Internet of Things - Intelligent and Connected Systems

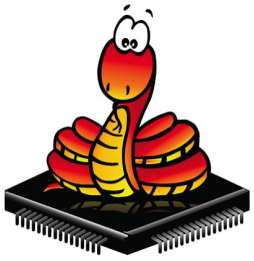
EECS E4764

Columbia University

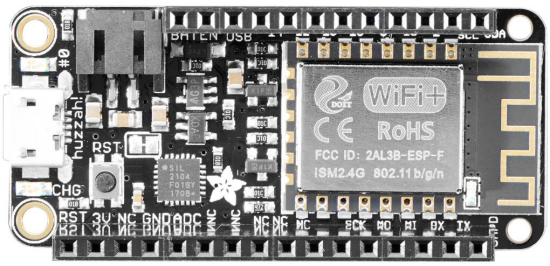
Prof. Xiaofan Jiang

Fall 2022

Lab 6 - Cloud Server, Databases, Visualization, and Putting it All Together



MicroPython



Introduction

When devices respond to changes in the environment or read in inputs from the physical world data is generated. We can create systems or execute functions based on this data to achieve a desirable output or result. Sometimes we need to collect large amounts of data for long periods of time from a large variety of sources. In these cases it would be in our best interest to store our data in an organized fashion so that we can later process and analyze our dataset. In cases like these, it is very useful to use a database on the cloud.

For the first part of this lab, we will be setting up a cloud server on Amazon Web Services (AWS), on which we set up a MongoDB database service. Database services like MySQL, provide a table-based relational database that can be accessed through the SQL language, which can be rigid and inflexible for certain applications. MongoDB is based off of NoSQL, which essentially gets rid of the rigid tabular structure of SQL, allowing the user to set up any organizational structure (though the two structures are actually very similar in their core). Since the ESP8266 does not have an operating system, and thus cannot install any of the existing database software, we will host the database on the Amazon AWS server, and access it using HTTP commands on the ESP8266. We can store almost anything on a database, and will focus on storing sensor values from our smartwatch in this lab.

In the second portion of this lab, we will add gesture recognition functionality to our watch, based on accelerometer data. We will produce some training data first, label the training data, and train a model to classify letters “C”, “O”, “L”, “U”, “M”, “B”, “I”, “A”, then deploy the model onto the cloud to enable real-time classification of the letters.

Finally, we will integrate everything we did in previous labs together to finish our smartwatch.

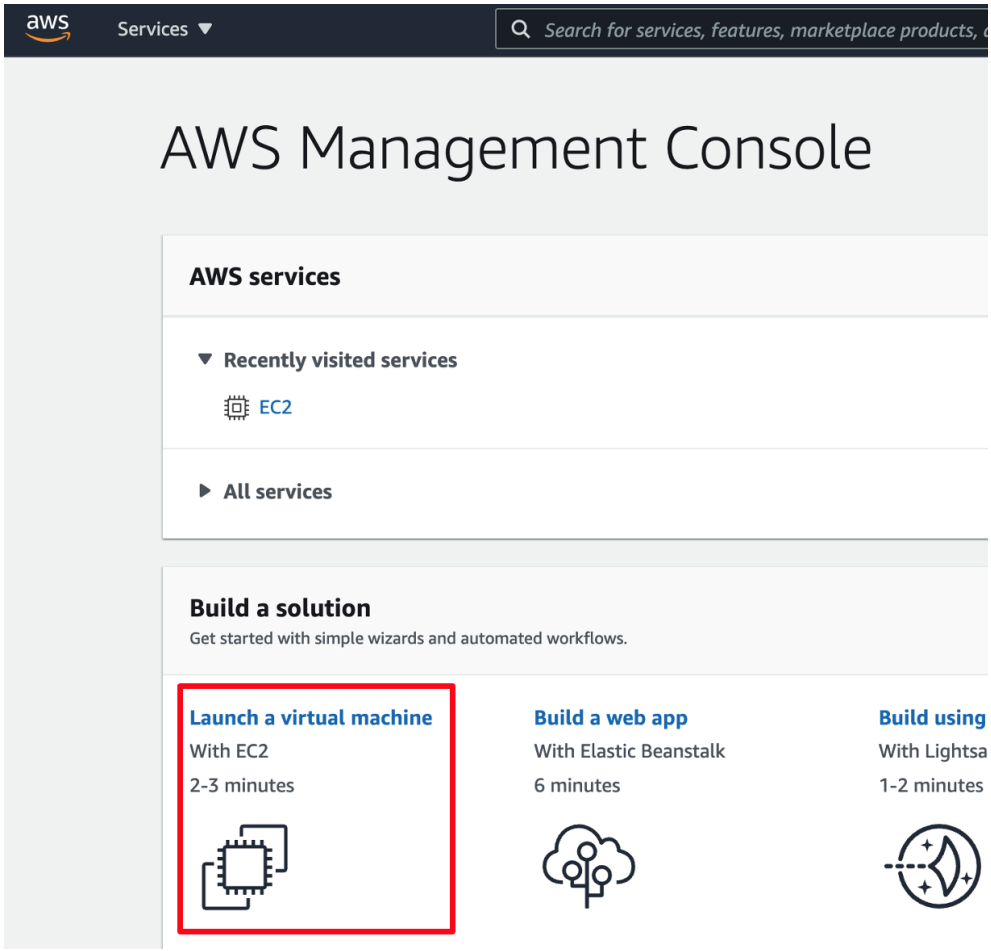
Part 1: Set up AWS Server

Task 1

Register an Amazon Web Services (AWS) account on https://aws.amazon.com/ and log into the AWS management console. Launch an EC2 Linux server instance following the instructions below.

More information on how to set up an EC2 instance can be found at:
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html

1. Click on “Launch a virtual machine with EC2”.



2. Select Ubuntu 18.04 under the list of system images.

3. Choose instance type: t2.micro.

1. Choose AMI | 2. Choose Instance Type | 3. Configure Instance | 4. Add Storage | 5. Add Tags | 6. Configure Security Group

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance families | Current generation | Show/Hide Columns

Currently selected: t2.micro (- ECUs, 1 vCPUs, 2.5 GHz, -, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
☐	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate
☑	t2	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
☐	t2	t2.small	1	2	EBS only	-	Low to Moderate

4. Use the default configurations for instance details, storage and tags (step 3~5).

5. Configure security groups (step 6) to allow traffic.

Make sure to create a security group for your instance that at least allows the server to accept any inbound HTTP, and SSH requests from Columbia IP addresses. The easiest way to do this is to allow any IP address to connect to your server via these methods by inputting “0.0.0.0/0” into the source fields, though in practice you may not want to do this for security reasons.

Notice the Port Range. Make sure the security rule allows traffic from the port that the web server you will be creating uses. If you select “HTTP” it will default to port 80 -- this means your web server should listen to port 80. You can also use a custom port by selecting “Custom TCP Rule”.

Click “Review and Launch”.

Step 6: Configure Security Group

Assign a security group: ◉ Create a **new** security group

○ Select an **existing** security group

Security group name: launch-wizard-2

Description: launch-wizard-2 created 2021-10-26T21:31:55.420-04:00

Type	Protocol	Port Range	Source	
SSH	TCP	22	Custom	0.0.0.0/0
HTTP	TCP	80	Custom	0.0.0.0/0, ::/0
Custom TCP F	TCP	A CUSTOM PORT	Custom	0.0.0.0/0, ::/0

Add Rule

⚠ Warning

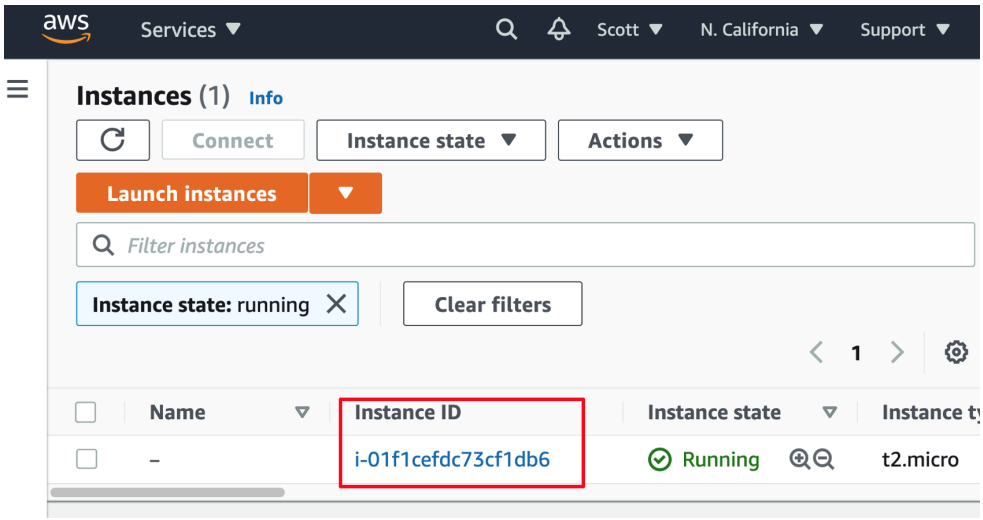
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

6. Create credentials for SSH

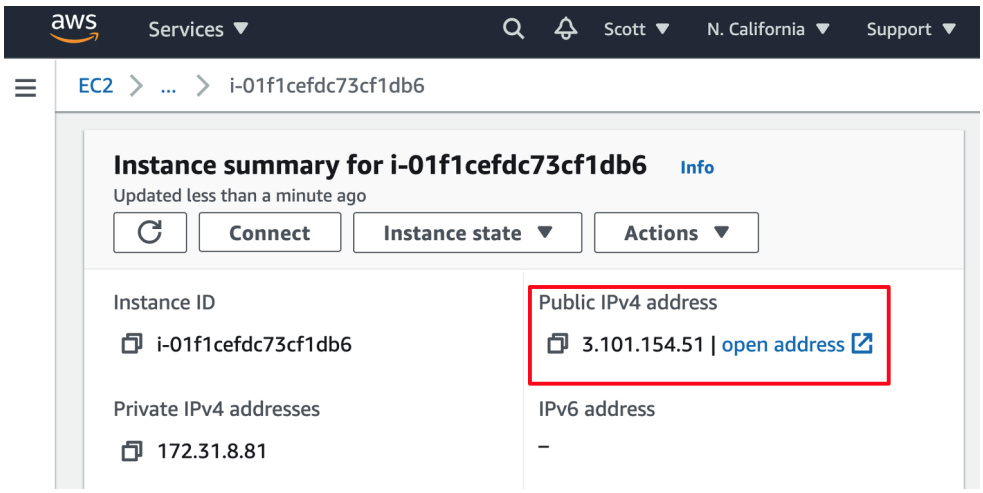
After reviewing your settings, click “Launch” on the bottom right corner. You will be asked to create a key pair. Download the key pair and save it securely. You will need this file to log into your server. Note that you will not be able to download this file again. **You should never push this file onto git or make it public anywhere.**

7. Log into EC2

Go back to your AWS Management Console, and on top left, click *Services - All Services - Compute - EC2*. Then, click on “Instances (running)”, and find the instance you just launched.



Click on the instance ID, and find your server’s public IP address.



Side note: Remember to terminate the instance at the end of the semester once we are done with the class, to avoid unwanted charges!!

Task 2

Using the IP address and the .pem key, log into the EC2 instance using SSH. You can use tools like Termius, PuTTY, or Terminal, but we also recommend setting up VSCode SSH as it not only can act as a terminal, but also allows you to remotely edit / run codes very conveniently.

The username should be "**ubuntu**". Depending on the SSH tool you are using, you will either have to give it the .pem file, or copy the content of the .pem file to the tool in the "private key" field.

Once you get access to the terminal on the cloud server, set up MongoDB following the instruction here:
https://docs.mongodb.com/manual/administration/install-on-linux/

Task 3

From now on, you will be working on Python code remotely on the EC2 server. Check the Python installation on the EC2 server by typing

```
python3 --version
```

Note: You will need to use "python3" instead of "python" on the EC2 server because the default "python" is Python 2.7.

Now, let's install the two libraries we will be using later.

```
python3 -m pip install Flask pymongo
```

We will use Flask to create a web server to allow ESP to interact with the cloud. If you haven't worked with Flask before, here is a simple guide.
https://pythonbasics.org/flask-rest-api/

Run the flask server and test with your browser to see if you can connect to it. Notice that if you close the SSH application, your program will automatically shutdown instead of running in the background. You can use a tool called "screen" to keep the program running after the terminal is closed.
https://linuxize.com/post/how-to-use-linux-screen/

Task 4

Create one RESTful interface on your Flask server to receive accelerometer data from ESP, and insert them into MongoDB. You can use pymongo to connect and insert data into MongoDB. (In the next part for gesture recognition, you will create an additional endpoint to retrieve the data from MongoDB and make a prediction).

Create a MicroPython program on ESP that will send accelerometer data to the AWS database at a frequency of at least 10 Hz. You can use *urequests* library to interface with the Flask endpoint.

Checkpoint 1

Create a server on AWS (cloud) and send continuous accelerometer data of at least 10Hz. For the demo, print the data received onto the terminal as it is being inserted into the database.

Part 2: Gesture Recognition

In this part, you will output letters in the word "COLUMBIA" with your accelerometer gestures.

Task 1

Before you start training the gesture recognition model, you need to collect some labelled data from the accelerometer.

You can write a simple program on ESP, so that once you click a button, it reads and prints the accelerometer data at a constant rate for a fixed amount of time. You should choose wisely your sampling frequency and duration -- the sampling frequency should be high enough so it captures your motion well but not too high so it's a lot of data to send to the cloud; the duration should be roughly the same as the time you take to "write" one letter.

You should also consider printing the data in a csv format, so you can directly paste the data into a file and import the data using Pandas.read_csv.

For the data collection, you should be repeating the same motion for one of the latters, save the accelerometer data into a file and name the file with the letter you wrote. Then, repeat the same task for other letters.

Task 2

Now, with the labeled data you've collected, train a classification model. If you are already familiar with machine learning, you can do the usual steps you take. The following procedures are just for reference.

The easiest way to set up the environment is to use Anaconda. Anaconda gives a complete package that we are going to use (Jupyter Notebook, Pandas, Numpy, Scikit-learn, etc).
https://www.anaconda.com/products/individual

Jupyter notebook is the tool that most people use for data science to run code and visualize plots. If you haven't used it before, here is an introduction
https://www.youtube.com/watch?v=2WL-XTl2QYI

Before you continue, check out this video to have a general idea on the process of training a classification model.

https://www.youtube.com/watch?v=J5bXOOmkopc

First, you will need to load your data from your file into a Pandas DataFrame or Numpy Array.

Note: Consider the consistency of the data from part 1.

For exploratory data analysis, you can use the Pandas and Numpy library.
https://pandas.pydata.org/docs/getting_started/intro_tutorials/01_table_oriented.html
https://www.kaggle.com/kashnitsky/topic-1-exploratory-data-analysis-with-pandas

Next, you will do preprocessing and training a machine learning model, the easiest way is to use the scikit-learn library.
https://scikit-learn.org/stable/tutorial/basic/tutorial.html

What you usually need to do for preprocessing is normalizing the data (MinMaxScaler or StandardScaler).
https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02

IMPORTANT: Do not forget to split the data for training and testing sets **before** the training process.
https://realpython.com/train-test-split-python-data/

The challenge is how you will format your input vectors and the corresponding labels. One way to do this is by generating a time series of accelerometer axes data, hence you will use those as the feature vectors.

For model training, because this will be a multi-class classification, you can use RandomForestClassifier.
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

After doing the training, you will evaluate your model. Because this will be a multi-class classification, you will need to check accuracy and also confusion matrix not only the overall but also for each class.
https://scikit-learn.org/stable/modules/model_evaluation.html

If you are not yet satisfied with the evaluation results, what you can do is one of the following:

1. Think of how you can make your gesture more “distinguishable”, and recollect some data (Part 1)

2. Hyperparameter tuning (https://scikit-learn.org/stable/modules/grid_search.html)

Then, you can retrain the model and reevaluate until you feel confident.

When you are already convinced with your model, **you will then save the model** for later prediction.
https://scikit-learn.org/stable/modules/model_persistence.html

End-to-end process example of machine learning using titanic dataset for reference:
https://towardsdatascience.com/predicting-the-survival-of-titanic-passengers-30870ccc7e8

(Note: Any gesture that can clarify each letter is acceptable, you don't have to make gestures similar.)

Checkpoint 2
Train a classifier on the AWS server that can recognize any letter in the word "COLUMBIA".

☑ Task 3

Add a new endpoint in AWS Flask Web Server that receives data from Huzzah and then return the predicted value.
https://towardsdatascience.com/a-flask-api-for-serving-scikit-learn-models-c8bcdaa41daa

In this endpoint, you will load your saved model from Task 1.
https://scikit-learn.org/stable/modules/model_persistence.html

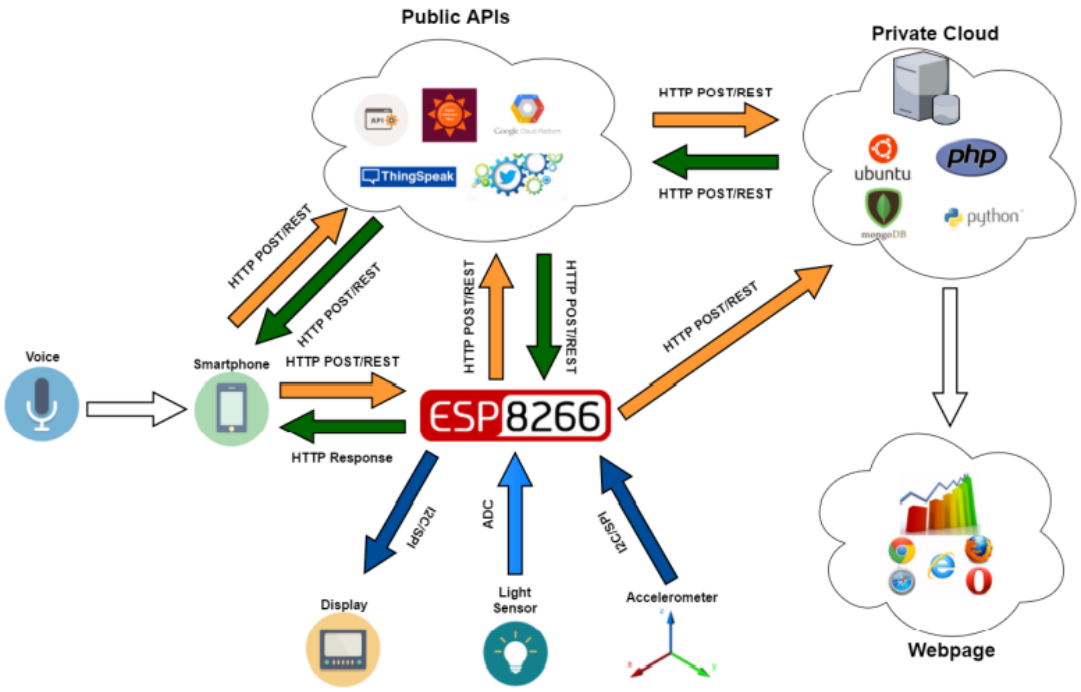
☑ Task 4

Create a MicroPython program that sends accelerometer data to the **new endpoint from Task 2**, and receive back the letter label from AWS server. You will then output the predicted letter into the OLED screen.

Checkpoint 3
Create a system that will be able to predict the letters from the word "COLUMBIA" based on input accelerometer gestures.

Part 3: Final Integration

We now have all of the individual components that we will combine to form the final version of our smartwatch. The only thing left to be done is to create the system. The diagram and sections below provide a summary of what the final system should contain.



Voice Commands:
The smartphone application and smartwatch should respond to the following voice commands:

- Turn on/off screen(Turn OLED Display ON/OFF)
- Show location
- Display weather: temperature and description (e.g. mostly cloudy)
- Send spoken tweets
- Display current time on smartwatch
- Other recognized voice commands in addition to the above commands which are feasible to implement within the scope of the hardware constraints. (Optional)

Smartphone Application:
You will need to build an application to bring to life all the different features that we want to integrate in our system. Your app should be able to do the following functions at the least to be able to integrate all the features:

- Interface with Google Speech API to translate voice commands into text.
- Send the voice commands to the smartwatch
- Voice command features:
    - While your spoken voice commands will lead to an observable action on the OLED or on Twitter, we would still like to see the voice command translated to Text and visible on the application screen.
    - Your application should also relay the success or failure of a spoken command on the screen. For example, if your voice command was “Display Time”, but due to some error, the system was unable to show the time on the OLED, a failure message should be displayed on the application screen.
- Any other application feature that makes a more immersive experience for the smartwatch consumer. (Optional)

Embedded Server/ESP8266:

- Interface with the OLED display to display the time and allow users to set the time.
- Alarm: Users should be able to set an alarm through the OLED display; once the alarm goes off, a visual and audio notification (using the piezzo) should play until the user interacts with the display again.
- Receive and process voice commands from the smartphone application
- Voice command features:
    - Display weather: After receiving this command from the smartphone, obtain the weather in the area around your geolocation and display it on the screen.

    - Send spoken tweets: After receiving this command, the custom spoken tweet should be tweeted on the account linked to the project. Additionally, display the tweet on the smartwatch.
    - Display the current time on smartwatch: After receiving this command, the watch should switch to the current time menu and display it to the user on the OLED.
- Allow users to display the weather information obtained from the previous weather voice command, through the OLED display.
- Allow users to display the last tweet sent through the OLED display.
- Read values from the light sensor; adjust screen brightness/contrast with respect to the ambient light around. For example, the display should get brighter if the reading of the sensor increases, and it should get dimmer if the reading of the light sensor decreases.
- Sync time on OLED with the internet to show real-world local time.
- Include a gesture recognition mode to recognize any letter in the word “COLUMBIA”. (Optional)
- Use accelerometer data on the smartwatch to automatically orient the screen for optimal display conditions. (Optional)

Cloud Server/Database:

- Receive and store accelerometer data from the smartwatch in a database.
- Setup and train a neural network that can recognize the gestures through the data received from huzzah.

**Notes: The above sections provide an outline of what should go into the final system. By this point, some of you may have experienced one of the constraints of small embedded systems: memory. While putting the system together, be mindful of your implementation to optimize your code to use less RAM. Additionally, strive to think about certain strategies (off-loading tasks from one part of the system to other parts, reducing RAM usage, more efficient implementations, modularization, etc.) that could drastically reduce the amount of memory your program requires; the guidelines above only disclose the features that should be present in the final system, not how to go about implementing them.*

Task 1

Features outlined for each of the subsystems should be functioning independently:

- Voice Commands
- Smartphone Application
- Embedded Server/ESP8266
- Cloud Server/Database

Task 2

All discrete components are combined and fully functioning. The final system should be able to perform the following functions:

- Voice actuated actions:
    - Turn ON/OFF screen
    - Display Time
    - Display location
    - Display weather
    - Send tweet
    - Increase/Decrease Screen Brightness(Optional)
    - Set alarm(Optional)
    - Additional commands(Optional)
- Smartphone application:
    - Speech to text conversion
    - Display spoken command on screen
    - Display tweet on screen(For command to send tweet)
    - Display command success/failure on screen.
    - Have a Menu screen to navigate Modes within the application.(Optional. Having multiple modes on the application might let you achieve more functionalities on the smartwatch with fewer GPIO pins)
- General Functionalities:
    - Sync watch time with actual local time over the internet.
    - Your smartwatch should be able to set an alarm. A visible and audible notification should be triggered when the alarm is set off.
    - Adjust OLED brightness according to ambient light conditions.
    - Optimize smartwatch display based on actual orientation of the hardware system. (Optional)
    - Have multiple modes on your smartwatch. In one of the modes, you are able to perform Gesture Recognition and identify “Columbia”based on accelerometer data without changing or reinserting any hardware or software configuration on the Huzzah or your smartphone application. (Optional)

Checkpoint 4

A complete functioning smartwatch with all components that have been built in the lab.

Lab 6 Checkpoints:

1. Create a server on AWS (cloud) and send continuous accelerometer data of at least 10Hz. Display the data on the cloud through SSH terminal.

2. Train a classifier on the AWS server that can recognize any letter in the word “COLUMBIA”

3. Display the word “COLUMBIA” on the OLED in a total of 10 trials, one letter at a time.

4. A complete functioning smartwatch with all components that have been built in the lab.

Submission:

1. **Submit a zipped file containing all the components that were required for the entire lab6.**

The submitted file should be named like

```
labx_{member1_uni}_{member2_uni}_{member3_uni}.zip
```

There should be 1 file for this lab.
e.g.

- Lab6_mz2878_eg3205_jw4173.zip