CG4002 Embedded System Design Project

January 2021 Semester 2

"Dance Dance"

Final Design Report

Group 15	Name	Student #	Sub-Team	Specific Contribution
Member #1	Cheong Ray Mun, Nicholas	A0180280M	Hw Sensors	Section 2.1, 2.5, 2.6 Section 3.1 Section 6.2 Section 7.2
Member #2	Chu Jui Hung @Jefferson Chu	A0182590B	Hw FPGA	Section 3.2 Section 2.2, 2.6 Section 6.2 Section 7.2
Member #3	Cheng Weixuan	A0183254E	Comms Internal	Section 4.1 Section 2.1 Section 1.1 Section 6.2 Section 7.1, 7.3
Member #4	Lim Si Ying	A0188693H	Comms External	Section 1.1 Section 2.3 Section 4.2 Section 6.2 Section 7.2
Member #5	Nguyen Ngoc Linh Chi	A0170767W	Sw Machine Learning	Section 1.2 Section 5.1 Section 6.1, 6.2 Section 7.1, 7.2
Member #6	Cheng Xianhao	A0167000X	Sw Dashboard	Section 1.1, 1.3 Section 5.2 Section 6.1, 6.2 Section 7.1, 7.2

Table of Contents

Section 1 System Functionalities

The system is designed to identify modified dance moves from the hit song "Dynamite" by South Korean boy band BTS, released on August 21, 2020. There are three main deliverables for this system:

1. Wearable device that senses each dancer's position and dance move
2. Hardware accelerator that performs dance move and position analytics on each dancer
3. Software dashboard that displays analytics of each dancer

The system is designed with dancers with mind. It targets 4 main groups of users:

1. Individual dancers
2. Team dancers
3. Dance instructors
4. Dance judges

1.1 Use Case

1.1.1 Use Case Diagram

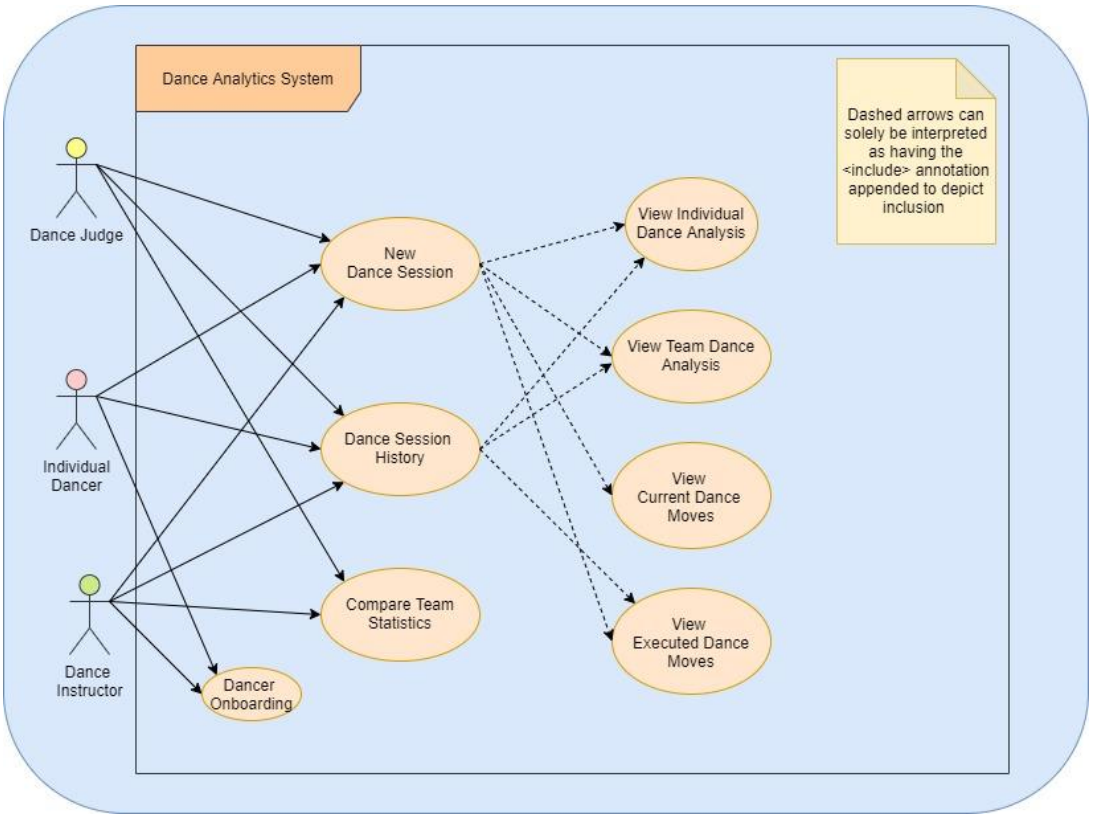


Figure 1.1.1a Use Case diagram

1.1.2 Use Case Descriptions

Use Case	UC01 - Dancer Onboarding	
Actors	Individual Dancer, Dance Instructors	
Pre-condition	User has access to the dashboard	
Post-condition	Dancer is added to the system	
Main Scenario	S/N	Activity
	1	User clicks Dancer from the side navigation menu

	2	User clicks Add Dancer from the main screen
	3	User input new dancer's profile (name, gender and profile picture) and wearable information (wearable model and identifier)
	4	User clicks Create Dancer
	5	New dancer is added to the list of dancers in the main screen Use case ends
Extension	4a	User did not input mandatory information (name, wearable identifier) i. Create Dancer button is disabled Use case resumes from step 3
	4b	Wearable identifier is not unique i. System prompts user that the wearable identifier is already in used Use case resumes from step 3

Use Case	UC02 - View Team Dance Analysis (Synchronize Delay) via Dashboard	
Actors	Multiple dancers (Users connected to the same dance system)	
Pre-condition	User has access to the dashboard	
Post-condition	Dancer is added to the system and is wearing wearable device that has been turned on	
Main Scenario	S/N	Activity
	1	Dancer begins dance move
	2	Dancer completes dance move
	3	Data packet with sensor information is sent to laptop intermediary
	4	Laptop intermediary takes note of time stamp and sends it embedded in the data packet to the Ultra96
	5	Ultra96 receives packet and notes down timestamp for when packet is received

	6	Ultra96 sends back acknowledgement and notes timestamp for when packet is sent
	7	Laptop intermediary receives packet and sends back acknowledgement with timestamp of when packet was received embedded in it
	8	Ultra96 uses timestamps to calculate synchronize delay
	9	Ultra96 sends delay time to the dashboard server
	10	Dashboard server updates dashboard accordingly with insightful and actionable data
Extension	4a	Packet sending failure occurred resulting in retry since TCP is used for communication
	6a	
	7a	

Use Case	UC03 - View Individual Dance Analysis	
Actors	Dancer	
Pre-condition	Dancer has access to the dashboard	
Post-condition	Dancer is added to the system and is dancing while wearing wearable device that has been switched on	
Main Scenario	S/N	Activity
	1	Dancer performs dance move
	2	Sensor collects data and sends to Ultra96 for dance move classification processing
	3	Ultra96 sends sensor data and dance move classification result to dashboard server
	4	Dashboard server processes data
	5	Dashboard server produces insightful and actionable suggestions to dancers

Use Case	UC04 - New Dance Session	
Actors	Individual Dancer, Dance Instructor, Dance Judge	
Pre-condition	User has access to the dashboard	
	Participating dancers have been onboarded (see UC01)	
	Participating dancers wear assigned wearables	
Main Scenario	S/N	Activity
	1	User clicks Session from the side navigation menu
	2	User clicks New Session from the main screen
	3	User selects dancers participating in the session
	4	User input participating dancer's initial position
	5	User input participating dancer's expected dance moves
	6	User clicks Start Session
	7	Initial analytics screen is displayed
	8	Participating dancers start dancing
	9	Analytics screen is updated with real-time analytics (see UC02, UC03)
	10	Steps 8 - 9 are repeated until user clicks on End Session
	11	Analytics screen displays aggregated analytics Use case ends
Extension	5a	User did not input participating dancer's expected dance moves Use case resumes from step 6 but "expected dance moves" section will not be displayed in the analytics screen
	8a	Dancers perform the final logout dance move i. System prompts user to end session ii. User clicks on End Session Use case resumes from step 11

1.2 Features

Accurately identify dance moves
Detect and isolate the relative locations of 3 dancers (detect when the dancer is moving, not dancing and compute the relative position)
Identify dance moves (only get data when dancer is dancing into the model) while maintaining low power consumption
Compute sync delay between dancers
Detect the dancer's muscle fatigue
Affordable system and components

1.3 User Stories

As a ...	**I want ...**	**So that I can ...**
Dancer	A wearable that is non-invasive and lightweight	Dance without being limited in my movement by the wearable
	To be able to review past dance move that I was poor at	Work on those dance steps to improve myself
	To know how I can improve a dance move effectively	Work on those dance steps easily
	To know how my team members are performing	Better understand if anyone in my team requires assistance
Dance Instructor	To be able to identify students who are performing the wrong dance routine	Correct them in time
	To be able to identify students who are in the wrong position	Correct them in time
	To be able to quickly identify how a student can improve his/her	Attend to more students

	dance routine	
	To be able to gauge my students' fatigue level	Know when to call for breaks
	To review my students' past performance	Gauge if the student has improved
Dance Judge	To be able to view the aggregated performance of each team	Determine which team is better easily

Section 2 Overall System Architecture

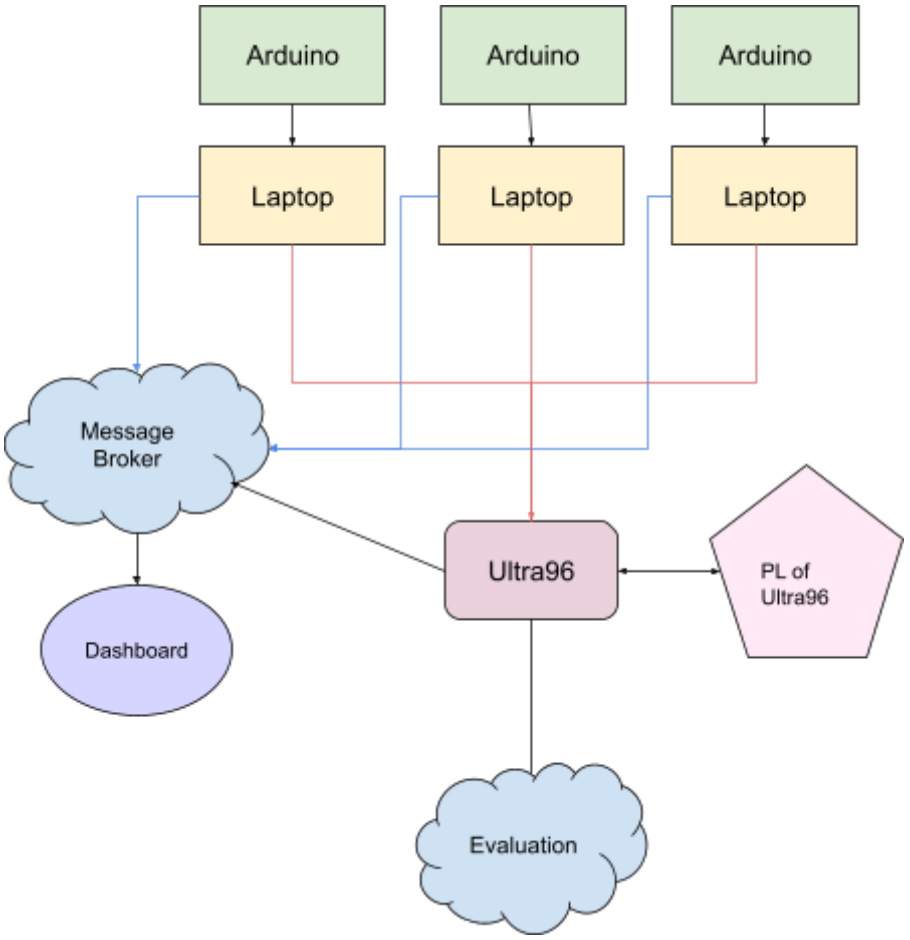


Figure 2a Architecture of System

The figure above is a brief overview of our architecture system. Firstly, the *Beetles* will forward sensor readings & data to laptops. Laptops/*Beetles* will perform some simple processing of the data before forwarding it to the *Ultra96-V2* server. Laptops will also send raw data to the dashboard server. Subsequently, the *Ultra96-V2* server will then run the necessary algorithms to meet the requirements of the capstone project, accelerated with the *FPGA*/Programmable Logic on the *Xilinx Zynq UltraScale+ MPSoC ZU3EG SBVA484*.

2.1 Architecture Diagram for Wearable

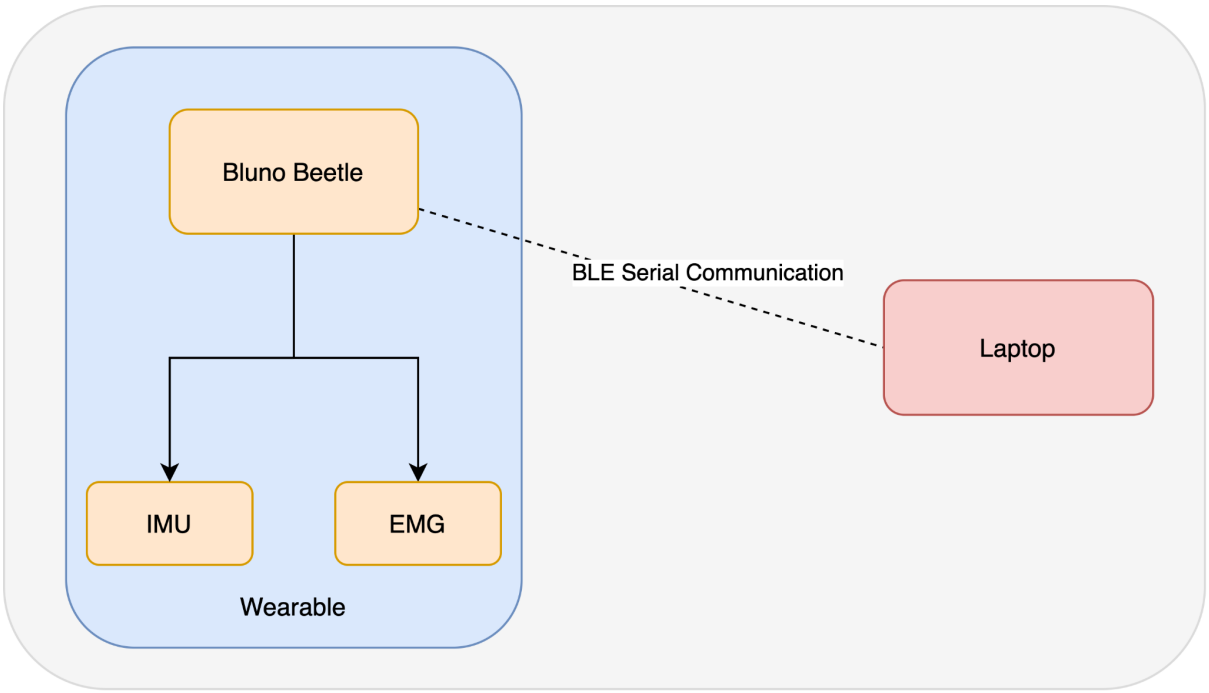


Figure 2.1a Architecture of Bluno Beetle X laptop (internal comms)

Each wearable has hardware consisting of a single Bluno Beetle unit with an IMU and EMG, and would be connected to a laptop via BLE connection during system runtime.

On system start up, the laptop would initiate the handshaking protocol to establish the BLE connection for serial communication of sensor data.

Following the handshake, the beetle will poll the sensor data from the IMU and the EMG respectively. Data received from the sensors will be filtered before performing basic position detection such as detecting when the dancer is idle, dancing, moving left or moving right. Data gathered from the EMG sensor, along with further analysis done on the Arduino side, is used to analyse the fatigue level of the dancer.

Once the data has been processed, it is packaged within a data packet of a format agreed upon between laptop and beetle. Data packets are sent to the laptop during system runtime via BLE serial communication, with the laptop acting as the central device and the beetle as the peripheral device.

2.2 Architecture Diagram for Ultra96

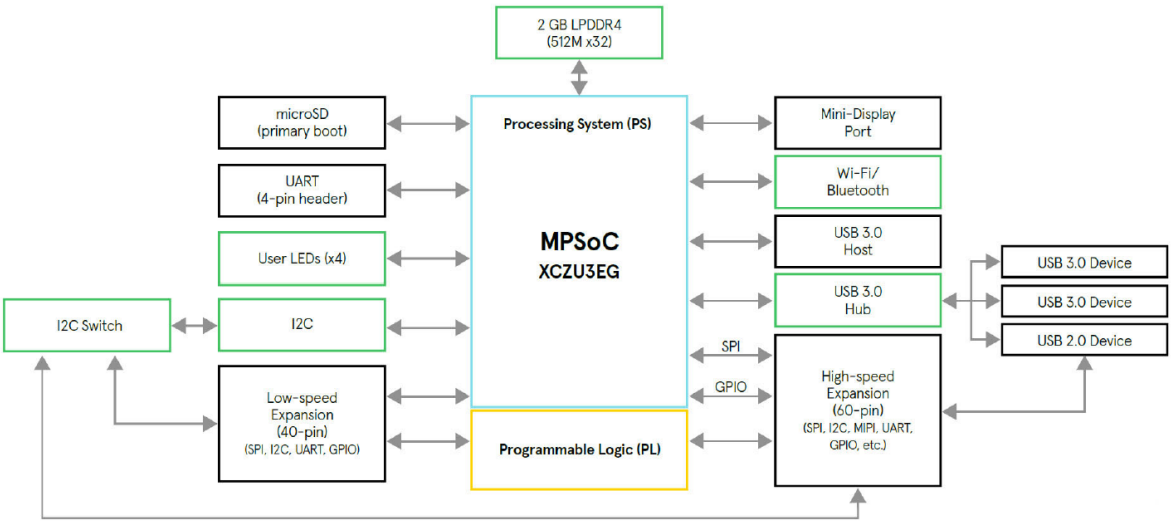


Figure 2 – Ultra96-V2 Block Diagram

Figure 2.2a Architecture of Ultra96-V2

The above diagram is the overview of the *Ultra96-V2* board we have at hand, taken from *Avnet* which is the company that sells this particular board. For the processing system, it is preloaded with an image provided by *Avnet* called *Ultra96-PYNQ*. *PYNQ* is built on top of *Linux* and offers some easy to use functionalities which are unfortunately irrelevant to our project.

On our *Ultra96-V2*, we will be running 2 tasks.

1. Communication with laptops, dashboard & evaluation server

2. AI algorithms to detect dance moves, accelerated with the Programmable Logic (*FPGA*) on the board

To achieve the aforementioned goals, we will be running our scripts on top of the *PYNQ* (*Linux*) operating system using mainly *Python*.

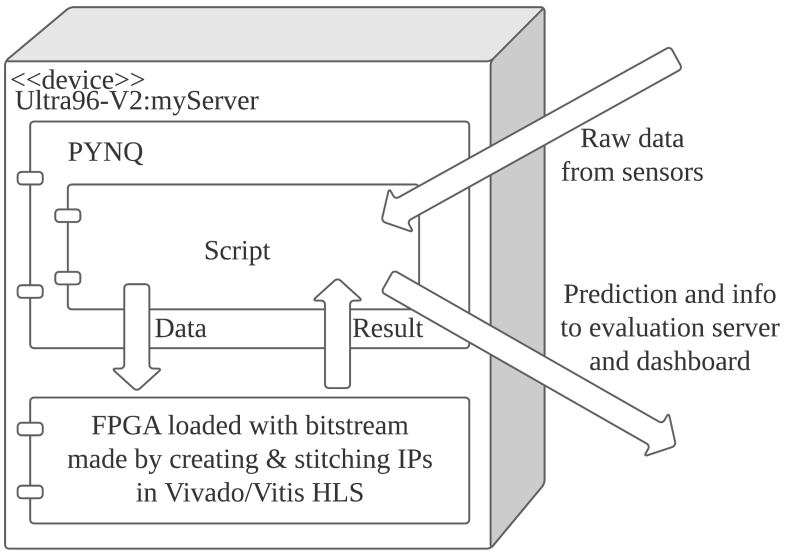


Figure 2.2b Architecture Diagram of Software

As shown in the diagram, raw data from the sensors/*Arduino* will be sent through the laptop using *TCP*. Details of the protocol will be explained further in the external communication section. Next, the server will aggregate the necessary data and feed into the *FPGA*. It will get the prediction from the *FPGA* and send the results to the evaluation server & dashboard.The prediction will contain 3 key pieces of information:

1. Predicted dance move
2. Dancer location
3. Sync delay

Note that not all of the predictions will be done by the *FPGA*, some of the algorithms will be run on the main processor itself and on the *Arduino*.

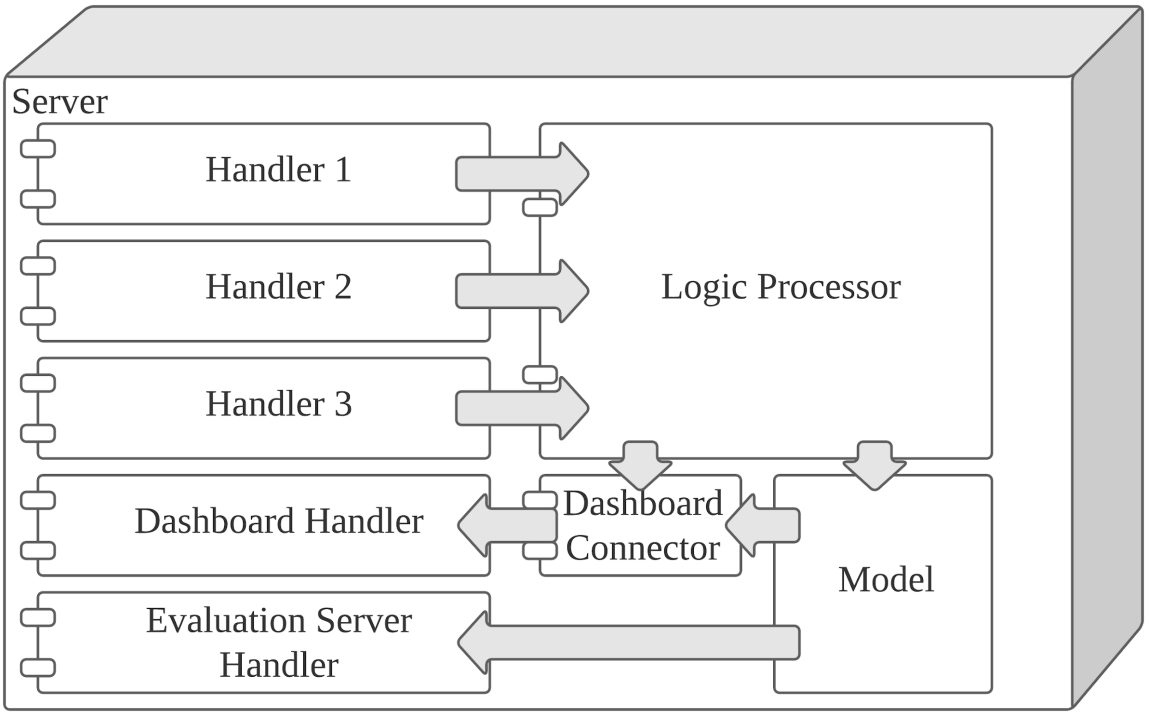


Figure 2.2c Diagram of Software Interaction

Above diagram is how the code works. The code is designed in such a way that there would be a Separation of Concerns to ensure code readability. This is made possible by encapsulation & abstraction concepts from Object-Oriented Programming. Furthermore, we would design our classes (Handler) with the Open-Closed principle in mind so that it is open for extension but closed to modification. Here is a brief explanation of the individual roles of each object.

- Handler 1 to handle incoming data from dancer 1
- Handler 2 to handle incoming data from dancer 2
- Handler 3 to handle incoming data from dancer 3
- Logic Processor to handle logic
- Dashboard Connector to handle data that needs to be shown on dashboard
- Dashboard Handler to handle sending of data to dashboard server
- Evaluation Server Handler to handle sending of data to evaluation server

2.3 Architecture Diagram for External Communications

The architecture design for external communications is as condensed in Figure 2.3a below.

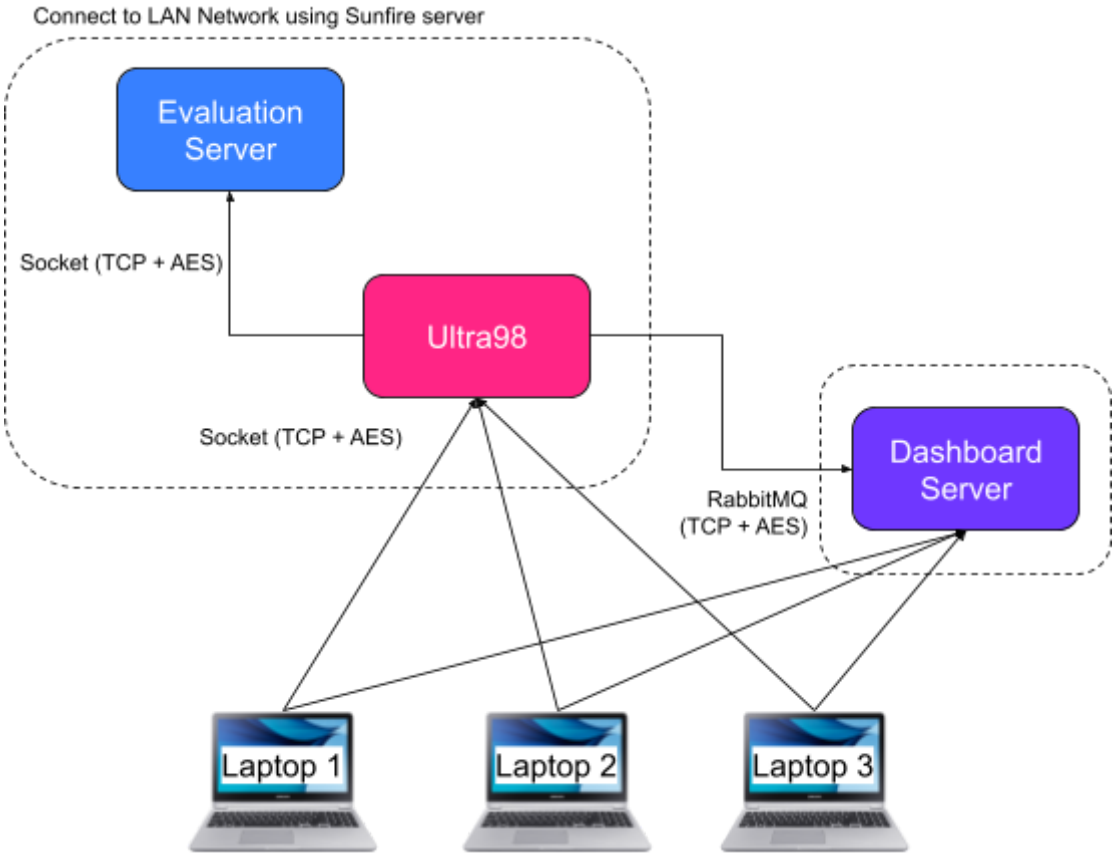


Figure 2.3a. Architecture diagram for external communications

The communication protocol between the Beetles and Ultra96 will make use of an intermediary laptop. This is because the Beetles can only use a bluetooth module and not a wifi module. Thus, we cannot send data over the internet. Furthermore, we will use multihop SSH tunneling to connect from this laptop to the Ultra96. This is because the Ultra96 is on the school's wifi which has a firewall.

2.4 Architecture Diagram for Hardware Components

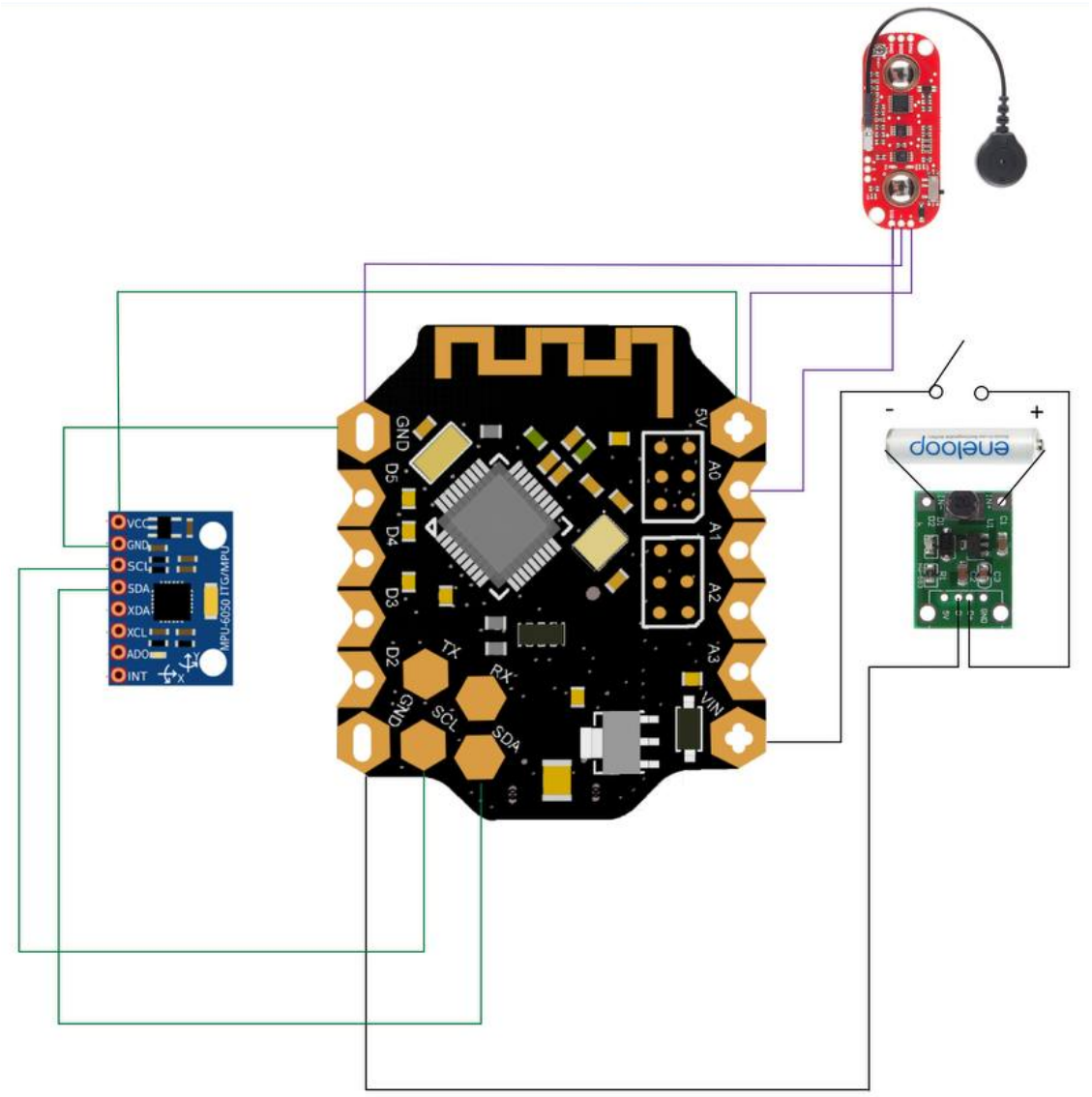


Figure 2.4a Circuit diagram of Hardware components

The above diagram shows the hardware connection between the Beetle and other subcomponents.

2.5 Hardware Design

Figure 2.5a Our wearable design

This section serves to cover the design details of the wearable.

The device is designed to have a profile that is similar to a watch. It is worn on the left wrist of the user. The electronics would be placed on the top of the wrist, in the same plane as the back of the hand. The Electromyography Sensor (EMG) sensor will be attached to the anterior deltoid of the dancer. The reference electrode of the EMG will be placed on an electrically neutral point of the body. The area that we are intending to place it is on the upper chest of the dancer.

The main components of the wearable are an arduino with an integrated BLE module, an inertial measurement unit (IMU) and an electromyography Sensor (EMG) sensor.

The arduino with the integrated BLE module used is the Beetle BLE board which is used to run the firmware of the wearable as well as communicate with the laptop to transmit the data it reads from the connected sensors.

The first sensor that it reads data from is the IMU. It is able to measure the 3-axis acceleration and 3-axis rotational velocity. The measurements are used by the Beetle BLE board to detect arm movements patterns necessary for dance move and activity detection.

The other sensor that it uses is the EMG sensor. This sensor is able to detect and record the electrical signal produced by the muscle as it contracts. The main purpose of including this sensor is to estimate the dancer's fatigue level after the dance moves have been completed by analyzing the electrical signal produced from the dancer's anterior deltoid muscle. The anterior deltoid was chosen as the location for the EMG sensor as it appears to be the most used muscle when performing the dance moves, so it would be able to collect the electrical signals produced by the dancers muscle for every dance move performed.

At the moment, the device is expected to work for around 5 hours.

2.6 Algorithm for Activity Detection

The algorithm for activity detection is proposed to use a spike in sensor readings to determine when the dancer starts his dance move. Upon detecting this spike, the Beetle sends a flag data packet with the local timestamp embedded to the Ultra96. We will use a modified Network Time Protocol (NTP) to calculate synchronize delay between dancers. This is done by exchanging timestamps between the Beetle and Ultra96 and then calculating and dividing by 2 the round trip time, which is further elaborated in Section 5. From the sensor data, we will determine if the dancer is in one of four states: moving left, right, dancing or idle. This will be based on various data thresholds which will need testing to be determined. The general algorithm for activity detection:

1) Determine if dancer is the following 4 states: move left, move right, dancing or idle

2) If dancer is dancing, we used the movement flags describe in section 3.1.4.1 to detect when the dancer starts dancing and cross validate with values from the gyroscope and accelerometer

    a) Compute sync delay using Network Time Protocol

    b) Feed features into model to detect type of dance move

    c) Wait for validation dancer positions from evaluation server and the next set of dance moves

3) Else if dancer is moving, keep track of the dancer's movement over a fixed time frame to ensure consistency.

    a) Compute dancer final positions based on step 3 using a lookup table and current dancer positions.

    b) Wait for dancer to start dancing

4) Else if dancer is idling, do nothing

To detect activity, we would look at the values from the gyroscope and accelerometer. First, we would perform a fourier transform over a window frame of sampled data from the time domain to the frequency domain in order to remove noise. Values that exceed or fall below the desired frequency can then be removed and we transform the data back into the time domain. However, it appears to be too memory intensive for the *Beetles* to compute. One solution to this problem is to forward the data to our more powerful *Ultra96-V2* for computation, however, this would also increase the amount of data sent by the *Beetles* which is undesirable.

Therefore, the solution we are adopting is exponential smoothing. Compared to moving average or average technique, exponential smoothing requires less memory and is much faster.

$$
\begin{aligned}
s'_0 &= x_0 \\
s''_0 &= x_0 \\
s'_t &= \alpha x_t + (1-\alpha)s'_{t-1} \\
s''_t &= \alpha s'_t + (1-\alpha)s''_{t-1} \\
F_{t+m} &= a_t + mb_t,
\end{aligned}
$$

Figure 2.6a Exponential smoothing

The above diagram is a simple explanation of exponential smoothing which is basically a mathematical formula. The mathematical soundness, details and experimental results will be described in the hardware section.

After removing noise from the values, we perform computation of the average delta gyroscope values over a window frame. If the value exceeds 0 by a large margin, there is no dancing activity. This works because of the placement of our sensor. The orientation of the gyroscope will only change when the dancer is dancing and remain relatively stable when the dancer is idle or changing positions.

Once we determine that the dancer is not dancing, we can use the accelerometer values to determine whether the dancer is moving right/left or idling. If a dancer moves to the left, the Z-axis accelerometer values will be positive. If dance moves to the right, the Z-axis accelerometer values will be negative.

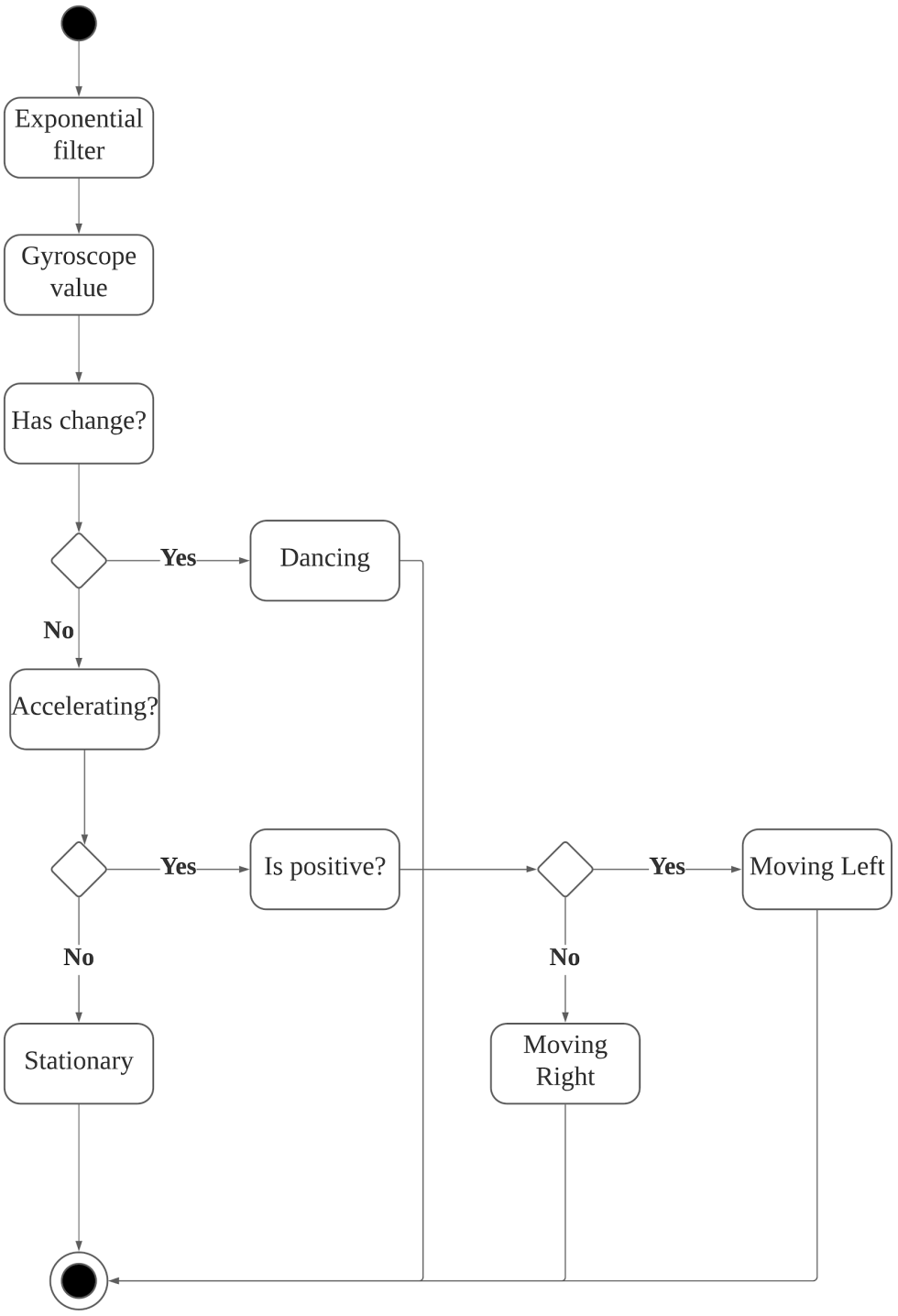


Figure 2.6b Activity detection algorithm

Given that we know the direction in which the dancer moves, we can easily figure out their final positions as we only have 3 dancers in total. Below is a simple pseudo code to figure out where the dancers are currently given that we know which directions the dancers move. It is very simple and only uses an array. There are only 6 possible states and each state can only transform

from 1 state to another state. So we can simply use a lookup table to express all possible transitions.

```
1
2  Tuple {
3    int a;
4    int b;
5    int c;
6  };
7
8  Tuple table[][][] = {t1, t2, t3 ...};
9
10 dancer_1_position = table[dancer_1_move][dancer_2_move][dancer_3_move];
11 dancer_2_position = table[dancer_1_move][dancer_2_move][dancer_3_move];
12 dancer_3_position = table[dancer_1_move][dancer_2_move][dancer_3_move];
```

Figure 2.6c Pseudocode

Over here, the dancer positions are represented using the variables *dancer_x_position* where *x* is used to differentiate between the 3 dancers. Following that *dancer_x_move* represents the encoding of the current state and also transition. For example, if dancer 1 is originally at position 1 and moves right, it can be represented as 0. And if dancer 1 is originally at position 1 and does not move, it can be represented as 1. It is clear that there are only 9 possible such transitions as there are only 3 positions and 3 courses of action, resulting in 9 = 3 x 3. And the size of a 9x9x9 array is extremely small in the context of *Ultra96-V2* and does not pose an issue to the operating system. However, this is just a simple algorithm to illustrate how we can compute the dancer positions and one may simply choose to use an *if-else* construct. However, note that the number of *CPU* cycles to execute an *if-else* will be significantly more compared to this lookup table algorithm we have as the branch is deeply nested and branch prediction accuracy is low given that the positions are randomly generated within the evaluation server script.

Section 3 Hardware Details

Section 3.1: Hardware sensors

3.1.1 Parts List

The following table shows the hardware components used in the wearable.

Component Image	Hardware Component	Requirements	Datasheet
	Beetle BLE, DFR0339	Provided	https://wiki.dfrobot.com/Bluno_Beetle_SKU_DFR0339
	IMU, GY-521 Module (MPU6050)	Provided	https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
	MyoWare Muscle Sensor (EMG)	Provided	https://cdn.sparkfun.com/datasheets/Sensors/Biometric/MyowareUserManualAT-04-001.pdf
	Panasonic Eneloop Battery AAA 350mAh	Single AAA 1.2V Rechargeable Battery	NIL

	AAA battery holder	Holds one AAA battery	
	HW-553 Voltage Regulator	Steps up input voltage to 5V	http://www.tp4056.com/d/tp4056.pdf
	Mi Band Strap	Stylish Secure Easily removable	NIL

Figure 3.1.1a wearable parts list table

3.1.2 Pin Table and Schematics

The following shows the pin table and hardware schematics of the wearable.

Category	**Beetle BLE**	**Peripheral**
Power	VIN	HW-553 Voltage Regulator Positive Terminal (D+)
Power	GND	HW-553 Voltage Regulator NegativeTerminal (D-)
IMU	SDA	GY-521 SDA

IMU	SCL	GY-521 SCL
IMU	5V	GY-521 VCC
IMU	GND	GY-521 GND
Power	VIN	EMG positive terminal
Power	GND	EMG negative terminal
EMG	A0	EMG signal terminal (SIG)
Category	**Peripheral**	**Peripheral**
Power	HW-553 Voltage Regulator IN+	Eneloop AAA battery positive terminal
Power	HW-553 Voltage Regulator IN-	Eneloop AAA battery positive terminal

Figure 3.1.2a wearable pin table

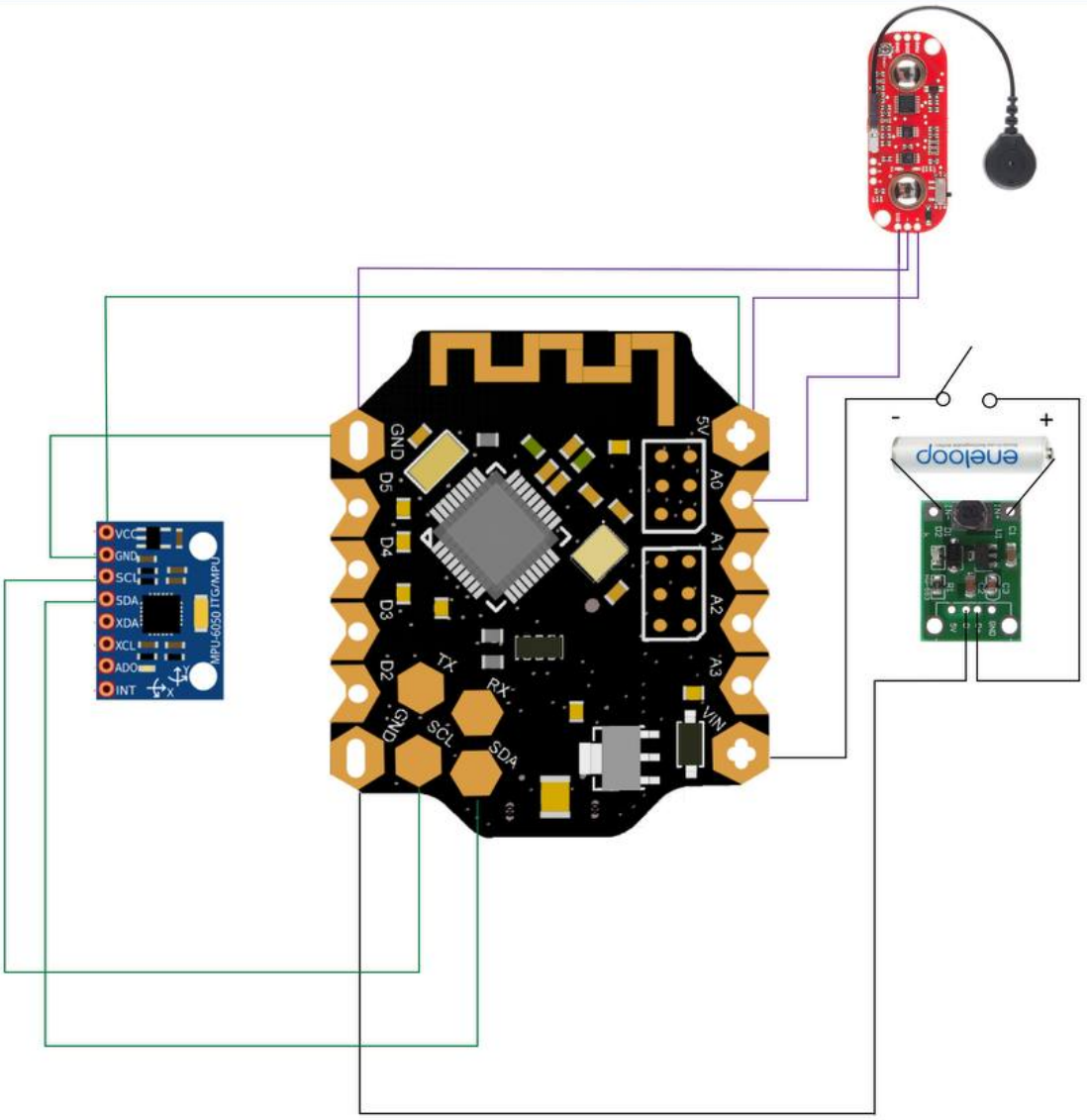


Figure 3.1.2b wearable schematic

3.1.3 Power Considerations

- **Beetle BLE Specifications:**
    - Input: <8V
    - Output: 5V
    - Operating Current: 10-35 mA
- **IMU Specifications:**

    - Input: 5V
    - I2C Voltage Level: 0-3.3V
    - Current: 3.9mA
- **EMG Specifications:**
    - Input: 5V
    - Output Signal Voltage Level: 0-5V
    - Current: 9-14mA
- **Battery Specifications:**
    - Output: 1.5V stepped up to 5V by the voltage regulator
    - Capacity: 350mAh
- **Power System Calculations**

Nominal operating voltage of the battery is at 5V.

Estimated system operating current:

- Beetle: 10-35 mA
- IMU: 3.9mA, less than 10mA
- EMG: 9-14mA
- Total System: 35mA + 10mA + 14mA = 59mA

Estimated battery time: 350mAh ÷ 59mA = 5.9 Hours

As a result, we can expect the system to last for around 5-6 hours before it needs to be recharged.

3.1.4 Beetle BLE Algorithms

3.1.4.1 Sensor Data Noise Removal

Sensor data from the IMU and EMG will be smoothed using an exponential filter. The filter configuration was set up to heavily smooth the sensor data, making it resistant to fluctuations in

the readings. This filtering is done before the sensor data is used in any algorithms that are running on the Beetle.

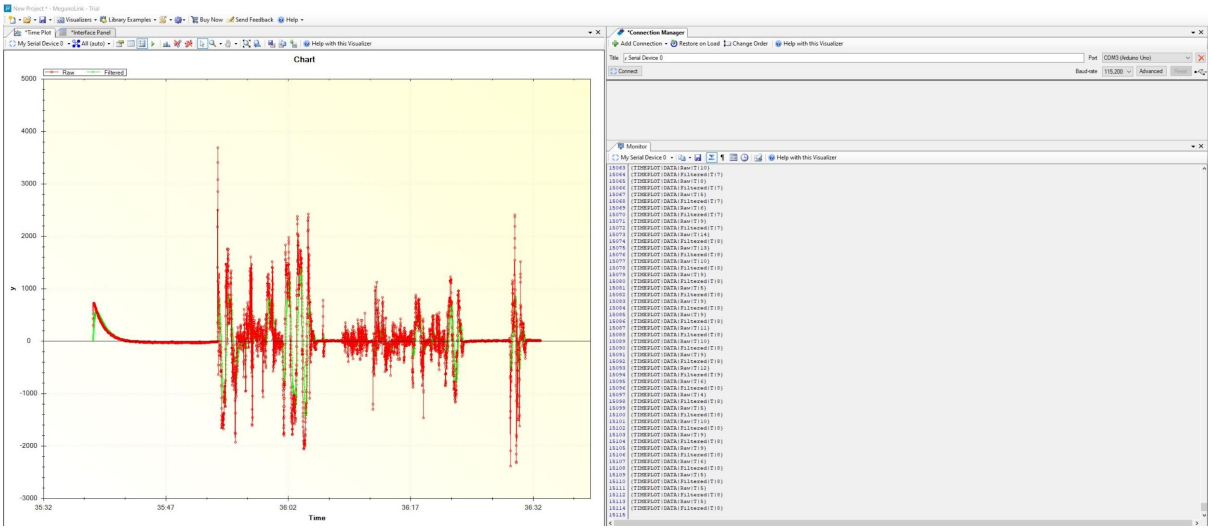


Figure 3.1.4.1a raw against smoothed data of x-axis acceleration

This graph shows the raw accelerometer data in the x axis against the smoothed data after it has been processed by the exponential filter.

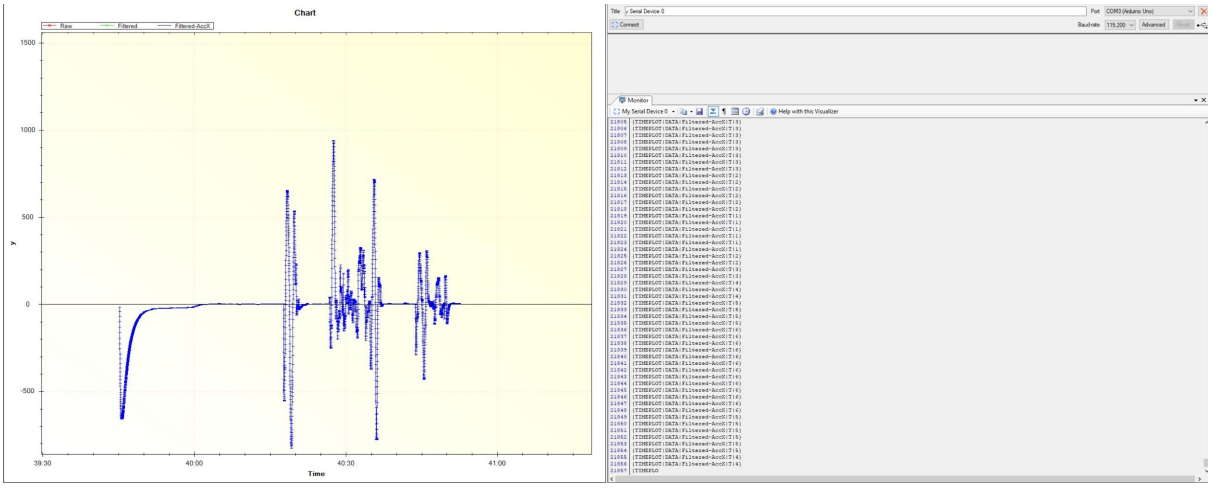


Figure 3.1.4.1b smoothed data of x-axis acceleration

This graph shows the smoothed accelerometer data of a single axis after it has been processed by the exponential filter for a different set of readings.

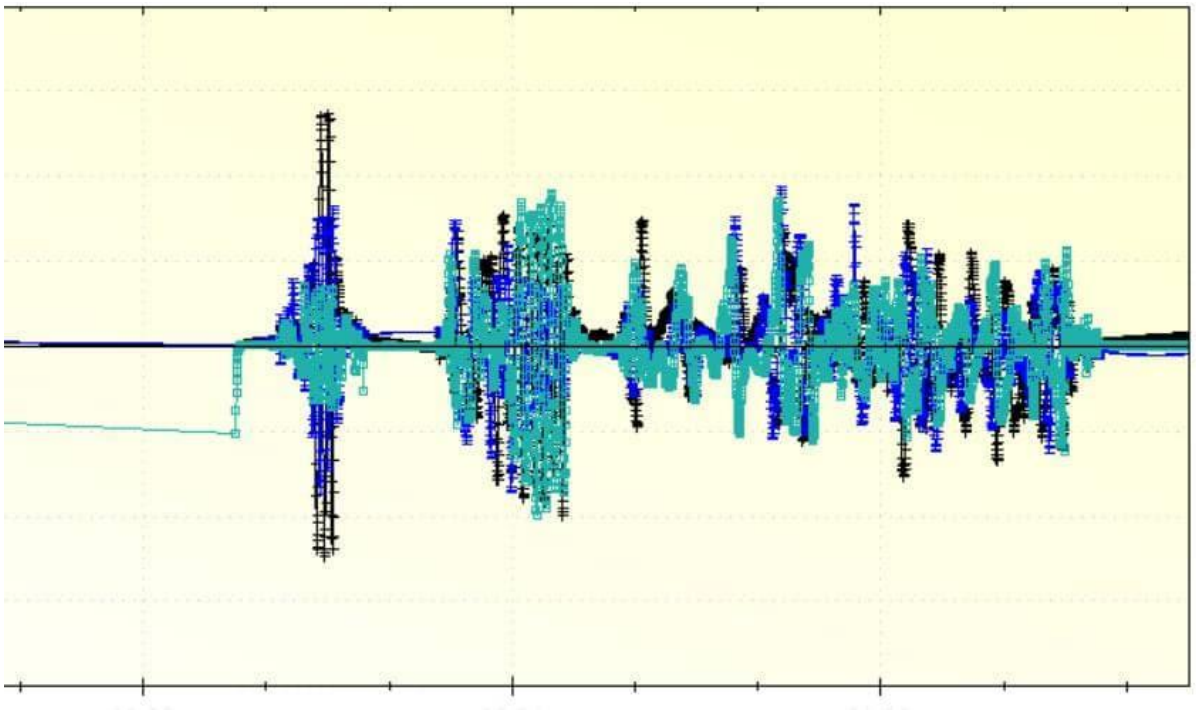


Figure 3.1.4.1c smoothed accelerometer and gyroscope data

This graph shows the smoothed data from the accelerometer and gyroscope that was collected when performing a selected dance move when collecting training data for the machine learning model.

When integrating it together with the final system, there were some issues with the data decay rate being too long as the exponential filter favours the older data over the newer data too much. This affected the dance move prediction and the accuracy of the left and right movement detection during a full test run. Both the dance move prediction and the left and right movement detection are heavily reliant on the accelerometer values which can get large. Since the intervals between moving dance moves and the changing of positions is short, there is not enough time for the accelerometer values to return to a stable state before the next action. This causes issues as some of the older data is still present in the output of the exponential filter.

To solve this issue, we drastically decreased the weights on the older data for the exponential filter on the accelerometer readings as we wanted the sensor readings to be more sensitive. The weights for the exponential filter on the gyroscope readings only were decreased slightly as the range of values for the gyroscope was a lot smaller than the accelerometer readings. This made the detection of the dancer state faster as well.

3.1.4.2 Detection of Dancer State

1. The dancer has only 2 states which can be detected by the Beetle:

a. Idle

b. Dancing

2. When the dancer is Idle, they are in a neutral position:

    a. The acceleration measured by the IMU would be low.

    b. The gyroscope readings would correspond with the arm pointing towards the ground.

3. When the dancer is dancing:

    a. The acceleration measured by the IMU is high as they are performing movements with their arm.

    b. The gyroscope readings would experience rapid changes as their arm positions would be continuously changing while performing the dance move.

        i. An average delta of gyroscope values over a window frame will be used to detect the changes in gyroscope values.

4. The state transition occurs when the above features are detected for a specific window of time.

3.1.4.3 Detection of Dancer Movement (Left/Right)

This is run only when the dancer is in an idle state.

1. When the dancer moves left or right while in a neutral position, there will be a spike in the readings in the horizontal axis.

2. A window of the accelerometer data in the horizontal axis would be read to determine the movement of the dancer:

3. If the velocity in the horizontal axis is positive, the dancer is moving left. If the velocity in the horizontal axis is negative, the dancer is moving right.

4. However, since there is the presence of drift and noise present in the accelerometer data, as well as the variation in arm orientation of each dancer, we are unable to compute velocity from the accelerometer data using any kind of formula.

5. As a result, we computed a value that is reflective of the overall acceleration in the chosen time frame that is used as an indicator of the direction of movement of the dancer.

a. If a positive spike occurs before a negative spike, the dancer is moving left

b. If a negative spike occurs before a positive spike, the dancer is moving right

6. When any of the events are detected, the relevant flag would be set to indicate if a left or right move has occurred.

3.1.4.4 Start of Dance Move Detection

First the dance state of the dancer will be detected if they are dancing or idle. If the dancer is dancing, it will override the next 80 data points as dancing. This is done as there are some moves that trigger the idle state as the arm position relative to the body is low. Since these 80 points are labelled as dancing, it will be forwarded to the machine learning model for dance move prediction.

3.1.4.5 Detection of Dancer Fatigue Level

To detect muscular fatigue, we assessed the output signal of the EMG in the time domain. The indicator in the time domain that is related to amplitude that we have chosen is the Mean Absolute Value (MAV) of the EMG Signal.

The equation used for the MAV of the amplitude of the EMG signal is shown below. Where N is the total number of data points and x_i are the data points read from the EMG signal.

$$MAV = \frac{1}{N} \sum_{i=1}^{N} \left| x_i \right|$$

When this was implemented on the beetle, the MAV readings for all the members of the team were collected while they performed a series of dance moves. When the dancer feels fatigue, they will voice it out and the current MAV reading would be noted down. Using all these values, an appropriate value was chosen for the fatigue threshold that represents the muscular fatigue experienced by the daner.

3.1.5 Beetle BLE Libraries

3.1.5.1 IMU Library

The Jrowberg library for sensor fusion was used. This library enabled the use of the MPU6050 onboard digital motion processor to correct for deviation in accelerometer and gyroscope values.

From this library, we are able to obtain 2 categories of readings:

1. Gyroscope:
    a. Yaw
    b. Pitch
    c. Roll
2. Acceleration:
    a. X-Axis
    b. Y-Axis
    c. Z-Axis

3.1.5.2 Circular Buffer Library

This library is used to maintain a sliding window for both the accelerometer, gyroscope and EMG readings for the processing done by the algorithms on the Beetle.

3.1.5.3 MegunoLink library

This library was used for filtering the sensor values as well as visualising and exporting the data collected by the sensors.

Section 3.2: Hardware FPGA

In this section, we will first introduce popular frameworks that have been used by people all over the world to deploy their model onto FPGA. Afterwards, we will introduce the methodology our group has adopted.

3.2.1 FINN Framework

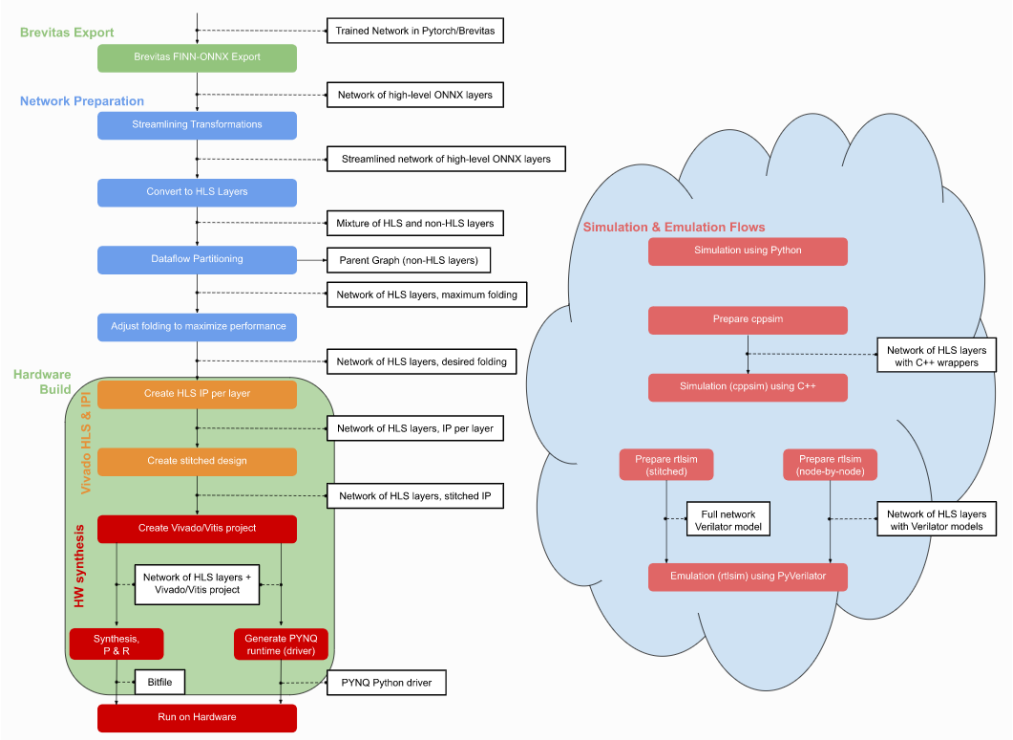


Figure 3.2.1a FINN framework overview

The initial plan is to use the *FINN* compiler to accelerate the model. First of all, the *FINN* compiler takes in *ONNX* layers as input instead of regular models. *ONNX* stands for open neural network exchange and is an open source format.

Brevitas is one of the many libraries that can train models and export them as *ONNX* and also made by *Xilinx*. *FINN* takes in a specific *ONNX* format that can only be guaranteed if *Brevitas* library is used. Else, it is unclear whether other *ONNX* formats will work on *FINN*. Next, we can adjust and fine tune the model using the *ModelWrapper* provided by *FINN*.

At this stage we can already use the functional verification flow to simulate the model using *Python*, this is marked in the graphic with the dotted arrow.

Next, we start to prepare the network by trying to convert nodes that correspond to *finn-hlslib* functions. The rationale is that the finn compiler relies on *Analysis Pass* and *Transformation Pass*,

by conforming to the standards, it allows the compiler to identify and optimise more portions of the network. To study the model, we used *Netron*, a tool to visualize neural networks.

Network preparation is complicated and involved the following operations:

- Tidy-up transformations
- Streamlining Transformations
- Convert to *HLS* Layers
- Dataflow Partitioning
- Folding

FINN compiler will then convert the *ONNX* format to *HLS* layers. The end result is a network of *HLS* layers with desired folding and it can be passed to *Vivado* for synthesis. The layers will then be stitched accordingly and compiled into a bitstream. To interface with the bitstream, we can then use the *PYNQ* built in functions to create a driver.

So here is a walkthrough of what we did while exploring *FINN*. We have used the *Convolutional Neural Network* model provided in the *GitHub* repository as the machine learning person has yet to come up with a model.

```
import onnx
from finn.util.test import get_test_model_trained
import brevitas.onnx as bo
from finn.core.modelwrapper import ModelWrapper
from finn.transformation.infer_shapes import InferShapes
from finn.transformation.fold_constants import FoldConstants
from finn.transformation.general import GiveReadableTensorNames, GiveUniqueNodeNames, RemoveStaticGraphInputs

cnv = get_test_model_trained("CNV", 1, 1)
bo.export_finn_onnx(cnv, (1, 3, 32, 32), build_dir + "/end2end_cnv_w1a1_export.onnx")
model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_export.onnx")
model = model.transform(InferShapes())
model = model.transform(FoldConstants())
model = model.transform(GiveUniqueNodeNames())
model = model.transform(GiveReadableTensorNames())
model = model.transform(RemoveStaticGraphInputs())
model.save(build_dir + "/end2end_cnv_w1a1_tidy.onnx")
```

Figure 3.2.1b FINN in action

The first step we do is import the model. Subsequently, we do a series of transformations on the model to make it more readable and understandable. The *InferShapes()* function ensures that every tensor in the model has a specific shape. Next, *FoldConstants()* functions replaces the output of a node with const-only inputs with a precomputed result. *GiveUnqiueNodeNames()* and *GiveReadableTensorNames()* functions give names to each node and tensor so that we can read it more easily. Last but not least, *RemoveStaticGraphInputs()* function simply removes the static inputs in the neural network. Now, we are ready to visualise this neural network and ready for more operations.

So this is an overview of how a convolutional neural network looks like in this example.

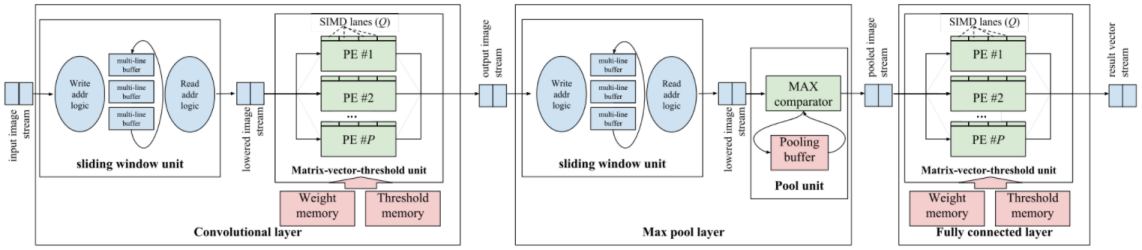


Figure 3.2.1c CNN explained by FINN

In the *FINN* workflow, it converts convolution to matrix operations. Essentially, we will be doing a bunch of matrix multiplication in the implementation on our *FPGA* at the end.

So the *FINN* workflow precisely provides us the tools to do this transformation from convolution to matrix operations. For the sanity of the reader of the report, we will not go through what each function does.

```
from finn.transformation.streamline import Streamline
from finn.transformation.lower_convs_to_matmul import LowerConvsToMatMul
from finn.transformation.bipolar_to_xnor import ConvertBipolarMatMulToXnorPopcount
import finn.transformation.streamline.absorb as absorb
from finn.transformation.streamline.reorder import MakeMaxPoolNHWC, MoveScalarLinearPastInvariants
from finn.transformation.infer_data_layouts import InferDataLayouts
from finn.transformation.general import RemoveUnusedTensors

model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_pre_post.onnx")
model = model.transform(MoveScalarLinearPastInvariants())
model = model.transform(Streamline())
model = model.transform(LowerConvsToMatMul())
model = model.transform(MakeMaxPoolNHWC())
model = model.transform(absorb.AbsorbTransposeIntoMultiThreshold())
model = model.transform(ConvertBipolarMatMulToXnorPopcount())
model = model.transform(Streamline())
# absorb final add-mul nodes into TopK
model = model.transform(absorb.AbsorbScalarMulAddIntoTopK())
model = model.transform(InferDataLayouts())
model = model.transform(RemoveUnusedTensors())
model.save(build_dir + "/end2end_cnv_w1a1_streamlined.onnx")
```

Figure 3.2.1d FINN on CNN

After this step, what we obtain can already be implemented on the *FPGA* step-by-step as it only consists of simple math functions. However, the *Vivado HLS* is not all powerful and able to synthesize every single operation. Thus, there is a need to perform partition operation to separate out what can be synthesised by *Vivado* and what cannot be synthesized.

```
import finn.transformation.fpgadataflow.convert_to_hls_layers as to_hls
from finn.transformation.fpgadataflow.create_dataflow_partition import (
    CreateDataflowPartition,
)
from finn.transformation.move_reshape import RemoveCNVtoFCFlatten
from finn.custom_op.registry import getCustomOp
from finn.transformation.infer_data_layouts import InferDataLayouts

# choose the memory mode for the MVTU units, decoupled or const
mem_mode = "decoupled"

model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_streamlined.onnx")
model = model.transform(to_hls.InferBinaryStreamingFCLayer(mem_mode))
model = model.transform(to_hls.InferQuantizedStreamingFCLayer(mem_mode))
# TopK to LabelSelect
model = model.transform(to_hls.InferLabelSelectLayer())
# input quantization (if any) to standalone thresholding
model = model.transform(to_hls.InferThresholdingLayer())
model = model.transform(to_hls.InferConvInpGen())
model = model.transform(to_hls.InferStreamingMaxPool())
# get rid of Reshape(-1, 1) operation between hlslib nodes
model = model.transform(RemoveCNVtoFCFlatten())
# get rid of Tranpose -> Tranpose identity seq
model = model.transform(absorb.AbsorbConsecutiveTransposes())
# infer tensor data layouts
model = model.transform(InferDataLayouts())
parent_model = model.transform(CreateDataflowPartition())
parent_model.save(build_dir + "/end2end_cnv_w1a1_dataflow_parent.onnx")
sdp_node = parent_model.get_nodes_by_op_type("StreamingDataflowPartition")[0]
sdp_node = getCustomOp(sdp_node)
dataflow_model_filename = sdp_node.get_nodeattr("model")
# save the dataflow partition with a different name for easier access
dataflow_model = ModelWrapper(dataflow_model_filename)
dataflow_model.save(build_dir + "/end2end_cnv_w1a1_dataflow_model.onnx")
```

Figure 3.2.1e FINN dataflow partition

Afterwards, there is also a need to check if the accelerator can be feasibly implemented on the *FPGA*. There is an issue of Fan-in-fan-out and we would like to adjust the folding factors to meet the desired requirement of the *FIFO* depth.

```
model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_dataflow_model.onnx")
fc_layers = model.get_nodes_by_op_type("StreamingFCLayer_Batch")
# each tuple is (PE, SIMD, in_fifo_depth) for a layer
folding = [
    (16, 3, 128),
    (32, 32, 128),
    (16, 32, 128),
    (16, 32, 128),
    (4, 32, 81),
    (1, 32, 2),
    (1, 4, 2),
    (1, 8, 128),
    (5, 1, 3),
]
for fcl, (pe, simd, ififodepth) in zip(fc_layers, folding):
    fcl_inst = getCustomOp(fcl)
    fcl_inst.set_nodeattr("PE", pe)
    fcl_inst.set_nodeattr("SIMD", simd)
    fcl_inst.set_nodeattr("inFIFODepth", ififodepth)

# use same SIMD values for the sliding window operators
swg_layers = model.get_nodes_by_op_type("ConvolutionInputGenerator")
for i in range(len(swg_layers)):
    swg_inst = getCustomOp(swg_layers[i])
    simd = folding[i][1]
    swg_inst.set_nodeattr("SIMD", simd)

model = model.transform(GiveUniqueNodeNames())
model.save(build_dir + "/end2end_cnv_w1a1_folded.onnx")
```

Figure 3.2.1f FINN folding

And that is pretty much the last step! Now we can choose the target board and generate our bitstream and driver file. And that is pretty much the entire workflow of the *FINN* framework, apart from the verification of the accelerator which can be applied not only to the *FINN* workflow but other workflows too.

3.2.2 Vitis and Vivado Workflow

The next workflow that our group is considering is the traditional framework where each layer is written using either *Vivado/Vitis HLS* and exported as *IPs* and integrated with other predefined *IP blocks* to interface with the *ZYNQ-MPSoC+*.

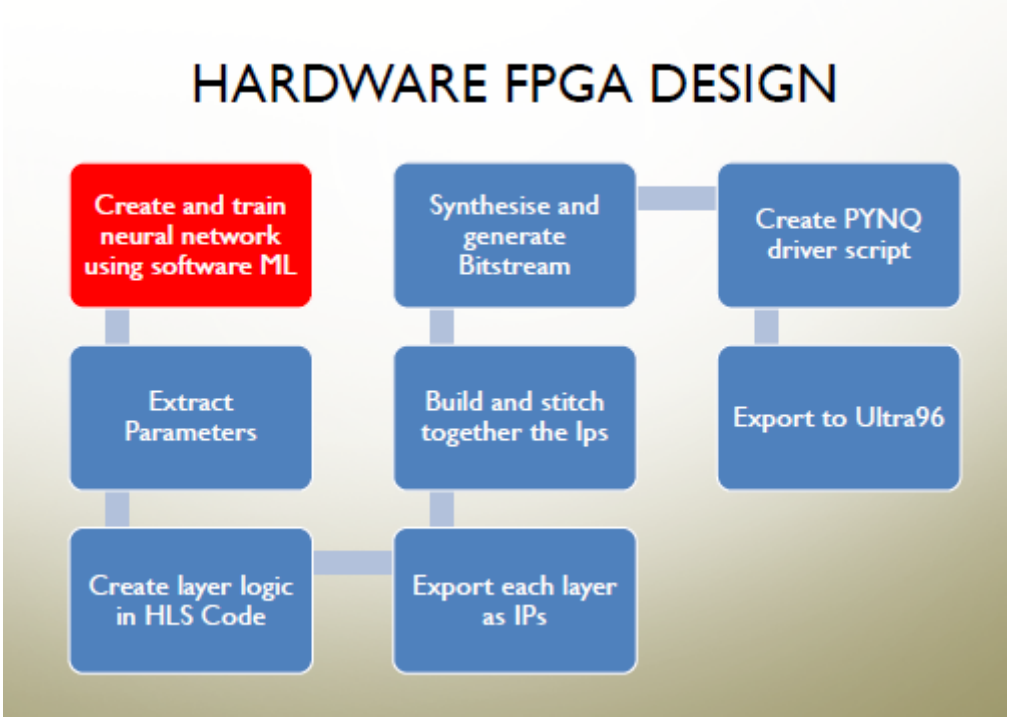


Figure 3.2.2a Diagram from lecture

As shown in the lecture, we would have to first extract parameters of the neural network. However, there are no existing tools that extract parameters of a neural network. Instead, we would use tools like *Netron* to visualize the different layers and implement each layer ourselves. This is highly dependent on the machine learning library used and what sort of model is being produced. Even within neural networks, there are gazillions of different kinds of layers and no knowledge will ever suffice.

After exporting each layer as *IPs* we would have to stitch the layers together. However, there is no information on how to integrate these *IPs* with the *ZYNQ Ultrascale MPSoC+*, making it an extremely challenging process.Therefore, we used the *FINN* framework earlier to help us figure out what are the necessary *IP blocks* the *ZYNQ Ultrascale MPSoC+* needs by inspecting the generated block diagrams by *FINN* and learning from it before proceeding to this method. We have already generated the block diagram using *FINN* and screenshots are provided in the optimization section of the report.

The next issue with this method is the lack of resources and information on how to create the driver code. It is unknown which libraries to use and the *APIs* these libraries can provide.

Although *PYNQ* does provide a short tutorial on how to interface without much explanation as they assumed users to be extremely familiar with *Vivado IP* design. Therefore, it is recommended to learn from the *FINN* framework again to glimpse these information so that we can emulate and design our own driver code.

Furthermore, we were and will not be taught the best practise in designing the *FPGA* to accelerate neural networks and diving straight into such a workflow has a lot of pitfalls and should be avoided given the duration and goal of the project. Now we would proceed to explain what each step of the workflow is about.

3.2.2.1 Extract Parameters

This is unknown as different machine learning libraries store their parameters differently. But in general, we should be able to use *Python* to print the description of the neural network. And we should be able to simply matrix-multiply the output of the previous layer with the weights and accumulate with the bias and treat that as input to the subsequent layer. For a foreign neural network model, we can visualise it with the *Netron* tool mentioned earlier

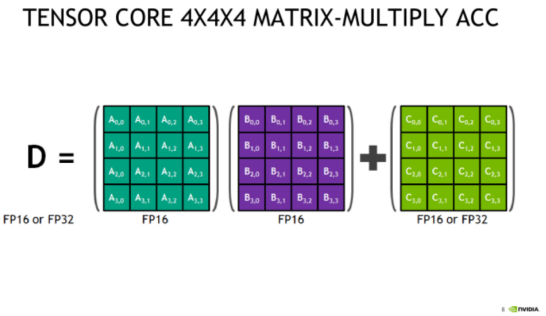


Figure 3.2.2.1a Tensor multiply and accumulate

We can actually do the multiply and add instructions extremely well on the main processor (*Cortex-A53*) of *Ultra96-V2* we have at hand due to the presence of *Neon Intrinsics* designed for *SIMD*. However, we would still stick with running the model on *FPGA* because of project requirements. Although it is agreed that the best performance is to have both the processor and *FPGA* running parallel with one another to perform the operations.

```
xilinx@pynq:~$ lscpu
Architecture:        aarch64
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  1
Core(s) per socket:  4
Socket(s):           1
Vendor ID:           ARM
Model:               4
Model name:          Cortex-A53
Stepping:            r0p4
CPU max MHz:         1199.9990
CPU min MHz:         299.9990
BogoMIPS:            199.99
Flags:               fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
```

Figure 3.2.2.1b Processor on Ultra96-V2

3.2.2.2 Create Layer Logic in HLS

After extracting the parameters, we move on to creating the individual layers in *Vivado HLS*. Below is an excerpt taken from Prof Rajesh EE4218 page and left here so that future readers of the report will not be as lost as our group. Each step is critical as *Vivado* itself is very problematic.

1) Open Vivado HLS.

2) Select Create New Project. Specify a name and path/location which doesn't involve underscores or spaces. Next Add/Remove C based source files (design specification). Add File > browse and select myip_v1_0_HLS.cpp. Click on the Browse button for the Top Function. Select myip_v1_0_HLS. OK, and Next.

3) Add/Remove C based testbench files (design test). Add File > browse and select test_myip_v1_0_HLS.cpp. Next. Note that the above files will remain in their original location and won't get copied over to the project, there is no option to easily copy it over to the project. The files should be in a location that doesn't involve underscores or spaces. It is also fine to choose New File instead of Add file and for the two steps above. Give an appropriate name, with a .cpp extension. You will later need to copy over the contents of the template files to the new files created. There is also an additional step of selecting the top function later.

4) Provide a solution name, clock period (10 ns), and select the part number. Older versions of HLS ask for clock frequency instead, which should be entered as 100MHz.

5) Click Boards, and select *Ultra96-V2*. Next.

6) You will see myip_v1_0_HLS.cpp file under Sources, and test_myip_v1_0_HLS.cpp under Test Bench in the project Explorer. Ensure that the contents are as expected.

7) Project > Project Settings. Browse and ensure that the Top Function is myip_v1_0_HLS (this is esp required if you change the function names, or if you created a file in the earlier part of this manual instead of adding a file).

3.2.2.3 Export layers as IPs

After we are satisfied with our design, we move on to the next step and export the layers as *IP*s. Again the following excerpt is taken from Prof Rajesh EE4218 page and left here so that future readers of the report will not be as lost as our group.

1) Once coding is complete, you can Run C synthesis and generate a report about resource utilization, timing constraints etc

2) Once you are satisfied with your code, you can export it. Solution > Export RTL. Accept the defaults and click ok. You can see that the RTL is now successfully exported, and is ready to be used in the Vivado IP integrator.

3) Open Vivado (not HLS). Create / Open a project (you can use a copy of the Lab 3 project) and go to IP integrator interface. Right-click and choose IP settings.

4) Choose IP > Repository and click the '+' sign. You can also do so from Tools (in the top menu bar) > Settings > Project Settings > IP > Repository.

5) Navigate to the directory containing the Solution (inside the HLS project folder)

6) That is it. Now you will be able to see the IP in the IP catalog.

3.2.2.4 Building and Stitching IPs

Once all our layers are ready, we will proceed to stitching the various *IP*s together. As this is the initial report, it is still unclear what sort of *IP*s we would finalise on for interfacing with the processor. Thus, information of the blocks would be omitted. This is because we do not know how much data we will be exchanging between the main processor and *FPGA* nor how fast the *FPGA* will perform. As such, it is difficult to decide which interface is the best.

To connect the *IP*s, one would use their eyes and try to connect them in *Vivado* > *Create block design* as shown in the image taken from *Xilinx* guides.

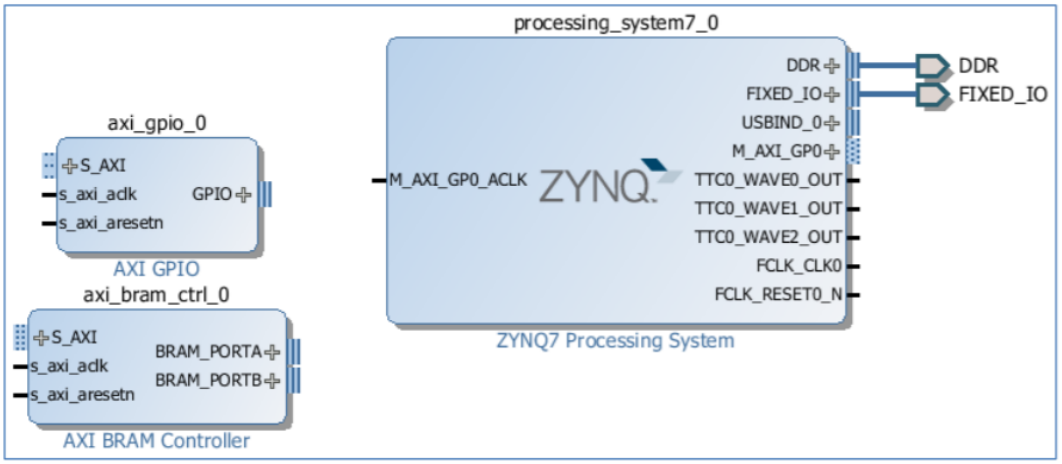


Figure 3.2.2.4a Stitching IPs

After we are done, remember to save the block design by selecting *File* > *Save Block Design*.

3.2.2.5 Synthesis and Generate Bitstream

After we are satisfied with our block design we can proceed to generate the *vhdl* files and also create a bitstream. The following is taken from the *Xilinx* guide and left here so that future readers of the report will not be as lost as our group.

1) In the Sources window, right-click the top-level subsystem design and select Generate Output Products. This generates the source files for the IP used in the block design and the relevant constraints file. You can also click Generate Block Design in the Flow Navigator to generate the output products.

2) Leave all the settings to their default values. Click Generate.

3) The Generate Output Products dialog box opens informing that Out-of-context runs were launched. Out-of-context runs can take a few minutes to finish. You can see the status of the runs by clicking on the Design Runs tab at the bottom of the Vivado IDE.

4) Click OK.

5) In the Sources window, right-click the top-level subsystem, zynq_design_1, and select Create HDL Wrapper to create a top level HDL file that instantiates the block design.

6) Select the default option of Let Vivado manage wrapper and auto-update.

7) Click OK.

8) In Flow Navigator > Program and Debug, click Generate Bitstream to implement the design and generate a BIT file

9) Click Yes. This will launch synthesis, implementation and generate the bitstream which could take a few hours.

3.2.2.6 Create PYNQ Driver Script

As this is the initial design report, we are still unsure which interface we will be using. Therefore, the respective *PYNQ* interface we will be using is still unclear, however, it will be one of the following from the *PYNQ* link.

PYNQ provides a neat feature called the *Overlay* class to load the bitstream into the *FPGA*. From the documentation, it states that a *tcl* script is required accompanying the *bit* file, this should also be automatically generated when exporting the *IP* Integrator block diagram. So we can simply just copy these files onto the *Ultra96-v2*.

```
from pynq import Overlay

base = Overlay("base.bit") # bitstream implicitly downloaded to PL
```

Figure 3.2.2.6a PYNQ Overlay class

And this *Python* driver will be used by the main server script on the *Ultra96-v2* to perform inference.

3.2.3 Vitis AI Workflow

This is the workflow that is designed and recommended by *Xilinx* in accelerating neural networks and other *AI* models.

"Vitis AI is Xilinx's development stack for AI inference on Xilinx hardware platforms, including both edge devices and Alveo cards. It consists of optimized IP, tools, libraries, models, and example designs. It is designed with high efficiency and ease of use in mind, unleashing the full potential of AI acceleration on Xilinx FPGA and ACAP."

The *Ultra96-V2* is an edge device and therefore is also a target of the *Vitis AI* workflow. The *Vitis AI* is composed of the following key components:

- AI Model Zoo - A comprehensive set of pre-optimized models that are ready to deploy on Xilinx devices.

- AI Optimizer - An optional model optimizer that can prune a model by up to 90%. It is separately available with commercial licenses.
- AI Quantizer - A powerful quantizer that supports model quantization, calibration, and fine tuning.
- AI Compiler - Compiles the quantized model to a high-efficient instruction set and data flow.
- AI Profiler - Perform an in-depth analysis of the efficiency and utilization of AI inference implementation.
- AI Library - Offers high-level yet optimized C++ APIs for AI applications from edge to cloud.
- DPU - Efficient and scalable IP cores can be customized to meet the needs for many different applications

The workflow is illustrated as follows, with *Vitis AI* providing tools for each step of the workflow. As shown in the diagram, the workflow highly resembles the FINN framework apart from the presence of *Deep learning Processing Units* (*DPUs*) which are pre-configured *IP*s that can be easily integrated and supports most deep learning models. Unlike other workflows, this workflow provides the full suite of tools for creating accelerators including the creation of a specialized *OS* for the *Ultra96-V2* board using the *Petalinux* tools.

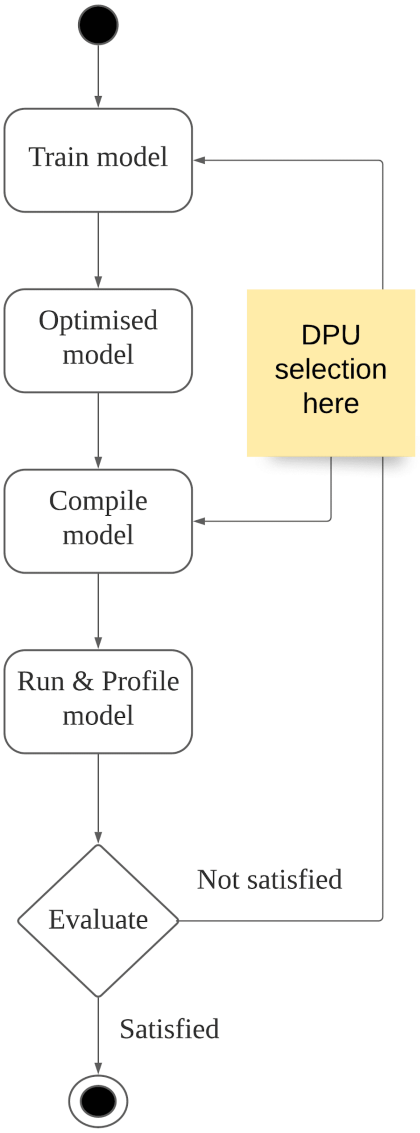


Figure 3.2.3a Vitis AI workflow

First, the model is trained using modern frameworks such as *Tensorflow* or *Pytorch*. Afterwards, they are further optimised by pruning to reduce the size of the neural network. After being optimised, it is time to quantise the model in order to fit the *FPGA*. Using the *Vitis AI* compiler, it is being compiled to either an *.elf* or *.xmodel* depending on the version of *Vitis AI* that was used. From this point onwards, we will be referring to the older framework (*v1.2*) where an *.elf* is generated.

We can use the DPU-PYNQ framework to interface with the *DPU* on the *Ultra96-v2*. As mentioned earlier, *DPU* is a prebuilt bitstream for deep learning models. We can also fine tune the *DPU* to suit our workloads and generate a *DPU* ourselves using the configurations they have provided. The *.elf* will be responsible for configuring the *DPU* such that the model we train can run on top of the *DPU*. More specifically it will interact with *Vitis AI Run Time* (*VART*) shown in

the diagram taken from *Xilinx* below. The *DPU-PYNQ* takes away the hassle of managing the installation of the relevant libraries and dependencies.

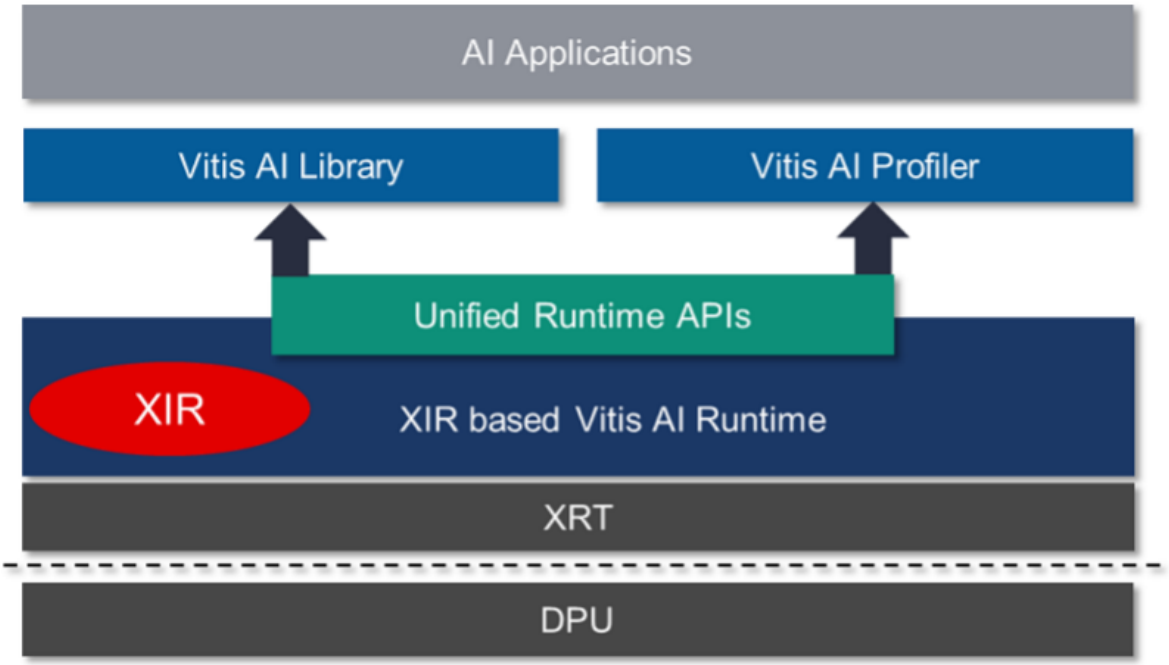


Figure 3.2.6b Vitis AI runtime

3.2.4 Neural Network Design

As of now, our group will have to follow the *Vivado* workflow even though it is the most foreign and challenging due to the lack of knowledge as we have not taken EE4218 nor CS3244. We had to do a lot of self-reading to catch up. Having a foundation in either of the modules will significantly reduce the difficulty of the Capstone project.

The neural network my group has selected for the *FPGA* is the simple feed forward neural network commonly referred to as the multilayer perceptron model (*MLP)*.

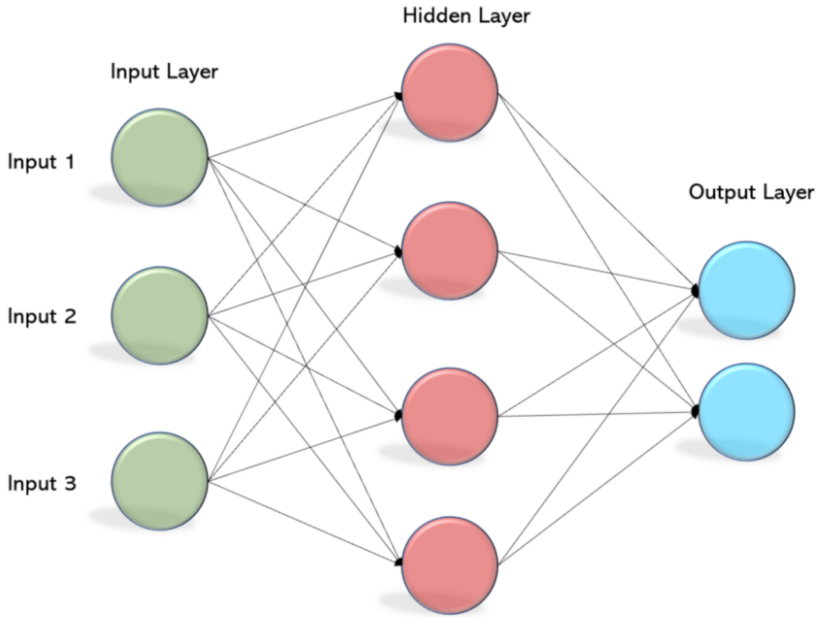


Figure 3.2.4a Multilayer perceptron/Feed forward neural network

In the diagram above, each circle represents a perception and can be represented using an activation function. An activation function is a mathematical function that maps an input value to an output value. On the other hand, the arrows in the diagram can be represented by a tensor. By multiplying the output of the previous layer nodes with a tensor, we can get the input values for the nodes of the next layer.

Next, we will be coding the tensors and activation function out in *C++* on *Vivado* and use *Vivado HLS* to generate an *IP* block and bitstream.

3.2.5 Mapping of Neural Network to FPGA Board

The initial mapping will be done using the *FINN* framework as the *FINN* framework is designed by people who have a clear understanding of how things should be mapped onto a *FPGA*. As such, we will use the *IPs* generated by the *FINN* framework as a starting point and improve the design from there on.

Next, we would also like to compare *Vitis AI* against *FINN* to see if the *DPUs* outperform the bitstream created by the *FINN* compiler. Depending on the experimental results of the 2 different frameworks, we would stick to the one that best fit the project requirement and fine tune the *DPUs* or *FINN IPs* accordingly.

However, as per requirements, our team will have to focus on *Vivado* workflow first before exploring the aforementioned options. Thus, the mapping to the *FPGA* board is as follows:

- Edges in the model ----> tensors
- Perceptrons in the model ----> activation function

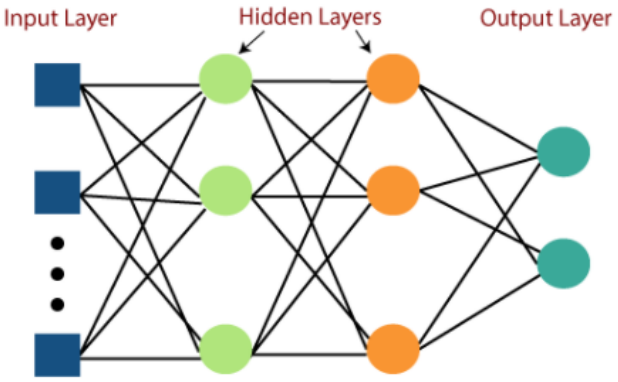


Figure 3.2.5a Neural network

- Input from processing system ---> through AXI interconnect to somewhere
- Input layer ---> *X* amount of activation functions
- Output of input layer to input of hidden layer ---> W x Y tensor
- Hidden layer ---> Y amount of activation functions
- Output of hidden layer ---> V x Z tensor
- Output layer ---> Z amount of activations functions
- Output to processing system ---> through AXI interconnect to somewhere

Here are the other possible activation techniques and structures we might be introducing into our neural networks as the project continues and the data we have at hand:

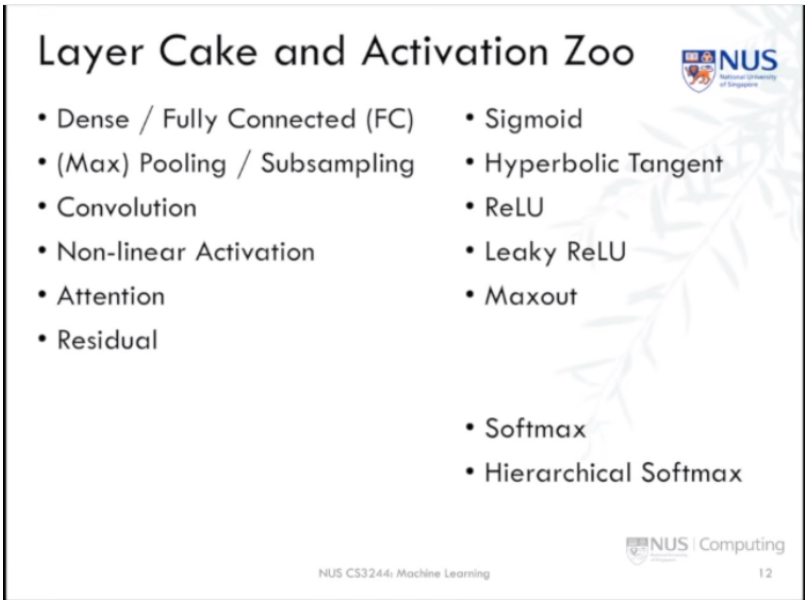
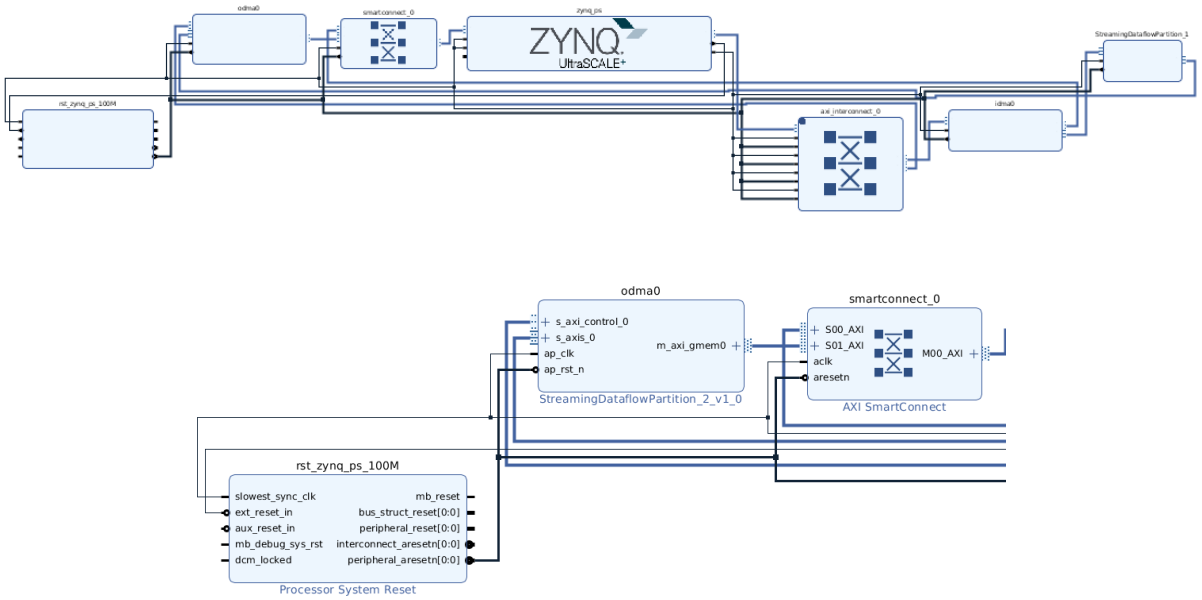


Figure 3.2.5b Model Zoo

But for the start, we will stick to *MLP* and use the *Sigmoid* or *ReLU* function for the neurons which seems to be more *FPGA* friendly as it does not involve negative numbers which may lead to unnecessary utilization of hardware resources on the *FPGA*.

3.2.6 Potential Optimizations



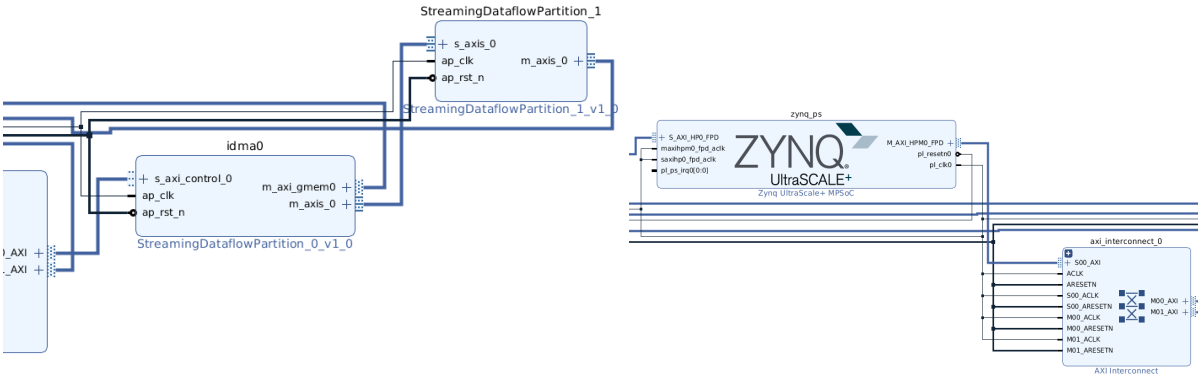


Figure 3.2.6a Figures of block designs

The diagrams above are the block designs we have generated using the *FINN* compiler for a sample convolution neural network model mentioned earlier. Please note that they were generated before the slides were provided and thus left in the report for reference purposes. Each *IP* block of the *StreamingDataflowPartition* actually consists of hundreds of layers written in *Vivado HLS*. We can access each layer and manually see what's going on and regenerate the *IP* blocks ourselves. However, for such a complex neural network, it is difficult to understand the consequence of changing 1 single layer and what sort of overall impact it will have on accuracy. Thus, there is really not much point in trying to modify the layers manually. Instead it is easier to return to the *FINN* workflow of tuning the model parameters as we are able to both simulate the model performance with very little cost and work at a higher level of abstraction than *Vivado HLS*.

```
#include "bnn-library.h"

// includes for network parameters
#include "weights.hpp"
#include "activations.hpp"
#include "mvau.hpp"
#include "thresh.h"

// defines for network parameters
#define MW1 600
 #define MH1 64

           #define SIMD1 15
 #define PE1 32
 #define WMEM1 80

           #define TMEM1 2
 #define numReps 1
#define WP1 2


void StreamingFCLayer_Batch_0(
                  hls::stream<ap_uint<15>> &in0,
                  hls::stream<ap_uint<960>> &weights,
                  hls::stream<ap_uint<64>> &out
                  )
{
#pragma HLS INTERFACE axis port=in0
#pragma HLS INTERFACE axis port=out
#pragma HLS stream depth=2 variable=in0
#pragma HLS stream depth=2 variable=out
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE axis port=weights
#pragma HLS stream depth=8 variable=weights
#pragma HLS ARRAY_PARTITION variable=threshs.m_thresholds complete dim=1
#pragma HLS ARRAY_PARTITION variable=threshs.m_thresholds complete dim=3
Matrix_Vector_Activate_Stream_Batch<MW1, MH1, SIMD1, PE1, Recast<Binary>, Slice<ap_uint<2>>, Identity, ap_int<2> >
                (in0, out, weights, threshs, numReps, ap_resource_lut());
}
"top_StreamingFCLayer_Batch_0.cpp" 42L, 1205C                                                    42,1          Bot
```

Figure 3.2.6b One of the hundreds of Vivado HLS in the neural network

As shown below, instead of managing the individual *HLS* layers, we would be managing the different segments of the neural network layer that is supported to be converted to *Vivado HLS* by the *FINN* framework. A simple glance and we know that this model is not something we want to manually optimise using *Vivado HLS* layer by layer as it will probably take forever.

In essence, we will be using the transformation tools that *FINN* provided to manipulate the layers. As the layers are changed, so will the corresponding *HLS* layers when they are generated.

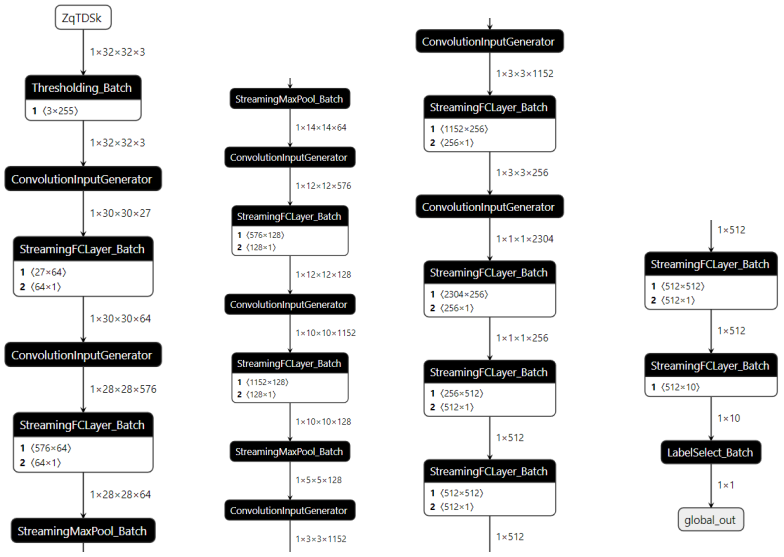


Figure 3.2.6c Netron on CNN model

But how are we going to test these models given that the synthesis of each bitstream takes about a day? Therefore, we will introduce the verification tools we are planning to use for this project.

There are mainly 3 tools:

- *PyVerilator* to ensure the integrity of the *IP* blocks (stitched *HLS*)
- Normal *python* to ensure integrity of the models (accuracy benchmark against different *onnx* models)
- *cppsim* to ensure the integrity of the *HLS*

Python will be heavily used in the quantization of the model. We would need to quantise the model to ensure that there is no explosion of resource usage in the *FPGA*. By trading off a tiny bit of accuracy, we can expect the resource usage to decrease exponentially. Therefore, we would use simple to verify the integrity of the quantised model before going ahead with the implementation on the *FPGA*.

Cppsim can be used to quickly compile and generate the code written in *C++* for *Vivado HLS*. Thus we can verify the functionality of each *HLS* layer simply by building the code with *cppsim* and verify the output to ensure that the *HLS* layers work as intended.

PyVerilator is precisely a tool for verifying hardware models. As the names implies, the *Verilator* corresponds to a tool for *Verilog* and *Py* corresponds to *Python*. We will use this tool to verify the hardware model we have and the code snippets that may have been written in *Verilog/vhdl*.

The first design optimizing technique that can be used in *Vivado HLS* is the pragma directives. This can be simply done by selecting the code that you want to insert the directive on and choosing the directive as shown in the image taken from Prof Rajesh EE4218 page.

Open the directive view on the right. Note that the values as shown below will appear only when the myip_v1_0_HLS.cpp is selected in the editor window. Otherwise, you will see a message that "Directive View is not available".

Select the part of the code whose hardware architecture you wish to control, for example, the for loop below.

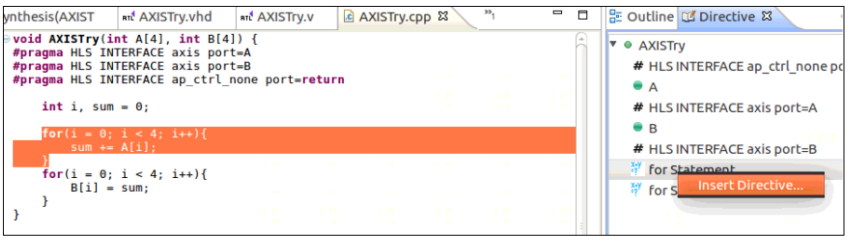


Figure 3.2.6d Directives in action

There are in total 9 different categories of directives as shown in the *Xilinx* guide.

Type	Attributes
Kernel Optimization	• pragma HLS allocation • pragma HLS clock • pragma HLS expression_balance • pragma HLS latency • pragma HLS reset • pragma HLS resource • pragma HLS top
Function Inlining	• pragma HLS inline • pragma HLS function_instantiate
Interface Synthesis	• pragma HLS interface • pragma HLS protocol
Task-level Pipeline	• pragma HLS dataflow • pragma HLS stream
Pipeline	• pragma HLS pipeline • pragma HLS occurrence
Loop Unrolling	• pragma HLS unroll • pragma HLS dependence
Loop Optimization	• pragma HLS loop_flatten • pragma HLS loop_merge • pragma HLS loop_tripcount
Array Optimization	• pragma HLS array_map • pragma HLS array_partition • pragma HLS array_reshape
Structure Packing	• pragma HLS data_pack

Figure 3.2.6e Possible HLS directives

The pragma exploits various optimization techniques that are commonly used in compilers such as function inlining and also special ones in *Vivado HLS*.

We will explain a few of the optimizations that might be more impactful to our project. These are very suitable directives as we will be mainly dealing with matrix multiplication and activation functions which are mathematical functions that possibly involve loops that can be parallelised easily.

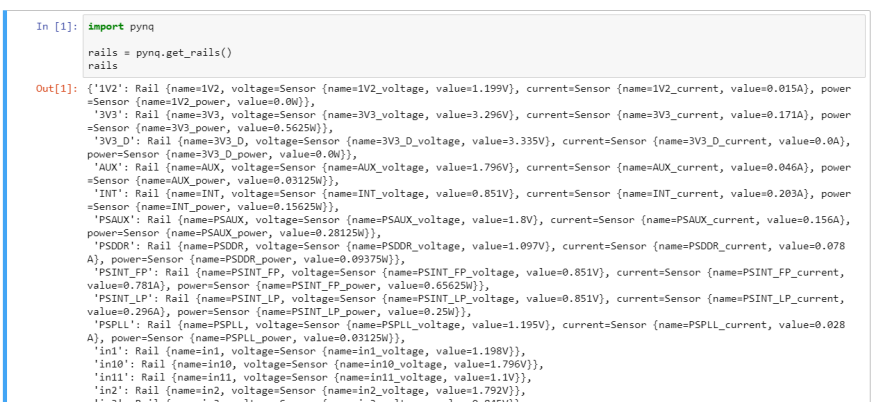
- Array partition directive partition an array into smaller arrays or individual elements
    - Results in RTL with small memories or register instead of one large memory
    - Effectively increases the amount of read and write ports for the storage
    - Potentially increase throughput at the cost of more memory instances or registers
- Loop unrolling which unroll the loops
    - Results in less iterations of the loop and potentially more resource consumption

- Pipelining reduces the number of loops by concurrent execution of operations
    - Results in higher throughput at the cost of more resources and might help in lowering latency

3.2.7 Power Management

For power management, we wanted to use PowerTop. PowerTop analyzes the programs, device drivers, and kernel options running on a computer based on the Linux and Solaris operating systems, and estimates the power consumption resulting from their use. This information may be used to pinpoint software that results in excessive power use. However, it was not very efficient as the data is not represented very expressively.

Therefore, we are using *PMBus* which Prof Sangit has recommended to us. It is very easy to use and already provided on our *Ultra96-V2*. First, we look at the rails we want to observe.



```
In [1]: import pynq

        rails = pynq.get_rails()
        rails

Out[1]: {'1V2': Rail {name=1V2, voltage=Sensor {name=1V2_voltage, value=1.199V}, current=Sensor {name=1V2_current, value=0.015A}, power
        =Sensor {name=1V2_power, value=0.0W}},
         '3V3': Rail {name=3V3, voltage=Sensor {name=3V3_voltage, value=3.296V}, current=Sensor {name=3V3_current, value=0.171A}, power
        =Sensor {name=3V3_power, value=0.5625W}},
         '3V3_D': Rail {name=3V3_D, voltage=Sensor {name=3V3_D_voltage, value=3.335V}, current=Sensor {name=3V3_D_current, value=0.0A},
        power=Sensor {name=3V3_D_power, value=0.0W}},
         'AUX': Rail {name=AUX, voltage=Sensor {name=AUX_voltage, value=1.796V}, current=Sensor {name=AUX_current, value=0.046A}, power
        =Sensor {name=AUX_power, value=0.03125W}},
         'INT': Rail {name=INT, voltage=Sensor {name=INT_voltage, value=0.851V}, current=Sensor {name=INT_current, value=0.203A}, power
        =Sensor {name=INT_power, value=0.15625W}},
         'PSAUX': Rail {name=PSAUX, voltage=Sensor {name=PSAUX_voltage, value=1.8V}, current=Sensor {name=PSAUX_current, value=0.156A},
        power=Sensor {name=PSAUX_power, value=0.28125W}},
         'PSDDR': Rail {name=PSDDR, voltage=Sensor {name=PSDDR_voltage, value=1.097V}, current=Sensor {name=PSDDR_current, value=0.078
        A}, power=Sensor {name=PSDDR_power, value=0.09375W}},
         'PSINT_FP': Rail {name=PSINT_FP, voltage=Sensor {name=PSINT_FP_voltage, value=0.851V}, current=Sensor {name=PSINT_FP_current,
        value=0.781A}, power=Sensor {name=PSINT_FP_power, value=0.65625W}},
         'PSINT_LP': Rail {name=PSINT_LP, voltage=Sensor {name=PSINT_LP_voltage, value=0.851V}, current=Sensor {name=PSINT_LP_current,
        value=0.296A}, power=Sensor {name=PSINT_LP_power, value=0.25W}},
         'PSPLL': Rail {name=PSPLL, voltage=Sensor {name=PSPLL_voltage, value=1.195V}, current=Sensor {name=PSPLL_current, value=0.028
        A}, power=Sensor {name=PSPLL_power, value=0.03125W}},
         'in1': Rail {name=in1, voltage=Sensor {name=in1_voltage, value=1.198V}},
         'in10': Rail {name=in10, voltage=Sensor {name=in10_voltage, value=1.796V}},
         'in11': Rail {name=in11, voltage=Sensor {name=in11_voltage, value=1.1V}},
         'in2': Rail {name=in2, voltage=Sensor {name=in2_voltage, value=1.792V}},
```

Figure 3.2.7a Get rails

Next, we can record the particular rails we want such as the processor, *PSINT_FP*.

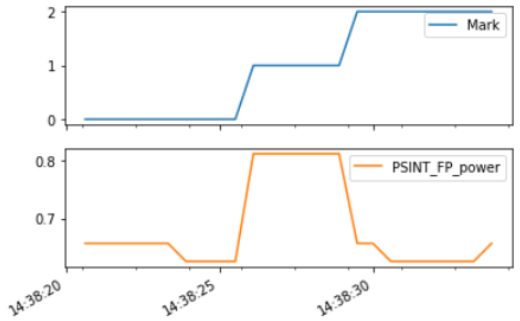
```
In [2]: if 'VSYS' in rails.keys():
            print("Recording Ultra96 v1 power...")
            rail_name = 'VSYS'
        elif 'PSINT_FP' in rails.keys():
            print("Recording Ultra96 v2 power...")
            rail_name = 'PSINT_FP'
        else:
            raise RuntimeError("Cannot determine Ultra96 board version.")
        recorder = pynq.DataRecorder(rails[rail_name].power)

        Recording Ultra96 v2 power...

        We can now use the recorder to monitor the applied sensor. For this example we'll sample the power every half second while sleeping and performing a dummy
        loop.

In [3]: import time
        with recorder.record(0.5):
            time.sleep(5)
            for _ in range(10000000):
                pass
            time.sleep(5)

        The DataRecorder exposes the sensor data as a pandas dataframe.

In [4]: recorder.frame
```

Figure 3.2.7b Record power

And we are done, we can use matplotlib to plot the power consumption over time.

```
recorder.frame.plot(subplots=True)
```

```
Out[6]: array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a083ac8>,
               <matplotlib.axes._subplots.AxesSubplot object at 0x7f79ff5860>],
              dtype=object)
```



This clearly shows the power spike when the for loop starts running.

Figure 3.2.7c Plot power consumption

To reduce power consumption, one obvious way would be to kill all other miscellaneous processes that are running on *Ultra96-V2*. As shown in the image below, there are indeed such processes which we can kill.

```
xilinx@pynq: ~/jupyter_notebooks/common
  826 ?        00:05:25 chromium-browse
  834 ?        00:00:07 jupyter-noteboo
  883 ?        00:00:12 kpktgend_0
  884 ?        00:00:13 kpktgend_1
  885 ?        00:00:11 kpktgend_2
  886 ?        00:00:16 kpktgend_3
  887 ?        00:00:00 chromium-browse
  891 ?        00:00:00 chromium-browse
  957 ?        00:00:12 chromium-browse
 1140 ?        00:00:16 chromium-browse
 1582 ?        00:00:00 python3.6
 3083 ?        00:00:01 chromium-browse
 9570 ?        00:00:01 kworker/u8:2-ev
11777 ?        00:00:05 kworker/2:1-eve
11880 ?        00:00:01 kworker/0:2-eve
12302 ?        00:00:00 kworker/3:1-eve
12701 ?        00:00:00 kworker/u8:1-ev
13692 ?        00:00:00 kworker/0:1
13751 ?        00:00:00 kworker/2:0-cgr
13784 ?        00:00:00 sshd
13812 ?        00:00:00 sshd
13813 pts/0    00:00:00 bash
13842 ?        00:00:00 kworker/1:2-eve
13848 ?        00:00:00 sshd
13876 ?        00:00:00 sshd
13894 ?        00:00:00 kworker/3:0H
13897 ?        00:00:00 kworker/1:1H
13912 ?        00:00:00 systemd-resolve
13913 ?        00:00:00 kworker/3:0
13925 ?        00:00:00 kworker/1:0-cgr
13926 ?        00:00:00 kworker/0:0H
13928 ?        00:00:00 kworker/2:2H
13937 ?        00:00:00 kworker/3:2H
13939 pts/0    00:00:00 ps
xilinx@pynq:~/jupyter_notebooks/common$
```

Figure 3.2.7d Unnecessary processes

Another method is to reduce unnecessary computations on the *Ultra96-V2* by reducing the time complexity of the algorithms or getting the *Beetles* to perform the calculations instead.

Another power optimization that we can perform is by modifying the synthesis and implementation strategy on Vivado itself. When selecting implementation options, we can enable power optimization to allow clock gating. We can also opt to use a lower clock frequency on the FPGA to reduce power consumption on the FPGA. Apart from that, we can also try to use the low performance slaves & master ports on the main processor of Ultra96.

We also can turn off unused cores and peripherals on the Ultra96.

We also can exploit SIMD intrinsics of the ARM processor to gain data level parallelism when preprocessing the data.

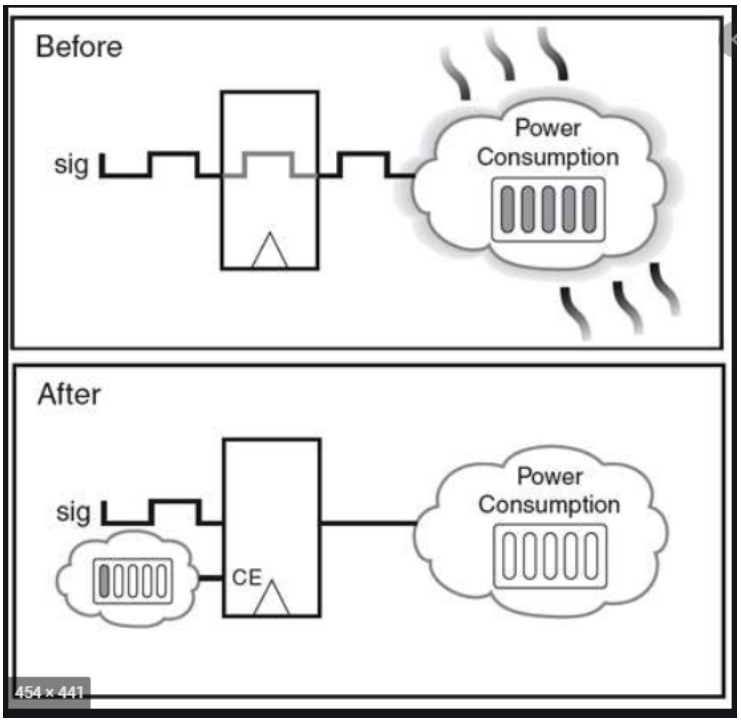


Figure 3.2.7e Clock gating

3.2.8 Implementation

For the implementation of the actual model on Ultra96, we chose to use MLP. This is due to the fact that MLP is lightweight compared to CNN and still can gain acceleration by running on the Ultra96, although being very negligible. The week 7 version can be found here.

We have various MLP architectures, one that has batch normalization, one that has 1 hidden layer instead of 2, etc. But we end up choosing a simple one with 2 hidden layers and up to 256

neurons per hidden layers. The model works by first reading the weights through DMA and initializing itself. Afterwards, we can send in the predictions and the FPGA will perform the prediction and output to the output buffer of the DMA. To read/write to DMA, we use the PYNQ library.

We also wrote a script to auto generate the initializing files from a Keras MLP model for the FPGA to initialize itself. We need such a script as there could be less than 256 neurons per layer and the weights and biases need to be zero-padded for the FPGA to work as intended. Such scripts are very trivial and explanation is omitted here.

We also wrote a driver for the interfacing of the initializing files generated by the script and to perform all necessary preprocessing from the raw data to the input of the DMA buffer. The driver API simply does the follows:

- Input: X number of raw data (acceleration x, acceleration y, acceleration z, yaw, pitch, roll) where X is the window size

- Output: Prediction (sidepump, logout, elbow kick, etc)

As for the optimization we done on the FPGA itself is very minimal apart from reducing resource utilization. This is due to the following reasons:

- FPGA power consumption is practically 0

- There is no reason to use FPGA given such a simple model (We are not allowed to use DNN)

- Activity detection period is significantly larger than prediction time

The power consumption of the FPGA is very low as the model we have implemented is very simple. The FPGA was designed to handle more advanced and complicated state of the art models that have hundreds of layers. On the other hand, in this module we are designing models that have a few layers. As such, unless a very inefficient model is constructed, there will be no way for the model to utilize much of the FPGA.

The bottleneck of the activity detection was never the model prediction in the first place. The architecture is dependent on how long it takes for the dancer to complete a single dance move. An average dancer takes a few seconds to finish a dance move. On the other hand, a prediction takes microseconds to complete. As such, one can see that the FPGA driver is not being called by the program 99.999% of the time and accounts for nearly 0% of the time spent in the program. And by Amdhal's law, the maximum theoretical speed up we can gain is 0%.

Similarly, due to the above mentioned reason, we can see why there is actually no need to perform any sort of power consumption optimization on the FPGA itself as the benefits to the overall system is practically non-existential.

Nonetheless, our group still optimized the power consumption by using specific directives to trigger Vivado to synthesise a reusable multiplier adder tree using Vivado HLS. We also used customized synthesis and implementation strategy on Vivado to generate the most optimal bitstream for the Ultra96. We also tried data preprocessing on the FPGA itself.

Feature extraction on the model is as follows:

- Maximum variance in acceleration over a window of time. As the dancer dances, there is a maximum and minimum acceleration for each dance move and it is likely to be different for each dance move

- Rate of change of yaw, pitch, roll. Each dance move will have their own rate of change of yaw, pitch, roll. For example, dab will induce a very sharp change in yaw, pitch, roll due to the property of dabbing.

- The raw values itself also differs across dance moves naturally and can be used as features

- Frequency is unable to be provide as a useful feature due to the dance moves

    - Each dance move takes a few seconds to complete, to reliably extract the frequency of the dance moves, the dancer would need to dance multiple times to allow us to perform fft

    - Such long dancing is not allowed as it takes too long, the evaluation server would have already timeout by that time

    - Rather than trying to extract garbage frequency data, it would be better to look into better model architectures

Figure 3.2.8a Dab

These features were also implemented on the FPGA together with the MLP. However, as there was a need to adjust the windows and features as time went by, it was moved back to the Ultra96 for the ease of modification. We did not use neon intrinsics explicitly because the Python

interpreter and the Numpy module does make use of instruction level parallelism in their libraries, so there is no need for us to reinvent the wheel and write the SIMD algorithms ourselves. Of course, we cannot deny the benefits of manually writing ourselves to customize according to our needs, the benefits are not worth the cost and time is better spent on improving the ML.

Section 4 Firmware & Communications Details

Section 4.1 Internal Communications

This section details the internal BLE serial communication between the Bluno Beetle unit on each user wearable and laptop.

4.1.1 Task management on Bluno Beetles

In order for each Beetle unit to simultaneously read sensor data and maintain BLE communication with the laptop, 2 tasks need to be run concurrently.

1. BLE communication task
    a. Handles handshaking protocol upon establishing connection with the laptop either on system start-up or reconnections
    b. Sends sensor data to laptop
2. Sensor task - handles data from hardware sensors
    a. Extracts sensor data from accelerometer
    b. Extracts EMG values from EMG sensor
    c. Determines dance state of the user (IDLE or DANCING)

As true multithreading is not possible on the single-core ATmega328, we need alternative methods to run both tasks at the same time. Originally, we considered using Arudino libraries capable of emulating multithreading on the beetle. One is the use of protothreading through the 'ArduinoThread' library, and the other is by setting up an RTOS environment on the microprocessor utilising the 'FreeRTOS' library.

In the end, we determined that the most straightforward implementation is usually the most reliable, and went with a simple round robin approach. Both tasks are handled in the main loop, where the sensor values are read and sent to the laptop at the same time at a fixed interval. The interval determines the rate at which data is being sent, with a frequency set to 20Hz. The main loop also polls the RX register on the beetle in order to execute the handshake protocol when required.

The simple round robin approach without scheduling keeps the task management implementation on the Beetles simple and reliable, without the need for additional libraries and overcomplicated code.

4.1.2 BLE Interface Between Beetle and Laptop

The Beetle unit on the wearable system would communicate with a laptop through a BLE interface, with the Beetle acting as the peripheral device and the computer acting as the central device. The diagram below depicts the architecture diagram of the BLE connection.

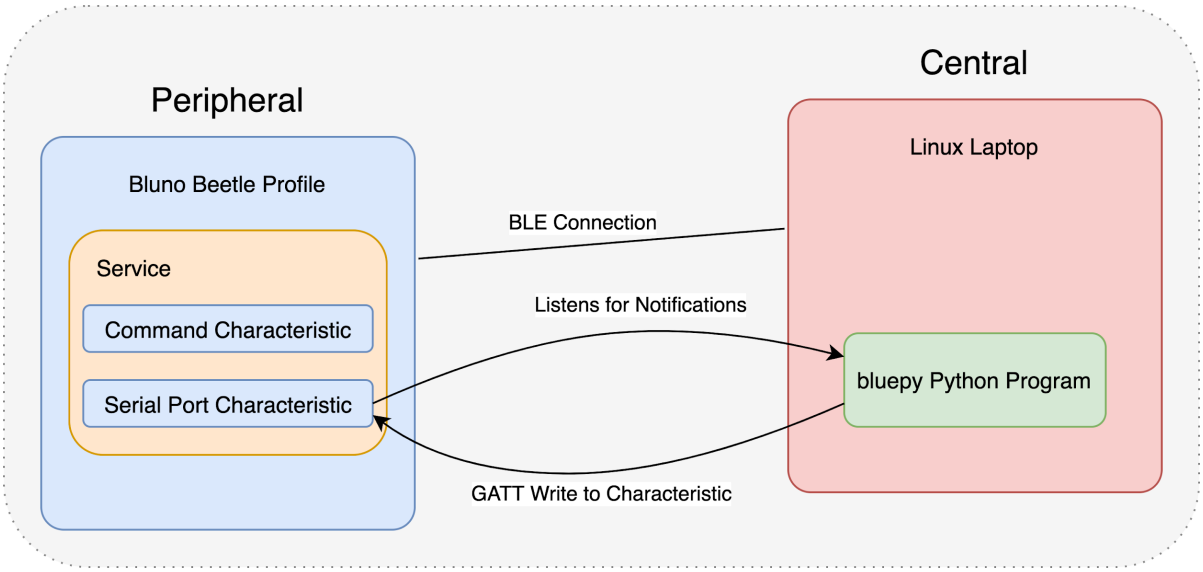


Fig 4.1.2a BLE Interface Diagram

To ensure smooth communication between laptop and beetle using the aforementioned BLE interface, configuration is to be on both sides.

4.1.2.1 Bluno Beetle Configuration

For the on-board BLE firmware settings, the default AT setting is followed. Packet transmission of handshake commands and data between beetle and laptop is handled by the BLE task running on the ATmega328 uploaded by Arduino Uno.

4.1.2.2 Linux Laptop Configuration

A Linux environment is set up on the laptop to manage the BLE connection. PyPI's 'bluepy' package is used to discover and form a BLE connection with the beetle upon system set up, after which the handshaking protocol takes place (further elaborated in the section 4.1.3).

Originally, we attempted the BLE connection using the PyPi package 'Bleak' due to the lack of computers with a Linux OS within the team. Alternative approaches were considered through the use of VMware fusion for macOS and WSL for windows. However, BLE support is not

included for VMware fusion on macOS and WSL does not support direct access to Windows local bluetooth hardware. As such, Bleak's platform agnostic nature allowed for the API to be used across multiple platforms, including macOS and Windows. However, the BLE Clients for CoreBluetooth on the macOS and WIndows 10 systems were not able to reliably subscribe to characteristic notifications on the Beetle.

As a result, we opted to use the tried and tested Linux based 'bluepy' Python package.

4.1.2.3 Data Transmission between Beetle and Laptop

The BLE profile on the Beetle houses 2 services, one for device information and another for communication. As displayed in the diagram, the service for communication has 2 characteristics, with the serial port characteristic being used for data transmission.

Commands sent from laptop to beetle are done in the form of commands sent by performing a write operation from the laptop onto the serial port GATT characteristic of the beetle.

The laptop would subscribe to the serial port characteristic for notifications to receive commands and sensor data, which will be in the form of command and data packets. The format of these packets are described in the sub-section below.

4.1.3 BLE Protocol for Data Transmission

To ensure smooth data transmission through the BLE link between laptop and beetle, a transmission protocol is agreed upon by both sides.

This protocol is defined by the following design components:

- Baud Rate
- Packet Types
- Command Packet Format
- Data Packet Format
- Handshaking Protocol
- Data Transmission Protocol

4.1.3.1 Baud Rate

With both the default baud rate on the hardware sensors and beetle being 115200, we opted to maintain our system baud rate at this default value for both fast and reliable data transfer over BLE that is standardised with hardware sensors.

4.1.3.2 Packet Types

Two types of packets are used in the BLE communication protocol:

1. Command Packet
2. Data Packet

Command packets are used to facilitate the Handshaking protocol between laptop and beetle during the initial connection on system set-up and system reconnections in cases of BLE disconnections.

Data packets are used to contain sensor data, which are sent from beetle to laptop during serial communication after establishing the BLE link.

The format of these 2 packets are expanded upon in the sub-section below.

4.1.3.3 Command Packet Format

Fig 4.1.3.3a Command Packet Format

The command packet sent between laptop and beetle has the length of 1 byte, fitting nicely into the size of the serial port on the beetle, and follows the format above.

The command type will be interpreted from the 3-bit packet type field. The table below shows the possible command types to be sent according to the packet type.

Packet Type	Packet Code
ACK	0
NAK	1
Hello	2
Read	3
Write	4
Data	5
Error	6
Awake	7

Only Hello and ACK commands are used in the BLE connection to facilitate the handshake protocol.

Each command will be verified on both sides during communication with a bitwise XOR operation on the checksum field with the predetermined checksum value, adding an additional level of reliability during command packet transmission.

4.1.3.4 Data Packet Format

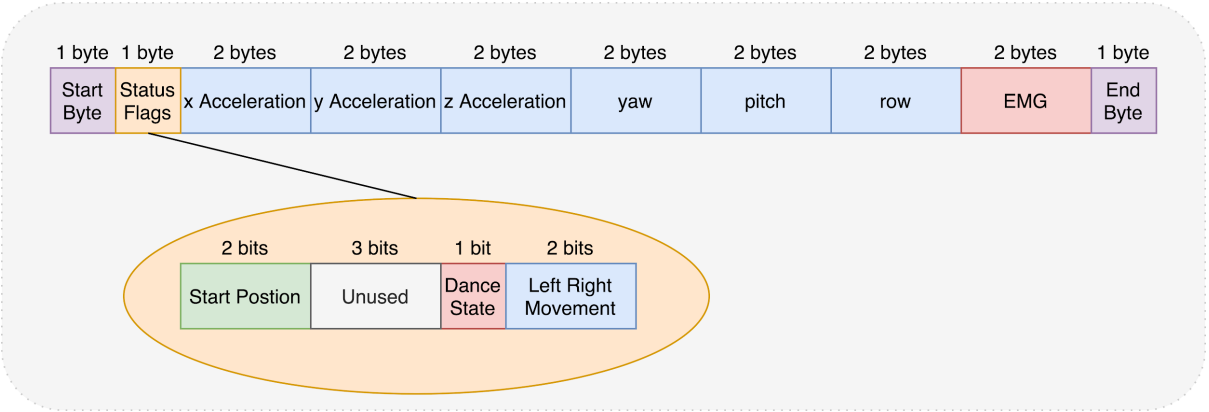


Fig 4.1.3.4a Data Packet Format

Sensor data is sent from beetle to laptop in the above format, with each transmission frame maintaining a small size of 17 bytes.

Byte stuffing is used as framing, with each frame being delineated with the start byte of 0xAC and end byte of 0xBE. The start and end bytes are used to ensure robust communication in the data transmission protocol, to be elaborated further in this section.

Status flags byte contains the following flags that represent the state of the user:

- Start Position: Initial dance position of the user (first, second, third)
- Dance State: Detects if the user is dancing (1) or IDLE (0)
- Left Right Movement: Details if the user is moving left (1), right (2) or stationary (0)

Data from hardware sensors is stored in the data packet in the above format, with the accelerometer data highlighted in blue and EMG sensor data highlighted in red.

4.1.3.5 Handshaking Protocol

The 3-way handshaking protocol is carried out between laptop and beetle to establish a stable connection before serial communication.

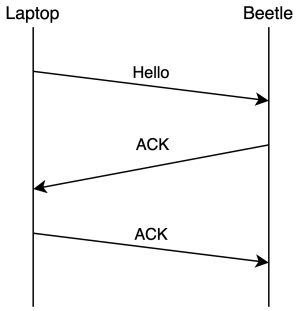


Fig 4.1.3.5a Handshake Timing Diagram

It is carried out by Hello and ACK commands exchanged between laptop and beetle, as depicted in the timing diagram shown above.

With the laptop acting as the central device, it initiates the handshaking protocol. The laptop scans for the beetle with its MAC address forms a BLE connection before starting the handshaking process. Implemented as a recursive function on the laptop side, the connection and handshaking process is depicted in the flow diagram below.

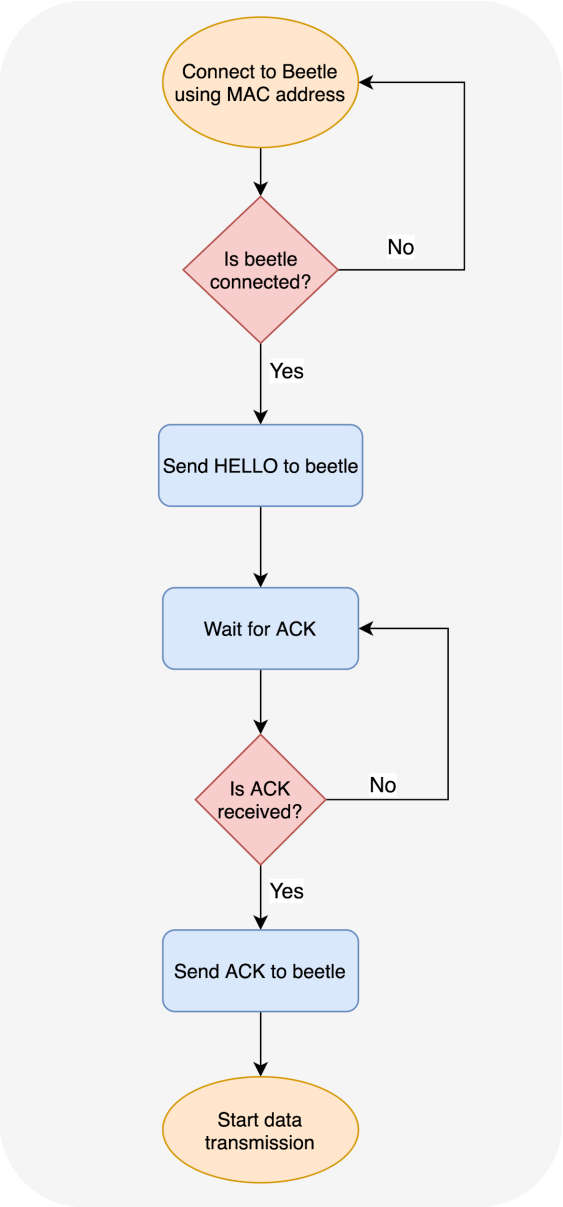


Fig 4.1.3.5b Handshaking Flow Diagram

Only when the beetle receives the ACK from the laptop will it start to transmit data. Hence, data transmission will only begin once the handshake is completed, ensuring both parties are ready before data communication between laptop and beetle begins.

4.1.3.6 Data Transmission Protocol

Upon completion of the handshaking protocol, consistent data transfer occurs from beetle to laptop in the form of data packets. Making use of the start and end bytes of the data packets, the protocol for data transmission is implemented through a custom byte buffer system on the laptop.

Two data structures are for the custom byte buffer system:

1. Buffer - byte array
   Stores all incoming data packets from beetle

2. PacketBuilder - deque
   Reconstructs bytes within buffer into data packets
   Maintained at 17 bytes

As the length of each data packet is at a constant of 17 bytes, we used both byte stuffing and byte counting in the process detailed below to effectively receive and reconstruct incoming data packets.

Data Framing Process:

1. Data received from beetle is appended into Buffer

2. Bytes from Buffer are popped and inserted into packetBuilder until it reaches 17 bytes

3. Checks if the first byte in packetBuilder is the start byte:

    a. If yes: proceed to step 4

    b. If no: discard byte and repeat step 3

4. Check if the 17th byte is the end byte:

    a. If yes: packet is successfully reconstructed

        i. Output packet

        ii. Clear packetBuilder

5. Repeat when packetBuilder reaches 17 bytes again

The custom buffer allows packets to be built even if the BLE internal buffer overflows, creating a robust data transmission protocol.

4.1.4 BLE Protocol Reliability and Robustness

To ensure robust BLE serial communication, the system is designed to provide reliable data transmission, even in cases of data corruption, packet loss or connection disruptions.

4.1.4.1 Byte Stuffing and Counting in Data Transmission

Originally, data transmission from beetle to laptop was handled by the BLE internal buffer on the laptop. However, the buffer failed to reliably receive data at higher frequencies, with bytes overflowing to the next buffer once frequency goes above 20Hz.

This drawback was circumvented through the use of byte stuffing and counting in the custom buffer system as detailed in the sub-section above, which allows data packets to be reconstructed consistently even if the BLE internal buffer overflows. A deque was also implemented as the data structure for the packetBuilder, as it allowed for bytes to be popped and inserted in O(1) time. As a result, BLE data transmission became capable of hitting much higher frequencies while remaining code efficient, with stress tests proving that it could receive transmissions at up to 100Hz.

However, we opted to keep the rate of data transfer at 20Hz to put less stress on both the hardware sensors and BLE connection in favour of stability, providing a reliable data transmission protocol from beetle to laptop.

4.1.4.2 Removal of CRC Check from Data Transmission

The first design of the data packet included a CRC field calculated with a 9 bit polynomial to yield an 8-bit CRC for maximum data corruption detection that can fit within the one byte.

However, the beetle uses Bluetooth 4.0, which has an integrated 16-bit CRC check. Hence, we decided to cut down on computation during data transfer to improve performance while eliminating double work, improving reliability during data transfer.

4.1.4.3 Checksum Implementations in Command Packets

Command packets are sent during handshaking, with a 5-bit checksum implemented in the command packet.

This checksum allows data corruption to be detected in command packets, adding an extra layer of security during handshaking.

4.1.4.4 Multiple Laptop Set-up vs Multithreading

A multithreading set up on the laptop in preparation of the full system integration with 3 concurrent beetle connections from each dancer. This was done to both simplify the coordination required to initialize the system as well as reduce the number of laptops needed that house a Linux OS.

However, we discovered during integration testing that a multithreading approach placed greater strain on a single laptop which as a result, negatively impacted reliability. Therefore, opted for a multiple laptop setup, where 3 separate Linux laptops were used to each maintain a BLE connection with a single beetle. While this caused more coordination on our part to initialize the system, we were able to ensure more reliable BLE connections to our wearables.

4.1.4.5 Connection Loss Protocol

Consistent BLE connection throughout the system runtime is ensured by the laptop provided the user remains within the working radius of BLE connection to the laptop. In cases of connection loss, the bluepy program detects the disruption and automatically scans for the disconnected beetle unit. When the unit is rediscovered, the laptop would initiate the handshake protocol again to resume communication.

As the packets in flight before connection loss are not properly handled by the data transmission protocol, there is a chance of incorrect data being received by the laptop upon reconnection. Hence, the first 100 packets following reconnection are discarded to main accurate data even in cases of disconnection.

Section 4.2 External Communications

4.2.1 Overview

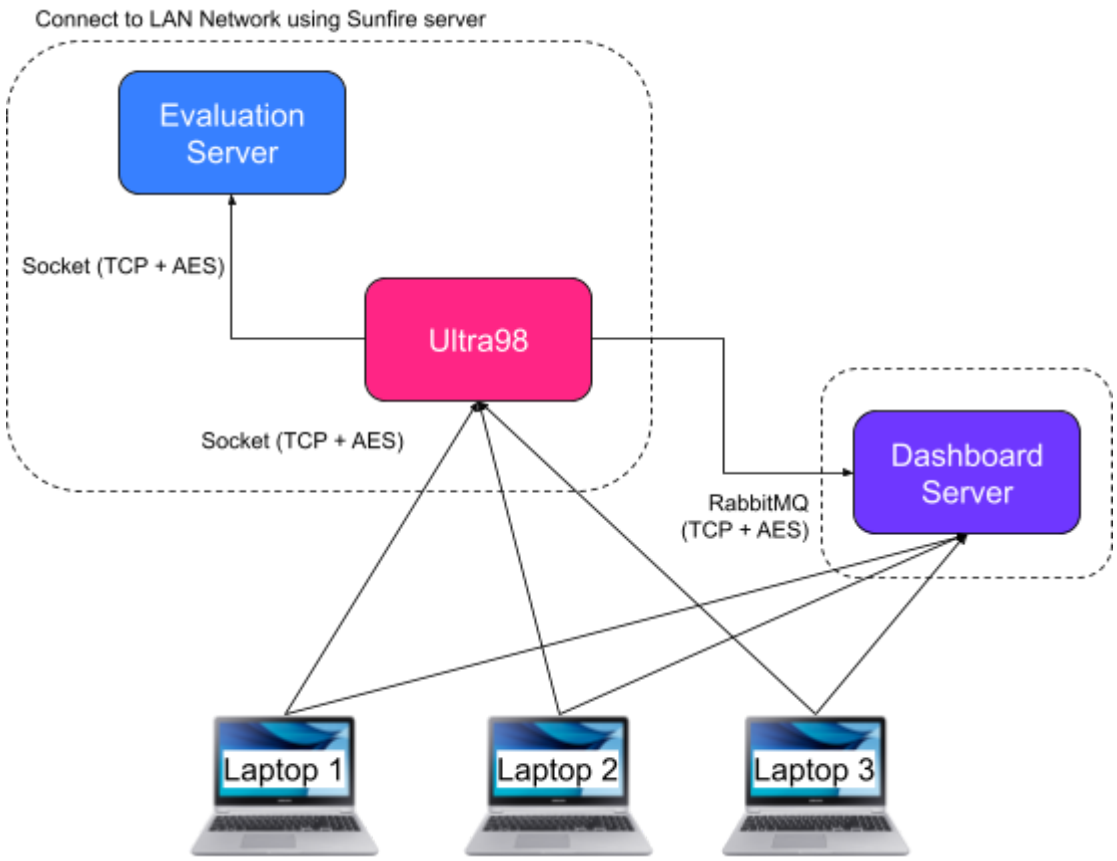


Figure 4.2.1a. Architecture diagram for external communications

The laptops, Ultra96 and end-point servers (evaluation and dashboard) communicate using TCP. All nodes use sockets to communicate other than the communication to the dashboard which uses MQTT. Since the Ultra96 is behind a firewall, we used ssh tunneling to connect to it from our laptop. We first tunnel from the laptop to an intermediary host, Sunfire, and then tunnel from Sunfire to the Ultra96. To achieve secure communication, the data packets sent between nodes were encrypted before being sent. The encryption protocol used was AES-128 in CBC mode.

For external communications, the Ultra96 will only run a single process, but use threads to run multiple tasks. A total of five threads will be used, three to communicate with the three laptops for the dancers, one to communicate with the evaluation server, and the last to communicate with the dashboard.

The synchronization delay between dancers is taken to be the time difference between when the fastest dancer and slowest dancer in starting their dances. The time difference is calculated using the timestamps recorded on the dancers' respective laptops when they start to dance. Because there can be latency when this data is sent to the Ultra96, and that the laptop clocks are not in sync with the Ultra96 clock, we perform a clock synchronization protocol at the start to sync the laptop clocks to the Ultra96 clock.

The architectural diagram for external communications is as seen in Figure 4.2.1a above.

4.2.2 Socket Communication

We establish sockets that will allow communication via Transmission Control Protocol (TCP) between the different nodes.

TCP is a suitable protocol to use because it is reliable. First, packets lost in transmission across the network are detected and retransmitted by the sender. This error recovery process is called Positive Acknowledgement with Retransmission (PAR) or Automatic Repeat ReQuest (ARQ). Second, packets are received in the same sequence they were sent. This ensures reliability and integrity in data transfer.

Socket communication scripts will be done in python using the socket module. The sequence of socket API calls by a socket server/ client to establish connection and receive/ send data is shown in Figure 4.2.2a below. The buffer for the connection is defined in *recv( )* and was set to an adequately large number to prevent packets from being dropped. Socket communications will be closed when the entire dance segment is finished and the final dance move is detected.

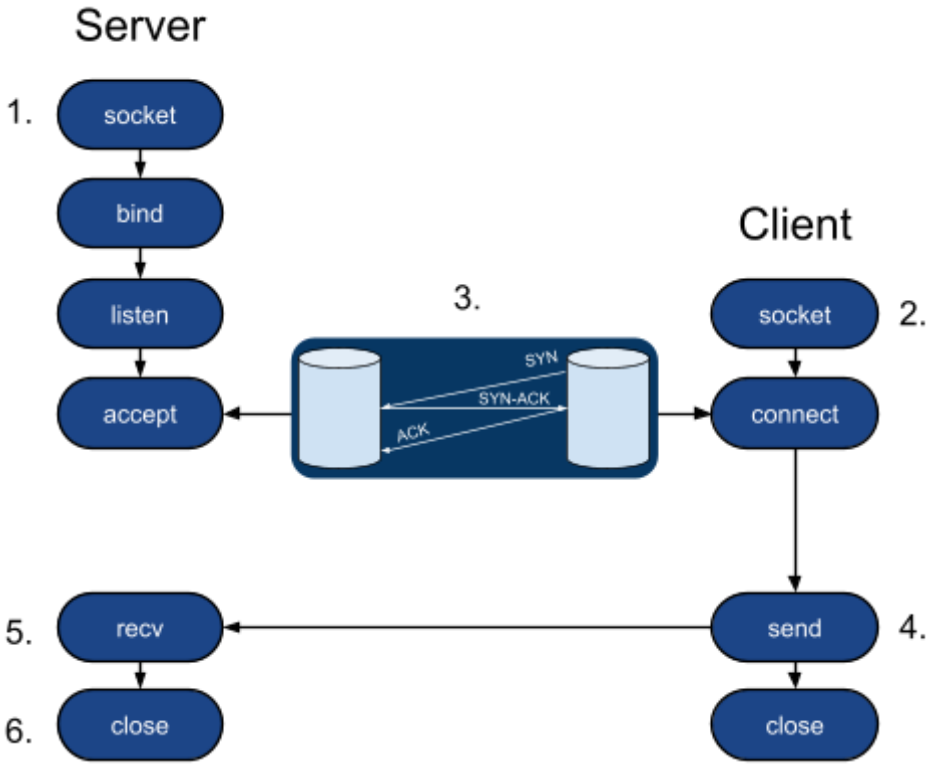


Figure 4.2.2a. Sequence of socket API calls by socket server and client with labels as follows:

1. *Server listens for connection from client with the API calls socket(), bind(), listen(), and accept()*
2. *Client connects to server with the API calls socket() and connect()*
3. *Client initiates three-way handshake and both sides synchronize (SYN) and acknowledge (ACK) each other, exchanging the SYN, SYN-ACK and ACK flags*
4. *Client sends data to server with API call send()*
5. *Server receives data from client with API call recv()*
6. *Server and client close the socket connection using API call close()*

4.2.2.1 Laptops to Ultra96

The bastion host used to access the Ultra96 is Sunfire which is on the same subnet as the Ultra96. The command for ssh tunneling is shown in Figure 4.2.2.1a below. The local port 1234 on the client is forwarded to port 1234 of the destination remote server.

```
ssh -t -L 1234:127.0.0.1:1234 <user>@sunfire.comp.nus.edu ssh -t
-L 1234:127.0.0.1:1234 xilinx@137.132.86.238
```

Figure 4.2.2.1a. Command used to set up ssh tunnel from laptop to xilinx via Sunfire

4.2.2.2 Ultra96 to Evaluation Server

The design for communications between the Ultra96 and the evaluation server was largely dictated by the eval_server.py script provided. The Ultra96 sets up a socket client to communicate with the socket server script running on the evaluation server. This link was configured to listen to the port number 5000. The Ultra96 sends the position, dance move and synchronization delay of the dancers to the evaluation server when these information is ready. The data packets sent were formatted as such: #<position>|<action>|<sync>.

4.2.3 Message Queue

In our initial stages, we realised that transmission performance using socket to the dashboard was poor and not all data was processed on time, causing packets to be dropped. Thus, we implemented a message queue to circumvent this problem. This allowed the data to add requests from the other nodes to a queue and not have to instantaneously process them, forcing other incoming packets to be dropped. This was done using the python module *pika*. The code used to set up the connection and send data is as shown below in Figure 4.2.3a.

```
import pika

rabbitParams = pika.URLParameters(<dashboard url parameters>)
pika_conn = pika.BlockingConnection(rabbitParams)
pika_channel = pika_conn.channel()
pika_channel.queue_declare(queue=queue_name, durable=True)
pika_channel.basic_publish(exchange='', routing_key=queue_name,
body=massage)
```

Figure 4.2.3a. Code snippet to establish connection to message queue, and send messages

4.2.3.1 Ultra96/Laptop to Dashboard Server

The Ultra96 and laptops communicate to the dashboard server using Advanced Message Queuing Protocol (AMQP). The message broker used was RabbitMQ. There were two queues used, one for raw data and one for processed data. To reduce transmission latency, only processed data was sent from the Ultra96 to the dashboard while raw data was sent from the laptops directly. The port for communications was configured to listen on port 3000. Processed data include dance move classification and position of the dancers. Data packets carrying processed data were formatted as such: #<accelerometer data>|<gyroscope data>|<EMG readings>|<action>|<sync>|<dancer>. Raw data includes the x, y and z direction/ axis respectively from an accelerometer and gyroscope, and also EMG readings. Data packets carrying processed information were formatted as such: @<x, y, x>|<row, pitch, yaw>|<emg> |<dancer id>|<timestamp>.

4.2.3 Processes on Ultra96

For external communications, there will be only one process running on the Ultra96. However, this will be a multithreaded process. A total of five threads were used. For a thread to run, they must obtain the lock needed to perform create, read, update or delete operations on shared data between the locks. Three threads are used to create a socket link to each of the three laptops. Two threads will be used to establish socket clients to the evaluation and dashboard servers. The biggest benefit to using threads is that they can access shared data, and thus the operational cost of communication between threads is low, which helps limit power usage on the Ultra96.

We used a data lock to ensure process synchronization. There were various scheduling methods we considered implementing to determine which thread should execute. To name a few: first-come first-serve, shortest-job-first scheduling. We want to maximise throughput and minimise turnaround, waiting and response time. This is to ensure the responsiveness and flow of the system. We decided on using round-robin scheduling to meet these goals. The benefits of using round-robin was its simplicity to implement and that it will ensure all threads have a chance to run thus preventing thread starvation. The data lock helped regulate thread access to critical sections. Before each thread tries to execute, it will try to acquire the lock and continue with execution if it is successful. Otherwise, it goes into a blocked state and waits for the lock to be free. Thereby we should not encounter race conditions where two threads access and manipulate any shared data.

Multithreading and process synchronisation using the python threading module which has an inbuilt Lock class. The shared data between the threads was stored in a queue which received all the dancer data from the three laptops. This queue was implemented using the python queue module.

4.2.4 Encrypted Communication

To protect our data from unauthorised users on the network, we encrypted the data using Advanced Encryption Standard (AES) in cipher block chaining (CBC) mode. The cipher is then encoded in Base64 before transmission. All nodes in the network will hold the same encryption key that will be used to encrypt messages or decrypt cipher data packets.

We used the pycryptodome python module to encrypt our data packets. A sample code is shown in Figure 4.2.4a below.

```
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Util.Padding import pad
import base64

def get_cipher(message):
        iv = get_random_bytes(16)
        aes = AES.new(key, AES.MODE_CBC, iv)
        message += ' '*(16 - (len(message) % 16))  # add padding
        cipher = aes.encrypt(message.encode("utf8"))
        cipher = base64.b64encode(iv + cipher)
         cipher += bytes(' '*(16 - len(message) % 16), 'utf8')  #
add padding
        return cipher
```

Figure 4.2.4a. Code to perform AES-128-CBC encryption on input message

4.2.4.1 Key Generation

AES is a cipher that allows flexibility in the length of the encryption key used. The key sizes are either 128, 192 or 256 bits. While longer keys provide stronger encryption, it could add to delay in encrypting and decrypting data packets sent between nodes. In addition, the encryption key needs to be manually entered into the evaluation server. Thus, a suitable key size would be 128 bits.

The secret key is generated using a pseudorandom generator and made known to relevant group members only. This is important as if the adversary is able to obtain the secret key, the communication channel will no longer be secure. Using a non random secret key such as “password” will make our system vulnerable to dictionary attacks.

Key generation will be done using the pycryptodome module in python. The utility get_random_bytes() returns a random byte string of desired length. Since our key length is 128

bit, we require a 128/8=16 bytes length key. The code for key generation is as shown in Figure 4.2.4.1a below.

```
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Util.Padding import pad
import base64
def get_key():
    key = Random.get_random_bytes(16)
    return key.decode('utf-8')
```

Figure 4.2.4.1a. Use of get_random_bytes to generate 16 bytes length key, whereafter the bytes are decoded to a string using python native utility, decode

4.2.4.2 Block Size and Padding

Because AES uses a block size of 16 bytes, data that we encrypt needs to be of a length that is a multiple of 16. Data that does not fulfill this criteria needs to have padding added to achieve this block size.

Padding will be done using the pycryptodome module in python. The utility pad() applies standard padding to an input message, thereby generating a byte string that will be a multiple of desired block size. It defaults to the PKCS#7 padding standard. A usage of this can be observed in Figure 4.2.4a in Section 4.2.4.

4.2.4.3 Mode of Operation

The mode of operation for a cipher determines how to repeat single-block operation on large amounts of data, whereby the size of the data is larger than the cipher block size. Different modes of operations present different encryption protocol characteristics such as integrity and confidentiality. Integrity ensures that data transmitted is unchanged, or detectable if tampered with, while confidentiality ensures that data transmitted is not accessible by unauthorised parties.

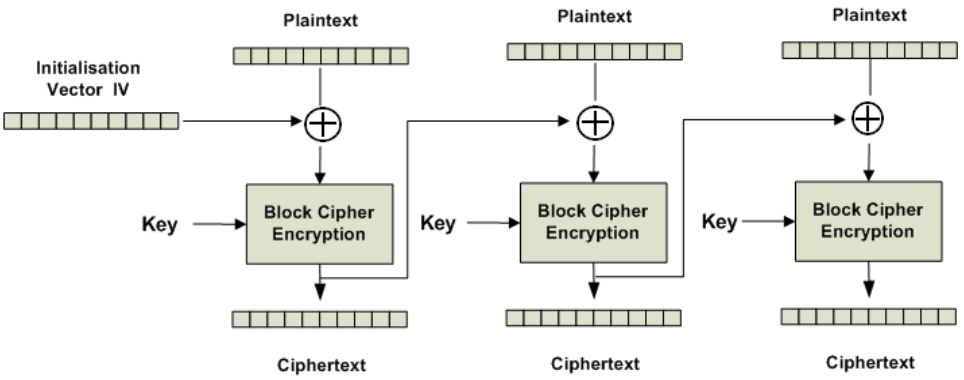


Figure 4.2.4.3a. Flow of Cipher Block Chaining (CBC) mode encryption

For continuity, the mode of operation will follow the decryption protocol provided in eval_server.py, which is Ciphertext Block Chaining (CBC). AES-CBC as illustrated in Figure 4.2.4.3a above. AES in CBC mode splits the data into blocks of 16 bytes. Then, each block is encrypted using AES and the result is used to XOR with the next data block before the new block is encrypted. The initialization vector (IV) serves as a nonce to add randomness to the encryption and thereby semantic security.

CBC is a mode of evaluation that ensures the confidentiality of the data but not its integrity. It is secure against chosen-plaintext attacks but vulnerable to chosen-ciphertext attacks. Ciphers can be modified through various attacks like padding oracle attack or byte-flipping attack. This cryptographic algorithm is highly malleable and changing the IV or ciphertext can cause decryption to fail.

Outside the scope of this project, there exists better modes of operations that ciphers utilise that can provide both confidentiality and integrity. Some alternatives include AES-GCM, AES-CCM, AES-OCB, and AES-CBC with HMAC-SHA256.

AES encryption in CBC mode will be done using the pycryptodome module in python. An example code usage can be seen in the previously mentioned Figure 4.2.4a in Section 4.2.4.

4.2.5 Synchronization Delay Protocol

In the scope of this project, for the dancers to be in sync, they need to start dancing at the exact same time. However, this is unlikely. To calculate the delay between the dancers and send it to the evaluation server we need a way to measure their different start times against a common starting point. While it will be ideal to calculate this difference directly from the Beetle, because the Beetle is capable of only sending data over bluetooth and not wifi. Thus asynchrony between

dancers will be addressed using Network Time Protocol (NTP) estimation to calculate clock offset.

NTP is a networking protocol for clock synchronization between computers and is used to synchronize them against the Coordinated Universal Time (UTC). It is commonly used to offset network latency. The NTP client and server send and receive timestamps to calculate network delay and clock offset, whereby the client matches the server clock. The clock on the Ultra96 should be first synchronized using the NTP Daemon to match the Internet standard time.

To synchronize the three beetles, a modification of the NTP was used. Calculation of delay was done at the start of system boot, during initialisation. Note that the sensors reading dance moves are unable to be synchronized with the Ultra96 clock over the internet. This is because the Beetle is only capable of BLE. Thus, when sending sensor data over to the Ultra96, the data packet needs to carry the time stamp for each reading.

During the initial testing stage, it was found that the timing for data packet transmission from the Beetle to the laptop is negligible. Thus it is accurate enough to take the timestamp from the laptop to the Ultra96. The timings needed for clock offset and network delay calculation is illustrated in Figure 4.2.5a below.

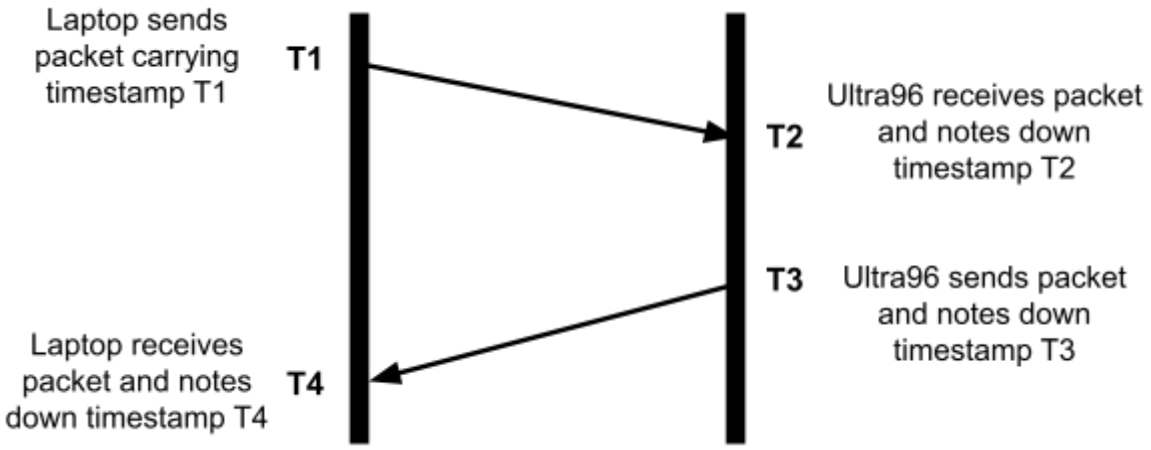


FIgure 4.2.5a. Timings needed for clock offset and network delay calculation, with calculations as follows:

Round Trip Time (RTT) = T2-T1-T3-T4
One-way delay = RTT/2
Clock offset = T2-T1-RTT/2

Here, we use the Ultra96 as the baseline. After finding the clock offset for each Beetle, we then determine the start of the dance move with respect to the Ultra96 clock. The synchronization delay was calculated as the time difference between the fastest and slowest dancer.

Section 5 Software Details

Section 5.1 Software Machine Learning

5.1.1 Objective

For machine learning part in this project, we aim to build a Machine learning model with signal processing pipeline in offline mode to process signals collected from inertial sensors (accelerator and gyroscope) and then produce useful datasets will be used as inputs of a machine learning model capable of recognizing some of human dance activities (hair, side-pump, wipe-table, …) with a low error rate.

5.1.2 Dataset and Inputs

The experiments will be carried out for every member in the group (6 people). The number of experiments will depend on how our Machine learning model performs, meaning if our model accuracy is not well, and have a tendency for under-fitting, we will collect more data (conduct more experiments to collect data).

All members use a wearable with embedded sensors (accelerator and gyroscope). The data will be collected and written in a text file. The dataset contains 3-axial linear acceleration and 3-axial angular velocity. The label will be attached manually based on their dance moves. The data collected at this stage will be stored in the same folder, named raw data.

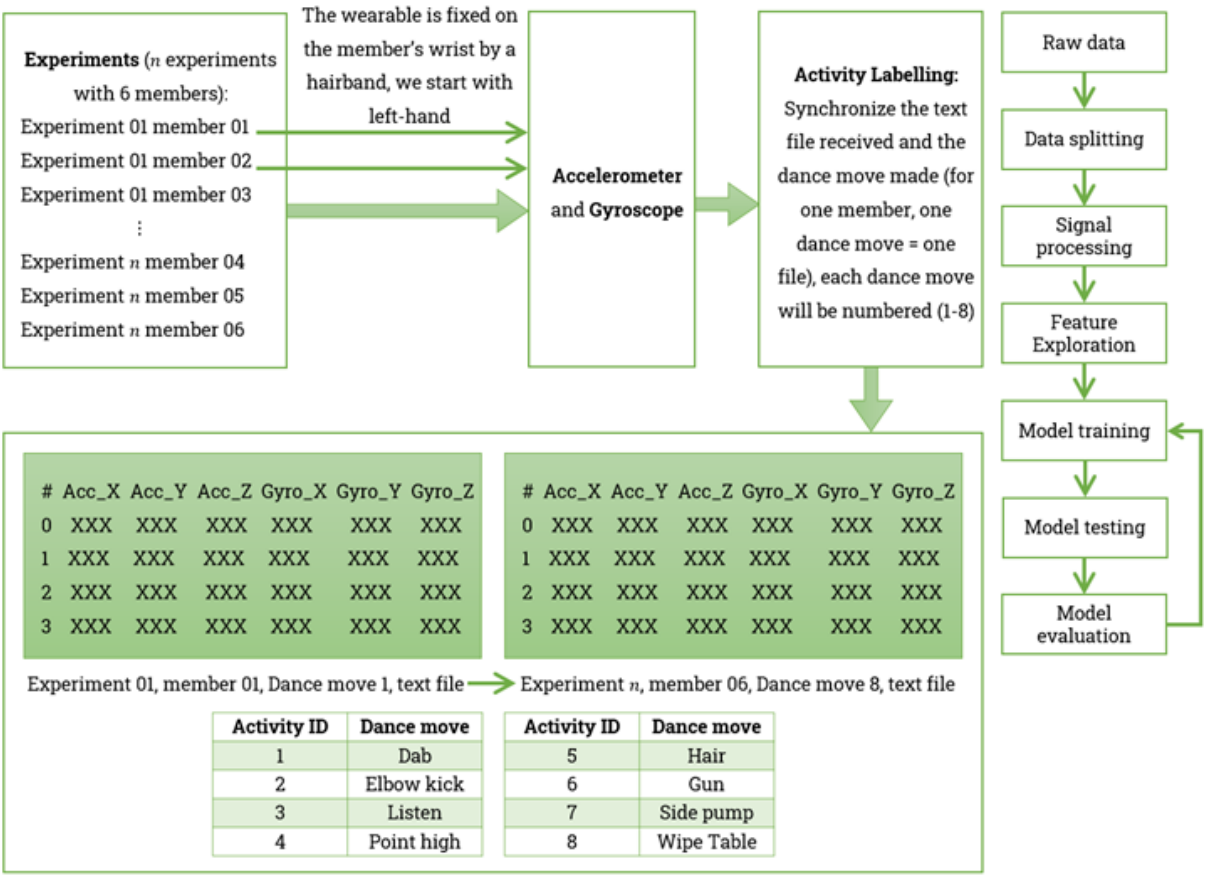


Figure 5.1.a: Experiment procedure and applying Machine Learning to train data procedure

5.1.3 Data Segmentation

From the raw data, we collected, we will randomly partition the whole dataset into 2 subsets: 70% for training data and 30% for test data.

The data processing procedure is shown as below:

Figure 5.1.b: Data processing procedure

5.1.3.1 Signal Processing

In every experiment, we cannot avoid noise. Hence, for the next step, we will apply noise filtering for the data features (coming from the accelerometer and gyroscope 3-axial signals).

Noted that exponential filtering is applied in hardware sections, it is still encouraged to proceed with other filters.

```
def apply_filter(signal, filter="median", window=3):
    """A denoising filter is applied to remove noise in signals.
    Args:
        signal (numpy array 1D (one column)): Raw signal
        filter (str, default='median'): Filter name is chosen from 'mean', 'median', or␣
↪'butterworth'
        window (int, default=3): Length of filter
    Returns:
        signal (pd.DataFrame): Filtered signal
    See Also:
        'butterworth' applies a 3rd order low-pass Butterworth filter with a corner frequency of␣
↪20 Hz.
    """
    if filter == "mean":
        signal = signal.rolling(window=window, center=True, min_periods=1).mean()
    elif filter == "median":
        signal = signal.rolling(window=window, center=True, min_periods=1).median()
    elif filter == "butterworth":
        fc = freq2  # cutoff frequency
        w = float(fc)/(sampling_freq/2.0)  # Normalize the frequency
        b, a = butter(3, w, "low")  # 3rd order low-pass Butterworth filter
        signal = filtfilt(b, a, signal, axis=0)
    return signal
```

5.1.3.1.1 Normalize data

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.

This is **the most important step** in this stage as the range of accelerator and gyroscope reading is very different, so we want to "treat each column in one instance equally" by normalizing the data.

For example, assume your input dataset contains one column with values ranging from 0 to 1, and another column with values ranging from 10,000 to 100,000. The great difference in the scale of the numbers could cause problems when you attempt to combine the values as features during modeling.

Normalization avoids these problems by creating new values that maintain the general distribution and ratios in the source data, while keeping values within a scale applied across all numeric columns used in the model.

In this project, we will use StandardScaler from sklearn library. Standardization is a scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

The formula of normalizing data is shown as below:

$$X' = (X - \mu)/\sigma$$

μ(mu) is the mean of the feature values and σ (sigma) is the standard deviation of the feature values.

After the signals are normalized, we will pass these values through filters.

5.1.3.1.2 Median Filter

The median filter is a non-linear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing.

Median filtering is very widely used in digital image processing because, under certain conditions, it preserves edges while removing noise, also having applications in signal processing.

In this experiment, this is applied to remove the background noise.

Working Description

The main idea of the median filter is to run through the signal entry by entry, replacing each entry with the median of neighboring entries. The pattern of neighbors is called the "window", which slides, entry by entry, over the entire signal. For one-dimensional signals, the most obvious window is just the first few preceding and following entries

To demonstrate, using a window size of three with one entry immediately preceding and following each entry, a median filter will be applied to the following simple one-dimensional signal: x = (2, 3, 80, 6, 2, 3).

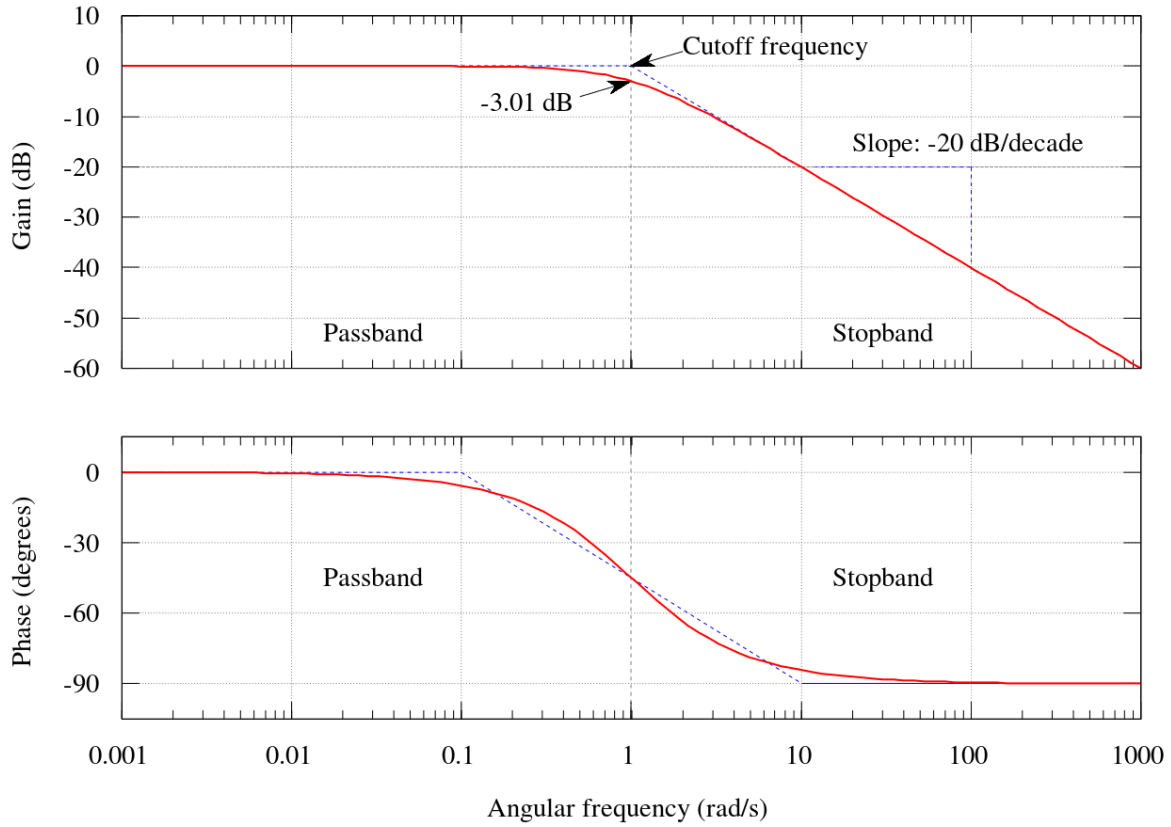
So, the median filtered output signal y will be:
y1 = med(2, 3, 80) = 3, y2 = med(3, 80, 6) = med(3, 6, 80) = 6,
y3 = med(80, 6, 2) = med(2, 6, 80) = 6, y4 = med(6, 2, 3) = med(2, 3, 6) = 3, i.e. y = (3, 6, 6, 3).

5.1.3.1.3 3RD Order Low pass Butterworth filter

This filter is defined with a cutoff to remove high frequency noise, the cut-off frequency will be decided later, now will be 20Hz, we will design our model to collect data at 20Hz.

The Butterworth filter is a type of signal processing filter designed to have a frequency response as flat as possible in the passband. It is also referred to as a maximally flat magnitude filter.

Working Description



Butterworth stated that:

"An ideal electrical filter should not only completely reject the unwanted frequencies but should also have uniform sensitivity for the wanted frequencies".

Such an ideal filter cannot be achieved, but Butterworth showed that successively closer approximations were obtained with increasing numbers of filter elements of the right values. At the time, filters generated substantial ripple in the passband, and the choice of component values was highly interactive. Butterworth showed that a low pass filter could be designed whose cutoff frequency was normalized to 1 radian per second

Gravity filtering

Due to gravity, the acceleration data collected comprise gravitational and body components.

We will use filters to separate into 2 parts: body and gravity acceleration signals using Butterworth filters with a corner frequency of 0.3 Hz, since the gravitational force is assumed to have only low frequency components.

5.1.4 Feature Exploration

By this stage, the data we should have data contains columns, such as

- Body acceleration XYZ
- Gravity acceleration XYZ
- Gyroscope XYZ

5.1.4.1 Windowing

Sensor data will be divided into consecutive segments so that each one of them can be analyzed separately and sequentially. These segments are known as time windows. We will be using sliding window-based segmentation. Sliding window-based segmentation divides sensor data into subsequences over time, we initially set fixed-width sliding windows of 2.56 sec and 50% of frames will overlap (for 20Hz, there will be 128 readings).

5.1.4.2 Feature Engineering

The data will be explored in time-domain and frequency-domain.

For common statistical features, we have mean, standard deviation, median deviation, max, min, interquartile range, entropy, sum of area under each signal, energy, and magnitudes for these features (meaning we will have mean and mean magnitude, …)

For time domain, body linear acceleration and angular velocity were derived in time to obtain Jerk signals (rate at which an object's acceleration changes with respect to time), composed into body acceleration Jerk signals and gyroscope Jerk signals. Time series features like an auto-regression coefficient with Burg methods with order from 1 to 4, correlation per each 3-axial signals are good to have. Since features generated are in 3-axial representation form, we can compute the magnitude of these using Euclidean norm.

For frequency domain, we apply Fast Fourier Transform (FFT) for sensor signals and produce body acceleration XYZ, body acceleration Jerk XYZ, body gyroscope XYZ, body acceleration Jerk magnitude, body gyroscope magnitude and body gyroscope Jerk magnitude. Additional frequency domain features can be extracted that are related to max value of that signal on XYZ, mean frequency, skewness, and kurtosis of signal magnitude.

Furthermore, we can have **angle features**, for one window, we compute:

- Angle 0: angle between mean value of body acceleration with mean value of gravity acceleration
- Angle 1: angle between mean value of body jerk acceleration with mean value of gravity acceleration

- Angle 2: angle between mean value of body gyroscope with mean value of gravity acceleration
- Angle 3: angle between mean value of body jerk gyroscope with mean value of gravity acceleration
- Angle 4: angle between X-axis with mean value of gravity acceleration
- Angle 5: angle between Y-axis with mean value of gravity acceleration
- Angle 6: angle between Z-axis with mean value of gravity acceleration

Feature extraction is necessary because raw data from the sensors is not appropriate for use by standard machine learning algorithms. Time domain features have a lower computational cost when compared to frequency domain features, but features in the frequency domain can better represent contextual information in terms of signal patterns. In total, we have about 640 features that can be generated from this above process. Since this is a very huge collection of features, an appropriate number of features need to be picked from each domain so that it is not too computationally heavy while also maintaining a reliable level of accuracy.

5.1.5. Machine Learning Models

5.1.5.1 Introduction

Machine learning tasks are generally categorized into three main areas: Supervised Learning, Unsupervised Learning and Reinforcement Learning. The methods all differ in how the machine learning algorithm is "rewarded" for being correct in its predictions or classifications.

Supervised learning algorithms involve labelled data. That is, data annotated with values such as categories (as in supervised classification) or numerical responses (as in supervised regression). Such algorithms are "trained" on the data and learn which predictive factors influence the responses. When applied to unseen data supervised learning algorithms attempt to make predictions based on their prior training experience. An example from quantitative finance would be using supervised regression to predict tomorrow’s stock price from the previous months’ worth of price data.

Unsupervised learning algorithms do not make use of labelled data. Instead, they utilize the underlying structure of the data to identify patterns. The canonical method is unsupervised clustering, which attempts to partition datasets into sub-clusters that are associated in some manner. An example from quantitative finance would be the clustering of certain assets into groups that behave similarly in order to optimize portfolio allocations.

Reinforcement learning algorithms attempt to perform a task within a certain dynamic environment, by taking actions inside the environment that seek to maximize a reward mechanism. These algorithms differ from supervised learning in that there is no direct set of input/output pairs utilized within the data. Such algorithms have recently gained significant

notoriety due to their use by Google DeepMind. DeepMind has utilized "deep reinforcement learning" to exceed human performance in Atari video games and the ancient game of Go. Such algorithms have been applied in quant finance to optimize investment portfolios.

In this project, since we are using sensor data to predict the dance moves and we use training data with labels, it is obvious that we will mainly use **Supervised learning algorithms** to tackle the problem.

5.1.5.2 Support Vector Machines

Support vector machines are used to find a hyperplane in an n-dimensional space that separates the data points to their potential classes. The hyperplane should be positioned with the maximum distance to the data points. The data points with the minimum distance to the hyperplane are called Support Vectors. Due to their close position, their influence on the exact position of the hyperplane is bigger than of other data points.

A good example of such a system is classifying a set of new documents into positive or negative sentiment groups, based on other documents which have already been classified as positive or negative. Similarly, we could classify new emails into spam or non-spam, based on a large set of emails that have already been marked as spam or non-spam by humans.

One of the simple 2D examples is shown below:

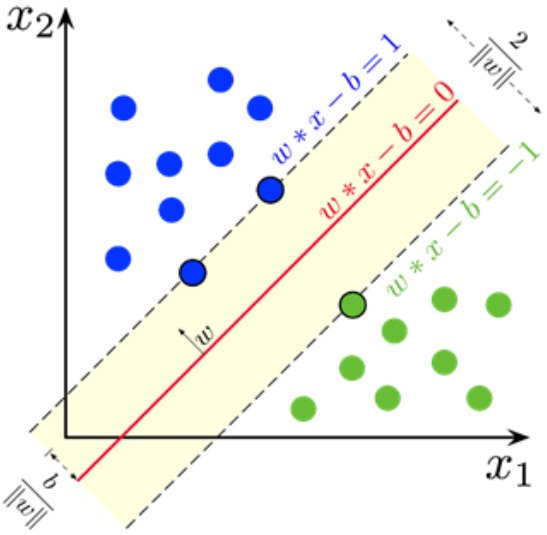


Figure 5.1.c: Hyperplanes separating 2 groups of data points with Support Vectors are the 3 points (2 blue, 1 green) laying on the lines.

SVMs are highly applicable to such situations. As with other supervised machine learning techniques a support vector machine is applied to a labelled set of feature data, which resides in feature space. In the context of spam or document classification, each "feature" dimension is the prevalence or importance of a particular word appropriately quantified.

The goal of the SVM is to train a model that assigns new unseen objects into a particular category. It achieves this by creating a partition of the feature space into two subspaces. Based on the features in the new unseen objects (e.g., documents/emails), it places an object onto a specific side of the separation plane, leading to a categorization such as spam or non-spam.

This makes it an example of a non-probabilistic classifier. It is non-probabilistic because the feature values in new unseen test observations explicitly determine their location in feature space without any stochastic component. Much of the benefit of SVMs is due to the fact that this separation plane need not actually be a linear hyperplane.

Utilizing a technique known as the **kernel trick** they can become much more flexible by introducing various types of non-linear decision boundaries. This is necessary for "real world" datasets that do not often possess straightforward linear separating boundaries between categories. Formally, in mathematical language, SVMs construct linear separating hyperplanes in large finite-dimensional vector spaces.

Data points are viewed as (x, y) tuples, $x = (x_1, ..., x_p)$ where the x_j are the feature values and y is the classification (usually given as +1 or -1). Optimal classification occurs when such hyperplanes provide maximal distance to the nearest training data points. Intuitively, this makes sense, as if the points are well separated, the classification between two groups is much clearer. However, if in a feature space some of the sets are not linearly separable and overlap, then it is necessary to perform a transformation of the original feature space to a higher-dimensional space, in which the separation between the groups becomes clearer. However, this has the consequence of making the separation boundary in the original space potentially non-linear.

For example, when we use IRIS dataset, which has non-separable data, it is impossible to have a linear kernel performing well. Therefore, we may opt for other kinds of kernels:

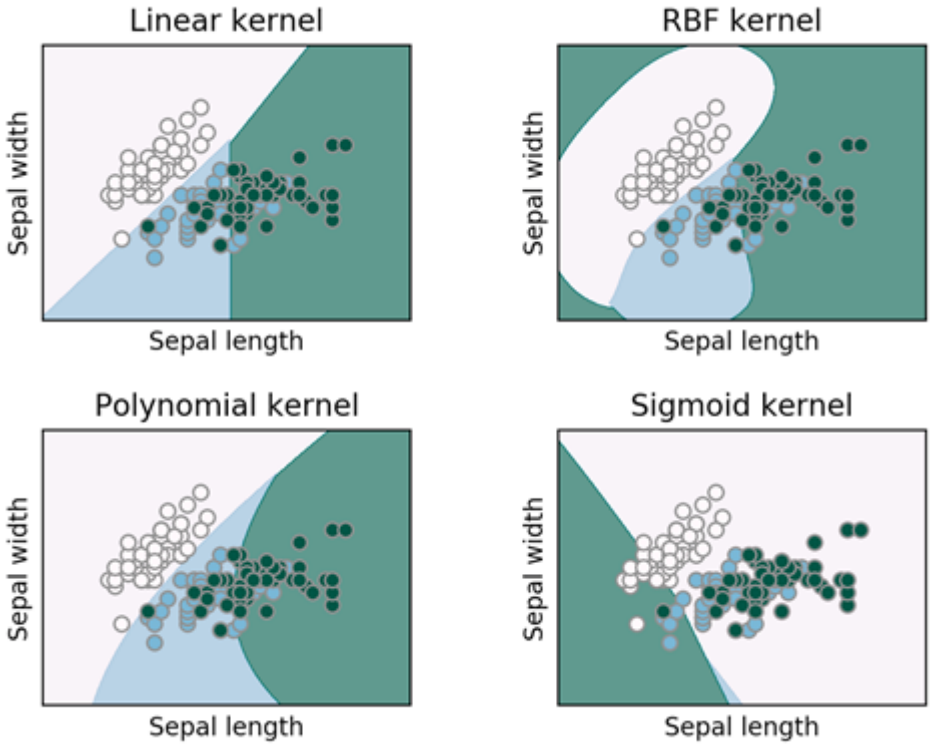


Figure 5.1.d: Visualization of the different kernel functions.

SVM has some advantages that we cannot overcome, such as:

- **High-Dimensionality -** The SVM is an effective tool in high-dimensional spaces, which is particularly applicable to document classification and sentiment analysis where the dimensionality can be extremely large ($>= 10^6$).

- **Memory Efficiency -** Since only a subset of the training points are used in the actual decision process of assigning new members, only these points need to be stored in memory (and calculated upon) when making decisions.

- **Versatility -** Class separation is often highly non-linear. The ability to apply new kernels allows substantial flexibility for the decision boundaries, leading to greater classification performance.

However, SVM still has some disadvantages hindering the classification process:

- $p << n$ - In situations where the number of features for each object p exceeds the number of training data samples n SVMs can perform poorly. This can be seen intuitively as if the high-dimensional feature space is much larger than the number of samples then there are less effective support vectors on which to support the optimal linear hyperplanes. This leads to poorer classification performance as new unseen samples are added.

- **Non-Probabilistic -** Since the classifier works by placing objects above and below a classifying hyperplane, there is no direct probabilistic interpretation for group membership. However, one potential metric to determine "effectiveness" of the classification is how far from the decision boundary the new point is.

5.1.5.3 Neural Network Zoo

The most advanced model that is currently mostly used is called neural networks. As we know the inspiration behind neural networks are our brains. Below image shows the biological aspect of neural networks.

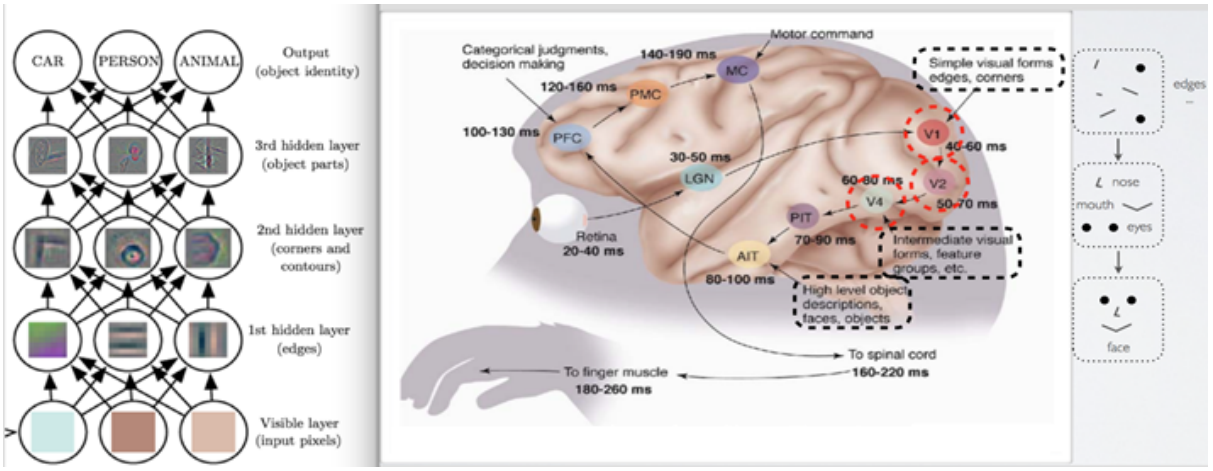


Figure 5.1.e: Multilayer perceptrons used to identify different objects by learning different characteristics of object at each layer and Human brain [Image Courtesy: Hugo Larochelle Slides.]

As being shown, neural network models learn how to identify objects with different characteristics, for each layer, one component of the object is being learned. For example, on the first hidden layer edges are detected, on the second hidden layer corners and contours are identified. Similarly, in our brain there are different regions for the same purpose, as we can the region denoted by V1, identifies edges, corners and etc.

There are many different neural network architectures that we can utilize and test to decide what is the best for our data.

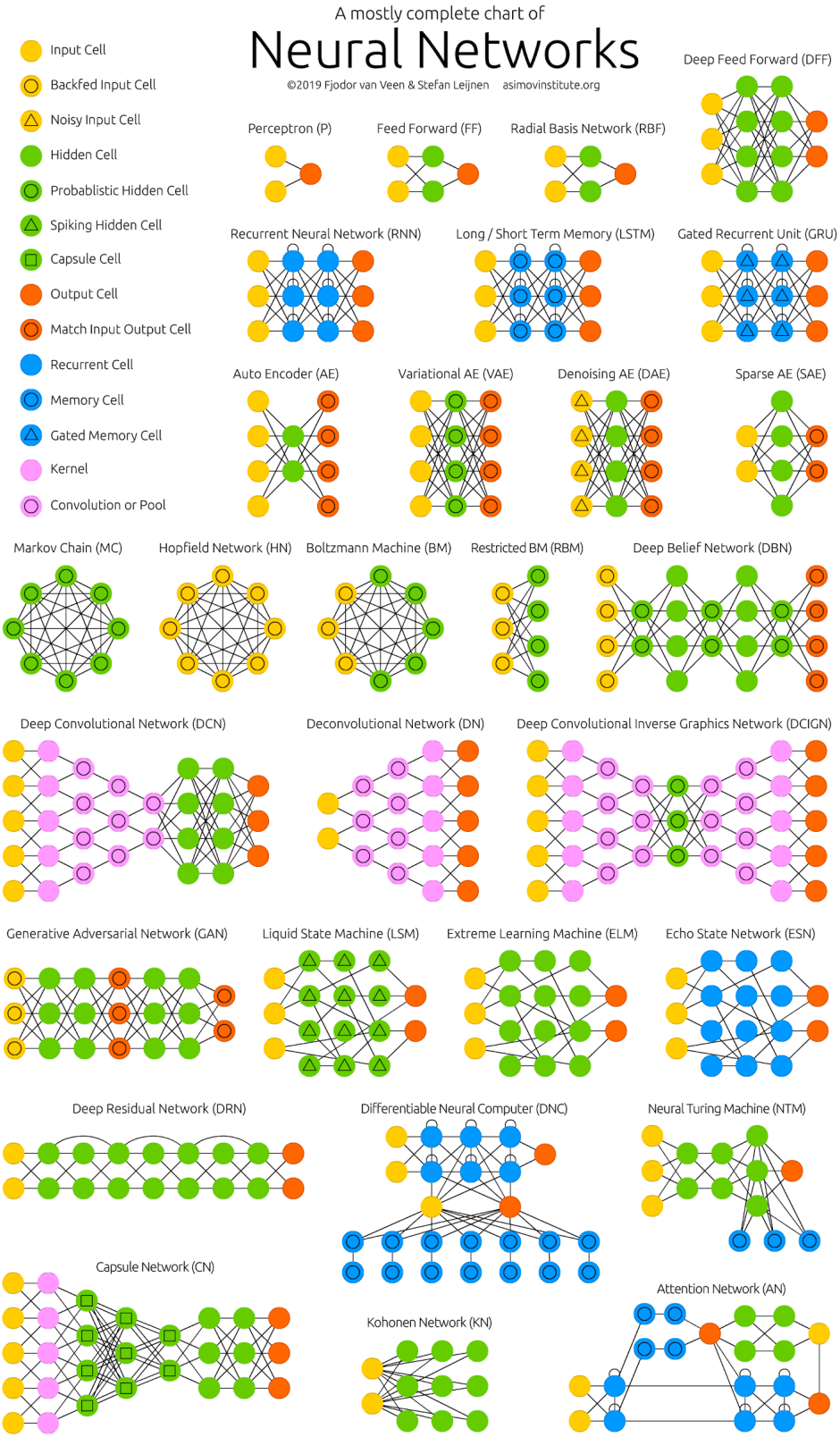


Figure 5.1.f: Neural network zoo

In our project, we will mainly focus on 3 different type of neural networks, namely Multi-layer perceptrons (Feedforward network), Recurrent neural network (specifically, Long-short term memory, LSTM) and Convolutional Neural Network

5.1.5.3.1 Feedforward Network

Deep feedforward networks also often called feedforward neural networks, or **multilayer perceptrons (MLPs)**, are the quintessential deep learning models.

The goal of a feedforward network is to approximate some function f*. For example, for a classifier, y=f*(x) maps an input x to a category y. A feedforward network defines a mapping y=f(x;θ) and learns the value of the parameters θ that result in the best function approximation. These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself. For example, the below network has 1 input layer, 2 hidden layers and a final output layer with a finite number of nodes on hidden layers; this depends on our choice.

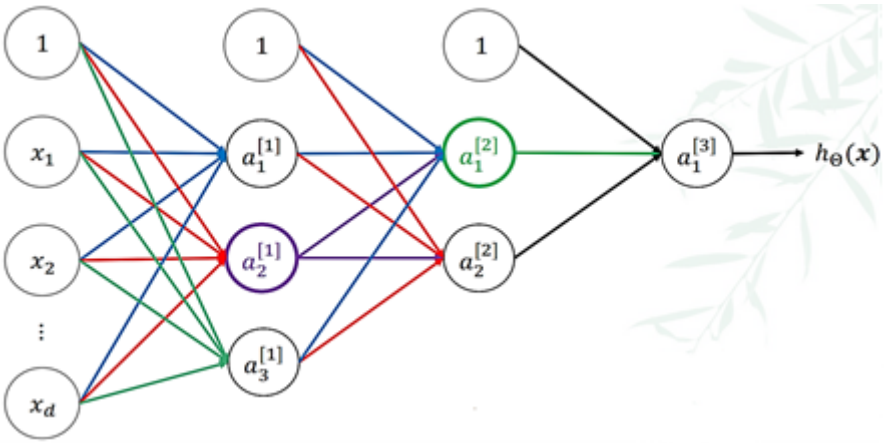


Figure 5.1.g: Feedforward Neural network example

For each layer in the model, we define an activation function, called g(). The node of the next layer will be the sum of the previous layer node after being applied through g().

$$a_2^{[1]} = g\left(\underbrace{\Theta_{2,0}^{[1]} \times 1 + \Theta_{2,1}^{[1]} x_1 + \Theta_{2,2}^{[1]} x_2 + \cdots + \Theta_{2,d}^{[1]} x_d}_{s_2^{[1]}}\right)$$

$$a_1^{[2]} = g\left(\underbrace{\Theta_{1,0}^{[2]} \times 1 + \Theta_{1,1}^{[2]} a_1 + \Theta_{1,2}^{[2]} a_2^{[1]} + \Theta_{1,3}^{[2]} a_3^{[1]}}_{s_1^{[2]}}\right)$$

$$\Theta_{i,j}^{[l]} \begin{cases} 1 \le l \le L & \text{layers} \\ 1 \le i \le d^{[l]} & \text{outputs} \\ 0 \le j \le d^{[l-1]} & \text{inputs} \end{cases}$$

$$x_i^{[l]} = a_i^{[l]} = g\left(s_i^{[l]}\right) x = g\left(\sum_{j=0}^{d^{[l-1]}} \Theta_{i,j}^{[l]} x_j^{[l-1]}\right) = g\left(\left(\Theta_i^{[l]}\right)^T x^{[l-1]}\right)$$

5.1.5.3.2 Convolutional Neural Network

Firstly, we know that convolution is an operation largely used in signal processing. The main goal is to filter out some useful features for characterizing a key object. Every filter is small spatially, but extends through the full depth of the input.

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

They are models that are composed of two main types of elements: convolutional layers and pooling layers. Convolutional layers read an input, such as a 2D image or a 1D signal, using a kernel that reads in small segments at a time and steps across the entire input field. Each read results in the input that is projected onto a filter map and represents an internal interpretation of the input. Pooling layers take the feature map projections and distill them to the most essential elements, such as using a signal averaging or signal maximizing process.

Convolution leverages two important ideas that help improve a machine learning system: **sparse interactions** and **parameter sharing**. Moreover, convolution provides a means for working with inputs of variable size.

Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing interaction between each input unit and each output unit. This means every output unit interacts with every input unit. Convolutional networks, however,

typically have **sparse interactions** (**sparse connectivity** or **sparse weights**). This is accomplished by making the kernel smaller than input. For example, when processing an image, the input image might have thousands or millions of pixels, but we detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. This means that we need to store fewer parameters, which both decrease memory requirements of model and improves its statistical efficiency. It also means that computing output requires fewer operations. These improvements in efficiency are usually quite large. If there are m inputs and n outputs, then matrix multiplication requires m×n parameters and algorithms used in practice have O(m×n) runtime (per example). If we limit # connections each output may have to k, then sparsely connected approach requires only k×n parameters and O(k×n) runtime. For many practical applications, it is possible to obtain good performance on machine learning tasks while keeping k several orders of magnitude smaller than m.

Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional neural net, each element of the weight matrix is used exactly once when computing output of a layer. It is multiplied by one element of input and then never revisited. As a synonym for parameter sharing, one says that a network has tied weights, because the value of weight applied to one input is tied to the value of a weight applied elsewhere. In a convolutional neural net, each member of the kernel is used at every position of input (except perhaps some of boundary pixels, depending on design decisions regarding boundary). Parameter sharing used by convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. This does not affect runtime of forward propagation—it is O(k×n)—but it does further reduce storage requirements of model to k parameters. Recall that k is usually several orders of magnitude less than m. Since m and n are usually roughly the same size, k is practically insignificant compared to m×n. Convolution is more efficient than dense matrix multiplication in terms of memory requirements and statistical efficiency.

Even though CNN is usually used in the image processing area, 1D CNN is actually very useful in time series data because of its sliding window functions. We all know that time series data has interrelationship to its previous history records. When a member dance, he or she will repeat over time, e.g., dance multiple times in 10 seconds, this will create a repeatable pattern on our data, and it is hard to train the model based on 1 time point data at a time, instead, we will have a sliding window together with pooling layers across a period of time. The convolution and pooling layers can be repeated at depth, providing multiple layers of abstraction of the input signals. The output of these networks is often one or more fully connected layers that interpret what has been read and map this internal representation to a class value.

5.1.5.3.3 Recurrent Neural Network

Recurrent neural networks, or RNNs for short, are a type of neural network that was designed to learn from sequence data, such as sequences of observations over time, or a sequence of words in a sentence.

Long-short Term Memory (LSTM):

A specific type of RNN called the long short-term memory network, or LSTM for short, is perhaps the most widely used RNN as its careful design overcomes the general difficulties in training a stable RNN on sequence data.

LSTMs have proven effective on challenging sequence prediction problems when trained at scale for such tasks as handwriting recognition, language modeling, and machine translation.

A layer in an LSTM model is composed of special units that have gates that govern input, output, and recurrent connections, the weights of which are learned. Each LSTM unit also has internal memory or state that is accumulated as an input sequence is read and can be used by the network as a type of local variable or memory register.

More specifically, The high level overview of the repeating unit of an LSTM is the following:

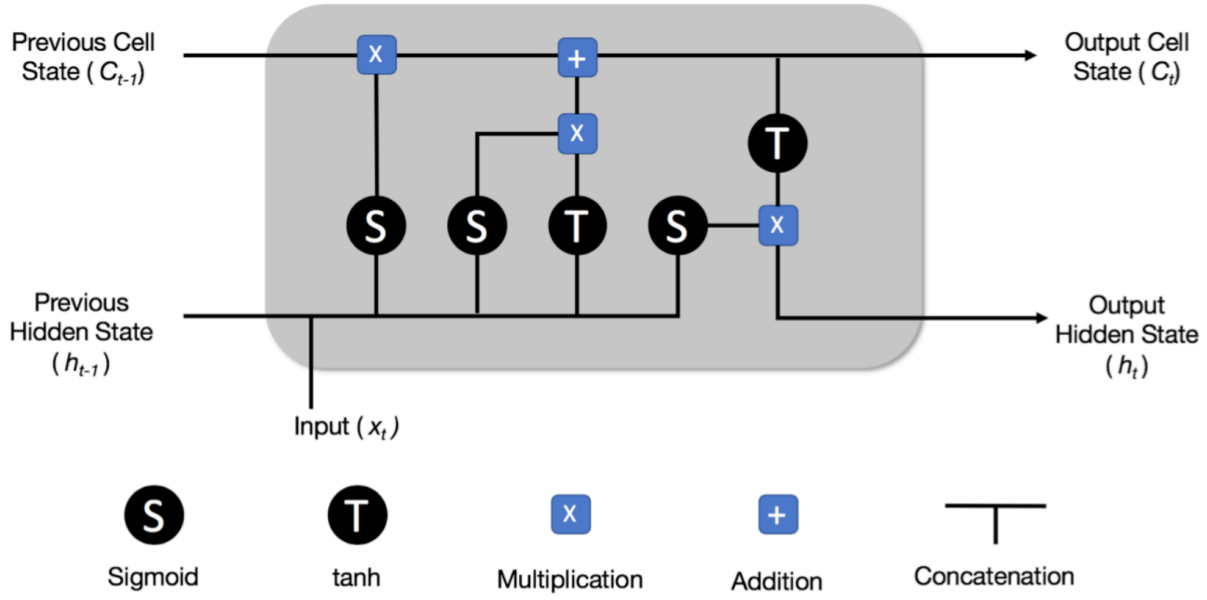


Figure 5.1.h: Repeating unit for LSTM

As you can see, LSTM includes the cell state as the current memory. It flows from one repeating structure to the next, conveying important information that has to be retained at the moment. The hidden state is the overall memory of the entire LSTM, having everything that we have seen (un)important inputs.

LSTM releases information between the hidden and cell state through setting three gates: forget gate, input gate and output gate.

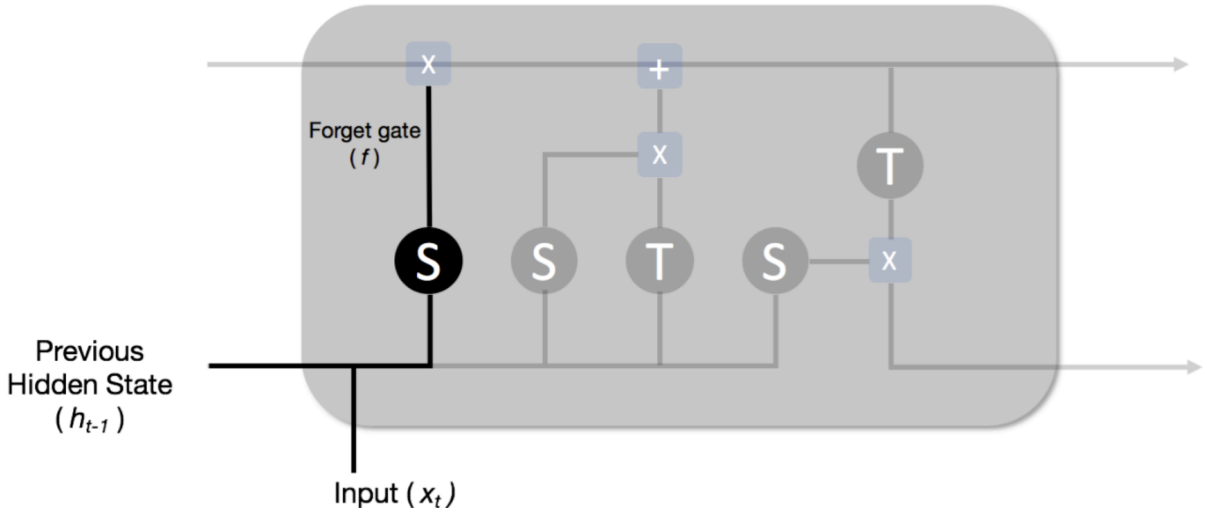


Figure 5.1.i: Step I in repeating unit in LSTM

Forget gate (f) forms the first part of the LSTM repeating unit, and its role is to decide how much data we should forget or remember from the previous cell state. It does by concatenating the previous hidden state and the current input, and then passing the concatenated vector through a sigmoid function.

Recall that the sigmoid function outputs a vector with values between 0 and 1. A value of 0 means to stop the information from passing through Forget f with $f = \sigma(concatenate(h_{t-1}, x_t))$ and a value of means to pass the information through remembering.

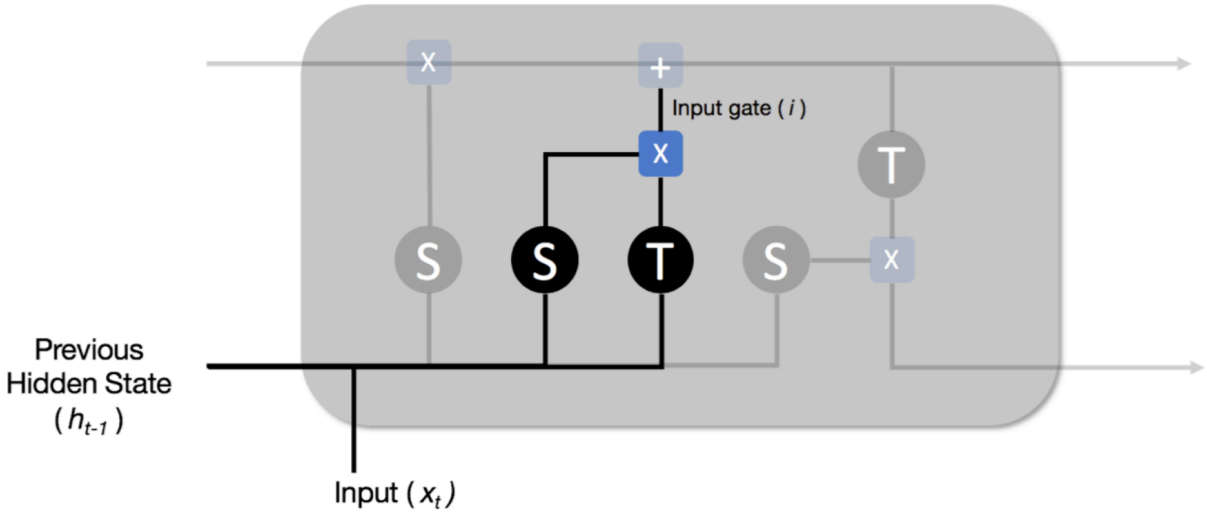


Figure 5.1.j: Step II in repeating unit in LSTM

Input gate (i) controls how much information to pass to the current cell state. It takes as input the concatenation of the previous hidden state and the current input. Then, it passes two copies

of the concatenated vector through a sigmoid function and a tanh function, before multiplying them together.

$$f = \sigma(concatenate(h_{t-1}, x_t))$$

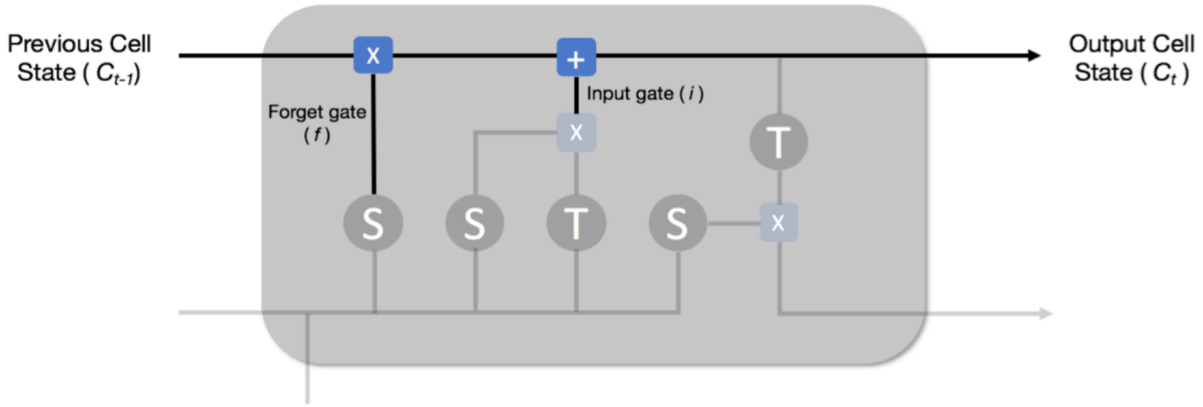


Figure 5.1.k: Step III in repeating unit in LSTM

Input gate (i) process offers what is required to compute the current cell state to be output.

$$C_t = (f * C_t) + i$$

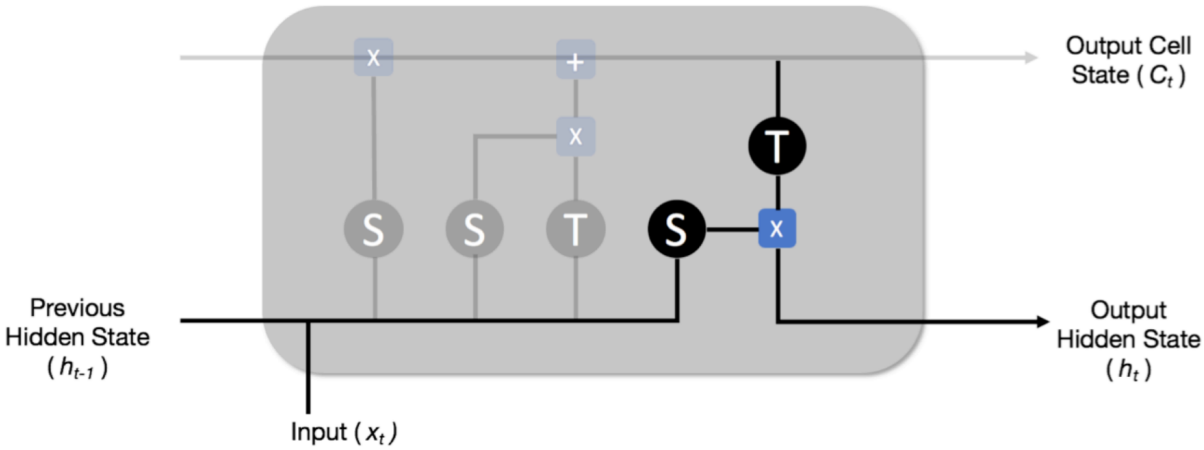


Figure 5.1.l: Step IV in repeating unit in LSTM

Output gate (i) controls how much information is to be retained in the hidden state. We concatenate the previous hidden state and the current input and pass it through a sigmoid function. Then, we take the current cell state and pass it through a tanh function. Finally, we take the multiplication of the two, which is passed to the next repeating unit as the hidden state.

$$h_t = \sigma(concatenate(h_{t-1}, x_t)) * tanh(C_t)$$

Like the CNN that can read across an input sequence, the LSTM reads a sequence of input observations and develops its own internal representation of the input sequence. Unlike the CNN, the LSTM is trained in a way that pays specific attention to observations made and prediction errors made over the time steps in the input sequence, called backpropagation through time.

Interestingly, LSTMs can be applied to the problem of human dance moves. The LSTM learns to map each window of sensor data to an activity, where the observations in the input sequence are read one at a time, where each time step may be composed of one or more variables (e.g., parallel sequences).

5.1.6 Model Training and Model Validation

5.1.6.1 Motivation

Traditionally, in statistics and machine learning we usually split our data into two subsets: training data and testing data (and sometimes to three: train, validate and test), and fit our model on the train data, in order to make predictions on the test data.

When we do that, one of two things might happen: we overfit our model or we underfit our model. We don't want any of these things to happen, because they affect the predictability of our model — we might be using a model that has lower accuracy and/or is ungeneralized (meaning you can't generalize your predictions on other data).

5.1.6.1.1 Overfitting

Overfitting means that the model we trained has trained "too well" and is now, well, fit too closely to the training dataset. This usually happens when the model is too complex (i.e. too many features/variables compared to the number of observations). This model will be very accurate on the training data but will probably be very not accurate on untrained or new data. It is because this model is not generalized (or not AS generalized), meaning you can generalize the results and can't make any inferences on other data, which is, ultimately, what you are trying to do. Basically, when this happens, the model learns or describes the "noise" in the training data instead of the actual relationships between variables in the data. This noise, obviously, is not part of any new dataset, and cannot be applied to it.

5.1.6.1.2 Underfitting

In contrast to overfitting, when a model is underfitted, it means that the model does not fit the training data and therefore misses the trends in the data. It also means the model cannot be generalized to new data. As you probably guessed (or figured out!), this is usually the result of a very simple model (not enough predictors/independent variables). It could also happen when, for example, we fit a linear model (like linear regression) to data that is not linear. It almost goes without saying that this model will have poor predictive ability (on training data and can't be generalized to other data).

5.1.6.1.3 Example

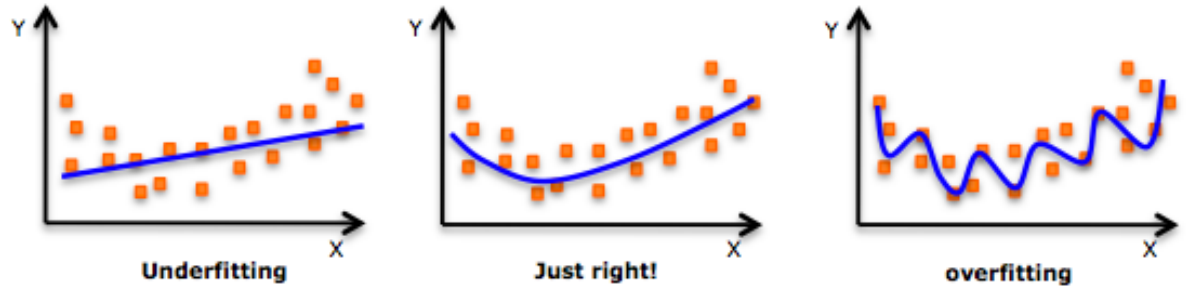


Figure 5.1.m: An example of overfitting, underfitting and a model that's "just right!"

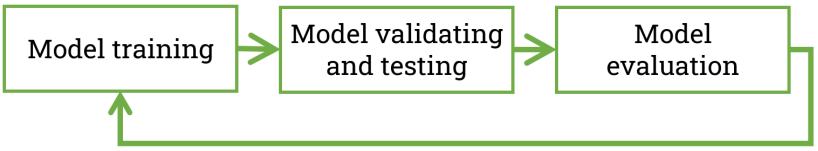
It is worth noting the underfitting is not as prevalent as overfitting. Nevertheless, we want to avoid both of those problems in data analysis. You might say we are trying to find the middle ground between under and overfitting our model. As you will see, train/test split and cross validation help to avoid overfitting more than underfitting. Let's dive into both of them!

When we train the model and fit data into model, there are 2 important factors we will always take note:

- **Test Error -** The average error, where the average is across many observations, associated with the predictive performance of a particular statistical model when assessed on new observations that were not used to train the model.

- **Flexibility -** The degrees of freedom available to the model to "fit" to the training data. A linear regression is very inflexible (it only has two degrees of freedom) whereas a high-degree polynomial is very flexible (and as such can have many degrees of freedom).

5.1.6.1 Training-Validation-Testing Dataset



In order to guarantee we will achieve best possible model; we will first discover 3 terms of dataset used in model training process:

- **Training Dataset:** The sample of data used to fit the model. The actual dataset that we use to train the model (weights and biases in the case of a Neural Network). The model sees and learns from this data.

- **Validation Dataset:** The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model

configuration. The validation set is used to evaluate a given model, but this is for frequent evaluation. We, as machine learning engineers, use this data to fine-tune the model hyperparameters. Hence the model occasionally sees this data, but never does it "Learn" from this. We use the validation set results, and update higher level hyperparameters. So, the validation set affects a model, but only indirectly. The validation set is also known as the Dev set or the Development set. This makes sense since this dataset helps during the "development" stage of the model.

- **Test Dataset:** The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained(using the train and validation sets). The test set is generally what is used to evaluate competing models (For example on many Kaggle competitions, the validation set is released initially along with the training set and the actual test set is only released when the competition is about to close, and it is the result of the model on the Test set that decides the winner). Many times the validation set is used as the test set, but it is not good practice. The test set is generally well curated. It contains carefully sampled data that spans the various classes that the model would face, when used in the real world.

First, we will have a very large dataset, in here, we will first split the dataset into train and test, let us say 70% for training data and 30% for testing data. After that, within this 70% training dataset, we will split into 80% of training dataset and 20% to be validation dataset, meaning training dataset is 70%×80% and validation will hold 70%×20% of dataset.

5.1.6.2 Cross-validation

Next, we will define the term, cross-validation. Cross-validation is to estimate the test error associated with a statistical model or select the appropriate level of flexibility for a particular statistical method. Recall that the training error associated with a model can vastly underestimate the test error of the model. Cross-validation provides us with the capability to estimate the test error more accurately, which we will never know in practice. Cross-validation works by "holding out" particular subsets of the training set in order to use them as test observations.

K-fold cross-validation improves upon the validation set approach by dividing the n observations into k mutually exclusive, and approximately equally sized subsets known as "folds". The first fold becomes a validation set, while the remaining k-1 folds (aggregated together) become the training set. The model is fit on the training set and its test error is estimated on the validation set. This procedure is repeated k times, with each repetition holding out a fold as the validation set, while the remaining k-1 are used for training.

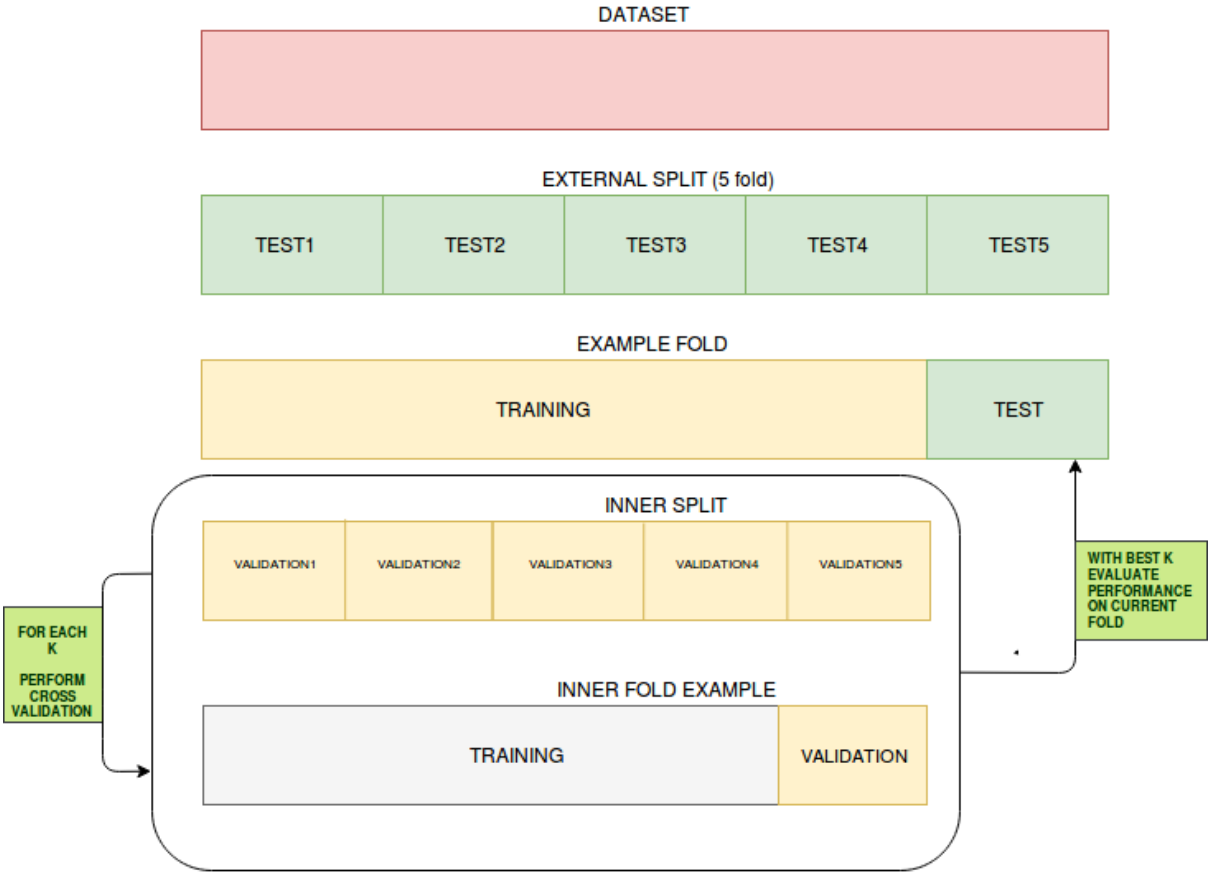


Figure 5.1.n: Cross validation with k=5

5.1.6.3 Leave-one-out Cross Validation

We can actually choose k=n, which means that we fit the model n times, with only a single observation left out for each fitting. This is known as leave-one-out cross-validation (LOOCV). It can be very computationally expensive, particularly if n is large and the model has an expensive fitting procedure, as fitting must be repeated n times.

While LOOCV is beneficial in reducing bias, due to the fact that nearly all of the samples are used for fitting in each case, it actually suffers from the problem of high variance. This is because we are calculating the test error on a single response each time for each observation in the data set.

k-fold cross-validation reduces the variance at the expense of introducing some more bias, due to the fact that some of the observations are not used for training. k-fold cross validation has less computational cost compared to LOOCV as the model has to be only trained k times, where k<n. There is also reduced bias as compared to the holdout method, and the variance of the test set is reduced compared to LOOCV due to multiple observations in the test set.

However, if the value of k is large, then there would be a high computational cost incurred, similar to the case of LOOCV.

Hence, it is desirable to pick a value of k which has a good trade off between accuracy and computational cost.

Our team will be using the k-fold cross validation method to split our testing and training datasets as it gives a good balance of reliability and efficiency as compared to the other methods. With k = 5 or k = 10 the bias-variance tradeoff is generally optimized.

5.1.7 Model Evaluation

First, we will have to know the definition of the confusion matrix. This matrix is made to evaluate the model performance after using the model fit into a testing dataset.

Confusion matrix	***Predicted: Sick***	***Predicted: Healthy***
Actual: Sick	*True Positive*	*False Negative (type II error)*
Actual: Healthy	*False positive (type I error)*	*True Negative*

In other words, we can define as:

- *True positive (TP)*: Sick people identified as sick
- *False positive (FP)*: Healthy people identified as sick
- *True negative (TN)*: Healthy people identified healthy
- *False negative (FN)*: Sick people identified as healthy

After training the model, it is vital that we test it to see that it is making accurate predictions. To evaluate the performance of a model, we can use metrics that inform, in mathematical terms, how reliable the model is in detecting an activity.

Even though accuracy is the first criteria in our model performance, we can try other measures as well, as the table below shown:

Evaluation Metric	***Formula***	***Interpretation***
Accuracy (ACC) ***Fraction of correct prediction***	ACC=(TP+TN)/(TP+FP+FN+TN)	Overall performance of model.
True positive rate (TPR) ***Sensitivity (SEN)*** ***Recall: Recall of positive class***	SEN=TP/P=TP/(TP+FN)	Among all positive instances, proportion of correct predictions is
True negative rate (TNR) ***Specificity (SPC), recall of negative class***	SPC=TN/N=TN/(TN+FP)	Among all negative instances, proportion of correct predictions is
False positive rate (FPR)	FPR =FP/(TN+FP)=1-TNR	Among all positive instances, proportion of incorrect predictions is
False negative rate (FNR)	FNR=FN/(FN+TP)=1-TPR	Among all negative instances, proportion of incorrect predictions is
Prevalence	(TP+FP)/(TP+FN+FP+TN)	Percentage of True (1's) in sample

Detection rate	TP/(TP+FN+FP+TN)	Correctly predicted 1's as a percentage of entire sample
Detection prevalence	(TP+FP)/(TP+FN+FP+TN)	What percentage of the full sample was predicted as 1?
Precision Positive prediction ***Value (PPV)***	PRC=PPV=TP/(TP+FP)	How accurate positive predictions are. A balance between correctly predicting 1' s and 0' s. What percentage of predicted 1' s are correct?
Balanced accuracy (bACC) ***Mean of sensitivity and specificity; average accuracy obtained in either class.***	bACC=1/2 (SEN+SPC)	Avoids inflated performance estimates on imbalanced datasets. “Balanced accuracy and its posterior distribution”
F1 Score (or F-score) ***Harmonic mean precision & recall***	2 (precision × recall)/(precision + recall)=2TP/(2TP+FP+FN)	Hybrid metric useful for unbalanced classes
AUROC	Area under curve	Model’s true performance considering all possible probability cut offs

5.1.8 From offline data to streaming dancing data

5.1.8.1 Offline dancing data VS streaming data

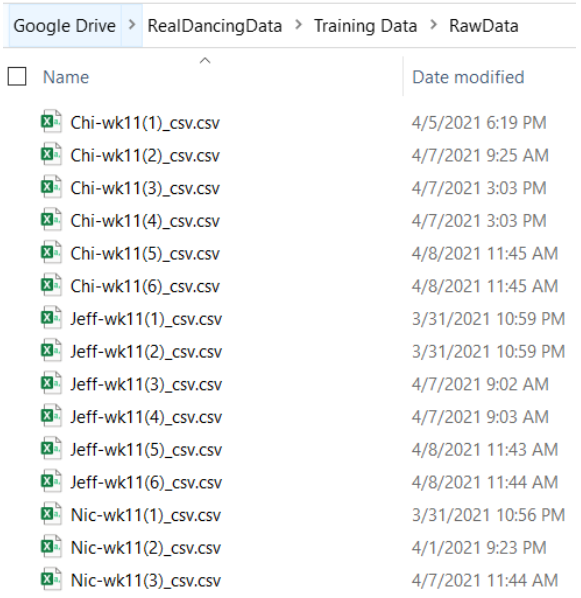
In order to ensure the model is fed with sufficient data, I have decided to collect 6 CSVs file for each dancer. In one CSV file, there will be 3 rounds of data, and each round is 9 dance moves. Each dancer will use one beetle for 2 csv and they exchange beetles to another. By doing this, this can guarantee that there are small changes when using different beetles, and to avoid overfitting as one dancer's data can change when they dance the same dance move on a different time-stamp and with different beetles. The ideal scenario should not be drastically different between CSVs files when dancers use different dataset to dance, each dance move should be easily recognized by "eyes". When we collect data and dancers finish all the moves for all 6 CSVs, we call this **"Offline" data** as there is no intervention of any other component (such as predicting models intervene to predict dance move) when the dancer dances.

In a real experimenting session, when we collect a sufficient array of data, representing one data instance, then we will use a model to predict one instance and give one dance move. This data is called **streaming data**.

5.1.8.1 Data collecting workflow and data management structure

	A	B	C	D	E
1	exp_id	user	activity	start	end
2	wk11(1)_csv	Chi	DAB	77	369
3	wk11(1)_csv	Chi	SIDE_PUMP	581	885
4	wk11(1)_csv	Chi	WIPE_TABLE	1058	1382
5	wk11(1)_csv	Chi	HAIR	1557	1858
6	wk11(1)_csv	Chi	KICK_ELBOW	2048	2259
7	wk11(1)_csv	Chi	POINT_HIGH	2454	2732
8	wk11(1)_csv	Chi	GUN	2921	3230
9	wk11(1)_csv	Chi	LISTEN	3420	3688
10	wk11(1)_csv	Chi	ROLL_HAND	3896	4201
11	wk11(1)_csv	Chi	DAB	4374	4691
12	wk11(1)_csv	Chi	SIDE_PUMP	4884	5184
13	wk11(1)_csv	Chi	WIPE_TABLE	5362	5652
14	wk11(1)_csv	Chi	HAIR	5818	6144
15	wk11(1)_csv	Chi	KICK_ELBOW	6349	6615
16	wk11(1)_csv	Chi	POINT_HIGH	6804	7123
17	wk11(1)_csv	Chi	GUN	7286	7613

The labels csv should contain 5 columns with start and end columns indicating the start line and end line in each csv file.



The file is stored as format NAME_WK#_csv.csv with NAME is dancer name, WK# indicate the week data is collected.

5.1.8.2 Signal processing and Data exploration for Streaming Data

When data is coming continuously in streaming mode, data is not easily scaled if we apply standardization on streaming data alone as their mean and std is "running", instead, we will accept a level of reasonable incorrect in scaling by **apply the scaler from offline data on streaming data,** meaning we will use mean and std of offline data and apply on streaming data.

So if the dancer dances very close to the ground truth in offline data, the instance should be classified correctly. The more different the dance dance is, the harder it gets to give a correct prediction.

This is the reason why streaming mode will have lower accuracy than offline data mode.

Especially, when the last Milestone has 3 "unseen" TA Cases. Hence, the data exploration stage is very important to choose the features that can be invariant with dancer moves, meaning whoever dances, even though the stranger dance model should be able to predict.

5.1.8.2.1 Gyroscope readings problems

We note that the dance bears some repetitive pattern over time that can be classified by "eyes", however, we cannot guarantee that the reading of accelerator and gyroscope always same, the starting point of each dance move is has great impact on prediction, for example, the way you set of your hand initially can be different whenever the you dance, when you rest, the hand must be absolutely vertically parallel to the body, but you cannot ensure that the arm resting, gyroscope reading and angle reading are same at all time dancing, and the reading from the moment your hand is resting to make its first beginning dance move.

So even if the hand rotates 1 circle (360) degree but with different initial gyroscope reading (orientation), then the data is different already.

Therefore, the gyroscope readings are not trusted. One of the solutions to solve this problem is using **jerk signals:** $(x(t+1) - x(t))/dt$ with $x(t)$ and $x(t+1)$ are data at time point t and t+1 and dt is time-derivative computed by $(1/samplingFreq)$.

Even though the value changes, the rate of changes between data series will be nearly constant. Back to the example, when we rotate 1 circle, even though the gyroscope can be different, the jerk signals will always be the same regardless of starting angle.

Therefore, jerk signals should be in the features list. Hence, **gyroscope readings can be excluded and replaced by their jerk signals, Acceleration can be kept** as they do not involve how dancers place their body parts but the speed, this will be nearly the same between streaming mode and offline mode if the dancer practices a lot and keeps their movement at some consistency level.

5.1.8.3 Training, Testing and Evaluating models for real-time dancing data

5.1.8.3.1 Difference when dealing with streaming data and offline data in training and testing model

When we collect data and train with different models, most of the time we always have good accuracy.

However, when we try on streaming data, there is still a big difference between models. For example, some traditional models have good accuracy on Offline data, however, when we try for streaming one, some models are not a very good choice as we are dealing with "unseen" data, since no dancer is perfect to dance and get exact same data as offline collected one.

For example, KNN has high chance to output many data misclassified as the model is very simple as there is little training and testing as finding the most closest group the data point should belong to, may be at that time, a dance move A has data that can be closely related to one of another dance move B in an offline dataset. Examples are shown in section 5.1.8.3.3 where all confusion matrices are provided.

Therefore, we have to design a model that can cope with some "changes" when the person dances differently in offline and streaming mode. We say the model is **generalized** enough to not overfit on the offline mode.

5.1.8.3.2 Training and testing data with different ways

Cross validation 5-fold when training models

The most efficient way is to train data with 5-fold cross validation (5-fold CV) and we get 5 sub-models, then the final prediction is the average cross 5 these sub-models and we get output final dance moves.

We use the process of (repeated) CV to obtain a relatively stable performance estimate for a model instead of improving it. This approach allows us to try out different model types and parametrizations which are differently well suited for the problem.

Using CV we obtain many different estimates on how each model type and parametrization would perform on unseen data. From those results we usually choose one well suited model type which we will use, then train it again on all (training) data.

The reason for doing this many times (different partitions with repeats, each using different partition splits) is to get a stable estimation of the performance - which will enable us to e.g. look at the mean/median performance and its spread (would give you information about how well the model usually performs and how likely it is to be lucky/unlucky and get better/worse results instead).

However, this method usually takes more time to train, and test when it is in the streaming model, as we have to load 5 models, then average predictions.

Produce a single model

This approach encourages us to produce one model that covers all data collected so far. With this model, we can perfectly fit all data in offline mode and have very high accuracy. However, since this model only fits data that is trained, when it comes to dealing with streaming data, overfitting will likely occur.

However, this will give us less time to predict dance moves and shorten the latency. So, our group decides to move on with this approach and think of other ways to compensate for overfitting.

Preventing overfitting

There are many ways to prevent overfitting, be it academic or practical ways.

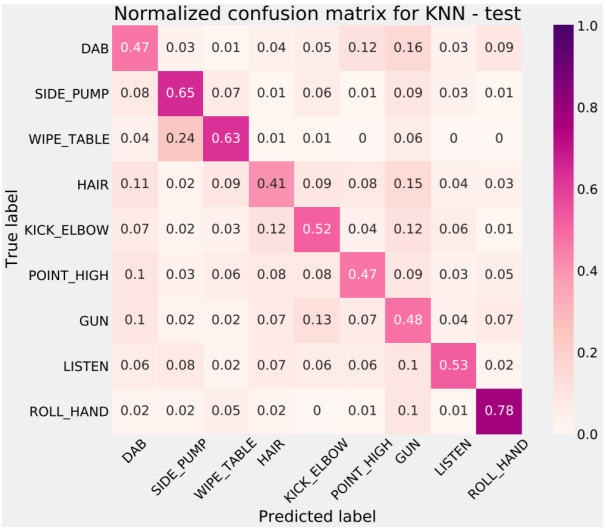
In practical ways, while dancing, our dancers tend to dance with different styles and different angles to create a “variety” of dancing data. This will reduce the fitting of the model an on dancers, we also exchange dancers among groups by inviting other group people to ry on our model, and we will collect how accurate of model on “strangers”

In academic ways, we will apply some techniques to prevent overfitting, such as adding regularization or some layers (dropout layer and normalization batch layer for neural network models)

5.1.8.3.3 Model performance and analysis

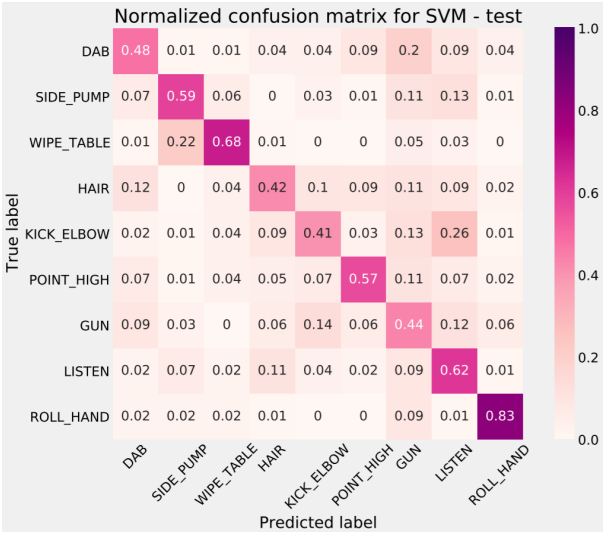
Among the neural network choices, we choose MLP as our main model instead of CNN or LSTM. The main reason is because the other two models are long to train and their accuracies are almost the same as MLP.

Each person has 6 CSVs, 1 CSV will be used as test data and other 5 are used as CSV. Since 1 CSV has "unseen" data, this can act as data coming from streaming mode. Below are confusion matrices for both traditional models and neural network models

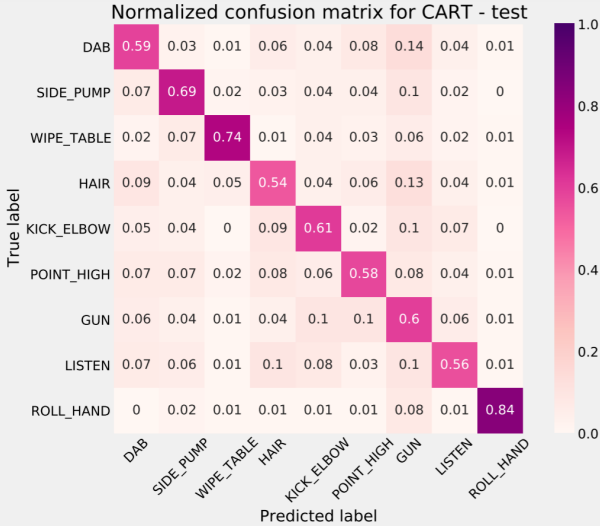


KNN has high chance to output many data misclassified as the model is very simple as there is little training and testing as finding the most closest group the data point should belong to, may be at that time, a dance move A has data that can be closely related to one of another dance move B in an offline dataset.

DAB, SIDE_PUMP, WIPE_TABLE, HAIR, KICK_ELBOW, POINT_HIGH, GUN, LISTEN, ROLL_HAND have probability of 0.42, 0.48, 0.24, 0.63, 0.66, 0.49, 0.8, 0.22, 0.14 being misclassified as other labels. This model is not good as the mis-classification rate is quite high.

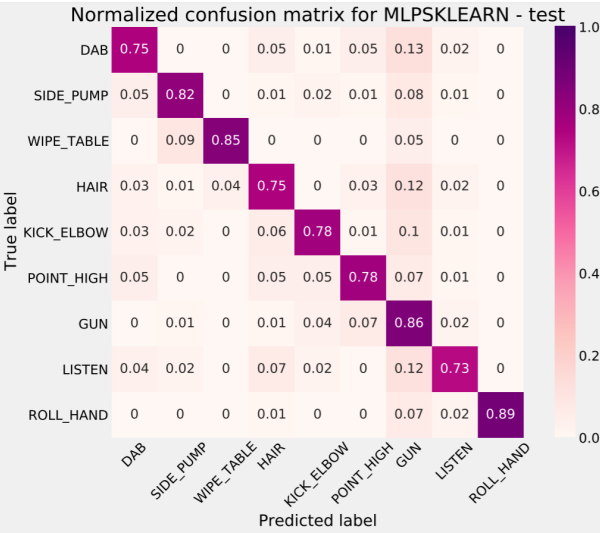


Moving onto the SVM model, this model tries to find the hyper plane and separate data. This model also has a high mis-classification rate, like KNN as this model also focuses on the spatial distribution of data points. Data located nearby are easily grouped into the same block.



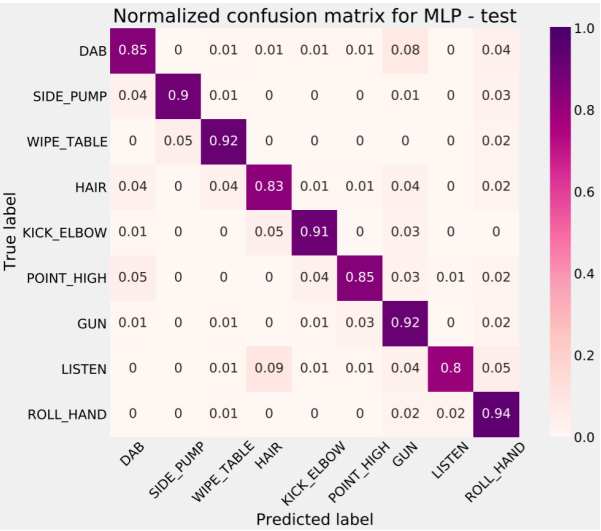
For the CART model, most actions have accuracy around 60% to 80%, this model is not good as well because the accuracy is not high enough.

The accuracy increases when the model becomes more sophisticated, but not good enough as the decision tree is only if-else cases and is not really robust to outliers coming from streaming data.



When we use the MLP model provided by sklearn library, there is an improvement in accuracies , which are around 80% with SIDE_PUMP, WIPRE_TABLE, GUN and ROLL_HAND to be mostly easy to predict.

The accuracy increase when we introduce the simple neural network from SKlearn and this indicates we should incest more time on neural network structures.



Therefore, we go one step further when designing our own neural network models. The structure will be shown as below.

This customized network performs better than the one in SKLearn with every move having accuracy over 80%.

In this customized network, we added Batch Normalization layer.

Batch normalization is a technique designed to automatically standardize the inputs to a layer in a deep learning neural network.

Once implemented, batch normalization has the effect of dramatically accelerating the training process of a neural network, and in some cases improves the performance of the model via a modest regularization effect.

The model structures are shown as below:

```
def build_baseline(clf_name, input_shape=(128, 6, 1), output_dim=6, lr=0.001):
    if clf_name == 'mlp':
        model = Sequential()
        model.add(Flatten(input_shape=input_shape))
        model.add(Dense(256))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dense(128))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dense(output_dim))
        model.add(Activation("softmax"))
        model.compile(
                loss="categorical_crossentropy", optimizer=optimizers.Adam(lr=lr),␣
 ↪metrics=["accuracy"]
        )
    elif clf_name == 'cnn':
        model = Sequential()
        model.add(Conv2D(int(window_size/2), kernel_size=(5, 1), input_shape=input_shape))
        model.add(Activation("relu"))
        model.add(Conv2D(int(window_size/2), kernel_size=(5, 1)))
        model.add(Activation("relu"))
        model.add(Conv2D(int(window_size/2), kernel_size=(5, 1)))
        model.add(Activation("relu"))
        model.add(Conv2D(int(window_size/2), kernel_size=(5, 1)))
        model.add(Activation("relu"))
        model.add(Flatten())
        model.add(Dense(window_size))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dense(window_size))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dense(output_dim))
        model.add(Activation("softmax"))
        model.compile(
                loss="categorical_crossentropy", optimizer=optimizers.Adam(lr=lr),␣
 ↪metrics=["accuracy"]
        )
    elif clf_name == 'deep_conv_lstm':
        model = Sequential()
        model.add(Conv2D(int(window_size/2), kernel_size=(5, 1), input_shape=input_shape))
        model.add(Activation("relu"))
        model.add(Conv2D(int(window_size/2), kernel_size=(5, 1)))
        model.add(Activation("relu"))
        model.add(Conv2D(int(window_size/2), kernel_size=(5, 1)))
        model.add(Activation("relu"))
```

```
        model.add(Conv2D(int(window_size/2), kernel_size=(5, 1)))
        model.add(Activation("relu"))
        model.add(Reshape((input_shape[0]-16, input_shape[1]*int(window_size/2))))
        model.add(LSTM(window_size, activation="tanh", return_sequences=True))
        model.add(Dropout(0.5, seed=0))
        model.add(LSTM(window_size, activation="tanh"))
        model.add(Dropout(0.5, seed=1))
        model.add(Dense(output_dim))
        model.add(Activation("softmax"))
        model.compile(
                loss="categorical_crossentropy", optimizer=optimizers.Adam(lr=lr),␣
↪metrics=["accuracy"]
        )
    else:
        models = dict()
        models['knn'] = KNeighborsClassifier(n_neighbors=7)
        models['cart'] = DecisionTreeClassifier()
        models['logReg'] = LogisticRegression()
        models['svm'] = SVC()
        models['bayes'] = GaussianNB()
        models['bag'] = BaggingClassifier(n_estimators=100)
        models['rf'] = RandomForestClassifier(n_estimators=100)
        models['et'] = ExtraTreesClassifier(n_estimators=100)
        models['gbm'] = GradientBoostingClassifier(n_estimators=100)
        models['mlpSKlearn'] = MLPClassifier()
        model = models[clf_name]
    return model
```

Figure 5.1.8.o: Structure of models used in experiment

Model diagnosis using learning history curve for neural network

When training the neural network, we decide the number of epochs based on how stable the validation loss is.

For example, if the data is well explored and defined, the model should not take too long to train, in this example, we can stop after 30 epochs. In our project, the model can stop maximum around 100 epochs by hard-coding the number of epochs.

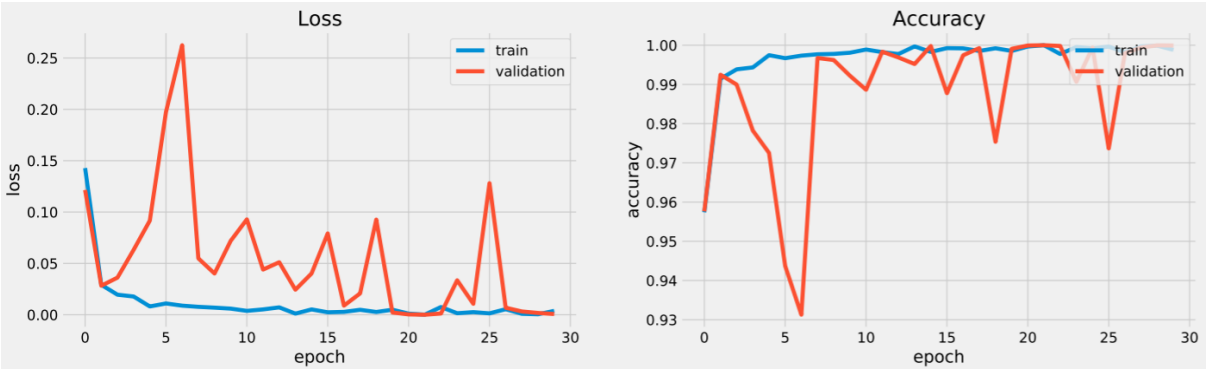


Figure 5.1.8.p: History learning curve when we try with epochs=30

Another example is we can use Early Stopping objects to stop whenever needed. Additionally, after adding Early stopping, we can add Model Checkpoint, Customized logger and CSV Logger (which create csv for network history training and validating).

```
class PeriodicLogger(Callback):
    """Logging history every n epochs"""
    def __init__(self, metric="accuracy", logger=None, verbose=5, epochs=None):
        self.metric = metric
        self.verbose = verbose
        self.epochs = epochs
        self.logger = logger
    def on_epoch_end(self, epoch, logs=None):
        epoch += 1
        if self.verbose == 0:
            return
        if epoch%self.verbose == 0:
            msg = " - ".join([
                f"Epoch {epoch}/{self.epochs}",
                f"loss: {utils.round_number(logs['loss'], 0.00001)}",
                f"{self.metric}: {utils.round_number(logs[self.metric], 0.00001)}",
                f"val_loss: {utils.round_number(logs['val_loss'], 0.00001)}",
                f"val_{self.metric}: {utils.round_number(logs[f'val_{self.metric}'], 0.00001)}", ])
            self.logger.debug(msg)
            print(msg)

def create_callback(
        model: Model, path_chpt, logger=None, patience=100, metric="accuracy", verbose=5,␣
↪epochs=None):
    """callback settinngs
    Args:
        model (Model)
        path_chpt (str): path to save checkpoint
    Returns:
        callbacks (List[Any]): List of Callback
    """
    callbacks = []
    callbacks.append(EarlyStopping(monitor="loss", min_delta=0, patience=patience, verbose=1,␣
↪mode="min", restore_best_weights=True))
    callbacks.append(ModelCheckpoint(filepath=path_chpt, save_best_only=True))
    callbacks.append(PeriodicLogger(metric=metric, logger=logger, verbose=verbose, epochs=epochs))
    callbacks.append(CSVLogger(f"{path_chpt}.csv", append=True, separator=','))
    return callbacks
```

5.1.9 Dancer position algorithm

The dancer position is determined based on the left right information from the Beetle. If left is detected, we update the position of the dancer by -1, and when right is detected, we update the position of the dancer by +1.

Before we send the position to the server, we would need to perform a reversal of the positions as the evaluation server takes in the dancer number instead of positions of dancers. We would also perform an integrity check to see if the dancers are in the correct positions, in the event where there are multiple dancers occupying the same positions, we would use heuristics to determine which left right information was mispredicted and correct it accordingly.

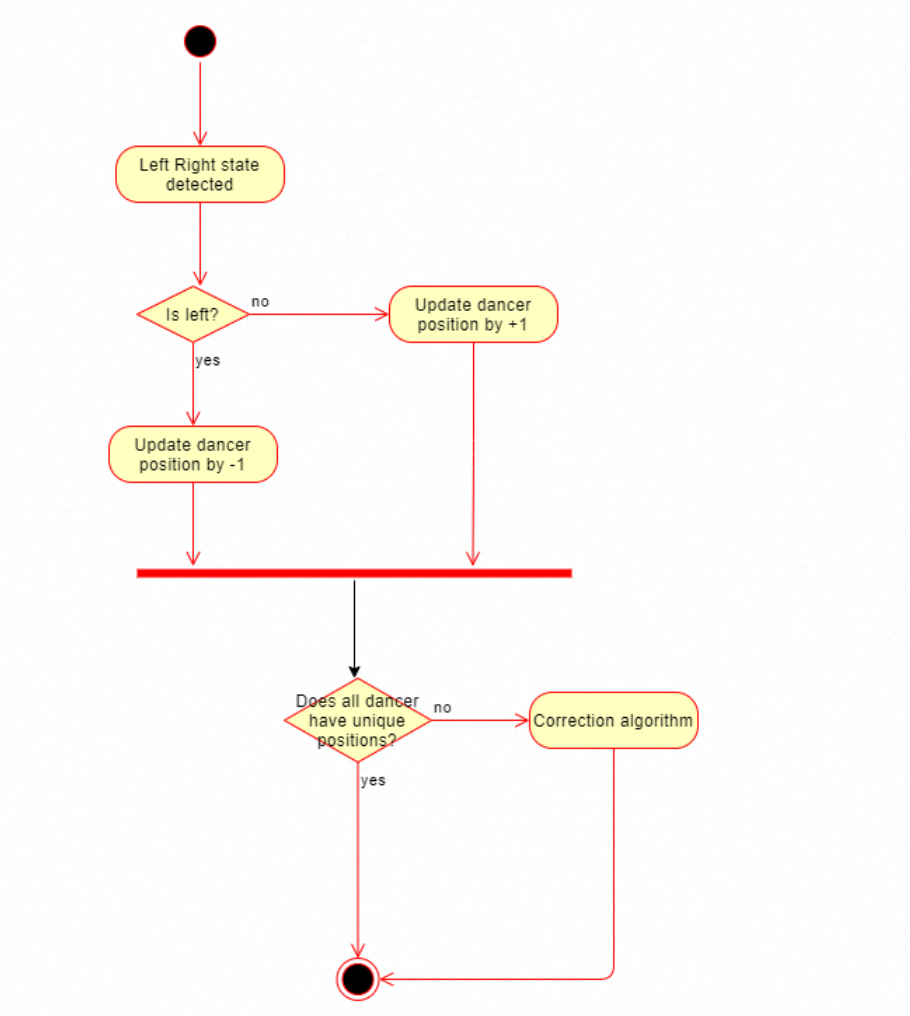


Figure 5.1.8.q: UML activity diagram for positions

Section 5.2 Software Dashboard

The objective of the software dashboard is to provide an intuitive environment for the user to interact with the sensor data streamed from the wearable and analytics provided by the machine learning algorithm. As such, there are two main requirements of the dashboard: the first is to provide a functional software that will satisfy the user stories; the second is to provide rich data visualization that helps users better understand, and if necessary make better decisions with the data and analytics presented.

5.2.1 Software Planning

The planning phase consists of two iterative stages - brainstorming and feedback. The brainstorming stage involves the team in deciding the target audience of the dashboard, as well as ideating possible features and data visualization types that are beneficial to the users. A minimum viable product in the form of wireframes will also be conceptualized at this stage. The feedback stage primarily revolves around conducting user surveys to validate the hypothesis generated during the internal planning stage and gathering feedback from users.

5.2.1.1 Brainstorming

5.2.1.1.1 Target Audience

From the target audience identified in section 1, the team has further narrowed down the target audience of the dashboard to the following two groups of users:

1. Dancers who are interested in their performance and sought to improve their abilities
2. Dance instructors who want to easily identify weaker dancers and gauge fatigue level of the dancers

5.2.1.1.2 User Stories

After identifying the possible target audience, the team has come up with user stories and associated functional requirements of the dashboard as shown in the table below:

User	User Stories	Functional Requirement
Dancer	As a dancer, I should be able to review	Input expected dance steps

	past dance moves that I am poor at so that I can work on those dance steps to improve myself	Flag out dance steps that are done incorrectly
		Save current session's analysis for future review
	As a dancer, I would want to know how I can improve a dance step easily so that I know what to work on	Display calculated and raw values (velocity, force, angle etc)
		Provide recommendation based on the calculated and raw values
Dance instructor	As a dance instructor, I should be able to identify students who are performing the wrong dance routine so that I can correct them in time	Input expected dance steps for each student
		Display predicted dance steps for each dancers in real-time
		Flag out dance steps that are done incorrectly for each dancer in real-time
	As a dance instructor, I should be able to identify students who are in the wrong position so that I can correct them in time	Input expected positions for each student
		Display predicted position for each dancers in real-time
		Flag out dancers that are at the wrong position in real-time
	As a dance instructor, I would want to be able to quickly identify how a student can improve his/her dance routine so that I can attend to more students	Display calculated and raw values (velocity, force, angle etc) for each student
		Provide recommendation for each student based on the calculated and raw values
	As a dance instructor, I should be able to gauge my students' fatigue level so that I know when to call for breaks	Display fatigue level of students in real-time
		Provide recommended break-time based on fatigue level
	As a dance instructor, I should be able to review my students' past performance so that I can gauge if the	Save current session's analysis for future review

	student has improved	Comparison between analyses of different sessions
	As a dance instructor, I should be able to group my students so that I can review the performance of each class easily	Each session can consist of a group of student or a student

Figure 5.2.1.1.2a Table of initial user stories and their associated functional requirements

5.2.1.1.3 Data Visualization

There are primarily two types of data that will be displayed on the dashboard - sensor data and prediction results. Sensor data are raw data generated from the sensors. Prediction results are the predicted labels (in the case of a classification problem) or values (in the case of a regression problem) generated by the machine learning algorithm.

Sensor data that will be displayed on the dashboard include the acceleration along x, y and z axes from the accelerometer, the rate of rotation around the x, y and z axes from the gyroscope and muscle fatigue from electromyography (EMG).

Predicted data that will be displayed include the predicted dance move, predicted dance position and predicted fatigue level.

In deciding how to best represent the aforementioned data, we employed heuristics from The Visual Display of Quantitative Information by Edward Tuft. There are two main heuristics that are considered in our evaluation: lie factor and data-to-ink ratio.

Lie factor refers to the ratio between the effect shown on the graph and the size of the effect in the actual data. It gauges whether a graph is misleading and in some instances, truthful. Data-to-ink ratio refers to the ratio of the data-ink and the total ink to print the graph. A high data-to-ink ratio indicates that a graph has excessive redundant elements which can confuse and distract the audience.

Sensor data are generally time-series data and are best represented with a line chart due to their continuous nature. However, the main point of contention here is whether or not to superimpose data that belong to the same group i.e. have the same y-axis units. For instance, while we can have three line charts that show the sensor data from the accelerometer with each chart representing one of the 3-dimensional planes, we can also have a single line chart that encompasses the accelerometer data for all planes.

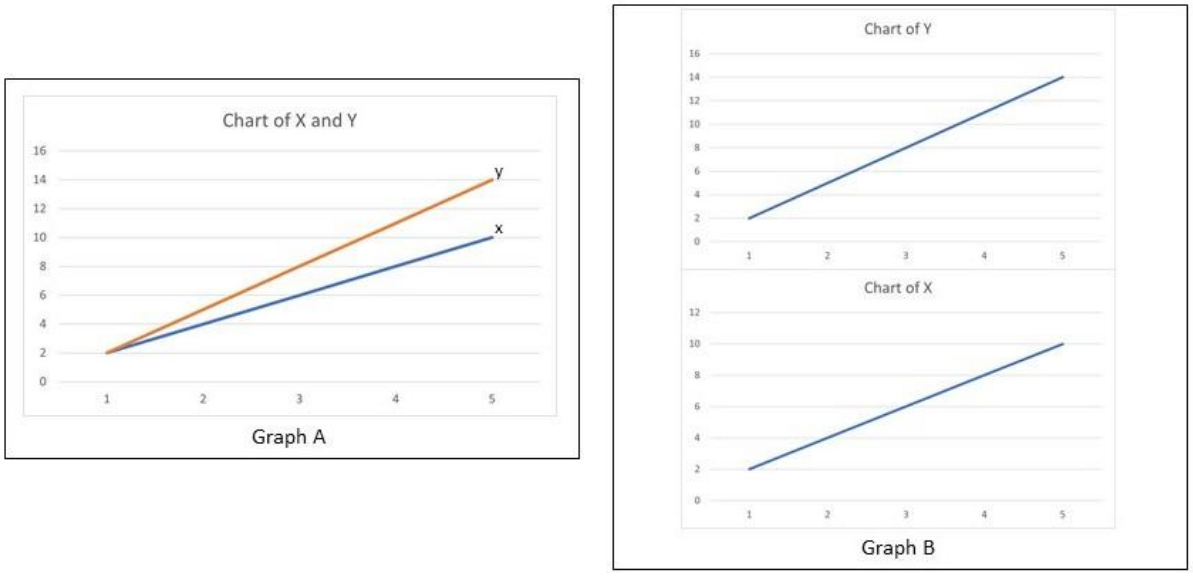


Figure 5.2.1.1.3a Both graphs present the same X and Y dataset; it might seem like X and Y are the same in graph B whereas the difference is more obvious in graph A

Consider the former design where there are three graphs each representing the acceleration of a single axis in the 3-dimensional plane, and that the total dimension (i.e. the height and width) of the graphs are constant. Such a design will incur a high lie factor since it is difficult to make comparisons between the graphs - the size of the graphs are the same, but the ticks on the axes can differ. While it is possible to go around this by increasing the size of the graph when necessary, this increases the data-to-ink ratio. The latter design which condenses all three sensor values into a single graph eliminates this problem without increasing any data-to-ink ratio and is our preferred design.

Predicted data are discrete data and can be represented in a variety of charts such as bar charts and scatter plots. However, in our use case where the predicted data involves classification labels (which dance move, and which position), and the audience is only interested in the predicted value at the current instance, using charts to represent such data unnecessarily increases the data-to-ink ratio since the chart representation adds little to no value to the audience - there are no comparisons to be made or trends to be identified. It thus makes more sense to not use any data visualization tool to represent such data, but to instead focus on the user experience behind displaying such data verbatim.

5.2.1.1.4 Wireframes

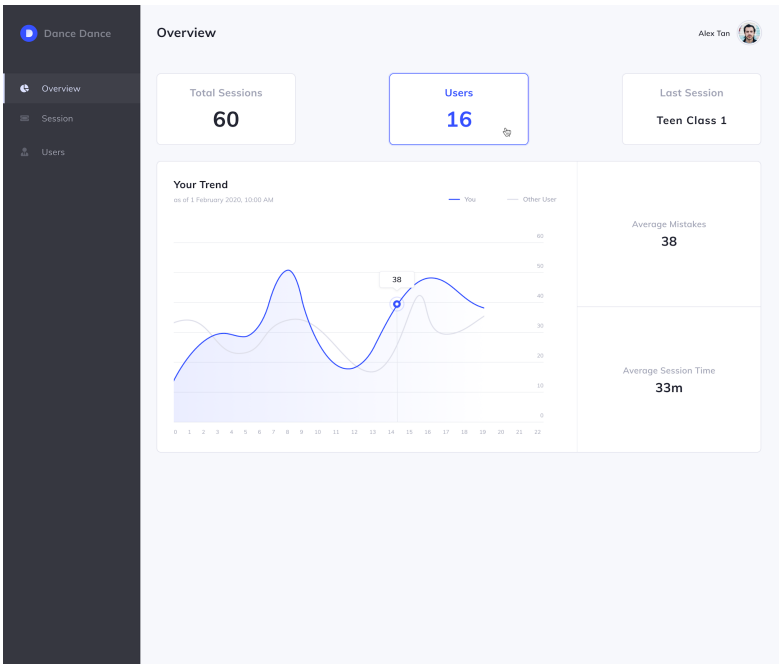


Figure 5.2.1.1.4a Design of dancer analytics screen - users can view their overall statistics and access to other screens using shortcut cards at the top of the screen or the side navigation menu

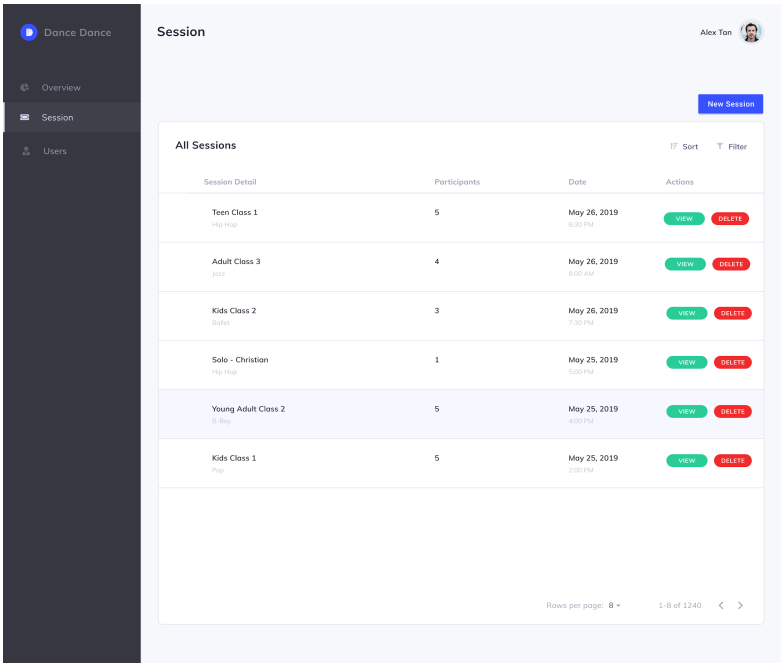


Figure 5.2.1.1.4b Design of session screen - user can view analytics of their past dance sessions by selecting the dance session of interest in the table or create a new dance session through the button on the top right hand corner

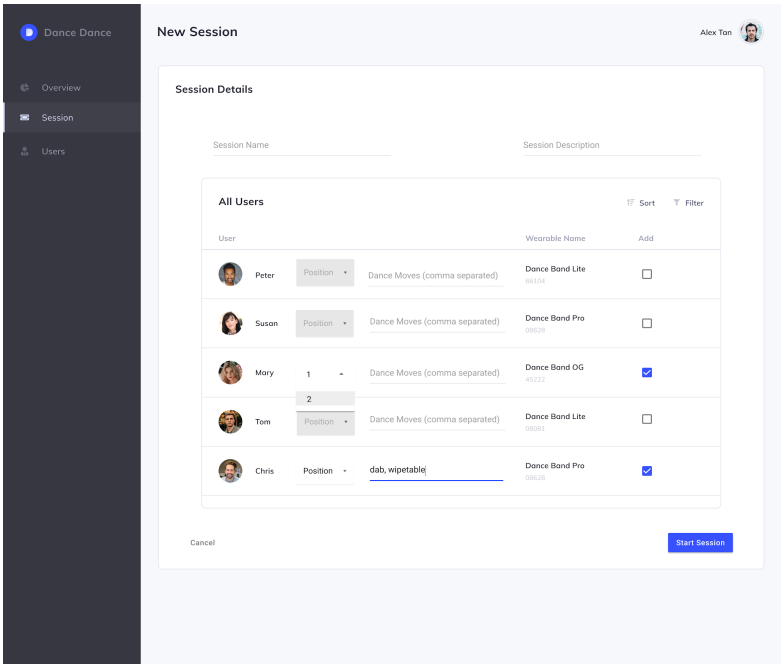


Figure 5.2.1.1.4c Design of dance session creation screen - user can select the dancers participating in the session, input their initial positions and optionally their expected dance moves as comma-separated values and start the session by selecting the bottom right button

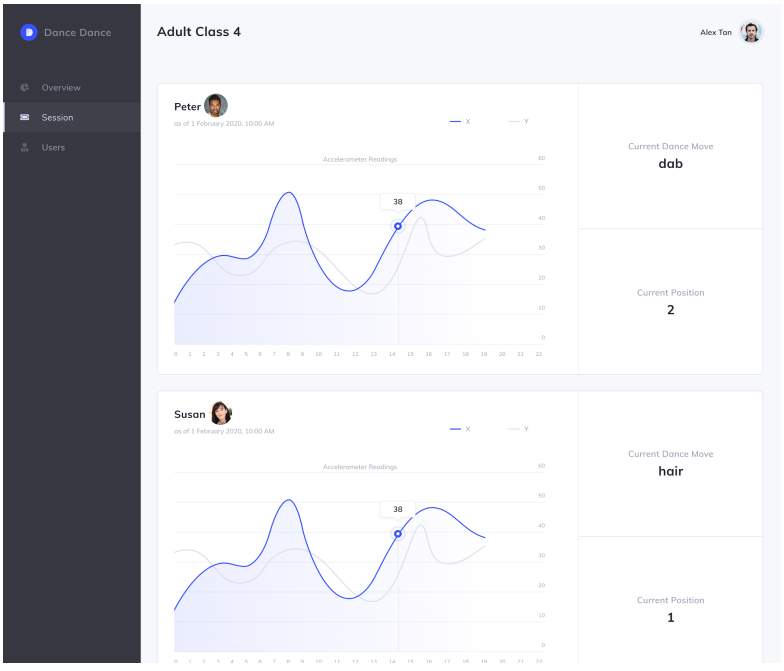


Figure 5.2.1.1.4d Design of session analytics screen - each participant's analytics will be presented as cards where the right hand column shows the predicted data while the left hand column shows the raw sensor data as a line graph

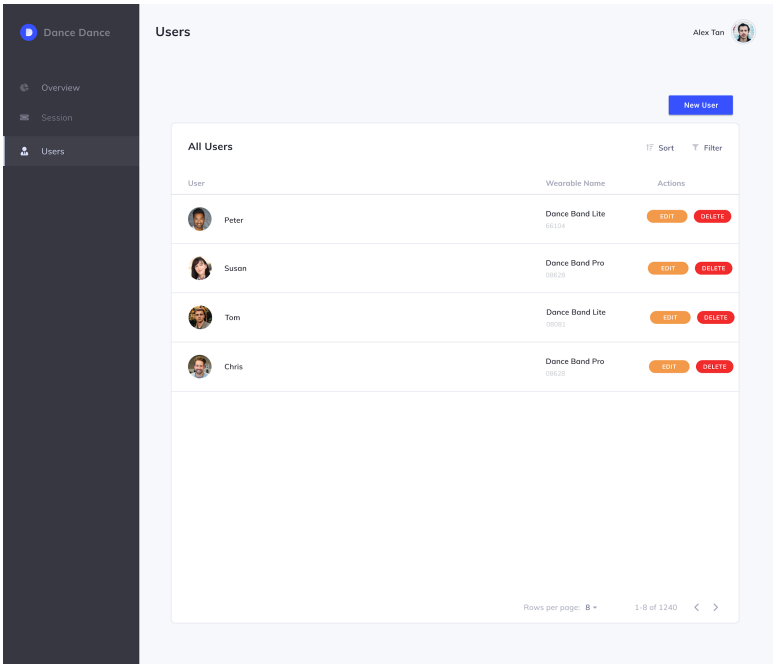


Figure 5.2.1.1.4e Design of dancer screen - user can see the list of onboarded dancers in this screen, make modifications to the existing dancer profile by selecting the correct dancer or add a new dancer through button on the top right corner

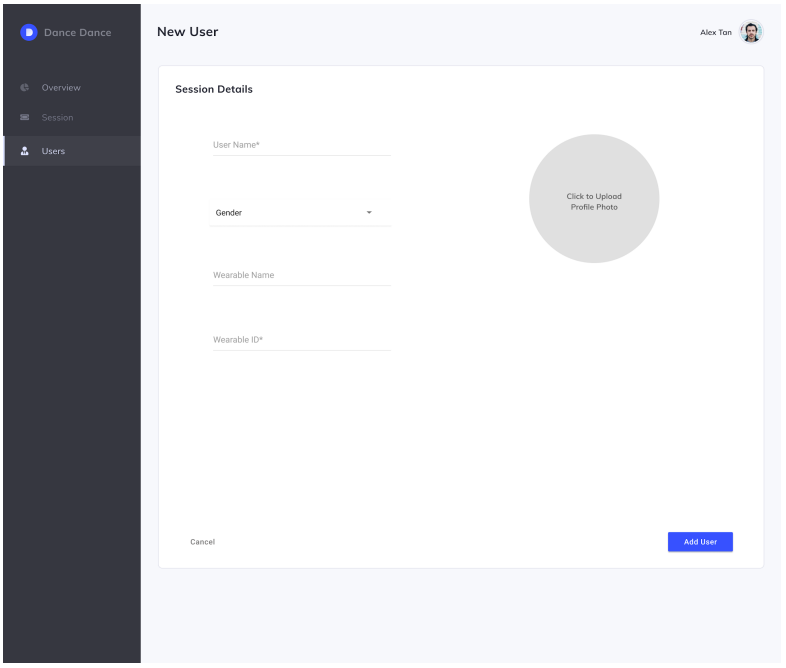


Figure 5.2.1.1.4f Design of dancer onboarding screen - fields required for dancer to be onboarded are the dancer's name and the wearable identification number

5.2.1.2 Feedback

5.2.1.2.1 Flow

The team has solicited 7 participants that fit the profile of the target user for the dashboard. Each participant is briefed on the background of the project. They are then asked a series of questions that gauges the usefulness of the features outlined in the user stories described in the previous section. This is followed by presenting the wireframes of the dashboard and describing the expected behaviour of each element in the wireframes. The survey concludes with gathering feedback from the participants on improvements that can be made to the dashboard as well as any functionalities that the team has overlooked but would otherwise be useful.

A final user testing session was done within the team. Team members were given access to an early prototype of the dashboard, and were asked to complete the entire user journey flow (from onboarding to viewing of analytics) on the dashboard. They were then asked questions related to the user flow and any improvements that can be made to the dashboard.

5.2.1.2.2 User Survey

Users are asked a series of questions that aims at determining the relevance of user stories that the team has come up with. Participants were aware of the background of the project, but were not told the exact features in the dashboard. Furthermore, most questions are open-ended in nature so as to gain organic insights from the users. Questions are also structured such that the team can draw reasonable conclusions on the usefulness of each feature derived from the user stories.

The list of questions asked in sequential order is as follows:

S/N	Question	Purpose
1	What is your primary objective in using such a dashboard?	Understand the desired outcome of using the dashboard from the user's perspective
2	How do you think that the dashboard will help you in achieving the objectives?	Understand the user's expectation of using the dashboard
3	What is the first feature that comes to your mind? Why?	Gauge the most important feature that will address the user's need
4	What are some of the other elements or features that you think are important?	Gauge the user's expectation in terms of the functionalities that are delivered by the dashboard

5	How important is the ability to view instantaneous results to you?	Determine the relevance of real-time aspect of the dashboard from the user's perspective
6	How important is the ability to view sensor data to you?	Determine the relevance of displaying sensor data from the user's perspective
7	How important is the intuitiveness of the dashboard to you?	Determine the importance of user experience for the project
8	Rank 5 - 7 in the order of importance	Determine precedence of each aspect of the dashboard

Figure 5.2.1.2.2a List of user survey questions

5.2.1.2.3 User Feedback

Participants are presented with the wireframes of the dashboard, and a brief explanation of the features associated with and the expected functionalities of each element in the wireframes. This is followed by asking the following questions:

S/N	Question	Purpose
1	Are there any concerns that you might have with regards to the designs that you were presented?	Solicit improvements that can be made to the existing design
2	Do you have any feedback on how the dashboard can better serve your needs?	Solicit features that were overlooked but are useful to the user

Figure 5.2.1.2.2b List of user feedback questions

5.2.1.2.4 User Testing

User testing was conducted internally within the team. Team members were asked to access an early prototype of the dashboard that was designed based on the feedback collected in User Survey and User Feedback and complete a typical user journey. They were then presented with a questionnaire to gather their responses on the onboarding process and the analytics were presented. Questions were generally presented in the form of 6-point likert scale options asking team members how satisfied or confused they were with a certain feature. A 6-point scale is used so as to prevent responses that are "on-the-fence". Some questions also involved team

members selecting multiple options to determine the usefulness of each feature. Feedback on how to further improve the dashboard was also solicited from team members.

The complete user testing process and questionnaire can be retrieved from the Google Form https://forms.gle/TvpdZHMcxZwmyhX86.

5.2.1.3 Findings

5.2.1.3.1 Findings from User Survey

Many of our survey participants found the project interesting, and the primary use case of a dashboard for them is to better understand how well they or their team performed in a dance session. A handful of the participants also mentioned how the dashboard acts as a record of all their dance sessions so that they can keep track of their performance. The bottom line here is that the participants view the dashboard as a tool to measure their performance.

Our survey results also indicated that the most important functionality of the dashboard is the ability to view the predicted dance move and scoring of dance moves in real-time. When asked the first feature that came to their minds, many participants talked about how the dashboard should be able to tell them the dance moves that they did and the accuracy of those dance moves. Many also gave the analogy of video-games like Just Dance and Dance Central where they were able to see how well they performed a particular dance move and instinctively thought that the dashboard would deliver similar features.

When asked about other important features that the dashboard should deliver, the majority of the participants mentioned the ability to view a summary of their performance for the past sessions. Some also expanded on their previous answer of the dashboard's ability to provide feedback on their dance moves, and explained that the ability to see their peer's feedback will be equally useful. Participants also drew inspirations from aforementioned video-games and talked about the possibility of making the dashboard into a competitive tool that allows individual dancers to challenge one another. An interesting thing to note is that almost none of the participants mentioned that the dashboard should display raw sensor data, presumably because such information is irrelevant to the target audience's objective in using the dashboard.

Most of the participants ranked the ability to see real-time analytics as the most important feature of the dashboard. Intuitiveness of the dashboard is the second most popular option. Ability to view sensor data was ranked the last for all participants.

Many of the participants agreed that the entire purpose of the dashboard hinges on the fact that it is able to produce real-time stats of the dance moves and thus rank that feature as highest. A significant number of participants also stressed the importance of intuitiveness of the dashboard, since the onboarding process, especially in a group setting, can be overwhelming and frustrating if the user experience of the dashboard is poor. Displaying the analytics of individual

dancers in a group setting can also be confusing if the user interface is poorly designed, and is a concern for a number of the participants. Almost all the participants were puzzled at the dashboard's ability to display sensor data as they were unsure of its purpose. Notably, many of the participants asked what they can do with the sensor data, with some suggesting that the addition of that might possibly make the dashboard unnecessarily convoluted.

The insights gathered from the user survey can be summarized as follows:

1. Target audience's main objective in using the dashboard is to measure their performance
2. The most important functional requirement is the ability to view real-time analytics of dance moves
3. In addition to their own analytics, target audience is also interested in learning their peers' analytics
4. Users want a summary of their previous analytics to be readily accessible
5. Users entertained the possibility of introducing competitive elements to the dashboard
6. Intuitiveness of dashboard is equally important as onboarding and displaying many dancers' analytics simultaneously can be a challenge
7. Displaying sensor data is not very useful for target audience

5.2.1.3.2 Findings from User Feedback

Most participants viewed the dancer analytics screen favorably, and were aware of the expected functionalities of each element. They were however a little confused with the term "Average Mistakes" and suggested that "Average Accuracy" as a percentage of correctly performed dance moves per session would make more sense. A few participants also pointed out that the dashboard should also indicate if one is faster, slower or on-time when executing his or her dance moves.

Participants were generally pleased with the onboarding process, describing it as intuitive and simple. They were especially impressed with the use of tables to enroll dancers for a session. However, there was some confusion about putting in the expected dance steps for each participant as comma separated values. Some participants also suggested the functionality of creating groups so that they do not have to re-enroll frequent dancers for each session. While there was not much inconvenience with the synchronous nature of creating dance sessions (i.e. a created session that has to occur immediately), participants thought that the ability to create asynchronous dance sessions (i.e. creating a session that occurs in the future) especially in the context of a recurring dance class.

The session analytics screen was met with mixed reaction from the participants. The primary complaint that many participants had was the lack of important information like whether a dancer is offbeat, the expected dance moves and the dance moves that were already executed by the dancer. Participants also found the line graph depicting the sensor data unnecessarily distracting since it does not add any value, resonating with the earlier finding in the user survey. Instead, many recommended the option to hide the graph, or translate those sensor data into something actionable (e.g. showing a pop-up that directs the dancer to swing his or her right hand faster, or to put more weight on his or her right feet etc). A handful of participants also felt that the arrangement of individual dancer's cards that displays the associated dancer's analytics in rows was not user friendly as having more dancers in a session will mean that the user has to scroll the dashboard constantly to monitor all dancers. Some participants thought that a visual representation of the current dance move being executed will give the user a better idea of the dance move executed.

The insights gathered from the user feedback and be summarized as follows:

1. Misleading nature of the term "Average Mistakes" which should be substituted with "Average Accuracy"

2. Dancer analytics screen should also indicate how well a dancer perform rhythm-wise

3. Input for expected dance moves for each session can be more intuitive

4. Enrolling frequent dancers can be made simpler with group functionality

5. Creating sessions that occur in the future will likely be useful

6. Important analytics like dancer's performance rhythm-wise, expected dance moves and executed dance moves are missing

7. Line graph depicting sensor data is unnecessary and should be translated into something that is actionable

8. Arrangement of dancer's analytics causes inconvenience to the user

9. Visual aid of the dance move executed is useful

5.2.1.3.2 Findings from User Testing

Most team members were satisfied with the onboarding process, giving an average satisfaction rating of 5.5 out of 6. Intuitiveness of the overall onboarding process was rated a 5.1 out of 6, with the major complaint being the un-intuitive nature of putting in expected dance moves and positions as comma-separated values. One of the team members also suggested a quick play feature that allows users to try out the system without going through the onboarding process.

Team members were also generally satisfied with the analytics presented for each dance session, giving an average satisfaction rating of 5.5 out of 6. Most team members found the rhythmic performance over time line chart redundant, and voted dance move and position accuracy over time line charts as the most useful analytics. Most team members also shared the same sentiment as our participants in the User Feedback on the displaying of sensor data, commenting that it is not really useful if it does not present anything actionable.

In terms of the analytics presented for each dancer, most team members were satisfied with the analytics provided, giving an average satisfaction rating of 5.2. Most team members found the total session count and average fatigue level redundant, and voted average dance accuracy across sessions as the single most important analytics. Team members also suggested replacing fatigue level with something more informative such as calories burned.

The insights gathered from the user testing session can be summarized as follows:

1. Input for expected dance moves and positions need to be clearer
2. An option that allows dancers to use the dashboard without going through the onboarding process
3. Sensor data that does not present anything actionable is redundant
4. Accuracy is the most important analytics
5. Fatigue level can be presented in a more informative manner such as calories burned

5.2.1.3.3 Revised User Stories and Functional Requirement

Based on the user survey, feedback and testing sessions, a revised functional requirement group based on component with priority indicated was drawn as follows:

Component	Functional Requirement	Priority
Dancer	Dancer list	High
	Add dancer profile	High
	Edit dancer profile	Medium
	Delete dancer profile	High
Group	Group list	Low
	Create group and group composition	Low

	Edit group composition	Low
	Delete group and group composition	Low
Session	Session list	High
	Create session	High
	Edit session	Low
	Delete session	High
	Manage future sessions	Low
Session Creation	Add and drop dancers	High
	Add and drop groups	Low
	Input wearable ID	High
	Input expected dance moves	Medium
	Input expected positions	Medium
	Input initial positions	High
	Quick play for unregistered dancers	Medium
Session Analytics	Show dancer's current dance move/position	High
	Show dancer's expected dance move/position	Medium
	Show dancer's executed dance moves/positions	High
	Show dancer's dance move/position accuracy	High
	Show dancer's fatigue level	High
	Show dancer's change in move/position accuracy	Medium
	Show team's dance move/position accuracy	High
	Show dancer's sensor data (with the ability to hide the sensor data)[1]	High

[1]Although this functionality was not well-received during the survey and feedback sessions, it is imperative for the project since it allows us to evaluate the working of the sensor. To address the issue faced by the audience, the ability to hide this data will be implemented together with this functionality.

Dancer Analytics	Show average dance move/position accuracy across all sessions	High
	Show average dance move/position accuracy of each participated session	High
	Show average calories burned per session	Medium
	Show average session time	Medium
	Show all participated sessions	High
	Shortcut to participated sessions	Medium

Figure 5.2.1.3.3a List of revised functional requirements

The revised user stories have been covered earlier under section 1.3.

5.2.1.3.4 Revised Wireframes

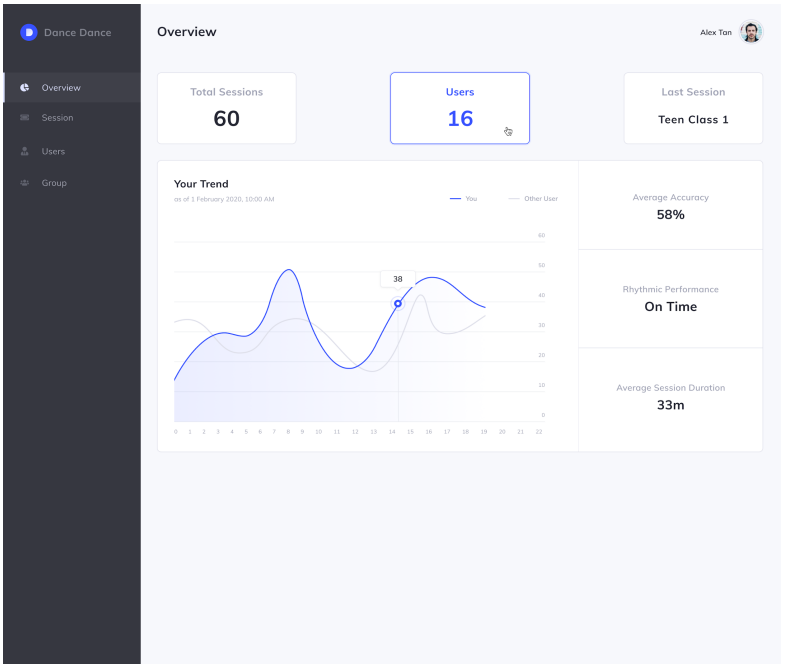


Figure 5.2.1.3.4a Revised design of the dancer analytics screen - overall rhythmic performance of the user is included in this revision

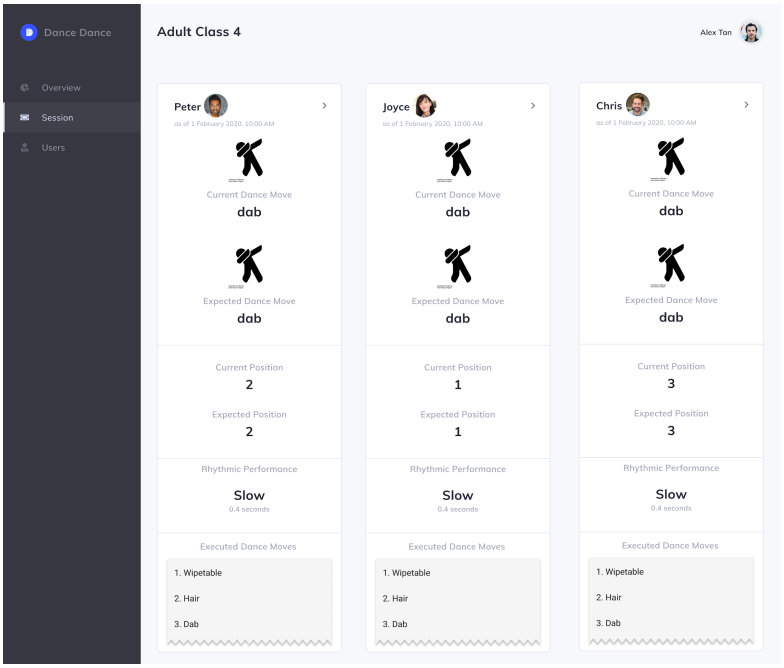


Figure 5.2.1.3.4b Revised design of dance analytics screen - compact view with columnar design of dancer analytics with sensor data being hidden by default; sensor data can be unhidden by selecting the chevron next to the dancer's name to go into expanded view

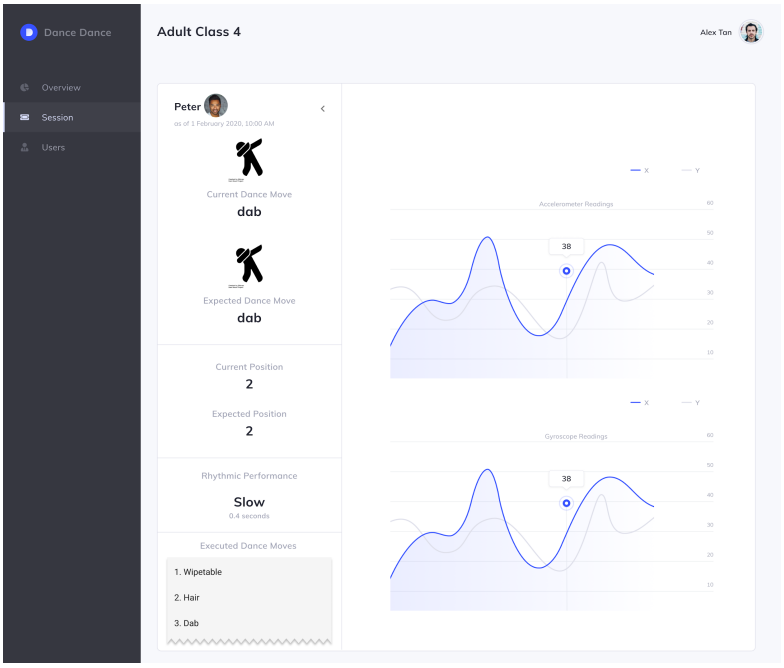


Figure 5.2.1.3.4c Design of dancer analytics screen expanded view - shows sensor data in this view; user can go back to compact view by selecting the chevron next to the dancer's name

5.2.2 Software Design

5.2.2.1 Software Stack

Taking on a "JavaScript Everywhere" approach, the team will be adopting JavaScript frameworks for the majority of the software stack. This is to keep the development simple since only mastery of a single programming language is required to manage and maintain all the codebases involved. Furthermore, as JavaScript frameworks are getting increasingly popular, adopting a "JavaScript Everywhere" approach means that we can take advantage of the community's support and gain access to the vast amount of existing libraries and packages to further streamline our development.

5.2.2.1.1 Client-Side

The dashboard's frontend will be powered by React, a JavaScript framework developed by Facebook. React does not have overly complicated features like two-way data binding and dependency injection which keeps the overall learning curve shallow. Yet, it is able to deliver high performance since React applications do not change the actual browser Document Object Model (DOM), but manipulate virtual DOM for rendering. These are the two primary reasons as to why React is chosen over other JavaScript frameworks like Angular and Vue.js.

5.2.2.1.2 Server-Side

The server will be powered by ExpressJS framework with NodeJS runtime. NodeJS with Express servers are easy to configure and offer high performance and comparable scalability to other server-side runtime and frameworks, making it the de facto choice for server-side programming in JavaScript.

5.2.2.1.3 Database

Database choice was a challenge as the considerations were not straightforward. Other than the fact that the database has to interface well with the server, it also has to offer the optimal performance for our use case.

Our initial intuition was to use MongoDB (which forms the modern MERN stack) since it plays well with our NodeJS server, and offers high performance for complex queries. It is also easy to scale the database instance horizontally due to MongoDB's auto-sharding feature and its document-based nature. This means that if we ever require a distributed environment to handle our workload, we can deploy our database fairly easily to such an environment.

However, our use case does not warrant a distributed environment since the majority of the analytics will be done on the Ultra96 and only three wearables will be active at any time. The more pertinent role that the database's plays is to store sensor data and prediction results and thus needs to be performant in handling read and write of streaming time-series data.

Our initial research showed that both TimescaleDB and ClickHouse perform better than MongoDB in both reading and writing of time-series data. Both database solutions also require less storage space as compared to MongoDB, and querying can be done with simple SQL language rather than convoluted JSON objects as in the case of MongoDB. Both database solutions can also scale horizontally fairly easily, allowing us to migrate to a distributed environment if such a need arises (for instance, when the number of active wearables at any point in time increases).

TimescaleDB builds on top of PostgreSQL, allowing it to inherit the reliability, security and connectivity of the latter. As a PostgreSQL extension, it enables PostgreSQL to process time-series dataset efficiently by automatically indexing at timestamp fields and introducing the concept of a hypertable where all data resides in a single table that is partitioned extensively to achieve high read and write speed.

ClickHouse is a relatively new column-based database that prides itself as being one of the fastest online analytical processing (OLAP) databases. Taking advantage of it's column-oriented design, it is optimized specifically for fast inserts and queries, which suits our use case of writing and reading time-series sensor data.

Both TimescaleDB and ClickHouse can seamlessly interface with our NodeJS server since both database solutions provide NodeJS database management systems (DBMS). However, in terms of reliability and support, TimescaleDB has an advantage here since it builds on top of PostgreSQL. We can therefore utilize NodeJS packages meant for PostgreSQL which have been around for a long time for our server-side development. Support is also more accessible with PostgreSQL's already vast community (node-postrgres's repository has 9300 stars whereas node-clickhouse's repository only has about 150 stars).

TimescaleDB also offers better flexibility than ClickHouse since it is built on top of PostgreSQL. For non-time series data that do not require TimescaleDB's automatic indexing of timestamps and hypertable features, we can still implement those tables as traditional PostgreSQL tables. The same however cannot be said for ClickHouse since its structure as a column-based database cannot be changed. Non-time series data like user accounts that are typically row accessed are not suited for column-based databases and might not perform very well, especially when join operations are required.

That said ClickHouse almost always trumps TimescaleDB for time-series data performance wise. In a benchmark that loads 100 million rows, ClickHouse is easily 2 times faster than TimescaleDB, achieving a total time slightly under 300 seconds.

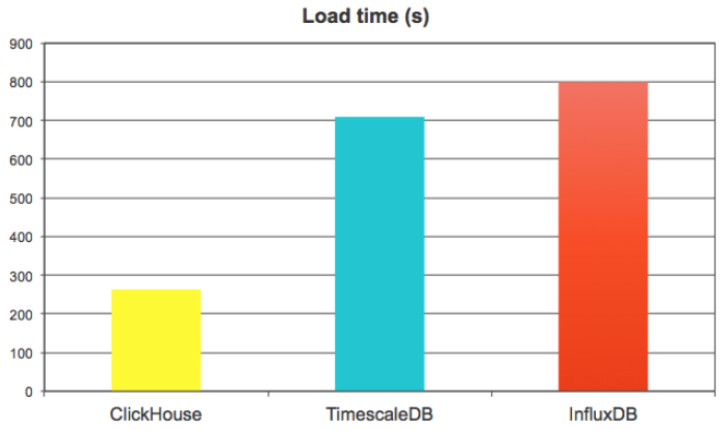


Figure 5.2.2.1.3a Insert performance of ClickHouse, TimescaleDB and InfluxDB [Image Courtesy: Altinity]

In terms of read speed, TimescaleDB generally performs slightly better at light queries, but ClickHouse significantly outperforms in more complex aggregation queries.

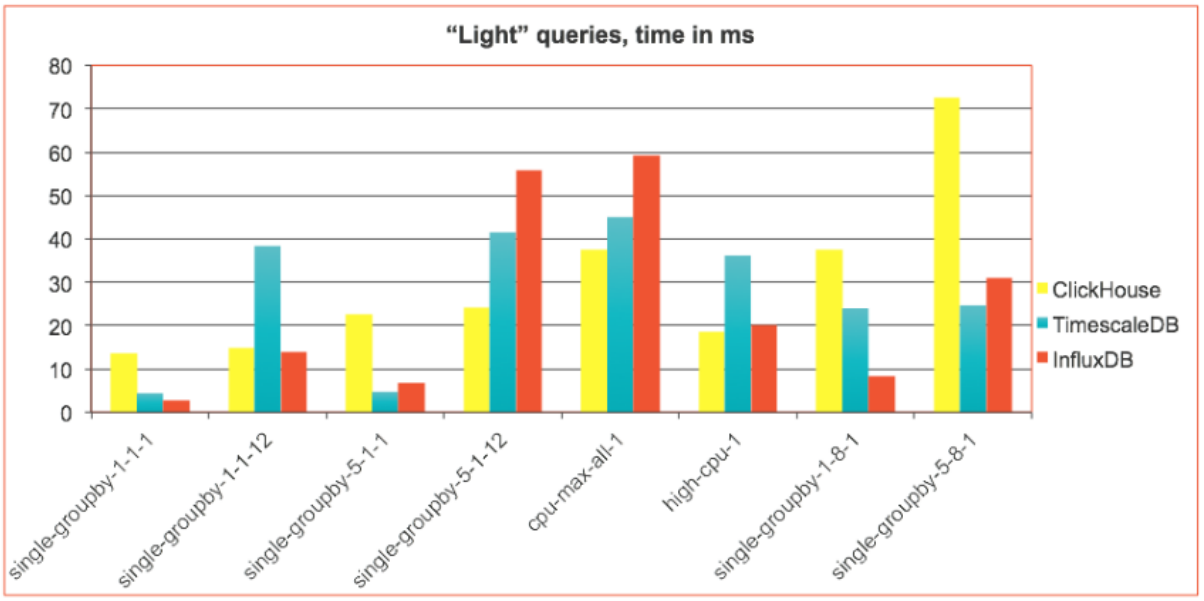


Figure 5.2.2.1.3b Light query performance of ClickHouse, TimescaleDB and InfluxDB [Image Courtesy: Altinity]

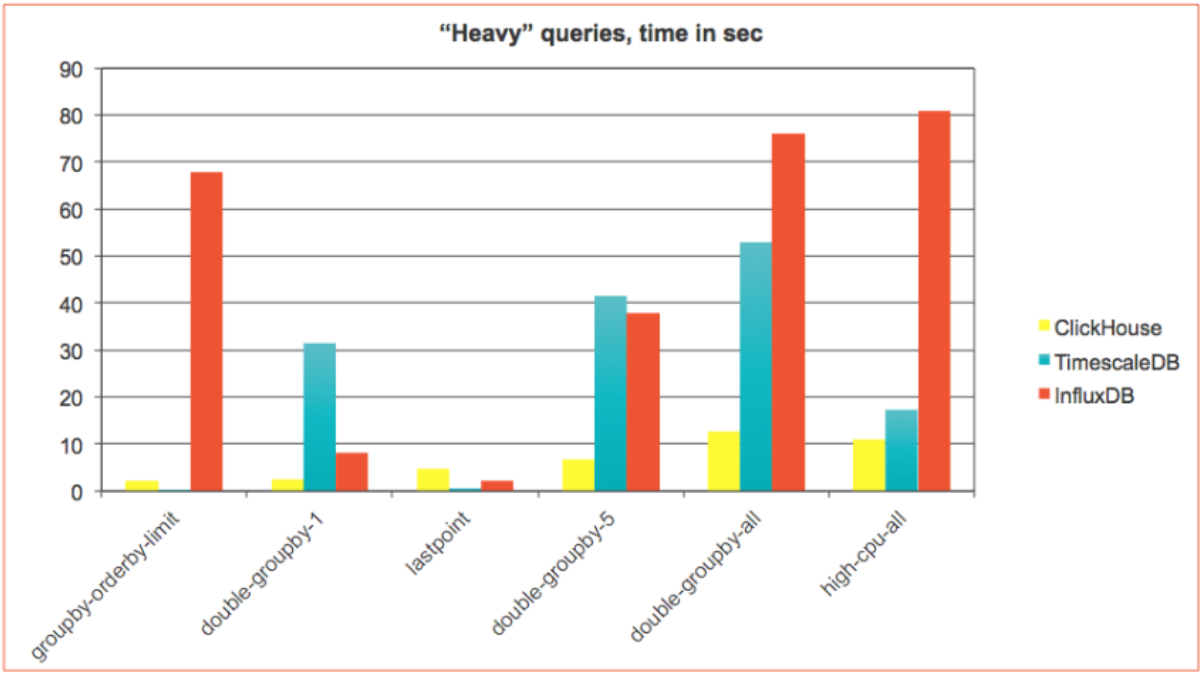


Figure 5.2.2.1.3c Heavy query performance of ClickHouse, TimescaleDB and InfluxDB [Image Courtesy: Altinity]

After taking into account both feasibility and performance, we have decided to work with TimescaleDB. This is primarily because the trade-off in performance is marginal - our use case does not require any complex queries that deal with aggregations since the dashboard will just be fetching raw values in the database for the most part. In fact, TimescaleDB might outperform ClickHouse for our use case since we will be doing simple queries most of the time. Furthermore, working with a stack that delivers better flexibility, reliability and support is more important given the tight timeline of the project.

5.2.2.2 System Architecture

The following diagram shows a high level overview of the system architecture of the dashboard.

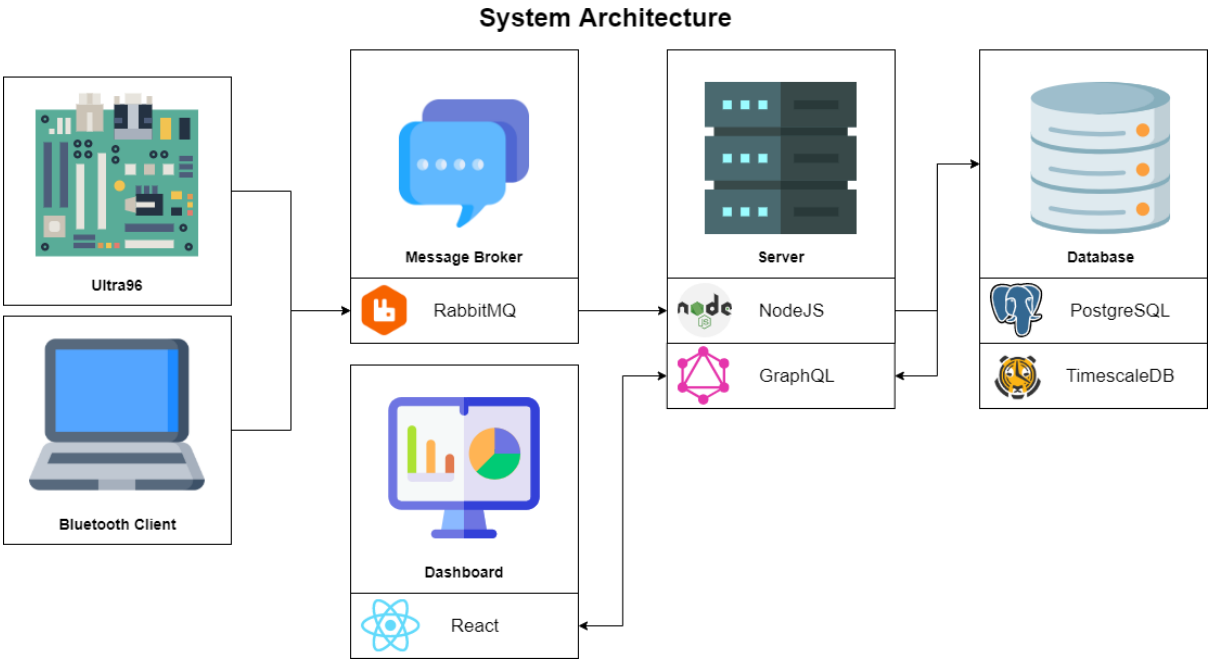


Figure 5.2.2.2a Overall system architecture of the dashboard

The direction of the arrow between each component shows the allowed flow of information between the respective components.

There are two additional stacks - RabbitMQ and GraphQL - in the architecture diagram which were not covered in the previous sections. The purpose and rationale behind these two stacks will be covered in the following sections.

5.2.2.2.1 Dashboard Interface: GraphQL

GraphQL is a data query and manipulation language for application programming interface (API). It utilizes a mixture of HTTP and WebSocket protocols to handle communication between the client and server.

Typically GraphQL uses HTTP protocol, which is useful for fetching data that requires only occasional communication with the server like obtaining a list of dancers, or creating a new session on the dashboard since communication between the client and server closes once a request has been met with a response due to HTTP's response-request model. In its subscription mode however, GraphQL uses WebSocket protocol to allow near real-time fetching of data. This is required for real-time streaming of sensor data and prediction results on the dashboard. Unlike HTTP which closes a connection once the client receives a response resulting in multiple handshakes between client and server, WebSocket only requires a single handshake and the communication channel is kept open until one of the hosts closes the channel. This keeps the

communication latency low, allowing the dashboard to update its state when new data is available almost instantly.

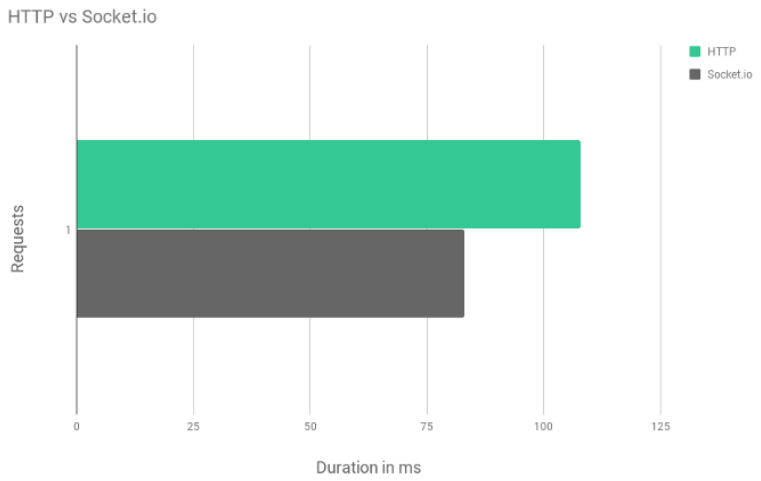


Figure 5.2.2.2.1a Performance comparison between SocketIO (WebSocket) and HTTP [Image Courtesy: David Luecker]

The support for multiple communication protocols makes GraphQL an attractive interface option for the dashboard to the server.

On an unrelated note, an additional benefit of GraphQL is that it allows users to query for a specific set of data without having to specify the result of that query on the server side, solving the issue of under- and over-fetching that plagues traditional Representational State Transfer (REST) API architecture. This added benefit means that we can develop our API rapidly since we no longer have to write specific endpoints targeting a specific query.

5.2.2.2.2 Message Broker: RabbitMQ

A message broker is put in place to route the sensor data and predictions (collectively known as "data") from the Bluetooth client and Ultra96 respectively to the server for processing and inserting the data into the database. In message broker terms, the client and Ultra96 publishes messages while the server consumes messages. There are primarily two reasons why a message broker is required.

Firstly, sending data directly to the server is not scalable as it puts unnecessary load on the server in terms of memory consumption. While this is not a significant issue in our use case since each wearable operates at 20 Hz, and with 3 wearables operating at any time, there will only be a maximum of 60 sensor data per second, the problem becomes more prominent when we increase the frequency of sensor readings and number of wearables. Data that has not been processed by the server will continue to reside in the server's memory, and if the memory is full, new sensor data will either be dropped (in the case of UDP) or re-sent (in the case of TCP, which adds additional overhead in the form of network IO giving rise to higher latency). Using a

message broker in this case will ensure that the data (messages) reside in the message broker's memory, and that the server only processes a fixed number messages at any one time.

Secondly, message brokers are durable. This means that even if the message broker restarts due to system crashes, the messages in the queue persist. By default if the server experiences system crashes, existing data that has been sent to the server will not be recovered once the server restarts since they reside in the main memory. While it is possible to configure the server such that received data is written to persistent memory and removed from the persistent memory once it has been processed, additional overhead is incurred in the form of disk IO. Message brokers thus provide us with a guarantee that data is not lost in the event of a system downtime.

The team has chosen RabbitMQ as the message broker service due to its ease of setting up and low latency via Advanced Message Queue Protocol (AMQP)l. The following diagram depicts how the message broker is configured in our system:

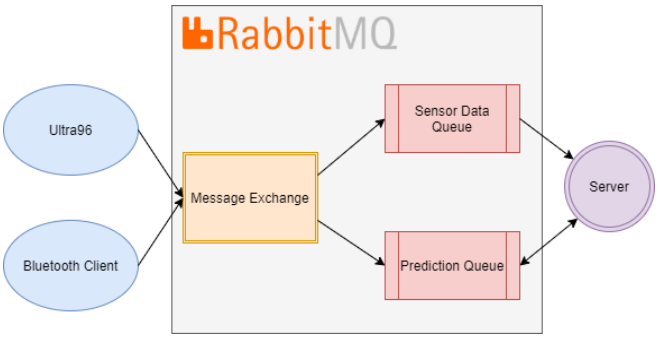


Figure 5.2.2.2.2a Message broker architecture

All messages sent by publishers are first routed to RabbitMQ's message exchange, before being routed to the specified queue based on each message's routing key. Sensor Data Queue stores all sensor data (sent from Bluetooth client) while Prediction Queue stores all prediction results (sent from Ultra96).

The double headed arrow between Prediction Queue and Server indicates that the server will send an acknowledgement back to RabbitMQ once the message has been processed successfully before RabbitMQ removes the message from the queue. This additional reliability is not required for the Sensor Data Queue since the throughput for sensor data is very much higher, and having such an additional reliability measure can add additional overhead that affects the real-timeness of data streaming on the dashboard. Furthermore, some data loss in sensor data is tolerable, whereas the same cannot be said for predictions. As such, messages in Sensor Data Queue are simply removed once the server has consumed the message.

5.2.2.4 Entity-Relationship Modelling

The following entity-relationship diagram depicts how different entities interact with each other in the final database design. The actual relational implementation can be found in the `dashboard-backend` source code under the directory `sql`.

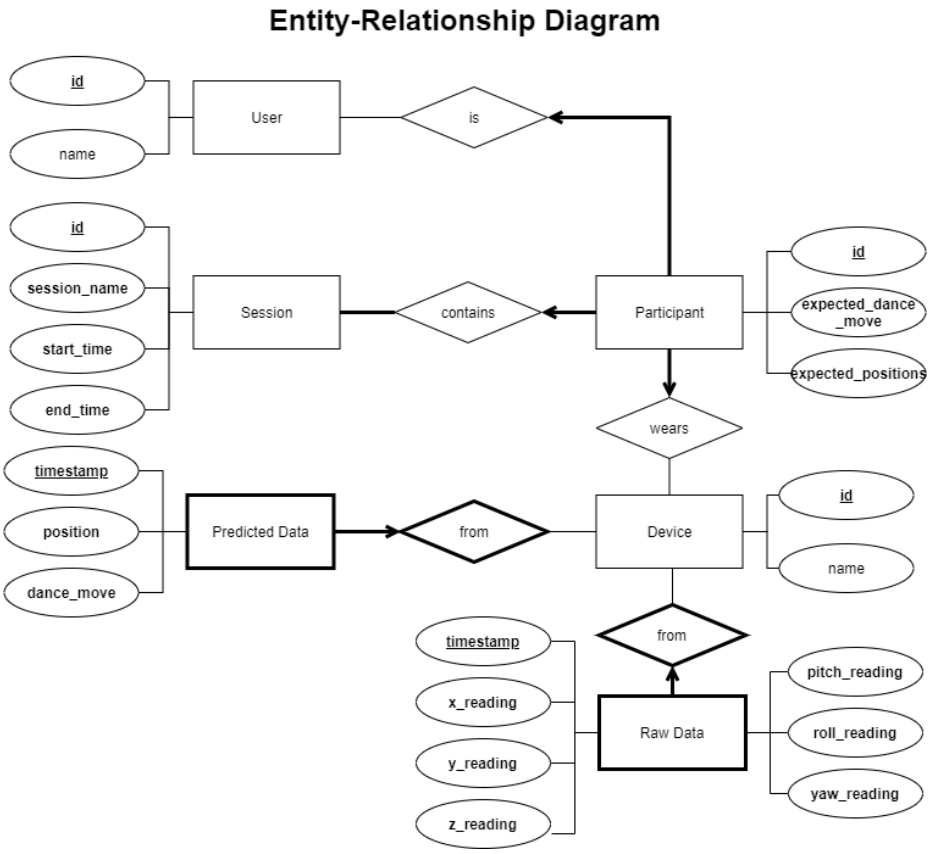


Figure 5.2.2.4a Entity-Relationship Diagram of the database

5.2.3 Software Deployment

All related services of the dashboard were deployed to the cloud so that all team members can have access to the dashboard for internal testing. The team has chosen Google Cloud Platform (GCP) as the cloud service provided due to the generous amount of free credits that GCP offers. The following page shows the overall deployment architecture of the dashboard. The grey arrows represent the deployment path while the blue arrows represent the data flow. In general, the deployment architecture is built with automation, scalability, speed and security in mind.

Continuous integration and deployment is embedded within the relevant repositories on GitHub using GitHub actions. Specifically, the backend repository automatically containerizes the source code and pushes the latest image to Google Container Registry, which is then deployed to the server instance. The frontend repository builds and pushes the latest release of the client straight to Google App Engine.

Most of the service instances have been configured with autoscaling to take advantage of the ease of scaling out with cloud computing. This allows the dashboard to function properly even under heavy load since more compute resources can be added automatically to handle the load. For instance, the Server Sensor instance, which deals with processing of sensor data, is configured to spin up a maximum of 5 instances to handle incoming sensor data messages from the message broker. Since RabbitMQ pushes messages to consumers in a round-robin manner, having more instances will mean that less time is required to process all existing messages in the message broker, achieving near real-time data streaming even as the number of wearables increase.

In addition, instances that are not exposed to the public internet communicate to each other through internal IP addresses since they reside in the same Virtual Private Cloud's subnet. Since all instances reside within the same region and the same subnet, with the exception of the dashboard's App Engine instance, latency due to network IO is kept at minimal. The reason as to why the dashboard's App Engine instance does not reside in the same region as the other instances is purely due to infrastructure limitation since App Engine is not available in the region Southeast-Asia-1. However, the performance degradation of that is unnoticeable considering that the dashboard polls at an interval of 1 second for streaming data.

In terms of security, all services that are exposed to the public internet are either served through a load balancer, a firewall or requires authentication. RabbitMQ instance for example only accepts AMQP ingress from NUS IP addresses (i.e. 137.132.0.0/16) in addition to encrypting individual messages. Other than requiring an admin secret to access the dashboard client, the client is also served through an App Engine's built-in load balancer to allow connection to the client through Transport Layer Security (TLS) traffic. Users who try to access the dashboard client (capstone-fifteen-staging.et.r.appspot.com) through HTTP will automatically be redirected to the HTTPS site.

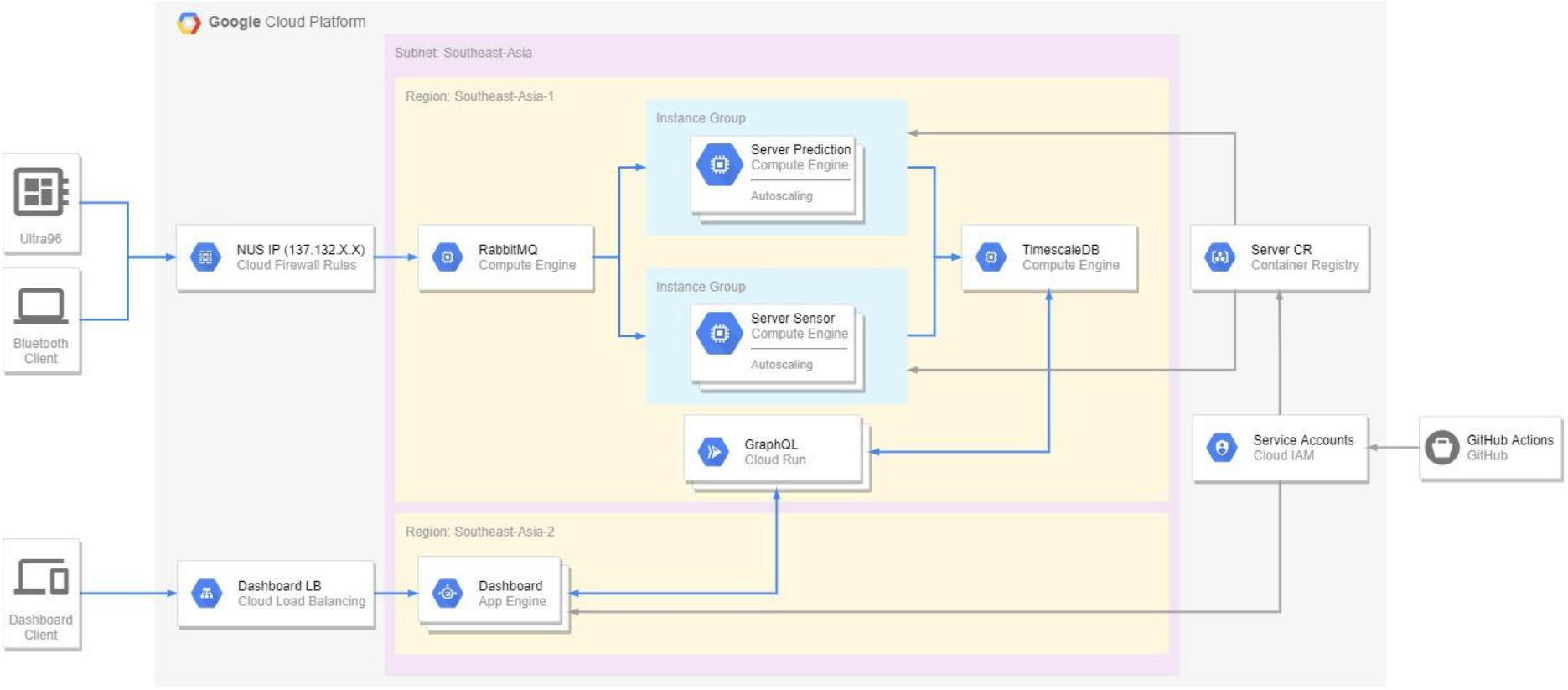


Figure 5.2.3a Overall deployment architecture on Google Cloud Platform

5.2.4 Software Implementation

This section covers all the implemented features on the dashboard. For the ease of following through the section, access the dashboard at https://capstone-fifteen-staging.et.r.appspot.com with the admin secret `Xjb84POExBxidM5gB3tdVrWHK1nqbx9o`. The Wearable component of the dashboard has been omitted as it is self-explanatory.

5.2.4.1 All Sessions

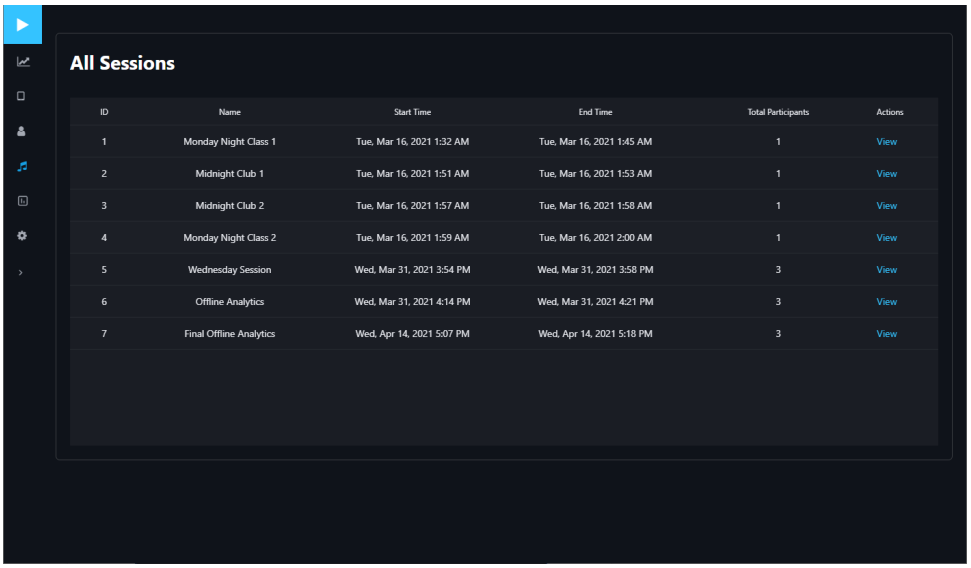


Figure 5.2.4.1a All Session accessed from Sessions > All Session on the left navigation bar

This screen displays all created dance sessions, the start and end time, and the number of participants for each session.

5.2.4.2 Session Analytics

Figure 5.2.4.2a Individual analytics in session analytics

Figure 5.2.4.2b Team analytics in session analytics

Continuing from All Sessions screen, selecting “view” on one of the sessions brings up the session analytics which displays the analytics for the team as a whole and each individual dancer. Each dancer’s dance move and position accuracy, sensor data, how the accuracies changed over time, incorrect dance moves and positions, fatigue level and sync delay over time are displayed.

5.2.4.3 New Session

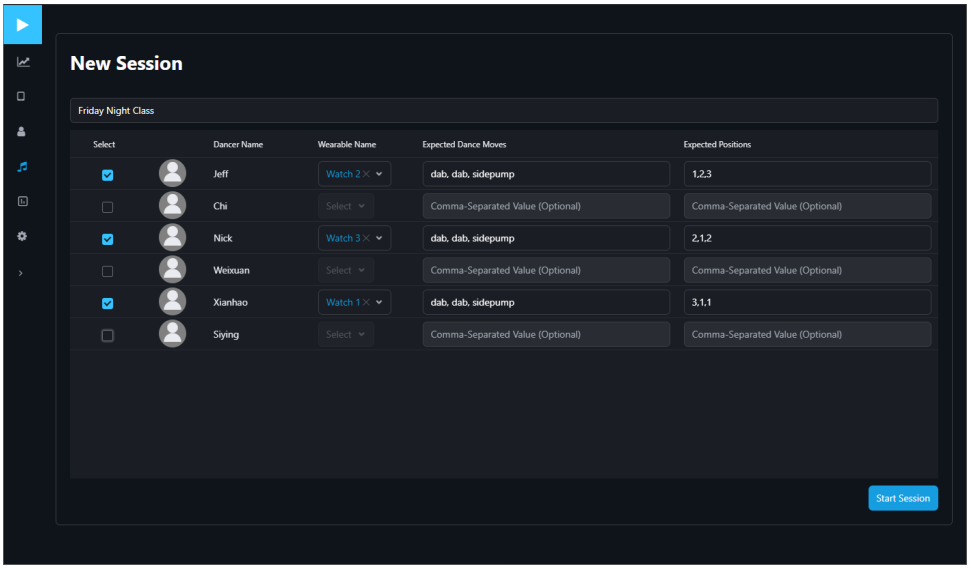


Figure 5.2.4.3a New Session accessed from Session > New Session on left navigation bar

New sessions can be created via this screen. Each session requires at least 1 dancer and at most 3 dancers. Expected dance moves and positions can be put in the respective text box as comma-separated values.

5.2.4.4 All Dancer

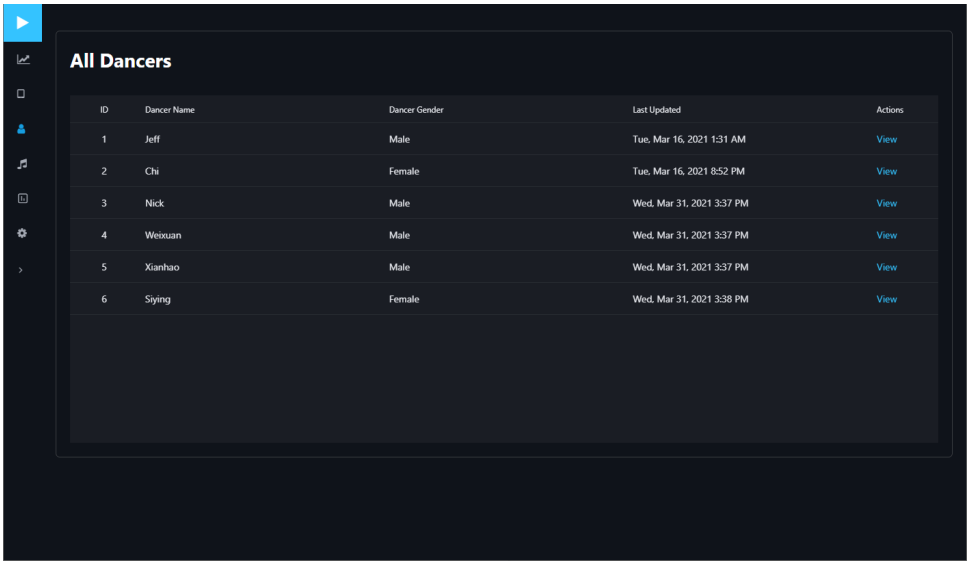


Figure 5.2.4.4a All Dancer accessed from Dancer > All Dancer on left navigation bar

This screen shows all dancers that have been onboarded onto the system.

5.2.4.5 Dancer Analytics

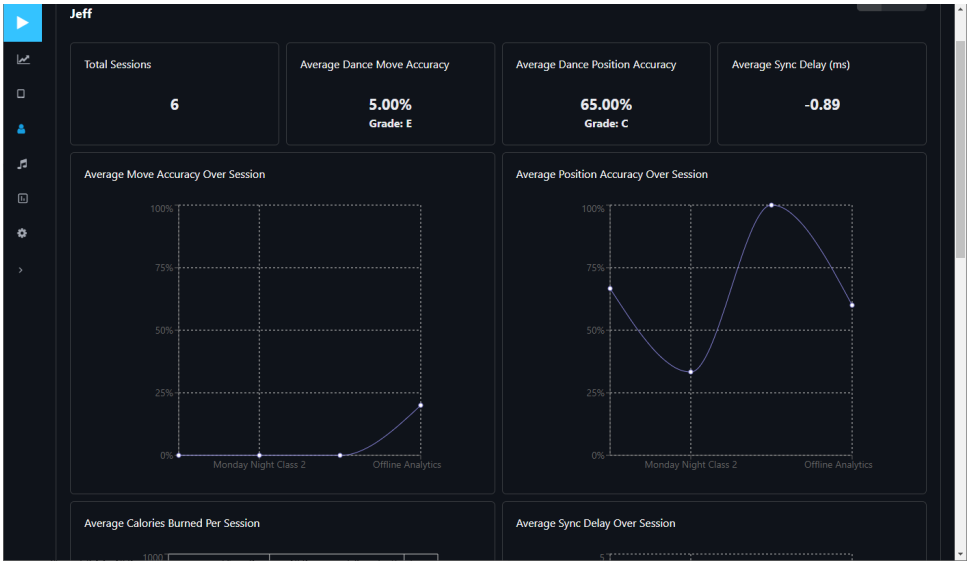


Figure 5.2.4.5a Dancer analytics screen

Continuing from All Dancer screen, selecting “view” on one of the registered dancers brings up the dancer analytics, which shows the aggregated dance move and position accuracy across all participated sessions, average calories burned for each session, and all participated sessions.

5.2.4.6 New Dancer

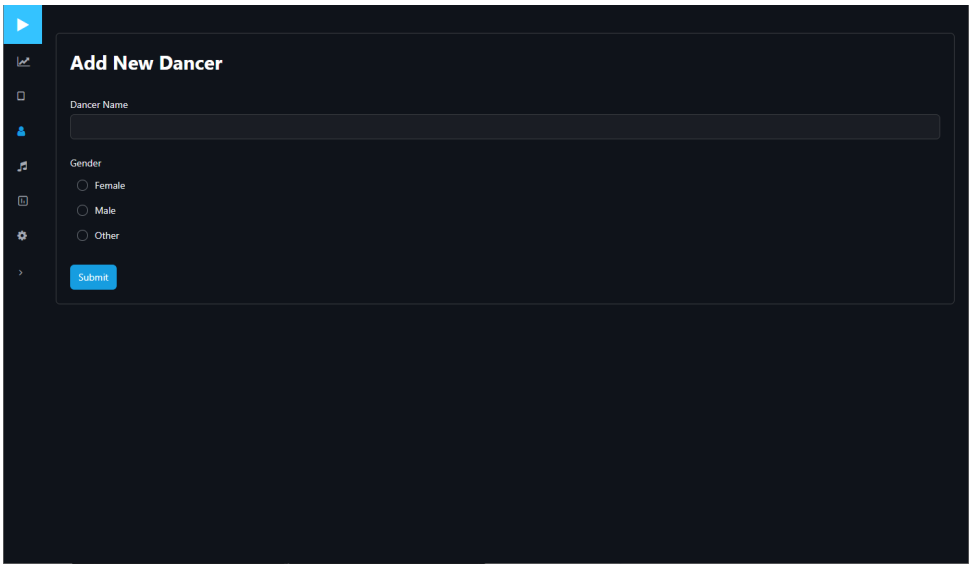


Figure 5.2.4.6a New Dancer accessed from Dancer > New Dancer on left navigation bar

New dancers can be registered into the system via this screen

5.2.4.7 Live Mode (also known as Quick Play mode)

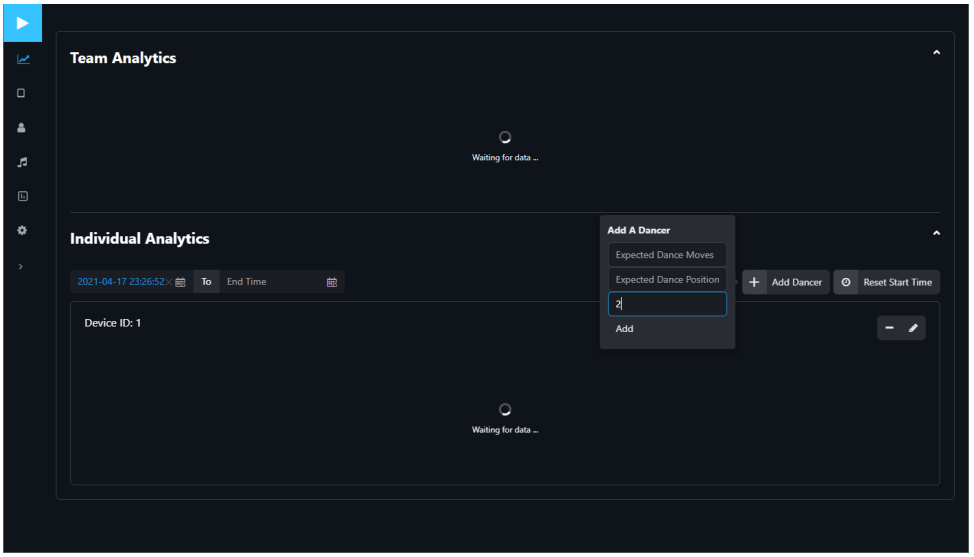


Figure 5.2.4.7a Live Mode accessed from the left navigation bar

This screen allows dancers to experience the system without registering. Only wearable ID is required to start this mode. Analytics generated are exactly the same as that in Session Analytics, but will not be saved for future viewing.

5.2.4.8 Data Collection

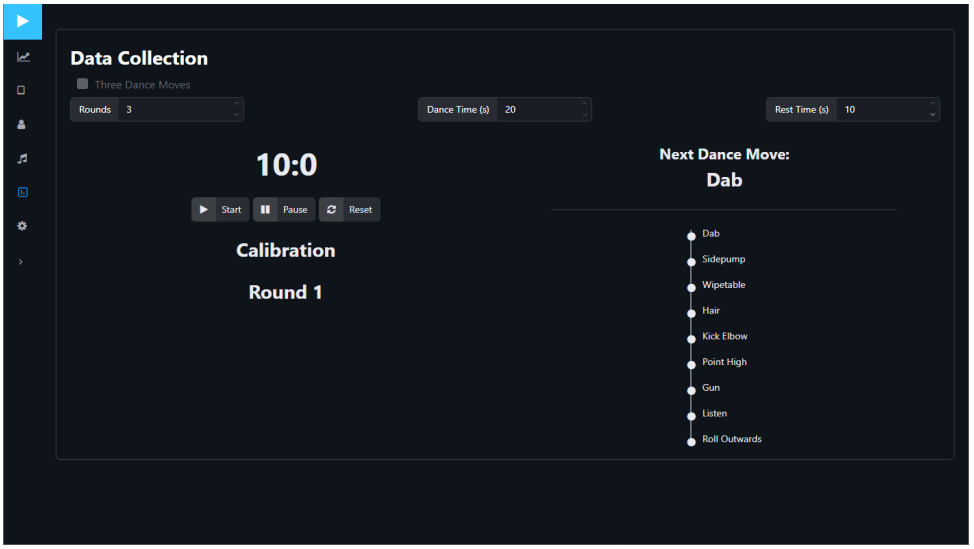


Figure 5.2.4.8a Data Collection accessed from Data Tool > Data Collection on left navigation bar

This screen is used internally by the team to collect and label dance data for training of machine learning models.

Section 6 Project Management Plan

6.1 Project Management Strategy

Agile development is the core part of the team's project management strategy. The team has adopted the Scrum framework with tweaks being made to support the project more efficiently.

Scrum is mainly based on timework iterations, otherwise known as sprints. Moreover, each sprint can be between two weeks and thirty days long. Because of that, teams can easily track progress and re-evaluate plans in fifteen-minute daily/weekly meetings also known as daily/weekly Scrums.

Given the timeline and scope of the project, the duration of each sprint is a single week. This allows the team to move even faster by checkpointing what has been completed and planning what is to be completed every week during sprint planning.

In addition, Scrum master rotates among team members. A different team member will assume the Scrum master role each week/day and perform the tasks of a Scrum master such as sprint planning and organizing standups. The rationale behind making such a tweak to the traditional Scrum framework where there is only one fixed Scrum master for the entire project is so that each team member will have a better understanding of the overall progress of the project in addition to the progress of each team member.

To support the project management strategy that the team has in place, the team will employ Jira as the main project management tool for issue tracking and GitHub as the remote version control system for codebase management.

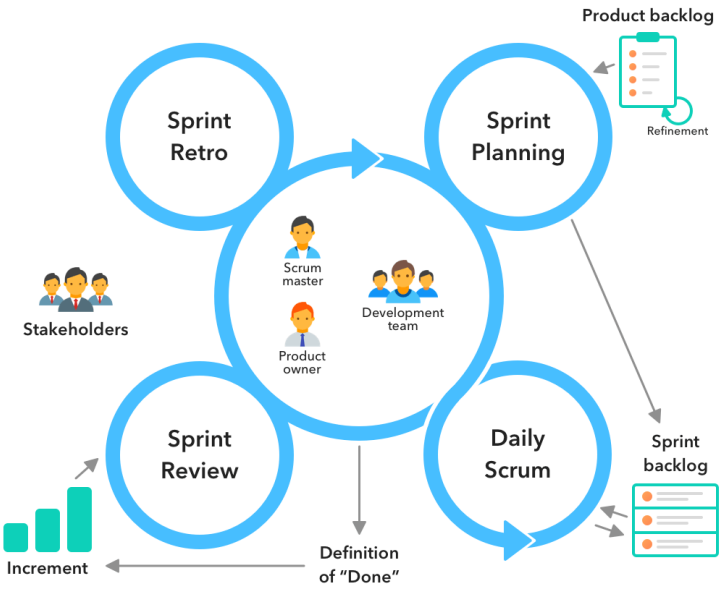


Figure 6.1.a: Agile Software Development (this is daily scrum example)

6.2 Project Timeline

The project timeline below is extracted directly from our JIRA board using the project roadmap section. To view the project timeline in greater granularity, access the shared JIRA product roadmap with the following link: https://capstone-fifteen.atlassian.net/jira/software/c/projects/TK/boards/1/roadmap?shared=&atlOrigin=eyJpIjoiODlmNjIxYTM5MzZjNGNmOGI2MGUyZThlN2VjOWU0MjkiLCJwIjoiaiJ9

Figure 6.2a Project Timeline on JIRA Board for Week 3 - 7

Figure 6.2b Project Timeline on JIRA Board for Week 8 - 13

Section 7 Societal and Ethical Impact

7.1 Societal Impact

7.1.1 Impact on Dance Industry

The use of motion sensing input devices in dance is nothing new. From more commercially affordable systems such as Xbox 360 Kinect developed back in 2010 to more expensive wearables such as motion capture suits used in bigger budget projects, applications of sensors to identify and classify movement has become a well established and developed application. Dance games released on the Kinect are able to reliably score dances based on various metrics and guide their users.

In such a well established area for motion sensing devices, what sets our design apart is in its simplicity and accessibility. From the use of affordable sensors to the low power BLE protocol, system cost remains low. Coupled with software side connection support to an external server and a machine learning algorithm that can be upscaled according to a manufacturer's desires, given more time to develop on additional wearable components and software refinement will allow the system to function as a reliable dance detector with low barriers to entry and ownership and manufacturing.

7.1.2 Impact on Fitness and Gaming Industry

A natural extension to dance are exercise routines. With the pandemic ongoing, a new form of fitness routine has become increasingly popular - fitness gaming. Essentially, fitness gaming involves doing fitness routines through video game software and peripheral. Ring Fit Adventure, a fitness role-playing game by Nintendo became the top selling video game for the year 2020 with various stores running out of stock in matters of days.

The technology that most of these fitness gaming use is similar to what our wearable offers - an accelerometer and a gyroscope. In the case of Ring Fit Adventure, the player "wears" the game controller containing the gyroscope and accelerometer on his or her thigh for lower body movement detection and uses a pilates ring containing the same game controller to detect upper body movement. Players then perform a series of exercise routines as directed by the game.

What makes our technology different is the incorporation of machine learning technologies to identify dance moves. We can apply the same technology to identify exercises as well as how well the player performs each exercise. This gives greater flexibility to the players as they are no longer restricted to performing the specific exercise determined by the software. Rather, players are free to perform a range of exercises, and the machine learning technology will be able to identify the exercises performed as well as how well the player performed those exercises. By giving players such flexibility, fitness gaming will be more natural. Using a single device to detect

such movement will also be more comfortable for the players performing intensive exercises where wearing multiple video game peripherals might be clunky and interferes with player's movement. By providing a more natural and comfortable experience to players, our device can make fitness gaming more accessible to the masses, doing away with expensive gym memberships and possibly revolutionizing the fitness industry.

7.1.3. Impact on Sports

With many sports, it's difficult for the human eye to observe all the movements an athlete might make during the course of an activity, but it is possible to record even unobservable data with sensors. And by using machine learning (ML) on this data, the athlete and coach can learn and improve based on precise measurements and analytics. The instrumented athlete is becoming the new competitive advantage.

If the current trend continues, in a few years most sports equipment sold in stores will have a smart sensor embedded. Electronics are becoming smaller, lighter and more flexible, and it is likely humans will see them embedded in fabrics, shoes, skis, tennis racquets and other types of smart gear.

To make adopting this technology easier, Microsoft have created an open source Sensor Kit, with components to process, measure, analyze and improve sensor measurements of athletic performance. Over time, Microsoft aim is to evolve this community Sensor Kit together with electronics and sports equipment companies, and sports associations and enthusiasts. The Sensor Kit is available at bit.ly/2CmQhzq.

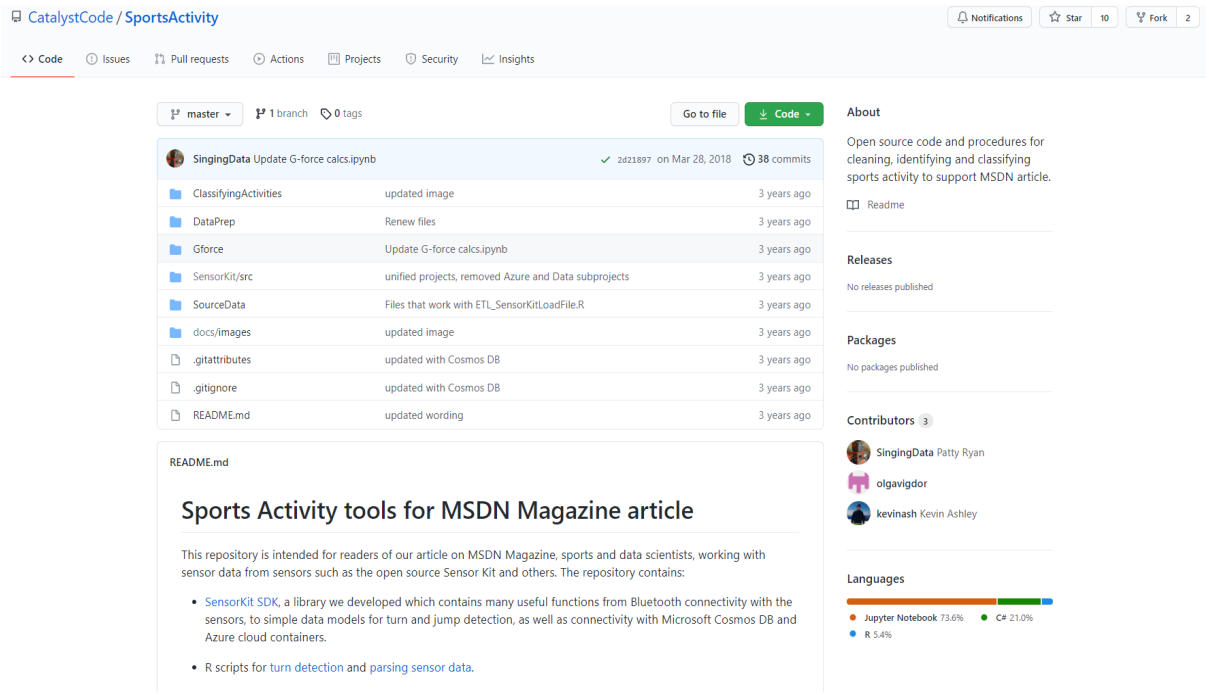


Figure 7.1.3.1a: Open source Github page for Sensor kit

Going further into the link above, the article explains how data can be collected from a sensor kit. Sensors will be attached to athletes to gather data with data analytics that will determine the movements of athletes and G-force or stress that an athlete can take. They used this kit on a skier to classify movement and to calculate the force that can be stressed on a person's body.

The goal of this article is to classify movement by skiers. This data can be used to develop a system that can teach beginners the right movements or help advance skiers fine-tune their craft. This system can be inducted to take in different sports as well. Therefore, with such a system, people will be to create commercial products to teach humans sports in a way that mimics high-performance athletes.

7.1.4 Impact on Other Industries

Developments in one industry often propagates to other related industries. Looking at the context of our project, there is a possibility of our design to affect the industries producing motion sensing devices. An example would be the use of full body mocap suits for motion sensing. If provided with the time for a full product development cycle, we can see our design expanding beyond the wristwatch to further wearables that sense motion in other areas such as the hips, legs and/or head. A full system that would be able to track full body movement would not only improve dance tracking, but also serve as a cheaper alternative to the more expensive, full body mocap suits.

Given the small scale of the project, proper insight to impacts on other industries cannot be accurately determined. But assuming our design is further developed and placed into proper manufacturing, the possibility of impacts on other industries will not be too far fetched.

7.1.4.1 Gesture control

It is well-known that the Covid-19 pandemic has placed a significant strain on economies and financial burden on people. As such, it was paramount that the spread of Covid-19 was mitigated to relieve the strain on the economy and people. To do so, governments such as Singapore have implemented a slew of measures such as the wearing of face masks. With our wearable, people are able to control devices without contact. When there is less contact between people, the risk of Covid-19 transmission will be reduced, hence there will be less Covid-19 patients. Therefore, Singapore can recover from the crisis earlier and resume normal life once, allowing the economy to improve once again. And most importantly, there will be less lives lost to Covid-19.

Figure 7.1.4.1a: Gesture control

For example, this project can be easily extended to gesture control by implanting the chip into human hands. This is very feasible given neuralink success of implanting a chip inside the brain. Our model and system will be much simpler, require less power consumption and be less dangerous compared to putting a chip inside the brain. Gesture control can then be connected to any device with bluetooth, which allows us to virtually control all electronic devices with gestures. The chip also can serve as a unique digital identity to grant access and make monetary transactions just like how we microchip cat & dogs. It can also act as the next generation of TraceTogether by GovTech.

Figure 7.1.4.1.b: Monkey playing pong using neuralink

7.1.4.2 Early intervention in emergencies

In Singapore, 71.6% of the population die from not getting treatment on time. Activity detection can also be used for early intervention in the case of emergencies. By recognising activities associated with trauma, falling down, the wearable can send out a SOS signal to nearby devices.

For example, if a poor elderly man would fall down wearing our device, our device can recognise that the elderly has fallen down. This would be extremely easy with the sensors like EMG which we already have that can detect someone's heartbeat or breath to see if the person went into shock. Next, through Singapore and Govtech smart nation initiative, this incident can be quickly verified using nearby surveillance cameras using computer vision. Once such an event is confirmed, the ambulance or emergency response team can then be notified automatically. With this our wearable can save countless lives too.

7.2 Ethical Considerations

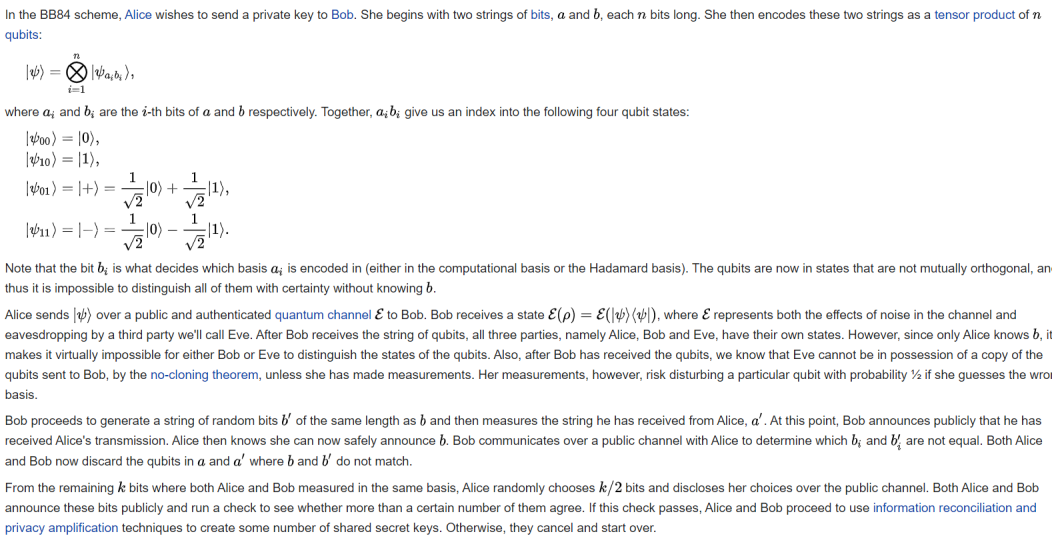
7.2.1 Inclusivity

Currently, our dance system is designed with and tested against the able-bodied user. People without hands cannot wear our wearables on their hands. This is not an inclusive system, as we have not tested on persons with disabilities. To fight against ableism, it is paramount to widen our target and test users.

7.2.2 Privacy and Security

As discussed in section 5.2.3, all services that are exposed to the public internet require access through encrypted traffic. The dashboard for instance requires users to access the client through TLS traffic, ensuring that that the communication between the dashboard client and server is encrypted and thereby preventing packet sniffers from reading the data that has been sent or fetched by the user, which can be sensitive information such as user login details. TLS also prevents man-in-the-middle attacks where an eavesdropper intercepts the communication between the client and the server posing as either party, ensuring that the user does not send data to an adversary. The only security concern that the current system has is the publishing of messages to RabbitMQ message broker as the current implementation, although only allows incoming traffic from NUS IP, does not have any form of TLS requirement in place. Encryption is done on the application layer, which means that data can be sent in plain text that can be deciphered by packet sniffers. This is not ideal in production environments and can be circumvented by enabling TLS support on RabbitMQ.

As always we should always assume Kerckhoff's principle: *"A system should be secure even if everything about the system, except the secret key, is public knowledge."* Current TLS handshake heavily relies on the fact that underlying theorems behind RSA and DHE are hard problems. This may not necessarily be true, especially when Peter Shor came up with a fast polynomial-time quantum computer algorithm for integer factorization. Although current technology has yet caught up to reliably run thousands of qubit quantum computers, it is not impossible in the near future. As such, we have to also evolve and adapt to these changes. In the event where quantum computers are the norm, we will not be able to rely on the current standard of TLS and might need to shift to quantum key distribution protocols such as the famous BB84 and BBM92 protocols.



In the BB84 scheme, Alice wishes to send a private key to Bob. She begins with two strings of bits, a and b, each n bits long. She then encodes these two strings as a tensor product of n qubits:

$$|\psi\rangle = \bigotimes_{i=1}^{n} |\psi_{a_i b_i}\rangle,$$

where a_i and b_i are the i-th bits of a and b respectively. Together, $a_i b_i$ give us an index into the following four qubit states:

$$|\psi_{00}\rangle = |0\rangle,$$
$$|\psi_{10}\rangle = |1\rangle,$$
$$|\psi_{01}\rangle = |+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle,$$
$$|\psi_{11}\rangle = |-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle.$$

Note that the bit b_i is what decides which basis a_i is encoded in (either in the computational basis or the Hadamard basis). The qubits are now in states that are not mutually orthogonal, an thus it is impossible to distinguish all of them with certainty without knowing b.

Alice sends $|\psi\rangle$ over a public and authenticated quantum channel $\mathcal{E}$ to Bob. Bob receives a state $\mathcal{E}(\rho) = \mathcal{E}(|\psi\rangle\langle\psi|)$, where $\mathcal{E}$ represents both the effects of noise in the channel and eavesdropping by a third party we'll call Eve. After Bob receives the string of qubits, all three parties, namely Alice, Bob and Eve, have their own states. However, since only Alice knows b, it makes it virtually impossible for either Bob or Eve to distinguish the states of the qubits. Also, after Bob has received the qubits, we know that Eve cannot be in possession of a copy of the qubits sent to Bob, by the no-cloning theorem, unless she has made measurements. Her measurements, however, risk disturbing a particular qubit with probability ½ if she guesses the wro basis.

Bob proceeds to generate a string of random bits b' of the same length as b and then measures the string he has received from Alice, a'. At this point, Bob announces publicly that he has received Alice's transmission. Alice then knows she can now safely announce b. Bob communicates over a public channel with Alice to determine which b_i and b'_i are not equal. Both Alice and Bob now discard the qubits in a and a' where b and b' do not match.

From the remaining k bits where both Alice and Bob measured in the same basis, Alice randomly chooses $k/2$ bits and discloses her choices over the public channel. Both Alice and Bob announce these bits publicly and run a check to see whether more than a certain number of them agree. If this check passes, Alice and Bob proceed to use information reconciliation and privacy amplification techniques to create some number of shared secret keys. Otherwise, they cancel and start over.

Figure 7.2.2a: BB84 protocol

In regards to privacy, the system is also capable of being PDPA and GDPR compliant. All user data is stored in and encrypted by our cloud service provider which has access to state of the art encryption technologies and provides round-the-clock monitoring for any breaches. Data that can identify an individual is never collected in our system, and data collected are anonymized as much as possible. For instance, sensor data stored in the database are not directly linked to individual users. As such, our machine learning engineers who require access to such data to train models will not be able to trace the sensor data to each individual. Data sent over the internet are encrypted and anonymized, preventing any adversary who successfully intercepted the communication from identifying any individuals with the data. Sensor data sent from bluetooth clients to RabbitMQ are only tagged to wearable identifiers and not to individuals, so even if packet sniffers manage to decipher the encrypted data, they are not able to pinpoint the owner of that data. Users who do not wish to provide any personal data are still able to use the system through the Quick Play feature (refer to section 5.2.4.7) without compromising on analytics provided by the dashboard. It is also possible for users to request for the absolute removal of their data from our database, which can be done with our internal SQL script.

While our project does not deal with complex privacy or security concerns, the underlying principles that we adopt when making decisions related to user privacy and security do not change even when the environment becomes more complex. That is, we believe that privacy is a human right, and we strive to guarantee that this right of our users is not compromised when using our product.

This comes in the form of greater transparency - making it easy for users to understand what is collected and how their data are processed; alerting them of any breaches and steps taken to remedy the situation, and providing sound advice on the next course of actions required from the users; explaining their rights to their data and giving them the deciding power over when and what data can be collected.

Fundamentally, we do not believe that there is necessarily a trade-off between privacy and technological advancement. If anything, the latter builds on top of the former. As Tim Cook, CEO of Apple, expressed this sentiment eloquently in his speech during the 2021 Computers, Privacy and Data Protection Conference:
"*We believe that ethical technology is technology that works for you. It's technology that helps you sleep, not keeps you up … It's technology that can fade into the background when you're on a hike or going for a swim, but is there to warn you when your heart rate spikes or help you when you've had a nasty fall. And that all of this, always, puts privacy and security first, because no one needs to trade away the rights of their users to deliver a great product.*"

7.2.3 Health and Wellness

The current version of the wearable has not been waterproofed. So it is not advisable to use it for extremely intense exercise. It may cause a short circuit which may harm the user. According to health authorities, 10 milliamperes is dangerous enough.

Health authorities says: "
While any amount of current over 10 milliamperes (0.01 amp) is capable of producing painful to severe shock, currents between 100 and 200 milliamperes (0.1 to 0.2 amp) are lethal.
"

Further improvements to the wearable to make it more durable and resistant have to be made to mitigate this risk. To achieve such improvements, we plan to encase the electronic components fully in insulating material such as resin. Given that R = V / I for an Ohmic conductor, we can greatly reduce the current going through the human heart by introducing additional insulation layers.

It is also known that bluetooth and other high frequency radio waves like Wi-Fi can kill sperms. However, compared to devices such as Apple watches and laptops, the waves emitted by our wearable is significantly lesser and poses less of a risk. Moreover technology today can allow people to use IVF and sperm dying is not really a huge issue in this context.

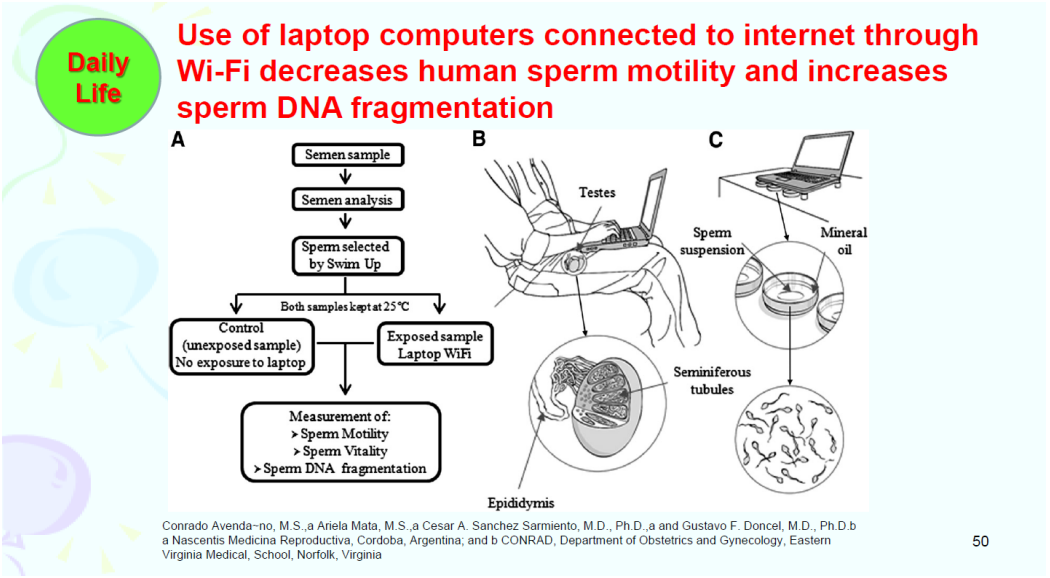


Figure 7.2.3a: How Wi-Fi kill sperms

However, if we are going forward with an implementation of chipping humans, the risks will naturally be larger. Apart from killing sperms, it is still unclear exactly what the long term impact of exposing the body to radio waves. This is due to the fact that living organisms today are extremely complex and a small change can lead to a drastic change in later states, which is what we know as the Butterfly effect. This is especially the case when it can possibly interfere with the DNA replication process or the DNA structure itself.

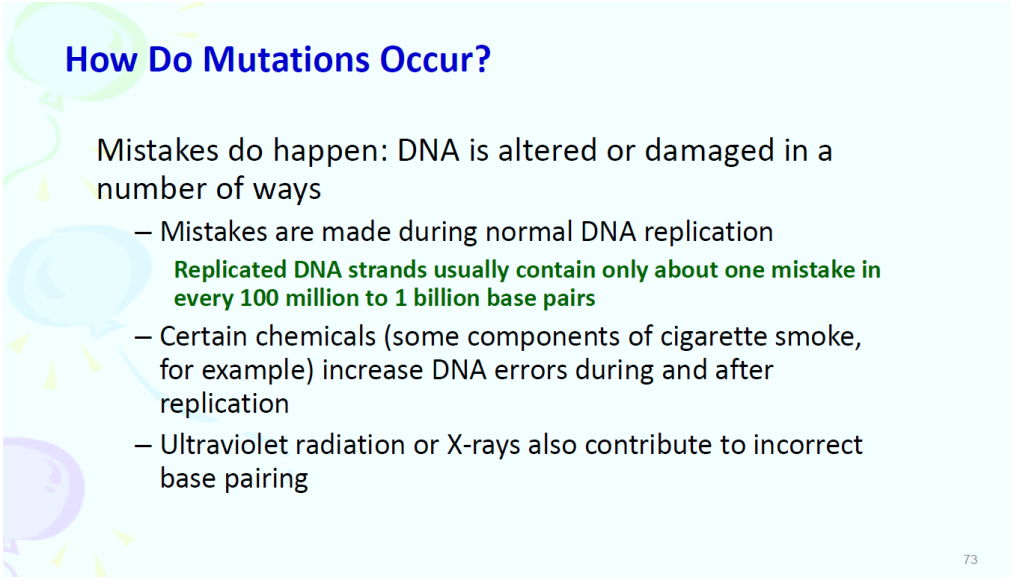


Figure 7.2.3b: Mutation

When DNA replication or structure is affected, mutations can occur. Unlike X-Men movies, most mutations are bad and leads to what we commonly referred to as cancer. With the DNA structure

changed, it may lead to the incorrect RNA being created during transcription, possibly resulting in wrong polypeptide chains to be created during translation, and wrong proteins to be formed, or possibly worse, prions to be formed. Prions are misfolded infectious proteins that induce other proteins to misfold. Prions often are responsible for most neurodegenerative diseases such as dementia. And hence, it is easy to see why Bluetooth can possibly be dangerous. This is also the basis behind all the health articles we see about using electronic products. However, such risks are known to be minimal and nobody really cares as there are much more lethal things in our everyday life such as the GM crops and artificial food we are eating, and of course, the Coronavirus.

7.2.4 Employment

With the drastic growth of AI and IoT in general, some claim that AI can replace humans someday. And this statement is no longer impossible especially when we are in the 4.0 technology industrial era, where there is a trend of automation and data exchange in manufacturing technologies, including cyber-physical systems, the Internet of things, cloud computing and cognitive computing and creating the smart factory. If all softwares and factories are created “smart” enough, will we need human workers anmore? If you are a factory owner, will you prefer robot workers that can work non-stop or human workers with higher pay? There will always be a dilemma between profit and ethics to some extent. And this issue has been prolonged over centuries.

7.3 Looking Forward

The engineering process, successful or not, serves to inspire. While our efforts in this project went into building a prototype that will not see any manufacturing or distribution, the experience itself of the system development remains the greatest takeaway.

The journey of the engineering process, where we fail at times, and yet come back to try again and again until the system succeeds; where integration efforts do not come to fruition, yet we preserve, trying new methods and debugging repeatedly, to produce a final working system that can express the fruits of our efforts, is an inspiring journey in creation that we will takeaway into any projects in the future.

References

A. Maharaj, “A Review on Advanced Encryption Standards (AES),” *ResearchGate*, 28-Jan-2020. [Online]. Available: https://www.researchgate.net/publication/338853730_A_Review_on_Advanced_Encryption_Standards_AES. [Accessed: 07-Feb-2021].

“Arch Linux,” *Xilinx Vivado Vitis 2020.1 Installation / Applications & Desktop Environments / Arch Linux Forums*. [Online]. Available: https://bbs.archlinux.org/viewtopic.php?id=256564. [Accessed: 07-Feb-2021].

Avnet, “Avnet/bdf,” *GitHub*. [Online]. Available: https://github.com/Avnet/bdf. [Accessed: 07-Feb-2021].

“Benchmarking Kafka vs. Pulsar vs. RabbitMQ: Which is Fastest?,” *Confluent*. [Online]. Available: https://www.confluent.io/blog/kafka-fastest-messaging-system/. [Accessed: 18-Apr-2021].

“Circular Buffer,” *Circular Buffer - Arduino Reference*. [Online]. Available: https://www.arduino.cc/reference/en/libraries/circularbuffer/. [Accessed: 07-Feb-2021].

“ClickHouse Crushing Time Series – ClickHouse Software And Services,” *Altinity*, 30-Jul-2020. [Online]. Available: https://altinity.com/blog/clickhouse-for-time-series. [Accessed: 07-Feb-2021].

“CPU Scheduling,” *Operating Systems: CPU Scheduling*. [Online]. Available: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html. [Accessed: 07-Feb-2021].

Craigloewen-Msft, “Install Windows Subsystem for Linux (WSL) on Windows 10,” *Install Windows Subsystem for Linux (WSL) on Windows 10 | Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/windows/wsl/install-win10. [Accessed: 07-Feb-2021].

“CS60 Computer Networks,” *Department of Computer Science*, 02-Apr-2012. [Online]. Available: https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html. [Accessed: 07-Feb-2021].

D. Badurina, “GraphQL over WebSockets,” *The Guild Blog*. [Online]. Available: https://the-guild.dev/blog/graphql-over-websockets. [Accessed: 07-Feb-2021].

D. Luecke, “HTTP vs Websockets: A performance comparison,” *Medium*, 27-Jan-2018. [Online]. Available: https://blog.feathersjs.com/http-vs-websockets-a-performance-comparison-da2533f13a77. [Accessed: 07-Feb-2021].

"DFRobot Beetle BLE - The Smallest Board Based on Arduino Uno with Bluetooth 4.0," *DFRobot*. [Online]. Available: https://www.dfrobot.com/product-1259.html. [Accessed: 07-Feb-2021].

E. Strauss, "Fitness video games can break your Covid pandemic exercise slump," *CNN*, 18-Dec-2020. [Online]. Available: https://edition.cnn.com/2020/12/17/health/fitness-video-games-covid-pandemic-wellness/index.html. [Accessed: 18-Apr-2021].

F. van Veen, "The Neural Network Zoo," *The Asimov Institute*, 27-Apr-2019. [Online]. Available: https://www.asimovinstitute.org/neural-network-zoo/. [Accessed: 07-Feb-2021].

"Getting Started," *Getting Started - FINN documentation*. [Online]. Available: https://finn.readthedocs.io/en/latest/getting_started.html#how-to-use-the-finn-compiler. [Accessed: 07-Feb-2021].

H. C. A. Tilborg and Sushil Jajodia, "Encyclopedia of Cryptography and Security," *SpringerLink*. [Online]. Available: https://link.springer.com/referencework/10.1007/978-1-4419-5906-5. [Accessed: 07-Feb-2021].

H. Marius, "Multiclass Classification with Support Vector Machines (SVM), Kernel Trick & Kernel Functions," *Medium*, 09-Sep-2020. [Online]. Available: https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functions-f9d5377d6f02. [Accessed: 07-Feb-2021].

"How Does It Work?," *NTP*. [Online]. Available: http://www.ntp.org/ntpfaq/NTP-s-algo.htm. [Accessed: 07-Feb-2021].

"Installation," *Acceleration Installation*. [Online]. Available: https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/acceleration_installation.html#vhc1571429852245. [Accessed: 07-Feb-2021].

J. Brownlee, "1D Convolutional Neural Network Models for Human Activity Recognition," *Machine Learning Mastery*, 27-Aug-2020. [Online]. Available: https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/. [Accessed: 07-Feb-2021].

J. Brownlee, "Deep Learning Models for Human Activity Recognition," *Machine Learning Mastery*, 05-Aug-2019. [Online]. Available: https://machinelearningmastery.com/deep-learning-models-for-human-activity-recognition/. [Accessed: 07-Feb-2021].

J. Brownlee, "LSTMs for Human Activity Recognition Time Series Classification," *Machine Learning Mastery*, 27-Aug-2020. [Online]. Available:

https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/. [Accessed: 07-Feb-2021].

J. Fruhlinger, “What is SSL, TLS? And how this encryption protocol works,” *CSO Online*, 04-Dec-2018. [Online]. Available: https://www.csoonline.com/article/3246212/what-is-ssl-tls-and-how-this-encryption-protocol-works.html. [Accessed: 18-Apr-2021].

J. Yates, “The complete ‘WSL2 + GUI’ setup,” *Medium*, 01-Aug-2020. [Online]. Available: https://medium.com/@japheth.yates/the-complete-wsl2-gui-setup-2582828f4577. [Accessed: 07-Feb-2021].

Jrowberg, “jrowberg/i2cdevlib,” *GitHub*. [Online]. Available: https://github.com/jrowberg/i2cdevlib/tree/master/Arduino. [Accessed: 07-Feb-2021].

“Loading an Overlay,” *Loading an Overlay - Python productivity for Zynq (Pynq)*. [Online]. Available: https://pynq.readthedocs.io/en/v2.6.1/pynq_overlays/loading_an_overlay.html. [Accessed: 07-Feb-2021].

M. Bellare, A. Desai, E. Jokipii, and P. Rogway, “A Concrete Security Treatment of Symmetric Encryption,” *UC Davis*. [Online]. Available: https://www.cs.ucdavis.edu/~rogaway/papers/sym-enc.pdf . [Accessed: 07-Feb-2021].

M. Kasana, “Tim Cook goes after Facebook in data privacy speech,” *Input*, 29-Jan-2021. [Online]. Available: https://www.inputmag.com/culture/tim-cook-goes-after-facebook-in-data-privacy-speech. [Accessed: 18-Apr-2021].

M. Wodding, “New Proofs for Old Modes,” *Cryptology ePrint Archive: Report 2008/121*. [Online]. Available: https://eprint.iacr.org/2008/121. [Accessed: 07-Feb-2021].

Microsoft, "[Machine Learning] Sensors in Sports: Analyzing Human Movement with AI," 04 2018. [Online]. Available: https://docs.microsoft.com/en-us/archive/msdn-magazine/2018/april/machine-learning-sensors-in-sports-analyzing-human-movement-with-ai. [Accessed 18 04 2021]

“Multithreaded Priority Queue in Python,” *GeeksforGeeks*, 31-Dec-2020. [Online]. Available: https://www.geeksforgeeks.org/multithreaded-priority-queue-in-python/. [Accessed: 07-Feb-2021].

“Multithreading in Python: Set 2 (Synchronization),” *GeeksforGeeks*, 28-Aug-2019. [Online]. Available: https://www.geeksforgeeks.org/multithreading-in-python-set-2-synchronization/. [Accessed: 07-Feb-2021].

"Network Time Protocol," *Network Time Protocol - an overview | ScienceDirect Topics*. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/network-time-protocol. [Accessed: 07-Feb-2021].

NCBI, "Is the Use of a Low-Costs EMG Sensor Valid to Measure Muscle Fatigue?" *NCBI*, 20-Jul-2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6679263/ [Accessed: 14-Mar-2021].

O. Romanyuk, "Angular vs React: Which One to Choose for Your App," *freeCodeCamp.org*, 19-Mar-2020. [Online]. Available: https://www.freecodecamp.org/news/angular-vs-react-what-to-choose-for-your-app-2/. [Accessed: 07-Feb-2021].

P. Martinsen, "Exponential filter for smoothing noisy data: Reference," *MegunoLink*, 25-Sep-2020. [Online]. Available: https://www.megunolink.com/documentation/arduino-libraries/exponential-filter/. [Accessed: 07-Feb-2021].

P. Martinsen, "Three Methods to Filter Noisy Arduino Measurements: Coding," *MegunoLink*, 22-Sep-2020. [Online]. Available: https://www.megunolink.com/articles/coding/3-methods-filter-noisy-arduino-measurements/. [Accessed: 07-Feb-2021].

"pycryptodome," *PyPI*. [Online]. Available: https://pypi.org/project/pycryptodome/. [Accessed: 07-Feb-2021].

"pytunneling," *PyPI*. [Online]. Available: https://pypi.org/project/pytunneling/. [Accessed: 07-Feb-2021].

R. Kiefer, "MongoDB Time-Series - A NoSQL vs. SQL Database Comparison," *Timescale Blog*, 20-Dec-2020. [Online]. Available: https://blog.timescale.com/blog/how-to-store-time-series-data-mongodb-vs-timescaledb-postgresql-a73939734016/. [Accessed: 07-Feb-2021].

R. Rhouma, "Cryptanalysis of a spatiotemporal chaotic image/video cryptosystem," *ResearchGate*, Feb-2008. [Online]. Available: https://www.researchgate.net/publication/215783767_Cryptanalysis_of_a_spatiotemporal_chaotic_imagevideo_cryptosystem. [Accessed: 07-Feb-2021].

"React vs Vue: Is Vue.js going to take over React in 2020?," *Medium*, 28-Jul-2020. [Online]. Available: https://medium.com/swlh/is-vue-js-going-to-take-over-react-in-2020-929c19806ac. [Accessed: 07-Feb-2021].

Real Python, “Socket Programming in Python (Guide),” *Real Python*, 12-Dec-2020. [Online]. Available: https://realpython.com/python-sockets/. [Accessed: 07-Feb-2021].

“Socket Programming HOWTO,” *Socket Programming HOWTO - Python 3.9.1 documentation*. [Online]. Available: https://docs.python.org/3/howto/sockets.html. [Accessed: 07-Feb-2021].

“An SSH tunnel via multiple hops,” *Super User*, 01-Jan-1959. [Online]. Available: https://superuser.com/questions/96489/an-ssh-tunnel-via-multiple-hops. [Accessed: 07-Feb-2021].

“Support Vector Machines,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html. [Accessed: 07-Feb-2021].

T. A. Gamage, “HTTP and Websockets: Understanding the capabilities of today's web communication technologies,” *Medium*, 05-Jan-2021. [Online]. Available: https://medium.com/platform-engineer/web-api-design-35df8167460. [Accessed: 07-Feb-2021].

“Ultra96-V2 Vivado and Vitis Development Workflow,” *Reiwa Embedded Systems and Electronics*. [Online]. Available: https://reiwaembedded.com/ultra96-v2-vivado-and-vitis-development-workflow/. [Accessed: 07-Feb-2021].

“Vitis Unified Software Platform Documentation,” *Xilinx*. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1400-vitis-embedded.pdf. [Accessed: 07-Feb-2021].

“Vivado 2019.2 and Vitis Unified Software Platform Installation,” *Reiwa Embedded Systems and Electronics*. [Online]. Available: https://reiwaembedded.com/vivado-2019-2-and-vitis-unified-software-platform-installation/. [Accessed: 07-Feb-2021].

Vivek BhageriaVivek is a Senior Embedded Engineer at Robert Bosch. He has been working on Embedded Systems for the past 10 years. He loves to share his knowledge and train those who are interested. Nerdyelectronics.com was started out of this interest.www, “Priority scheduling with semaphores,” *NerdyElectronics*, 12-Jun-2020. [Online]. Available: https://nerdyelectronics.com/articles/semaphores-priority-scheduling/. [Accessed: 07-Feb-2021].

W. Lin, “MegunoLink Arduino Library: Overview: Documentation,” *MegunoLink*, 26-Sep-2020. [Online]. Available: https://www.megunolink.com/documentation/arduino-library/. [Accessed: 07-Feb-2021].

"WebSockets vs REST: Understanding the Difference," *PubNub*, 27-Jan-2021. [Online]. Available: https://www.pubnub.com/blog/websockets-vs-rest-api-understanding-the-difference/. [Accessed: 07-Feb-2021].

"What is Bastion Host," *What is Bastion Host Server - Learning Journal*. [Online]. Available: https://www.learningjournal.guru/article/public-cloud-infrastructure/what-is-bastion-host-server/. [Accessed: 07-Feb-2021].

Xilinx, "Xilinx/Vitis-AI," *GitHub*, 17-Dec-2020. [Online]. Available: https://github.com/Xilinx/Vitis-AI/blob/master/demo/VART/README.md. [Accessed: 07-Feb-2021].

Xilinx, "Xilinx/Vitis-Tutorials," *GitHub*, 06-Dec-2020. [Online]. Available: https://github.com/Xilinx/Vitis-Tutorials/blob/master/Getting_Started/Vitis/Part2.md. [Accessed: 07-Feb-2021].

Xilinx, "Xilinx/Vitis-Tutorials," *GitHub*, 07-Dec-2020. [Online]. Available: https://github.com/Xilinx/Vitis-Tutorials/blob/master/Getting_Started/Vitis/Part1.md. [Accessed: 07-Feb-2021].