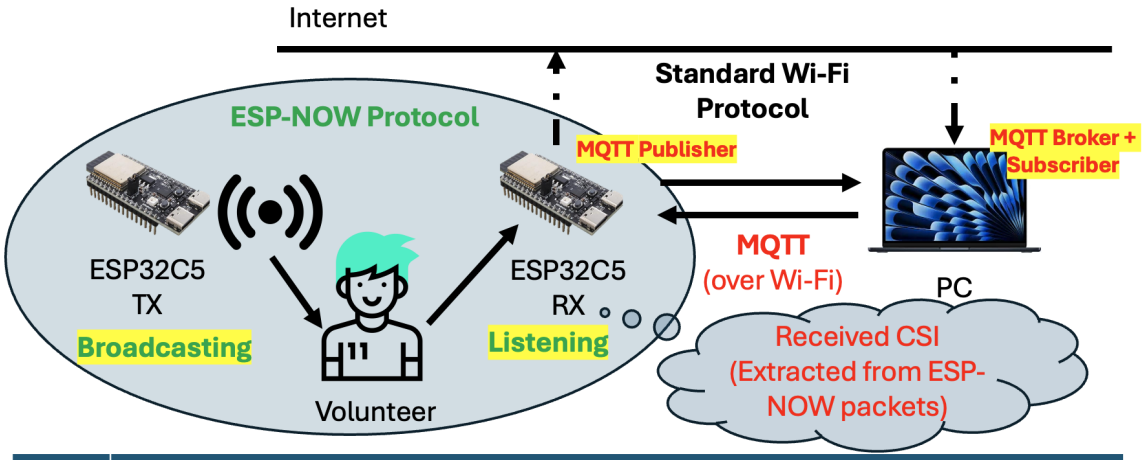
Group Project

We have explored various methods and conducted experiments on extracting vital signs using wireless modalities through lectures and labs. Now, let's dive in and get hands-on with the ESP32-C5 embedded system to further explore opportunities for extracting motion and breathing rates from Channel State Information (CSI) and transmitting the data to a PC via MQTT. The typology and scenario of the experiments is depicted as follows:



Device	Functions
TX	Sending ESP-NOW packets via broadcast
RX	1. Receiving and extracting CSI from the ESP-NOW packets; 2. Connecting to Internet via standard Wi-Fi protocol and transmitting the data (CSI or estimated motion or breathing BPM) via MQTT (MQTT Publisher); 3. Estimating the motion and breathing rate (if you can implement on-board)
PC	1. Receiving the data via MQTT (MQTT Broker + Subscriber); 2. Estimating the motion and breathing rate (if you can implement offline)

Figure 1: The scenario and typology of the project.

In this project, you will be required to complete four tasks:

- **Task 1: CSI Collection (10 pts)**

  Successfully send and receive CSI data between the transmitter (TX) and receiver (RX).

- **Task 2: Data Processing (50 pts)**

  Design an algorithm to process the CSI data you collected.

  - **Task 2.1: Motion Detection (20 of 50 pts)**

    Implement an algorithm to detect motion using CSI data.

  - **Task 2.2: Breathing Estimation (30 of 50 pts)**

    Develop an algorithm to monitor and detect breathing rates from CSI data.

- **Task 3: Data Transmission (20 pts)**

  Transmit data from the RX to your PC via the **MQTT** protocol.

- **Task 4: Data Visualization (10 pts)**

  Develop an end-to-end system that visualizes your results.

- **Task 5: Report Writing (10 pts)**

  Write a report to show your results and explain your algorithm design using provided template.

Group Registration: Team leader needs to fill out the Qualtrics.

Grading Rubric

Features	Grades (point)	Details
Task 1: CSI Collection	10	-
Task 2.1: Motion Detection	20	Performance: 20%, Method: 60%, Real-time: 10%, On-board: 10%
Task 2.2: Breathing Rate Estimation	30	Performance: 20%, Method: 60%, Real-time: 10%, On-board: 10%
Task 3: Data Transmission	20	-
Task 4: Data Visualization	10	Report and other methods.
Task 5: Technical Report	10	-
Total	**100**	Graded based on their contribution statements.
Live Demo	5	*Bonus* on your total scores

> NOTE: You all should try your best to build the on-board system but not the necessary for grading.

Hands-on ESP32C5 CSI Sensing

You are highly recommended to refer to the following websites for more detail:

Content	Source	Notes
ESP-IDF Tool	GitHub	Official documentation for the ESP-IDF toolchain. Refer to this resource for guidance on compilation, flashing, and installation issues.
ESP-NOW Protocol	Document	Detailed explanation of the ESP-NOW protocol. Use this document to understand the differences between ESP-NOW and standard Wi-Fi protocols.
ESP32-C5 Document	Document	Comprehensive documentation for ESP32-C5, featuring tutorials and troubleshooting guides to resolve common issues and bugs.
ESP-DSP	GitHub	Official repository for basic signal processing methods (e.g., FFT, filters). Integrated with the latest ESP-IDF Tool; add dependencies as needed.

Content	Source	Notes
VSC IDF Extension	GitHub	Official VS Code extension for ESP-IDF. Download this extension to streamline development directly within Visual Studio Code.

Hardware Requirements

- **Two ESP32-C5 Boards**: One board acts as a transmitter, and the other as a receiver.
- **Placement Guidelines**: Ensure the two boards are placed at least **1 meter apart** for effective CSI sensing.

Software Setup

In the traditional Wi-Fi CSI sensing, the transmitter typically operates in **station mode**, while the receiver functions in **monitor mode**, establishing a direct link for wireless sensing. However, **Espressif** has introduced **ESP-NOW**, a connectionless Wi-Fi communication protocol that enables data transmission without requiring a traditional Wi-Fi connection. In this approach, both the transmitter and receiver remain in **station mode**, deviating from conventional methods and offering a more flexible solution for CSI sensing.

Development Tool

The ESP-IDF environment is required. You should download this tool from the official GitHub website here. The basic usage of the ESP-IDF tool can be referred to in the support documents on Moodle.

Flashing the Firmware

Run the following commands to flash the firmware onto the ESP32-C5 boards:

```
# Flashing the transmitter
cd csi_send
idf.py --preview set-target esp32c5
idf.py build flash monitor -b 115200 -p /your/port

# Flashing the receiver
```

```
cd csi_recv
idf.py --preview set-target esp32c5
idf.py build flash monitor -b 921600 -p /your/port
```

The results can be seen as follows:

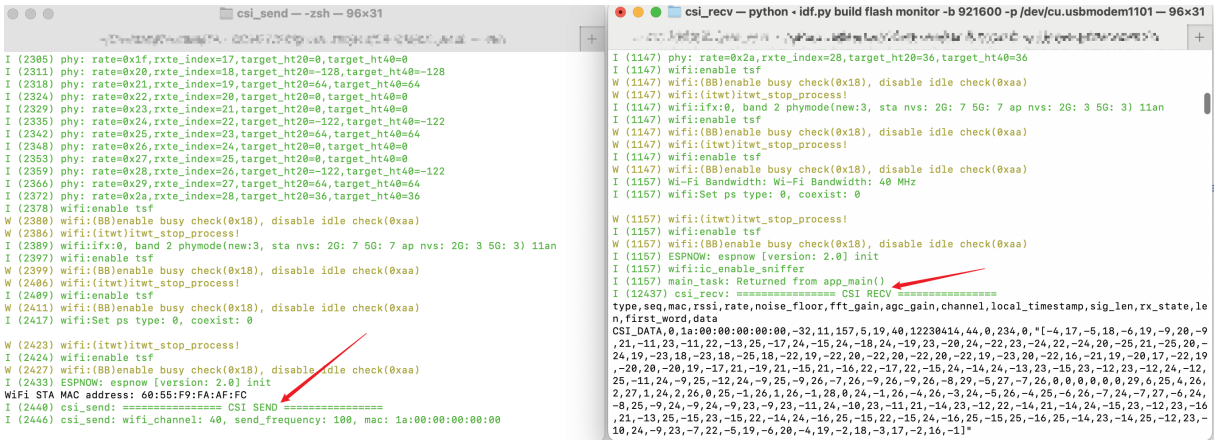


Figure 2: The terminal screenshot when minitoring the ESP32C5.

CSI Data Format

A single row of raw CSI data includes metadata and CSI values. Example:

> type,seq,mac,rssi,rate,noise_floor,fft_gain,agc_gain,channel,local_timestamp,sig_len,rx_state,len,first_word,data
> CSI_DATA,0,1a:00:00:00:00:00,-55,11,159,19,40,8,837230,44,0,234,0,“[-5,-28,-4,-27,-5,-33,-4,-30,-2,-33,-3,-33,-3,-33,0,-35,-1,-35,1,-36,3,-36,2,-34,1,-35,3,-36,5,-34,3,-38,3,-34,7,-36,5,-36,4,-34,8,-35,2,-34,5,-35,4,-34,3,-34,3,-33,3,-34,6,-32,3,-37,3,-32,7,-34,3,-33,3,-32,4,-33,2,-34,3,-32,2,-34,1,-33,-2,-31,1,-30,2,-30,0,-31,-2,-28,0,-31,-1,-29,-1,-28,-1,-29,-1,-28,-1,-28,-4,-27,-3,-27,-4,-24,-5,-25,-6,-25,-7,-24,-8,-24,-7,-24,0,0,0,0,0,0,-23,7,-22,8,-22,9,-20,9,-20,10,-20,9,-20,11,-18,9,-18,12,-17,11,-16,12,-15,11,-14,12,-15,13,-12,13,-14,13,-11,12,-12,12,-12,13,-12,16,-10,14,-11,15,-10,13,-9,15,-10,15,-8,14,-8,14,-7,15,-6,15,-6,15,-6,16,-4,15,-4,15,-3,14,-5,15,-4,16,-2,18,-2,16,-3,17,-3,17,-2,16,-1,17,0,17,0,16,-2,15,1,17,0,15,-2,18,-2,14,-2,14,-1,15,0,15,-1,16,-2,16,-1,15,-2,13,-3,12]”

  - **Metadata Fields**: Include `type`, `seq` , `mac`, `rssi`, `rate`, `noise_floor`, `len`, `first_word` , etc.

- **CSI Data Array**: Stored as an array of values in **[imaginary, real]** format for each subcarrier.

In the benchmark dataset, the CSI data will be stored as CSV files, which are structured as follows:

- **Headers**: Each CSV file contains multiple headers, with the last column representing the CSI values.
- **Data Shape**: The CSI data is stored in a **time (row) × subcarrier (last column)** format, with a sampling rate of 100 Hz.

MQTT Introduction

The following materials can be useful to setup MQTT service for the ESP32C5 device:

1. On Host PC (broker): Create an MQTT broker service using Mosquitto.
    - Download & install Mosquitto
    - Installation tutorial (Windows, Mac)
2. Establish Pulisher on ESP32-C5
    - Refer to this file: Link
3. On Host PC (subscriber): Run MQTT client python Paho-MQTT.
    - YouTube: MQTT Beginner Guide with Python

Materials

For each group participating in the benchmark, the following materials are provided:

- *TWO ESP32-C5 boards* for TX and RX respectively
- One package of basic source code
- Benchmark dataset

1. ESP32-C5 Boards

Each group will receive two ESP32-C5 boards: one for the transmitter and one for the receiver. Below are some key considerations:

1. Each ESP32-C5 board has two serial ports for flashing and data transmission: **UART** and **USB**. The assigned serial ports may vary depending on the PC used.

2. When flashing the board, if one serial port is unavailable, try using the other. Additionally, run `idf.py clean` or `idf.py fullclean` before the next flash attempt to ensure a clean build.
3. Do not change or mistake the bitrate when flashing the sender and receiver board:
    - **Sender board:** 115200 bps
    - **Receiver board:** 921600 bps
4. Ensure that your **ESP-IDF toolchain** is properly set up and can successfully connect to the board. By default, the sender board only needs to be flashed once; it will then automatically transmit ESP-NOW packets when powered by a battery.
5. If the compiled binary file exceeds the default flash memory limit, use `idf.py menuconfig` to increase the *Flash size* (default is 2 MB) under *Serial Flasher Config*.

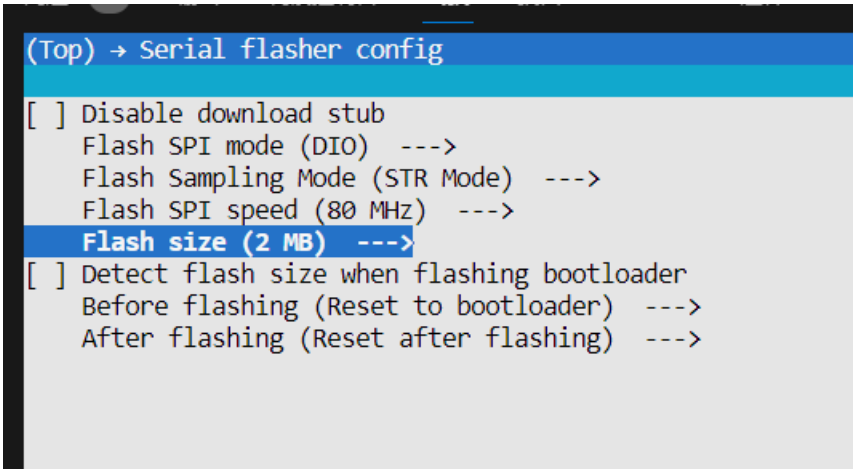


Figure 3: Serial flasher config of idf.py menuconfig.

6. Carefully check the **short-circuit wire** on the board and ensure it remains in place. Do not remove it.

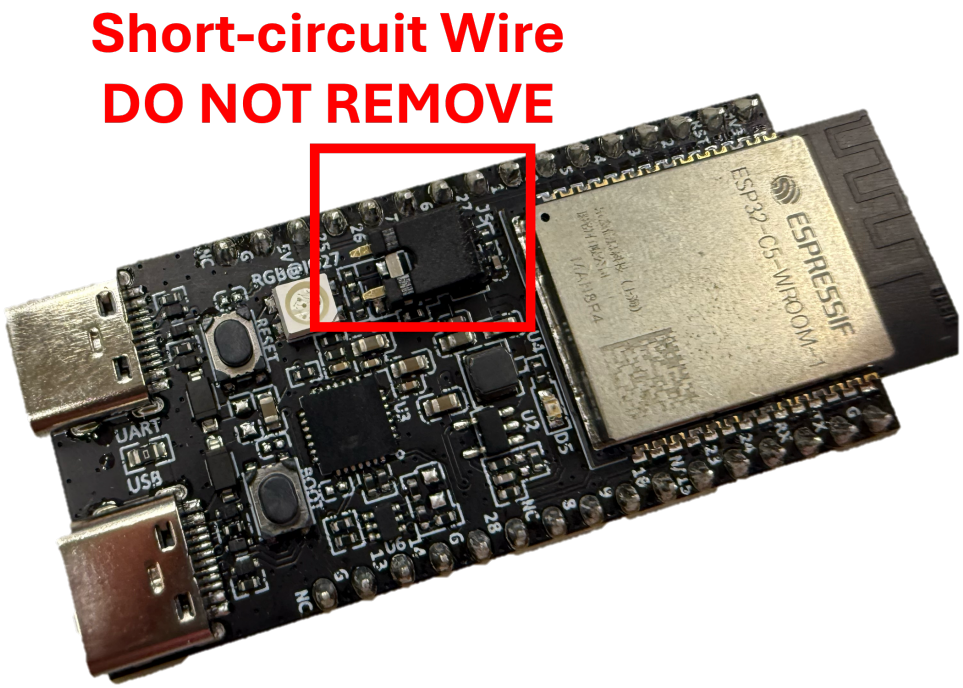


Figure 4: Short-circuit Wire of ESP32C5.

7. ESP32-C5 has two different layouts; however, they function the same for all applications.

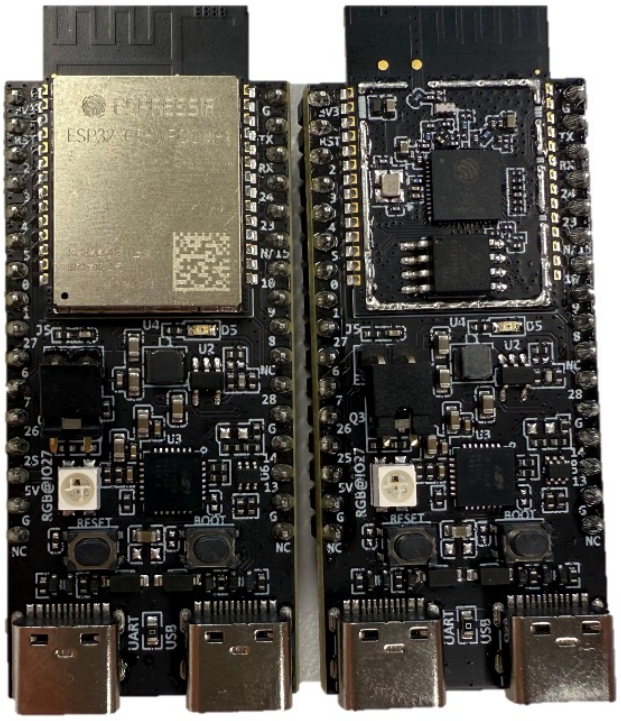


Figure 5: Two different layouts.

8. When flashing, please avoid the register error, which is harmful for the hardware.

2. Code Package

We provide a basic package of source code, located in the `esp32c5/csi_send` and `esp32c5/csi_recv` folders. Both subprojects share a similar folder structure, including `CMakeLists.txt`, `main/`, `main/app_main.c`, `main/idf_component.yml`, and more.

2.1 Implementing Required Tasks You should implement the required tasks in `main/app_main.c` for both the `csi_send` and `csi_recv` folders. Each block starts and ends with the comment:

```
// YOUR CODE HERE
// END OF YOUR CODE
```

Besides that, you can also add other functions at the proper position freely if necessary.

2.2 Compilation and Component Management In addition to `main/app_main.c`, both the manifest file `main/idf_component.yml` and the CMake file `main/CMakeLists.txt` play crucial roles in the compilation process:

- **`main/CMakeLists.txt`**: Manages the technical aspects of building the component.
- **`main/idf_component.yml`**: Provides metadata and dependency information for component management.

To include additional components (such as `esp-dsp`), modify the `main/idf_component.yml` file following YAML syntax (documentation). Additionally, update the `idf_component_register` function in `main/CMakeLists.txt` to ensure proper integration of the new components.

2.3 Diagram of TX `app_main.c` The figure below illustrates the implementation diagram for the TX `app_main.c` file. A key parameter, `CSI_Q_ENABLE`, controls whether CSI data is stored in a buffer `CSI_Q` or sent via a serial port.

- When **`CSI_Q_ENABLE = True`**, the received CSI data is stored in a queue buffer `CSI_Q` and updated using a **First-In-First-Out (FIFO)** mechanism by `csi_process()`. The buffered data can then be sent via MQTT using `mqtt_send()`, allowing two methods of data transmission:
  - **Method 1 (Blue Path - Data: CSI)**: CSI data is sent directly to the PC via MQTT, where on-PC algorithms can be implemented.
  - **Method 2 (Red Path - Data: Results)**: On-board algorithms such as `motion_detection()` and `breathing_rate_estimation()` process the buffered CSI data, and the results are sent to the PC via MQTT.
- When **`CSI_Q_ENABLE = False`**, CSI data is sent through a serial port instead. This setting is useful for verifying whether TX has been successfully flashed and is collecting data. However, it must be set to `True` for other tasks.
- The function `wifi_csi_rx_cb()` handles the reception of CSI data. Based on `CSI_Q_ENABLE`, it either processes the data in a buffer for onboard calculation or outputs it via serial.
- You need to properly implement `csi_process()` to store and analyze CSI data and modify the `mqtt_send()` function to transmit either raw CSI data or processed results.

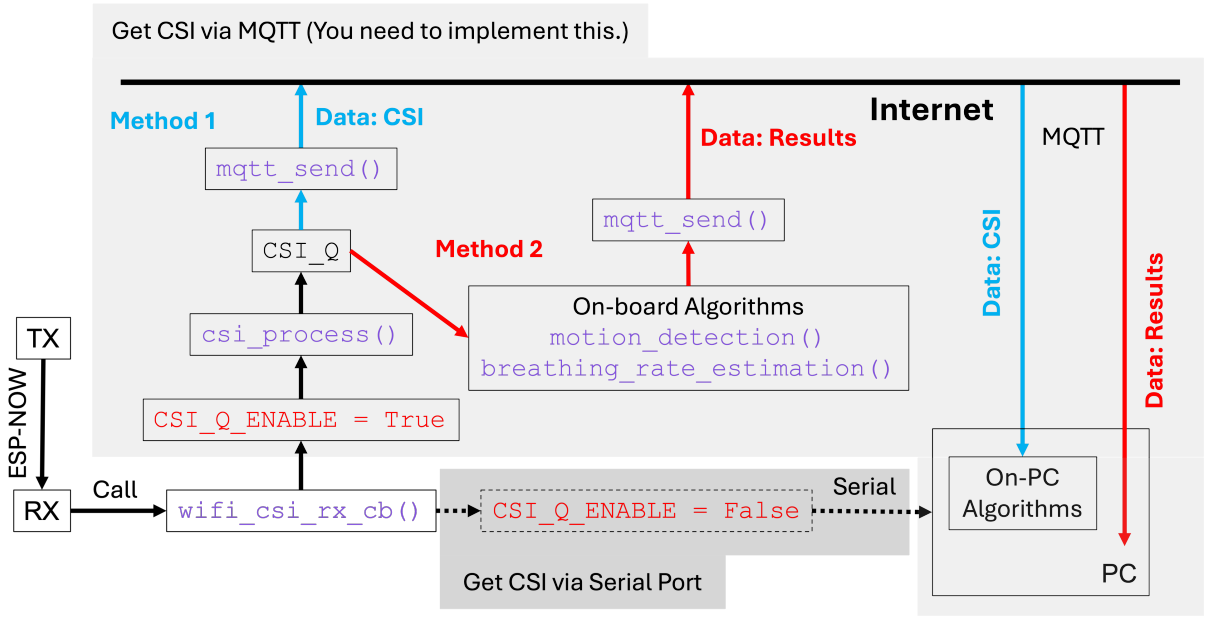


Figure 6: Diagram of functions in TX app_main.c

3. Benchmark Dataset

We provide two benchmark datasets, each designed to facilitate the evaluation of different wireless sensing tasks: motion detection and breathing rate estimation. These datasets contain Channel State Information (CSI) recordings, which serve as the core data source for developing and benchmarking CSI-based sensing algorithms.

The dataset structure is as follows:

```
benchmark
|---- breathing_rate
|    |---- evaluation
|    |---- test
|---- motion_detection
     |---- evaluation_motion
     |---- evaluation_static
     |---- test
```

Each dataset is split into two parts:

- **Evaluation Set**: Labeled data used to benchmark algorithm performance.
- **Test Set**: Unlabeled data reserved for performance validation, contributing 20% of the overall task assessment.

3.1 Motion Detection Dataset The benchmark dataset for CSI-based motion detection is stored in the `benchmark/motion_detection/evaluation_motion` and `benchmark/motion_detection/evaluation_static` directories. Each folder contains CSI files labeled according to their respective categories, representing either **motion** or **static** states.

1. The evaluation dataset comprises CSI files categorized into motion and static states, forming a binary classification problem.
2. Our benchmark method achieves **100% accuracy** on the labeled dataset (`evaluation_motion/` and `evaluation_static/`), effectively distinguishing between motion and static states. The accuracy can be formulated as:

$$\alpha = \frac{N_{\text{correct}}}{N} \times 100\%,$$

   where (N_{correct}) and (N) represent the number of correct predictions and the total number of samples in the `test` folder, respectively.
3. The CSI files in `benchmark/motion_detection/evaluation_motion` and `benchmark/motion_detection/evaluation_static` are strictly for evaluating motion detection algorithms. Additionally, unlabeled CSI files in `benchmark/motion_detection/test` will be used for performance assessment, contributing **20% of the task**.

3.2 Breathing Rate Dataset The benchmark dataset used for CSI-based breathing rate evaluation is located in the `benchmark/breathing_rate/evaluation` folder. It consists of CSI files paired with their respective ground truth files, where the ground truth BPM values are obtained using a breathing belt. Each BPM value is computed from 15-second intervals and updated approximately every second via queue mechanism (FIFO).

1. The evaluation dataset comprises the following CSI files and their corresponding ground truth files:

CSI File	Ground Truth File
CSI20250227_193124.csv	gt_20250227_193124.csv
CSI20250227_191018.csv	gt_20250227_191018.csv

2. The benchmark employs **Mean Absolute Error (MAE)** as the primary evaluation metric. MAE quantifies the absolute difference between the estimated and actual BPM values, providing an accurate measure of model performance. The formula of MAE can be expressed as:

$$e_{\text{MAE}} = \frac{1}{N}\sum_{i=1}^{N}|BPM_{\text{pred},i} - BPM_{\text{gt},i}|,$$

where (BPM_{pred, i}) and (BPM_{gt, i}) represent the predicted and ground truth BPM values for the (i)-th sample, respectively.

3. Our baseline approach yields a **median MAE $\tilde{e}_{\text{MAE}}$ of 0.94 BPM**, demonstrating its effectiveness in breathing rate estimation.
4. The evaluation files under `benchmark/breathing_rate/evaluation` is only employed to evaluate your algorithm. The CSI files without labels under `benchmark/breathing_rate/test` will be considered as the performance evaluation marks, contributing **20% of the task**.

Evaluation Protocol:

Task 1: CSI Collection (10 points)

Before starting this task, ensure that all necessary toolchains are installed. You can refer to the startup tutorial on Moodle for guidance. In this task, you will build and flash the firmware onto the board to verify that the transmitter successfully transmits and the receiver successfully receives the CSI data.

Criteria:

- You should include screenshots in their report demonstrating the successful build and flash process for both TX and RX.
- The `build` folders under `csi_send` and `csi_recv` must exist and contain files (i.e., they should not be empty).

Task 2: Motion Detection (20 points)

Implement motion detection using CSI data to identify movement within the environment. The expected output is a boolean value: 1 for motion detected and 0 for no motion. The evaluation metric is accuracy.

Criteria:

- You should provide an algorithm for motion detection.
- You should include a description of the algorithm's implementation and performance in the technical report.

- If the team chooses to implement the algorithm in real-time or on-board, an additional video demonstrating the functionality must be provided. The video must include all student ID cards for verification.

Task 3: Breathing Rate Estimation (30 points)

Develop an algorithm to estimate and monitor an individual's breathing rate using CSI data. The expected output is an **integer** in beats per minute (BPM), rounded to the nearest whole number. The evaluation metric is the median mean absolute error (MAE).

Criteria:

- You should provide an algorithm for breathing rate estimation.
- You should include a description of the algorithm's implementation and performance in the technical report.
- If the team chooses to implement the algorithm in real-time or on-board, an additional video demonstrating the functionality must be provided. The video must include all student ID cards for verification.
- In your report, for each breathing rate test file, you should include a graph that shows how the measured breathing rate changes over time. Additionally, for each file, provide the average breathing rate (in bpm).

Task 4: Data Transmission (20 points)

Set up the **Message Queuing Telemetry Transport (MQTT)** protocol to transmit data from the ESP32-C5 to a host device (e.g. laptop).

Criteria:

- You should successfully transmit the estimated breathing rate and/or raw CSI data via MQTT.
- You should include screenshots or logs demonstrating the MQTT message flow in their technical report.
- If the team is unable to implement the algorithm on-board, they must transmit some alternative information (e.g., teammates' UIDs and CSI for offline computation) via MQTT. Partial points will be awarded for this approach.

Task 5: Data Visualization (10 points)

Develop an end-to-end system that visualizes your results. This can be a Web App, a desktop application or a mobile app. Try to visualize your results, including the intermediate results of your algorithm. It

resembles a dashboard for the user to interact with.

Submission

Each team needs only one member to submit the `.zip` file to Moodle. The file should contain:

1. **Contribution Statement**:

   Each member in the group must clearly state their individual contributions, as this will be considered in grading. Each member's contribution should be specified in percentage form to ensure fair assessment.

2. **Technical Report**

   The technical report must clearly document your implementation and performance analysis. The maximum length is **5 pages** (including the first page), with a font size **no smaller than 12 pt**, in **A4 or US Letter format**. A standard LaTeX template will be provided. You should not modify the first page format in the template—only fill in your results and team information. Any content exceeding this limit will not be considered. **Only PDF submissions will be accepted.**

3. **Implementation**

   The submission must specify the location of the algorithm implementations. All **code** and the **`build` folder** for both **TX and RX** must be included.

4. **Video (Optional)**

   If the team wishes to demonstrate an on-board or real-time implementation, a video may be uploaded as part of the submission. The video must begin by displaying **all student ID cards**. The maximum length is **2 minutes**, and any content exceeding this limit will not be considered.

Bonus

We plan to hold an **in-class demo and competition** at the end of the semester. The group project should be designed with this objective in mind. **Finalists may receive up to 5 bonus points** for a live in-class demo, which will be graded as follows:

1. Participation in the live demo session. (1 pt)

2. The system can successfully run and function for motion detection (regardless of the performance). (1 pt)
3. The system can further work for breathing estimation. (1 pt)
4. The system achieves accurate and robust performance. (1 pt)
5. The overall very best demo(s). (1pt)