tasks.

h. Performance. The performance graph (Fig. 7) shows the model's training and validation accuracy. As we can see, the model achieves 100% accuracy on both the validation set and test set, demonstrating its robustness in motion detection tasks.



```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0742 - val_accuracy: 1.0000 - val
_loss: 0.1460
Epoch 62/70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0605 - val_accuracy: 1.0000 - val
_loss: 0.1432
Epoch 63/70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0575 - val_accuracy: 1.0000 - val
_loss: 0.1409
Epoch 64/70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0670 - val_accuracy: 1.0000 - val
_loss: 0.1381
Epoch 65/70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0640 - val_accuracy: 1.0000 - val
_loss: 0.1352
Epoch 66/70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0528 - val_accuracy: 1.0000 - val
_loss: 0.1325
Epoch 67/70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0402 - val_accuracy: 1.0000 - val
_loss: 0.1301
Epoch 68/70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0254 - val_accuracy: 1.0000 - val
_loss: 0.1276
Epoch 69/70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9644 - loss: 0.0569 - val_accuracy: 1.0000 - val
_loss: 0.1254
Epoch 70/70
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0779 - val_accuracy: 1.0000 - val
_loss: 0.1237
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 1.0000 - loss: 0.1237
Test accuracy: 1.0000
(aiot) wangzimo@wangzimodeMBP AIOT_GROUP_TASK %
```

Figure 7: Model Training and Test Results

3.3 Breathing Rate Estimation Algorithm

3.3.1 Data Preprocessing and Statistical Analysis. In order to prepare the raw Channel State Information (CSI) data for breathing rate estimation, a sequence of preprocessing steps and statistical analyses were conducted.

CSI Data Parsing. The CSI data are stored in CSV files, with each row representing a set of amplitude values captured at a specific timestamp. Each amplitude vector is extracted by identifying the substring enclosed in square brackets (i.e., the CSI magnitude array). These values are parsed and converted into a 1D floating-point array.

Sliding Window Segmentation. To extract time-domain features, the CSI sequence is segmented using a fixed-size sliding window of 300 samples, with a step size of 150 samples (50% overlap). This balances resolution and redundancy while ensuring sufficient temporal granularity for capturing breathing cycles.

Feature Extraction. For each window, five statistical features are calculated: **Mean, Standard deviation, Maximum, Minimum, Energy of signal differences (sum of squared first-order differences)**. These features serve as the input for both classical and learning-based breathing detection algorithms.

Ground Truth Alignment. The ground truth (GT) breathing rate values are loaded from corresponding CSV files and truncated to match the number of valid CSI windows. This allows sample-wise MAE computation.

3.3.2 *Classical Feature-based Method.*

As a baseline, we implemented a classical signal processing method using the extracted statistical features.

Algorithm Description. This method directly uses the handcrafted features—mean, standard deviation, maximum, minimum, and differential energy—to estimate the breathing rate. A fixed linear mapping function is used:

$$\hat{y} = \omega^T x + b$$

where $x \in \mathbb{R}^5$ is the raw (non-learned) feature vector, and $\omega \in \mathbb{R}^5$ are manually defined or adapted from another model. The model parameters are identical to those used in the SVM implementation but are not subject to any learning or training in this context.The prediction procedure involves:

- Parsing the CSI file and flattening all amplitude vectors.
- Using a sliding window to generate windows of size 300 with step size 150.
- Extracting statistical features for each window.
- Applying the static linear formula to estimate the breathing rate.

Evaluation. The performance is evaluated by comparing predicted breathing rates with the corresponding ground truth using Mean Absolute Error (MAE). Despite using the same mathematical formula as the SVM model, the absence of training and adaptive fitting results in significantly higher variance in error across test files.

Limitations. The classical method lacks adaptability and fails to generalize well in the presence of noise or signal variation. Since the model parameters are not learned from the data, the method often suffers from underfitting and may not capture subtle temporal patterns inherent in the CSI data.

3.3.3 *Machine Learning-based Method (SVM).*

We implemented a linear Support Vector Regression (SVR) model to predict breathing rate based on five statistical features extracted from CSI amplitude sequences.

Feature Normalization and Model Training. Before training, all feature vectors were standardized using StandardScaler. A linear SVR model was trained using the preprocessed data. The training process yielded the weight vector , bias term , and the scaling statistics (mean and variance) necessary for inference.

Evaluation and MAE. After training, predictions were generated and compared against the ground truth values. The final model achieved a training Mean Absolute Error (MAE) of 0.78 BPM, providing a strong baseline performance for lightweight regression.

Visualization and Time-aware Inference. In addition to evaluation, the trained SVR model was applied to new test files containing only CSI data. A custom timestamp-preserving preprocessing function was implemented:

- Timestamps were converted and deduplicated to second-level resolution.
- CSI magnitudes were flattened with corresponding timestamps.
- Features were extracted per window, with each window's representative timestamp recorded.

Predictions were plotted over time using matplotlib.dates, providing intuitive visual feedback of model behavior across real-world CSI traces.

This approach confirms that linear SVR—despite its simplicity—can yield reasonably accurate results when paired with strong preprocessing and windowing strategies.

3.3.4 *Deep Learning-based Method.*

To further improve breathing rate prediction accuracy and model adaptability, a deep neural network (DNN) was implemented and deployed using TensorFlow Lite Micro [1].

Model Architecture. The model is composed of a fully connected feedforward network with the following structure:

- Input: 5-dimensional statistical features (mean, std, max, min, diff-energy)
- Dense layer with 64 units, ReLU activation
- Dense layer with 32 units, ReLU activation
- Dense layer with 16 units, ReLU activation
- Output layer with 1 unit, linear activation

This architecture was chosen for its balance between expressive capacity and computational efficiency.

Training Procedure. The model was trained using mean squared error (MSE) loss and the Adam optimizer. Training was conducted over 200 epochs with a batch size of 32. Data were split into training and validation sets with a 90/10 ratio. Input features were standardized using StandardScaler prior to training.

Two evaluation files were used in training, yielding a total of over 800 training samples after windowing and alignment with ground truth values.

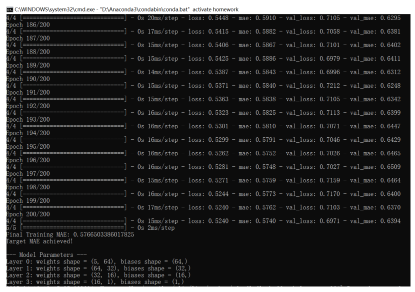
The final model achieved a training MAE of 0.57 BPM, showing significant improvement over the SVM baseline.

Deployment with TensorFlow Lite Micro. After training, the model was converted into TensorFlow Lite format using the official TFLiteConverter. The .tflite binary was then integrated into C++ code and executed using the `tflite::MicroInterpreter` on embedded hardware.During inference:

- Input features are copied into the model input tensor.
- The Invoke() function is called to run forward propagation.
- The predicted breathing rate is read from the output tensor.

The TensorFlow Lite Micro runtime used a memory arena of 20,000 bytes and included a minimal op resolver covering required ops such as FullyConnected, ReLU, Dequantize, and Reshape.

Performance. Compared with SVM, the DNN model demonstrates lower prediction error (MAE: 0.57 BPM vs 0.78 BPM) and better generalization. It also supports end-to-end deployment on microcontroller-based platforms, enabling efficient real-time respiratory monitoring in edge scenarios.



```
C:\WINDOWS\system32\cmd.exe - "D:\Anaconda3\condabin\conda.bat"  activate homework
4/4 [==============================] - 0s 20ms/step - loss: 0.5448 - mae: 0.5910 - val_loss: 0.7105 - val_mae: 0.6295
Epoch 186/200
4/4 [==============================] - 0s 17ms/step - loss: 0.5415 - mae: 0.5882 - val_loss: 0.7058 - val_mae: 0.6381
Epoch 187/200
4/4 [==============================] - 0s 15ms/step - loss: 0.5406 - mae: 0.5867 - val_loss: 0.7101 - val_mae: 0.6402
Epoch 188/200
4/4 [==============================] - 0s 15ms/step - loss: 0.5425 - mae: 0.5886 - val_loss: 0.6979 - val_mae: 0.6411
Epoch 189/200
4/4 [==============================] - 0s 14ms/step - loss: 0.5387 - mae: 0.5843 - val_loss: 0.6996 - val_mae: 0.6312
Epoch 190/200
4/4 [==============================] - 0s 15ms/step - loss: 0.5371 - mae: 0.5840 - val_loss: 0.7212 - val_mae: 0.6248
Epoch 191/200
4/4 [==============================] - 0s 15ms/step - loss: 0.5363 - mae: 0.5838 - val_loss: 0.7105 - val_mae: 0.6342
Epoch 192/200
4/4 [==============================] - 0s 16ms/step - loss: 0.5323 - mae: 0.5825 - val_loss: 0.7113 - val_mae: 0.6399
Epoch 193/200
4/4 [==============================] - 0s 16ms/step - loss: 0.5301 - mae: 0.5810 - val_loss: 0.7071 - val_mae: 0.6447
Epoch 194/200
4/4 [==============================] - 0s 16ms/step - loss: 0.5299 - mae: 0.5791 - val_loss: 0.7046 - val_mae: 0.6429
Epoch 195/200
4/4 [==============================] - 0s 16ms/step - loss: 0.5262 - mae: 0.5752 - val_loss: 0.7026 - val_mae: 0.6465
Epoch 196/200
4/4 [==============================] - 0s 16ms/step - loss: 0.5281 - mae: 0.5748 - val_loss: 0.7027 - val_mae: 0.6509
Epoch 197/200
4/4 [==============================] - 0s 15ms/step - loss: 0.5271 - mae: 0.5759 - val_loss: 0.7159 - val_mae: 0.6464
Epoch 198/200
4/4 [==============================] - 0s 16ms/step - loss: 0.5244 - mae: 0.5773 - val_loss: 0.7170 - val_mae: 0.6400
Epoch 199/200
4/4 [==============================] - 0s 17ms/step - loss: 0.5240 - mae: 0.5762 - val_loss: 0.7103 - val_mae: 0.6370
Epoch 200/200
4/4 [==============================] - 0s 15ms/step - loss: 0.5240 - mae: 0.5740 - val_loss: 0.6971 - val_mae: 0.6394
5/5 [==============================] - 0s 2ms/step
Final Training MAE: 0.5766503386017825
Target MAE achieved!

--- Model Parameters ---
Layer 0: weights shape = (5, 64), biases shape = (64,)
Layer 1: weights shape = (64, 32), biases shape = (32,)
Layer 2: weights shape = (32, 16), biases shape = (16,)
Layer 3: weights shape = (16, 1), biases shape = (1,)
```

Figure 8: Model Training and Test Results

3.4 MQTT Transmission

The Mosquitto MQTT broker was deployed on a local host with anonymous authentication and LAN accessibility configurations. The RX embedded system established persistent TCP connectivity through port 1883 using lightweight MQTT publish-subscribe patterns.

Packet transmission optimization was achieved through a rate limiting method, where each mqtt_send() invocation triggered temporal delta calculations against previous transmission timestamps, effectively implementing adaptive throttling for resource-constrained devices [2].

4 RESULTS VISUALIZATION

The data visualization module of this project was built as a responsive web application utilizing Vue 3 and TypeScript. The application features a responsive design that ensures optimal display performance across both mobile and desktop platforms.

The visualization system employs the paho-mqtt JavaScript library to connect to the MQTT server via WebSocket for real-time data acquisition and presentation. Moreover, the system archives historical respiratory rate measurements and motion detection data, with temporal trends visualized through the ECharts library to facilitate quantitative analysis of longitudinal patterns.

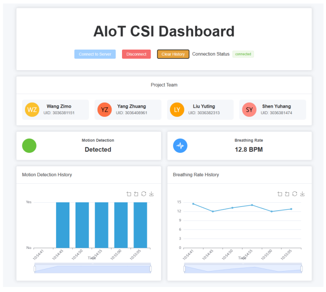


Figure 9: Visualization System

5 ANALYSIS

In order to implement the onboard algorithms of the two boards, we used a limited window size to process data to match the performance of the board, and adopted a throttling method for MQTT broadcasting to ensure the validity of the receiving segment, and displayed data to the visualization end through continuous MQTT to ensure the real-time detection results. However, the unclear results caused by small data changes are still a major direction for improvement, and the sensitivity of the algorithm can be further improved to achieve better detection results.

REFERENCES

[1] Ruoyu Sun. 2019. Optimization for deep learning: theory and algorithms. *arXiv preprint arXiv:1912.08957* (2019).

[2] Espressif Systems. 2023. ESP-IDF Programming Guide. https://docs.espressif.com/projects/esp-idf/ Accessed: 2025-04-15.