DEVELOPMENT OF MICROCONTROLLER-BASED WEARABLE DEVICE WITH MONITORING SYSTEM FOR BODY-FOCUSED REPETITIVE BEHAVIOR

Undergraduate Design Project
Submitted to the Faculty of the
College of Engineering and Information Technology
Cavite State University
Indang, Cavite

In partial fulfillment
of the requirements for the degree
Bachelor of Science in Computer Engineering

RYAN CHRISTOPHER D. BAHILLO
IRIKS MARIA B. DALANON
April 2024

BIOGRAPHICAL DATA

Ryan Christopher D. Bahillo was born in Muntinlupa, Manila on May 29, 2000, the eldest among the two children of Lilibeth D. Bahillo and Reynaldo E. Bahillo. He is currently residing at 261 Purok 5 Matagbak 1 Alfonso Cavite.

He took his primary education at Matagbak Elementary School in Barangay Matagbak 1, Alfonso Cavite. He finished junior high school education at Alfonso National High School and his senior high education at Olivarez College – Tagaytay, located along Emilio Aguinaldo Highway in Tagaytay. Pursuing higher studies, He pursued his tertiary education at the Cavite State University – Main Campus and took up a Bachelor of Science in Computer Engineering.

BIOGRAPHICAL DATA

Iriks Maria B. Dalanon was born in Sta. Cruz, Manila on October 21, 1999, the eldest of the four children of Richard F. Dalanon Sr. and Nora B. Dalanon. She is currently a resident at 114, Iruhin Central, Tagaytay City, Cavite.

She took her primary education at Tagaytay City Central School in Sungay West, Tagaytay City. She finished her junior and senior high school education at Tagaytay City Science National High School- Integrated Senior High located at Sungay West, Tagaytay City. She attended Cavite State University's Main Campus to complete her tertiary education, where she took Bachelor of Science in Computer Engineering.

ACKNOWLEDGMENT

This design project owes its realization to the valuable contributions and unwavering support of numerous individuals and entities, without whom its completion would not have been possible.

Foremost, we express our deepest gratitude to God the Father, whose guidance and strength sustained us through the challenges encountered during the development of this system, enabling us to achieve what we once deemed unattainable.

We extend our sincere appreciation to Prof. Marivic G. Dizon, our project adviser, for her unwavering support, patience, and invaluable guidance throughout the entirety of this endeavor.

We are indebted to Dr. Edwin R. Arboleda for his assistance and the provision of essential information crucial to our study.

Our heartfelt thanks go to the panel members and faculty of the Department of Computer Engineering and Electronics (DCEE) for their constructive suggestions and insightful comments, which significantly contributed to the improvement of our work.

We extend our gratitude to Dr. Willie C. Buclatin, the dean of the College of Engineering and Information Technology, for his unwavering support throughout this project.

We express our appreciation to the respondents whose cooperation greatly facilitated the success of our study.

We acknowledge with gratitude Cavite State University, particularly Dr. Hernando D. Robles, the university president, for the support extended to us.

Lastly, we are deeply thankful to our families and friends for their understanding, encouragement, and unwavering support, which played an instrumental role in the completion of this study.

BAHILLO, RYAN CHRISTOPHER D.
DALANON, IRIKS MARIA B.

ABSTRACT

BAHILLO, RYAN CHRISTOPHER D., and DALANON, IRIKS MARIA B. Development of Microcontroller-Based Wearable Device with Monitoring System for Body-Focused Repetitive Behavior. Undergraduate Design Project. Bachelor of Science in Computer Engineering. Cavite State University, Indang, Cavite. April 2024. Adviser: Prof. Marivic G. Dizon.

This study aims to develop a microcontroller-based wearable device with an integrated monitoring system to reduce Body-Focused Repetitive Behaviors (BFRB) activities. Utilizing the Arduino Nano 33 BLE Sense as its core, the device used sensors to classify user behavior and provide real-time feedback via a mobile app, enabling users to track progress and evaluate effectiveness over time.

Key components include sensors for data acquisition, a Feedforward Neural Network (FNN) for anticipatory classification, and file management modules to ensure data integrity. The device consumes around 30mA of battery power, with BLE transmission speed at 380 bytes per second.

While daily performance fluctuations were observed, extending the evaluation period beyond one week is recommended for a comprehensive assessment. Additionally, there was no significant difference in detection accuracy across different BFRB categories such as hair pulling, nail biting, and skin picking, indicating consistent performance regardless of behavior type.

The Multivariate Regression analysis shows varied responses to the BFRB detection device. Some respondents showed decreased behaviors while others increased, emphasizing the importance of considering individual differences in intervention tool efficacy for BFRB. Additionally, the Ordinary Least Squares (OLS) model demonstrated strong predictive power, with an F-statistic of 240.8 and a p-value of 3.33E-31, confirming its reliability in forecasting performance changes over time, as indicated by the R-squared value of 0.976.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF APPENDIX TABLES

LIST OF APPENDIX FIGURES

DEVELOPMENT OF MICROCONTROLLER-BASED WEARABLE DEVICE WITH MONITORING SYSTEM FOR BODY-FOCUSED REPETITIVE BEHAVIOR

Ryan Christopher D. Bahillo
Iriks Maria B. Dalanon

An undergraduate design project submitted to the faculty of the Department of Computer and Electronics Engineering, College of Engineering and Information Technology, Cavite State University, Indang, Cavite in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Engineering with Contribution No._____. Prepared under the supervision of Prof. Marivic G. Dizon.

INTRODUCTION

Body-focused repetitive behavior (BFRB) is a term that refers to a group of compulsive habits that unintentionally harm one's body and alter one's appearance but the factors that predispose individuals to these behaviors are poorly understood. The main distinction between BFRBs and other compulsive behaviors that hurt the body is that BFRBs are characterized by repetitive, self-grooming behaviors that result in damage to the body. Pulling, picking, biting, or scraping one's hair, skin, or nails are examples of these behaviors. Trichotillomania such as hair pulling, Dermatillomania such as skin plucking, also known as Excoriation disorder, and Onychophagia are among the disorders such as compulsive nail biting. According to Smitha Bhandari (2020), ***"As many as 1 in 20 people have a BFRB, but they can be dismissed as bad habits."*** It is also stated that obsessive-compulsive disorder (OCD) is not the same as BFRB. OCD is a mental health disorder characterized by intrusive, distressing thoughts (obsessions) and repetitive behaviors or mental acts (compulsions) performed to alleviate anxiety. Compulsions in OCD are often elaborate and may not be directly related to body-focused grooming behaviors. Someone with OCD may engage in rituals such as checking, counting, or arranging items to reduce anxiety

triggered by obsessions. Though they share some similarities with OCD, BFRBs typically involve a different underlying mechanism and response to treatment. BFRBs are often managed with habit-reversal therapy, cognitive-behavioral therapy, or medication, while OCD is typically treated with cognitive-behavioral therapy and exposure and response prevention therapy.

In approach of Son et al., (2019) to BFRB monitoring, a study shows the data collected in different locations on the head can be calculated by measuring the distance between each pair of the target locations on the head using the data from the proximity and the Inertial Measurement Unit (IMU) sensors. They disassembled the N68 Fitness Tracker to utilize its main component as their Micro Controller Unit (MCU) – Nordic nrf52832 and sensors. The Keen created by HabitAware (2020) is a wearable-based tracking device to detect BFRB activity. It uses gesture recognition for the initial use that makes the device recognize such habit. It then transmits a vibration signal to the user wearing the device. Despite that, it is unclear how well-suited they are for BFRB monitoring or treatment because there has been no published peer-reviewed study showing the effectiveness of the device.

This study aims to develop a microcontroller based wearable technology that conveys a signal to the user and is integrated with a mobile application for monitoring of repetitive behavior relative to its position and motion sensors in real time. According to Leibinger et al., (2023), a Habit-reversal therapy (HRT) is a type of behavioral therapy that aims to help individuals become more aware of their BFRB behaviors, learn to recognize triggers, and develop alternative responses to replace the habit. The development of the wearable device can assist in the treatment of the individual's repetitive behavior. Wearable devices can be designed to provide sensory feedback (such as vibration or buzzing) when they detect the individual engaging in the BFRB behavior. This feedback prompts individuals to halt repetitive behaviors, promoting awareness of their occurrence. The development of the wearable device utilizes cues

such as vibration and sound to interrupt these behaviors, promoting engagement in alternative actions. Equipped with sensors and a trained model, the device allows the users to manage their repetitive behaviors effectively. It is important to clarify that this study offers support for BFRB individuals' self-awareness rather than medication.

Statement of the Problem

Many individuals with Body-Focused Repetitive Behaviors (BFRBs) engage in anticipatory behaviors such as hair pulling, skin picking, or nail biting, often without full awareness or conscious control. These repetitive actions can result in considerable distress and may disrupt daily activities. In response to this challenge, the researchers have developed a wearable device aimed at detecting anticipatory behavior associated with BFRBs. This provides real-time feedback to individuals, enabling them to become more aware of their behaviors and potentially reduce their occurrence. By addressing the issue of anticipatory behavior detection, this study seeks to offer a proactive approach to managing BFRBs and improving the overall well-being of affected individuals.

Objectives of the Study

The study was conducted in the creation of a microcontroller-based wearable device that was used to help the user control its repetitive behavior. To accomplish this, the following objectives were met:

1. To develop a microcontroller based wearable technology that conveys an alert to the user.
2. To embed mobile application for monitoring and motion sensors in real time.
3. To assist in the treatment of Body-Focused Repetitive Behavior using the wearable device.
    a. Trend analysis of percentage change over time.

   b. Calculate the overall improvement experienced by the user over several days of use.
4. To evaluate the effectiveness and performance of the wearable device such as hardware and software stability when predicting with a neural network.

Significance of the Study

The findings of the study could be a great help in providing crucial information and knowledge about the development of the device. It can also help the scope of the study in treating their behavioral disorders. Specifically, the results of this study could benefit the following:

Individuals with BFRB. The study could help treating the behavioral disorders of individuals. It can benefit from the constructed device by the researchers. The device can understand the external triggers that lead a person to engage in their BFRB, as well as the external events that reinforce them that make this behavior more likely to happen again in the future.

Society. The study could give basic understanding of what is BFRB, the difference of other compulsive behaviors, and why a person urges to occur this kind of behavior. This could also help society to give insights on how the device can help to manage its anxiety for their repetitive behavior.

Medical Professionals. The findings of this study can give medical professionals a finding on how haptic feedback helps their patients reduce their repetitive behavior. They can also evaluate the effectiveness of this device towards their target and future audiences.

Future Researchers. This study can be beneficial to the new researchers that are conducting related research that may be used as their reference data. This can also serve as their cross-reference that can give them a background or an overview for the construction of the wearable device.

Time and Place of the Study

The wearable device underwent its initial conceptualization and construction phase in August 2023, within Alfonso Cavite, Philippines. Following a series of iterative adjustments and refinements to its design and construction methodology, the device reached its final stage of development in February 2024. A comprehensive study was undertaken to evaluate the efficacy and functionality of the device during a period from March 3rd, 2024, to April 15th, 2024. The testing and evaluation spanned from various locations to perform data gathering of each participant.

Scope and Limitations of the Study

This study covers the development of a microcontroller based wearable device that can detect body-focused repetitive activities such as trichotillomania, excoriation, and onychophagia. It documents how the researcher constructs the wearable device, development of the mobile application, and web server for accessing neural network training. This study also tests how the device accurately predicts the hotspot location for the repetitive behavior of the user. The numerical results of the usage taken by the user in several days are then calculated for trend analysis to find the percentage of change. It is done using device evaluation through test and trials, a survey to evaluate the acceptability of the developed software application, and qualitative field test of the wearable device for a period of selected days.

Definition of Terms

To have a full understanding of this paper, the following are the prominent terms used as presented in this study. This is intended to assist in understanding commonly used terms and concepts when reading, interpreting, and evaluating scholarly research in this study.

Accelerometer is a device that measures acceleration, which is the rate of change of the velocity of an object. They measure in meters per second squared (m/s^2) or in G-forces (g).

Anticipatory Classification is a process where a system predicts or classifies data before it is fully received or processed based on available information or patterns.

Bluetooth Low Energy (BLE) is a power-conserving variant of Bluetooth personal area network technology, designed for use by Internet-connected machines and appliances.

Body-Focused Repetitive Behavior (BFRB) refers to a group of psychological disorders characterized by repetitive self-grooming behaviors that result in damage to the body. These behaviors are often compulsive in nature and can lead to physical and emotional distress.

Checksum is a small-sized block of data derived from another block of digital data for the purpose of detecting errors that may have been introduced during its transmission or storage.

C++ is a programming language popular for its efficiency and versatility. Its low-level memory manipulation capabilities make it suitable for various applications.

Dart is a programming language developed by Google, popular for its simplicity and speed. It is used for building various types of applications, especially mobile apps with Flutter.

Deep Learning is a type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher-level features from data.

Dermatillomania also known as excoriation disorder or skin-picking disorder, is a mental health condition characterized by repetitive picking at one's own skin, leading to skin lesions and significant distress or impairment in daily functioning.

Feedforward Neural Network (FNN) is a type of artificial neural network that processes data in one direction, without feedback loops. It consists of interconnected nodes compute outputs by weighing input sums and applying activation functions. This structure enables tasks like classification and pattern recognition.

Gyroscope. It is a device that can measure and maintain the orientation and angular velocity. These can measure the tilt and lateral orientation of the object.

Hypertext Transfer Protocol (HTTP) is an application protocol for distributed collaborative hypermedia information systems that allows users to transmit data over the internet.

Onychophagia is the compulsive habit of biting one's nails, often resulting in damage to the nails and surrounding skin.

Representational State Transfer API is commonly known as RESTful or REST API. This is an architectural style for an application program interface (API) that accesses and utilizes data using HTTP requests.

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It is popular for web development, data analysis, Artificial Intelligence, and more due to its simplicity and strong community support.

Trichotillomania is a mental health disorder characterized by recurrent, irresistible urges to pull out hair from the scalp, eyebrows, or other areas of the body, resulting in noticeable hair loss.

Wearable is any technology that is designed to be used while worn. Common types of wearable technology include smartwatches and smart glasses.

Conceptual Framework of the Study

Figure 1 shows how the researchers came up with the construction of the wearable device. The researchers use an Input-Process-Output model for planning the development. Generally, it enumerates the hardware and software requirements to have an effective outcome. While conducting the study, researchers also evaluated each process to test the performance and effectiveness of each feature completion until certain requirements are satisfied.

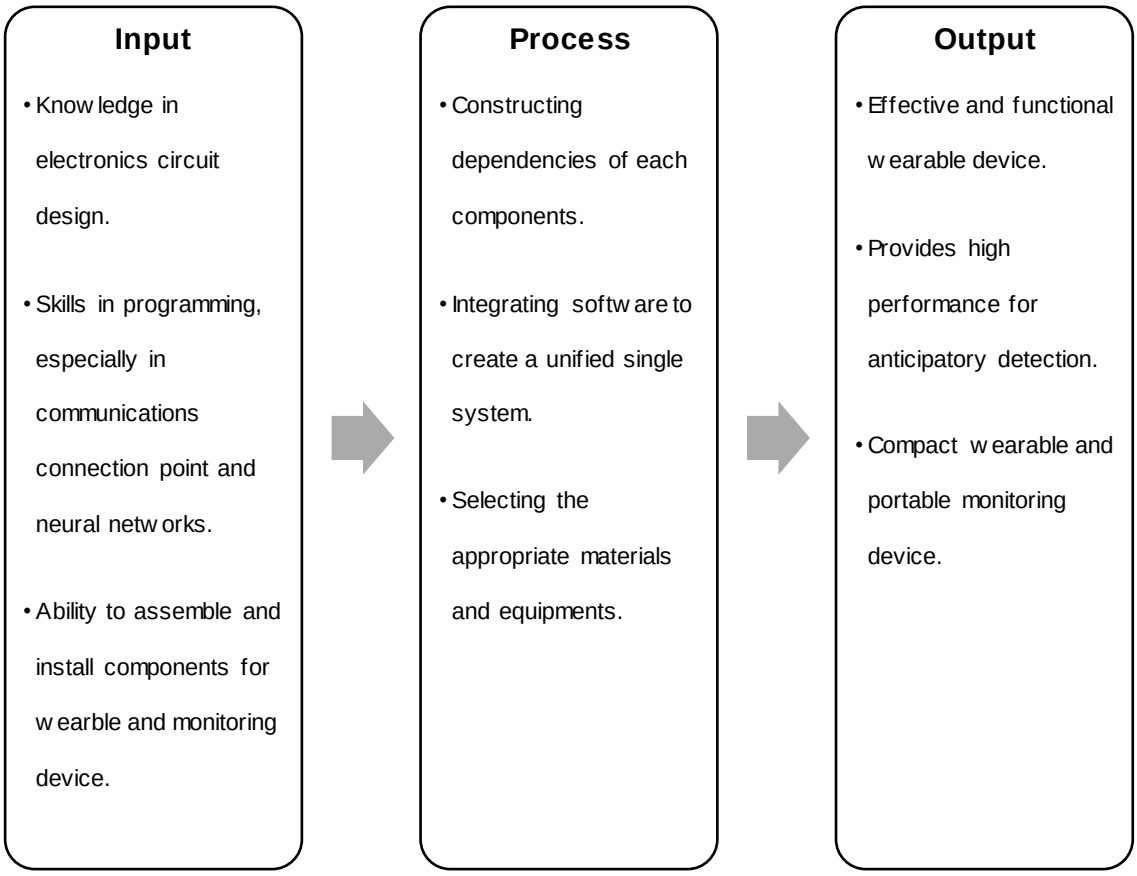


Figure 1. Conceptual framework of the wearable and monitoring device.

REVIEW OF RELATED LITERATURE

This chapter contains literature and studies in both local and foreign to support the study. This can also identify gaps or contradictions in current literature, which can then be discussed further after reviewing the study. Through the study, the researchers addressed these gaps and resolved these conflicts.

Body-focused repetitive behavior interventions

As Houghton, D. (2018) behavioral interventions have been utilized to block symptom performance and induce extinction of the BFRB habit to permit fewer instances of symptom performance. It is hypothesized that by continuing to abstain from the BFRB symptoms, reinforcement is no longer supplied, and the behavior should become less common over time.

A study about using N-Acetylcysteine (NAC) as solution for the treatment of Trichotillomania, Excoriation Disorder, and Onychophagia found that it was effective in lowering compulsive behaviors in BFRB disorders. According to Lee & Lipner, (2022), although NAC has been shown to be effective in the treatment of BFRB problems, evidence is taken from a small number of clinical studies and case reports involving a small number of individuals. Larger, longer-term trials are required to properly demonstrate NAC's effectiveness in these illnesses.

The study by Searle B. et al. (2021) addresses the detection of BFRBs, such as face-touching or skin-picking, which can lead to appearance damage if left untreated. Existing automatic detection technologies are limited, often focusing on single modalities like motion. To overcome this, they propose a multi-sensory approach integrating motion, orientation, and heart rate sensors. Through a feasibility study involving 10 participants, they analyzed 380 minutes of signals and evaluated various sensing modalities, cross-validation methods, and observation windows. The models

achieved an AUC > 0.90 in distinguishing BFRBs, particularly evident in 5-minute observation windows preceding the behavior. Additionally, a qualitative survey highlighted the importance of context-awareness in designing timely interventions to prevent BFRBs.

Wearable device integration

In a related study of Son et al., (2019) published in npj Digital Medicine, a team lead by Child Mind Institute researchers found that utilizing thermal sensors in addition to inertial measurement and proximity sensors, a wearable tracking system they designed achieves greater accuracy in position tracking. Tingle, a wrist-worn gadget, could also tell the difference between actions aimed at six distinct parts of the head. The paper, titled "Thermal Sensors Improve Wrist-worn Position Tracking," provides preliminary evidence of the device's potential use in the diagnosis and treatment of excoriation disorder, nail-biting, trichotillomania, and other body-focused repetitive behaviors.

As Closa & Tambaoan (2018), microcontrollers such as Arduino Nano are effective to make a low-cost wearable device. They also added a digital thermometer, oximeter, Organic Light Emitting Diode (OLED) display that is capable of sensing body temperature, heart rate, and oxygen saturation. It then displays the information in real-time through an OLED screen. Data is stored in their database using a Hypertext Markup Language (HTML) website. This gives the researchers that Arduino can make HTML requests and responses through the internet.

According to Kok M. et al., (2017), to integrate inertial measurements with other sensors and models for position and orientation estimation, it is necessary to precisely define the quantities obtained by the inertial sensors as well as characterize the usual sensor errors. These physical properties are monitored along three sensitive axes in 3D accelerometers and 3D gyroscope sensors. They are measured in terms of an

output voltage, which is translated to a physical measurement using factory calibration values. Even if the sensors are normally calibrated at the factory, inaccuracies, which are time-varying, might still exist.

The study by Daskalos A-C. et al. (2022) introduces a wearable IoT device for real-time monitoring of novel coronavirus (COVID-19) symptoms. It uses a unique Continuous Displacement Algorithm to differentiate temperature fluctuations caused by physical activity. The device communicates with stakeholders via platforms like nRF Connect and demonstrated robust performance on an Arduino Nano BLE 33 Sense. With over 12 hours of battery life, rapid charging, and efficient Bluetooth Low Energy (BLE) 5.0 data transmission, it offers a promising solution for COVID-19 monitoring. Additionally, a 1D Convolutional Neural Networks (CNN) effectively identifies fever patterns, providing valuable insights for managing cases.

Machine learning for wearable devices

According to the study of Wang H. et al., (2023), traditional machine learning (ML) algorithms like support vector machine (SVM), backpropagation neural network (BP), decision tree (DT), and random forest (RF) can perform gesture recognition classification. They used an inertial sensor-based gesture data acquisition system with the goal of constructing a gesture-recognition model based on the collected static and dynamic gesture datasets.

In a study conducted by Liu S. et al. (2020), a method for monitoring the dynamic movement of human limbs using wearable technology was introduced. This method integrates motion capture with velocity detection, eliminating the need for additional equipment. They developed a wearable device by combining micro tri-axis flow sensors with micro tri-axis inertial sensors. This device enables precise measurement of three-dimensional motion velocity, acceleration, and attitude angle during various daily activities, as well as strenuous and prolonged exercises.

Additionally, they employed a back propagation (BP) neural network to discern the coordination within limbs during walking and running activities.

The study by Sabry F., et al. (2022) explores the utilization of artificial intelligence and machine learning in healthcare wearable devices. It highlights the potential of these technologies in tracking human activities, diagnosing diseases, and improving patient health monitoring. Despite significant research efforts, few ML applications have reached the market. The study reviews recent ML research in healthcare wearables, discussing challenges, potential solutions, and areas for further improvement and research. This work contributes to advancing the efficacy and adoption of ML in healthcare wearables.

Hutabarat J. P., et al. (2023) proposed a feedforward neural network (FFNN) model with four fully connected layers to improve Human Activity Recognition (HAR) accuracy. Their experiment, using the UCI-HAR dataset, demonstrated that the FFNN achieved an accuracy and F1 score of 0.96, outperforming traditional algorithms like K-Nearest Neighbors (KNN) and Support Vector Machines (SVM). This underscores the effectiveness of a simplified neural network architecture in enhancing HAR performance.

Monitoring applications

As Navarro M. et al., (2019). A low-cost prototype was created by embedding a pulse rate sensor, Global Positioning System (GPS), and Global System for Mobile communication (GSM) modules in a wearable wrist band. Additionally, an SMS message is sent to an emergency contact, and a locating map may be seen on a smart phone or computer. The response time for sending a distress notification varies depending on the strength of the mobile network signals.

A study of Cruz F. et al., (2016) from Mapua University in Manila, accelerometer integration human belt is utilized to examine the accelerometer's capacity for body

detection. The prototype sends messages if the predetermined values are reached by the accelerometer reading. This study created a human belt tracking belt gadget that enables users to text a selected contact with their current position using short message service (SMS). The gadget determined the user's activities of daily living, which were limited to walking, sitting, standing, and lying.

According to the study by Deng Z. et al. (2023), the demand for smart wearable systems in health monitoring is rising. These systems offer portability, long-term monitoring, and comfort in detecting bio signals. While advancements in materials and system integration have increased the availability of high-performance wearable systems, challenges remain in balancing flexibility, sensing performance, and system robustness. Their study summarizes recent achievements and strategies for material selection, system integration, and bio signal monitoring. The next generation of wearable systems is expected to enable more accurate, portable, continuous, and long-term health monitoring, enhancing opportunities for disease diagnosis and treatment.

According to Tanna R. & Vithalani C. (2023), the rapid expansion of the Internet of Things (IoT) is transforming healthcare, particularly in sports medicine. Wearable devices like fitness trackers and body sensors play a crucial role in assessing physiological parameters and enhancing health among athletes and patients. Their study focuses on using IoT, wearables, and machine learning to continuously monitor athletes' health, aiming to improve well-being, prevent injuries, and optimize training. They introduce a smartwatch-based system for athlete health monitoring and employ an ensemble naïve Bayes classifier (ENBC) for predicting health activity. Results show high accuracy of 98.63%, making this approach efficient, reliable, and aligned with consumer preferences for cost-effectiveness and ease of use.

METHODOLOGY

In this chapter, the researcher praesents the approach in making the wearable device. It includes the research design and pattern to achieve the objectives of this study. The materials and estimated cost are also shown in this chapter to give understanding on the features and emphasize its functions. The evaluation of performance is discussed at the end of this chapter whereas the crucial part of making the device effective.

Materials

Arduino Nano 33 BLE Sense is a lot more powerful processor than the ordinary Arduino Nano. It uses the nRF52840 from Nordic Semiconductors with a 32-bit ARM Cortex-M4 CPU running at 64 MHz. The size of its program memory is 1MB and 256KB of SRAM for more variables. It has an integrated LSM9DS1 Inertial Measurement Unit sensor, which typically includes an accelerometer, gyroscope, and magnetometer. These sensors allow the board to measure acceleration, rotation, and magnetic fields in three-dimensional space, enabling precise motion tracking and orientation detection. The main feature of this board is its capability of transmitting and receiving data using the Bluetooth Low Energy (BLE) communication chipset.

VL53L0X Time-of-Flight Sensor works optically by emitting short infrared pulses and measuring the time it takes for the light to be reflected. The sensor can measure distances up to 2 meters, though it depends significantly on several conditions like surface reflectance, field of view, temperature etc. In general, developers can expect surfaces up to 60cm to work, after that they need to make sure the surface is reflecting well enough.

Micro Vibration Motor is typically used to describe a small electric motor designed to generate vibrations on a miniature scale. These motors are commonly

found in various electronic devices such as smartphones, wearables, and gaming controllers. They are used to provide haptic feedback, notify users of incoming calls or messages, and for vibration alerts in various applications. The rated voltage is 2.5 to 3.8V, it vibrates from 2V up to 5V, higher voltages result in more current draw but also a stronger vibration.

Passive Buzzer is an electronic component used in circuits to generate audible tones. It requires an external oscillating signal to produce sound. By modulating the input signal's frequency, different tones can be generated. Passive buzzers are cost-effective and commonly used in alarms, notifications, and interactive feedback systems.

LP503035 Lithium Polymer Battery is a rechargeable battery, it has an output range of 4.2V when completely charged to 3.7V. This battery has a capacity of 500mAh and can be easily incorporated into a variety of electrical products. The battery has one prismatic cell in a one-series, one-parallel arrangement. Over-charge, over-discharge, over-current, and short-circuit protection are all provided by integrated battery PCBs or protection circuit boards.

TP4056 Charger Module is a lithium battery charger for a single cell battery, protecting the cell from over and under charging. It has two status outputs indicating charging in progress and charging complete. It also has a programmable charge current of up to 1A. It can be used to charge batteries directly from a USB port since the working input voltage range is 4V to 6V.

Single Pole, Double Throw (SPDT) switch has a single input and two dissimilar outputs which is used to control two dissimilar circuits through a similar single input.

Micro SD Card Module enables to read or write to the memory card and connect with it. The SPI protocol is used for the module interfaces. High-capacity

memory cards cannot be used with these modules. Typically, these modules have a maximum capacity of 2GB for SD cards and 16GB for micro-SD cards.

MLX90614 Temperature Sensor is an infrared thermometer for non-contact temperature measurements. This component has a pulse width modulation digital output setting (PWM). The 10-bit PWM is typically set up with an output precision of 0.14°C and designed to send temperature readings constantly in the range of -20 to 120°C.

BC547 Transistor is a general-purpose NPN bipolar junction transistor (BJT) commonly used in electronic circuits for amplification or switching purposes. The maximum current gain of BC547 is 110 up to 800.

Carbon Film Resistor (220 ohms) is a cylindrical in shape, resembling a small capsule, and their resistance value is indicated by a series of colored bands painted around their body. The color code is used to determine the resistance value, tolerance, and sometimes even the temperature coefficient of the resistor. Carbon film resistors are widely used in electronic circuits due to their reliability, accuracy, and low cost.

Pololu U3V12F9 9V Step-Up Voltage Regulator is a compact and efficient electronic module used for boosting input voltages to a fixed 9V output voltage. It can convert input voltages as low as 2.5V up to 9V, making it suitable for a wide range of applications where a stable 9V power source is required. This regulator utilizes a step-up switching regulator topology, which enables it to achieve high efficiency while minimizing power loss and heat generation. It features a synchronous rectification mechanism, which further enhances efficiency by reducing voltage drop across the device.

Methods

The research methodology provides a comprehensive prototyping process where the literature is critically reviewed, important prototype goals are examined, key techniques are reviewed critically, and links between techniques are analyzed. These insights are then combined. The construction of the wearable device for body-focused repetitive behavior enables the device for anticipatory detection, it detects anticipating behavior for a specific part of the body in real-time.

To make the device operational, a combination of programming languages is utilized. C++ is used for Arduino programming, ensuring seamless integration and control of hardware components. Dart is utilized for developing a mobile application for monitoring system, providing a user-friendly interface and real-time communication capabilities. Additionally, Python is used for the web server, enabling efficient data processing and remote access to device functionalities. These programming languages complement each other, working together to achieve the objectives of the study by facilitating robust functionality, accessibility, and user interaction across various platforms.

The process of testing and evaluating the device takes weeks due to the utilization of qualitative field-testing methods. It was carried out in multiple locations to ensure diverse feedback and perspectives. Various functionalities of the device undergo a thorough evaluation to ensure optimal performance and user experience. Despite the testing procedures, the device maintains a wearable design with a compact form factor, suitable for users of varying wrist sizes. Thus, provides a simple and flexible user experience, making manual testing easy to perform.

System Overview

Figure 2 illustrates the interactions within the system. It comprises three (3) main components necessary for its operation: the wearable device, mobile application,

and web server. The wearable device is the main component that classifies the anticipatory behavior of the user. The microcontroller used by the researchers is the Arduino Nano 33 BLE Sense, the wearable needs to connect to the web server to access the file of the user. However, this microcontroller only supports Bluetooth Low Energy (BLE) and does not support connecting via the internet. Therefore, the researchers used a bridge to support connection to the server through a mobile application connecting via BLE. To maintain communication without relying on a physical server, researchers have integrated the server into the cloud. This allows users to effectively utilize the system over the internet.

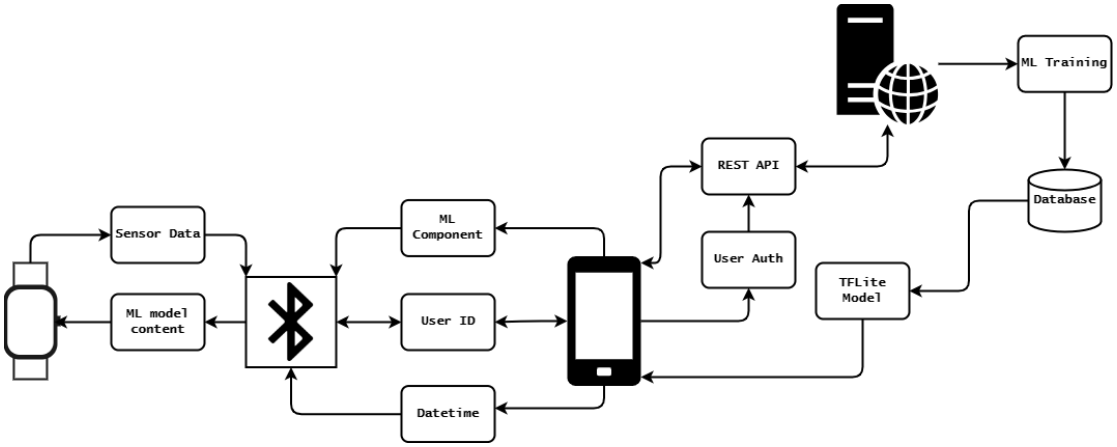


Figure 2. Block Diagram of the System Integration.

Hardware Structure of the Wearable Device

The structure for building the hardware components is shown in Figure 3. For longevity of the device, the researchers used a stack of rechargeable batteries. The researchers also used temperature and proximity sensors to add features for classifying the behavior of the user. There are a total of eight (8) parameter inputs used for the classifier: tri-axial accelerometer, tri-axial gyroscope, and the two external sensors mentioned earlier. These are the crucial parts for classifying the anticipating behavior of the person. The device mainly classifies the relative position of the person wearing the device. Haptic feedback using buzzer and vibration are sent to the user

when the device classifies the anticipating behavior. Additionally, the data of the specific user is stored in a non-volatile memory storage.

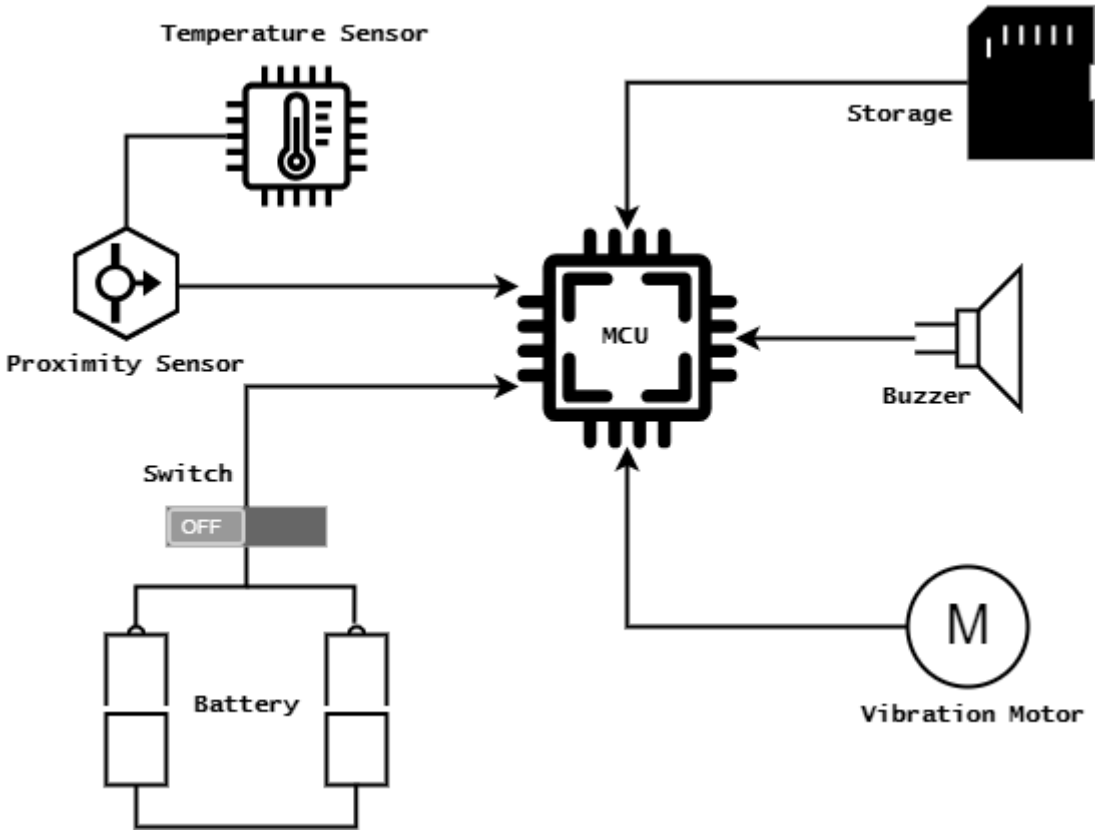


Figure 3. Components Structure for Building the Wearable Device.

Chassis Design of the Wearable Device

Figures 4 and 5 show the dimensions (length, width, and height in millimeters) of the microcontroller based wearable device. The researcher used this dimension to fit all the components inside. The components are protected with a chassis to protect it from any dirt/debris and water particles that may cause from destroying the entire circuit. The chassis is 3D printed with strong filament of plastic for durability. It can also prevent some form of electric shock commonly known as Electrostatic Discharge (ESD) to the user, but this has a very small chance of occurrence because microcontrollers are designed to be low powered. The researchers used to create a compact wearable design like any other smartwatches that can fit any person using it.

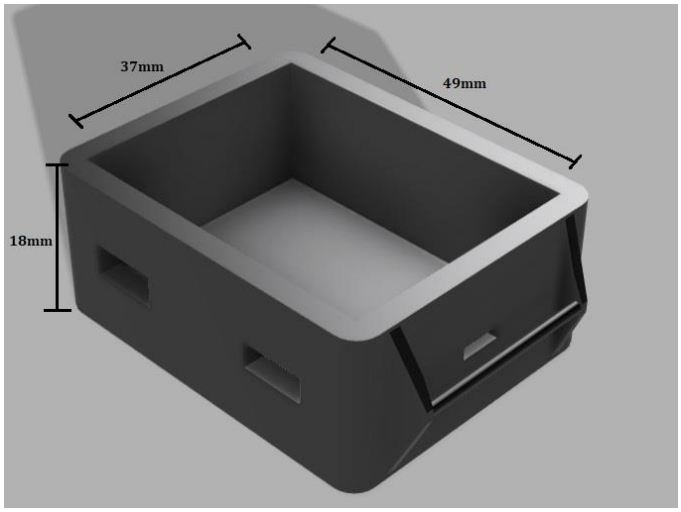


Figure 4. Three-dimensional design of the main board chassis

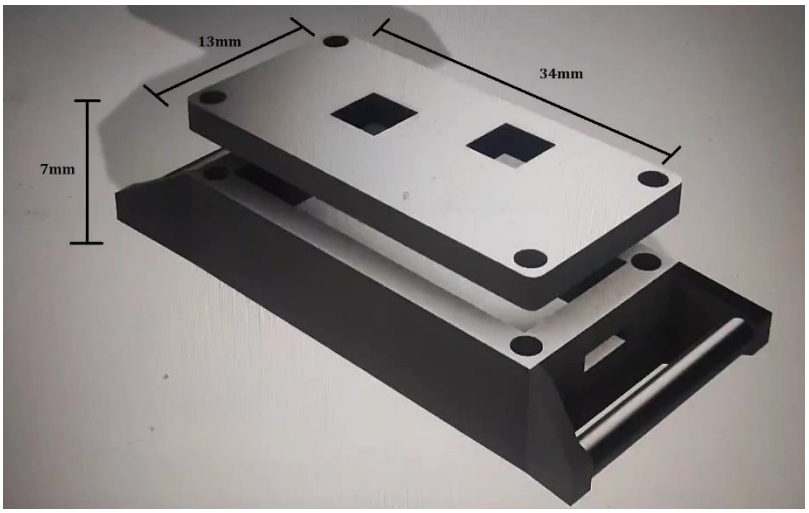


Figure 5. Three-dimensional design of the external sensors' chassis

Development of the Software

In Figure 6, the design for building the mobile application is shown. The researchers carefully considered the simplicity of the mobile application for user experience. The system views or the pages serve as the frontend that divides its individual functions for the application. Accounts management, file management, and other API workflow are done in the backend. These workflows effectively reduce the complexity of the user interface making it easier to use. The mobile application also has its limitations for creating machine learning components. Therefore, the

researchers used a server to connect the application and access the database using the Hypertext Transfer Protocol (HTTP) services.

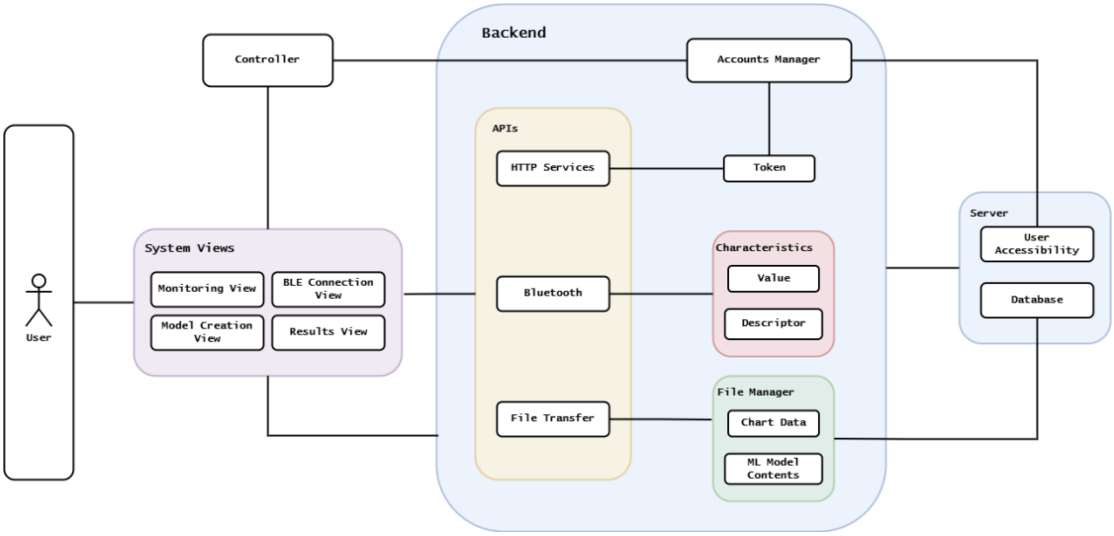


Figure 6. Software Feature Structure of the Mobile Application.

The flowchart of the system is shown in Figure 7 to sum up the process of using the system. The user must turn on the wearable device to begin advertising BLE connectivity. Additionally, the user needs to turn on Bluetooth on their device to use the mobile application; this prompts the user to allow the use of Bluetooth upon first use. After BLE is connected, the user can read the sensors in real-time and begin collecting data of on-target and off-target. Data collection for each class takes 60 seconds to complete. This is because the researchers used a set of 300 samples for each class at a speed of 200 milliseconds per sample. The collected data is then uploaded to the web server to initiate training and export the TensorFlow Lite model. The user must wait for a short period to finish training the model. Subsequently, the model is optimized to reduce the size of the TensorFlow Lite model using quantization, and the content is converted into byte sized characters to send via BLE. Finally, when the model is sent to the wearable, it initializes the new model for use and begins classification. Additionally, the user can view analytics of their improvements on the dashboard page of the mobile application.

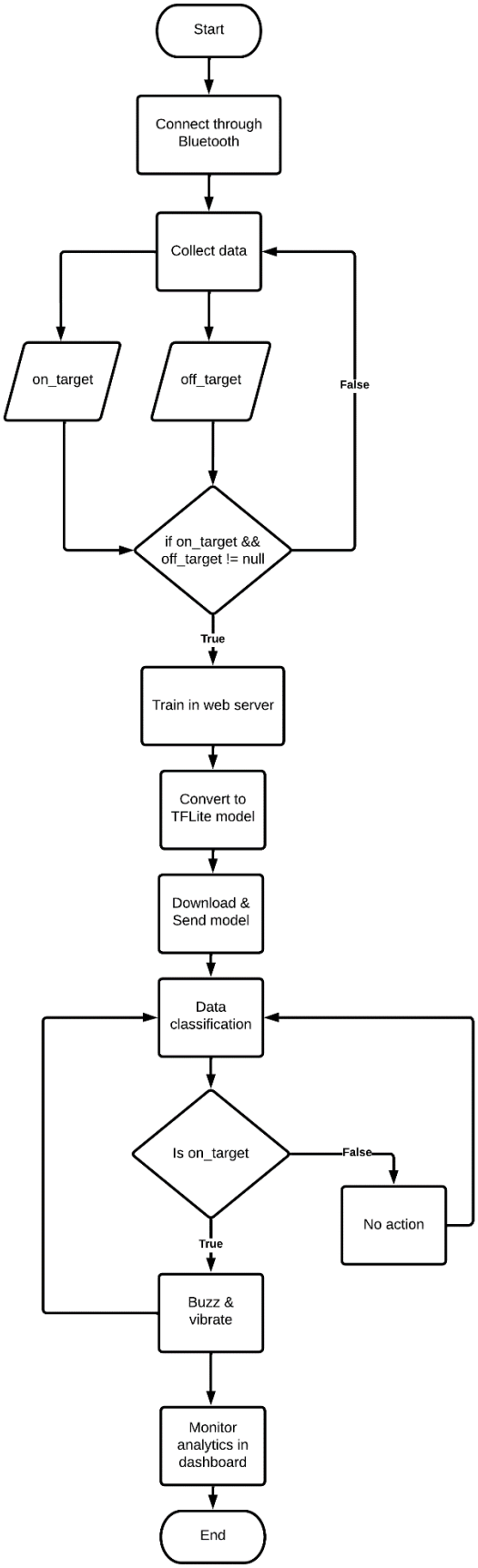


Figure 7. Flowchart of the System.

Anticipatory Classification using Feedforward Neural Network (FNN)

The researchers used machine learning to classify or detect anticipatory behavior of the user. However, every user can have different data for the parameters used, especially for the temperature. Therefore, the researchers solved this problem by letting the user generate data for the parameters to create their own machine learning model. To generate the input data, the user must locate the device to its target. For example, to generate data for detecting hair pulling, the user must wear and position the device near the top of the head, especially focusing on the area where the behavior is most prevalent.

As shown in Figure 8, the researchers used FNN as their ML architecture. The neural network model consists of five (5) fully connected layers with addition of dropout layer and kernel regularization to avoid overfitting. Training the model can consume time especially when the user is waiting for their model to be complete. To address this, the researchers have implemented an early stopping to monitor the validation accuracy. This stops the unnecessary training if the model is not learning. When the model is complete, trained models are saved in the web server media storage to manage the file for every user. In the file management system, the files of a single user are bundled and downloaded using the mobile application. This way, the researchers have avoided file collision when a new user has logged-in using shared devices.

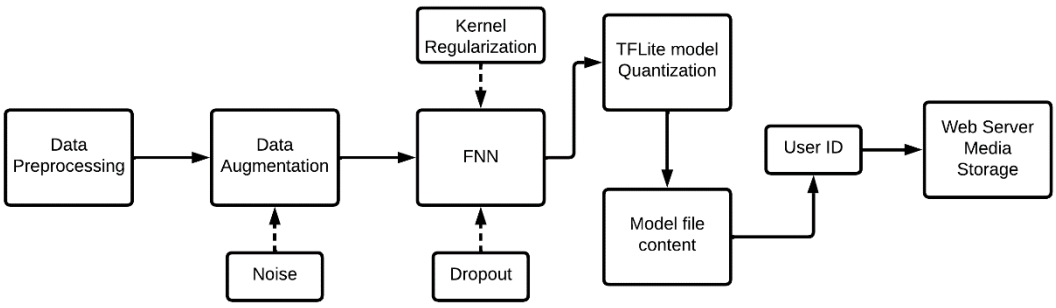


Figure 8. Process for creating a machine learning component in the web server.

Creating the model takes only a short time to complete since the input data needed to train is small and simple. Although small data are generated, the data underwent pre-processing adding noise to the parameters. The researchers used this type of Artificial Neural Network because it is relatively straightforward to train and implement compared to more complex architectures like recurrent or convolutional neural networks that require more memory. This simplicity can be advantageous, because the anticipatory classification of the device does not require capturing temporal dependencies or spatial patterns.

Testing and Evaluation of the System

The performance of the wearable device's neural network was tested and evaluated through Individualized Accuracy Assessment. There are a total of two (2) classifications, on-target which the device must alert the user and off-target which the device stays in its neutral state or does nothing. The researchers tested the device on a total of ten (10) individuals, conducting ten (10) trials for both on-target and off-target areas, including hair, skin, and nails, resulting in an overall total of sixty (60) samples.

The effectiveness of minimizing the repetitive behavior of the user using the wearable device was tested and evaluated through Qualitative Field Testing. The researchers sent consent forms to seven (7) individuals who experienced hair pulling, skin picking, and nail biting, inviting them to participate in the evaluation. For each respondent, the alerts or buzzes are collected daily and stored automatically in the storage of a wearable device. The evaluation lasts one week or seven days for each respondent resulting in an overall total of forty-nine (49) samples.

The researchers also evaluated the mobile application through a survey targeting thirty (30) respondents. Each respondent rated the software's functionality, reliability, efficiency, portability, usability, performance, as well as the information provided in the software, and its technical aspects and materials.

Statistical Treatment

The Mean of the Percentage Change was used for monitoring system to determine the improvement of the user by using the device. The data used for this equation are the stored data from the wearable device which are the number of alerts/buzzes that occurred per day. It is computed using the formula below.

$$\bar{x}_{pc} = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{x_{i-1} - x_i}{x_{i-1}} \times 100\right)$$

Where:

x are data points of anticipated behavior occurred per day.

The Multivariate Multiple Linear Regression was used to determine the numerical relationship between the sets of variables categorizing each respondent. This can also determine if there is a significant difference between improvement of each respondent by using the device over a period of days. It is computed using the formula below.

$$\mathrm{Y} = \beta_0 + \beta_1\mathrm{X}_1 + \beta_2\mathrm{X}_2 + \cdots + \beta_\mathrm{n}\mathrm{X}_\mathrm{n} + \delta_1\mathrm{D}_1 + \delta_2\mathrm{D}_2 + \cdots + \delta_{\mathrm{C}-1}\mathrm{D}_{\mathrm{C}-1} + \epsilon$$

Where:

- Y is the dependent variable.
- $\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_\mathrm{n}$ are the independent variables that are numeric.
- $\mathrm{D}_1, \mathrm{D}_2, \ldots, \mathrm{D}_{\mathrm{C}-1}$ dummy variables (one-hot encoded) from the categorical variable.
- β_0 is the intercept.
- $\beta_1, \beta_2, \ldots, \beta_\mathrm{n}$ are the coefficients for the numeric independent variables.

- $\delta_1, \delta_2, \ldots, \delta_{C-1}$ are the coefficients for the dummy variables.

- ϵ is the error term.

To measure how well a model fits data in multiple linear regression, the metrics are computed using the formula below.

$$R^2 = 1 - \frac{\text{Sum of Squares of Residuals (SSR)}}{\text{Total Sum of Squares (SST)}}$$

Where:

- SSR = $\sum(y_i - \hat{y}_i)^2$
- SST = $\sum(y_i - \bar{y})^2$

The effectiveness of the device measuring the stability for classifying anticipatory behavior by using FNN is calculated using the Analysis of Variance (ANOVA). Table 1 shows the list of equations to measure the stability for this classification. The researchers collected data for classifying the anticipating behavior for Hair, Nail, and Skin to proceed the device evaluation. To analyze the variance of the collected data, the formulas are shown below.

Table 1. Segment of formula for ANOVA.

SOURCE	SUM OF SQUARES	DEGREES OF FREEDOM	MEAN SQUARES	F-RATIO	P-VALUE
Treatment	SSR	df_r	MSR	MSR/MSE	F_{dfr}, df_e
Error	SSE	df_e	MSE		
Total	**SST**	**df_t**			

Where:

- $SST = \sum_{i=1}^{k}\sum_{j=1}^{n_i}\left(Y_{ij} - \overline{Y}\right)^2$
- $SSR = \sum_{i=1}^{k} n_i\left(\overline{Y_i} - \overline{Y}\right)^2$
- $SSE = \sum_{i=1}^{k}\sum_{j=1}^{n_i}\left(Y_{ij} - \overline{Y_i}\right)^2$
- $df_{between} = k - 1$

- $df_{within} = N - k$
- $MSR = \frac{SSR}{df_{between}}$
- $MSE = \frac{SSE}{df_{within}}$
- $F = \frac{MSR}{MSE}$

To determine if the related groups (Hair, Nail, and Skin) are significantly different from each other on a continuous variable, the One-way repeated measure ANOVA is used. Each segment of the formula is shown below.

1. Total Sum of Squares (SST): This is similar to the standard ANOVA but is calculated using all the measurements from all conditions.

$$SST = \sum_{i=1}^{k}\sum_{j=1}^{n}\left(Y_{ij} - \overline{Y}\right)^2$$

2. Sum of Squares Between Subjects (SSB) This measures the variation due to differences between subjects.

$$SSB = \sum_{i=1}^{n}\left(\overline{Y_i} - \overline{Y}\right)^2$$

3. Sum of Squares Within Subjects (SSW): This captures the variability due to differences within each subject across conditions.

$$SSW = SST - SSB$$

4. Sum of Squares Due to Treatment (SSTr): This is the part of the within-subject sum of squares that is due to the different treatment conditions.

$$SSTr = \sum_{j=1}^{k} n\left(\overline{Y_j} - \overline{Y}\right)^2 - SSB$$

5. Degrees of Freedom: These are calculated as follows:
   - Total: $df_{total} = nk - 1$
   - Between Subjects: $df_{subjects} = n - 1$
   - Within Subjects: $df_{within} = nk - n$
   - Due to Treatment: $df_{treatment} = k - 1$
6. Mean Squares: These are calculated by dividing the sum of squares by their respective degrees of freedom.
   - Mean Square Due to Treatment (MSTr): $MSTr = \frac{SSTr}{df_{treatment}}$
   - Mean Square Error (MSE): $MSE = \frac{SSW - SSTr}{df_{within} - df_{treatment}}$
7. F-Statistic: Finally, the F-statistic is calculated to test the significance of the treatment effect.

$$F = \frac{MSTr}{MSE}$$

RESULTS AND DISCUSSION

The chapter presents the results of the study, discussing the data collected through tests and evaluations conducted during the research.

Schematic Diagram of the Wearable

In Figure 9, careful consideration was given to the components used and their connections to ensure effectiveness in achieving the objectives. The Arduino Nano 33 BLE Sense serves as the central component of the device. Its primary tasks include enabling the functionality of other components, establishing connections via BLE, and performing calculations based on sensor readings. The sensors used include the accelerometer and gyroscope of the LSM9DS1 inertial measurement unit (IMU) sensor, the VL53L0X time-of-flight (ToF) distance sensor, and the MLX90614 infrared (IR) temperature sensor, which are crucial for detecting user's anticipating behavior. The data are then used to classify anticipating behavior and use this as a signal to operate the vibration and passive buzzer.

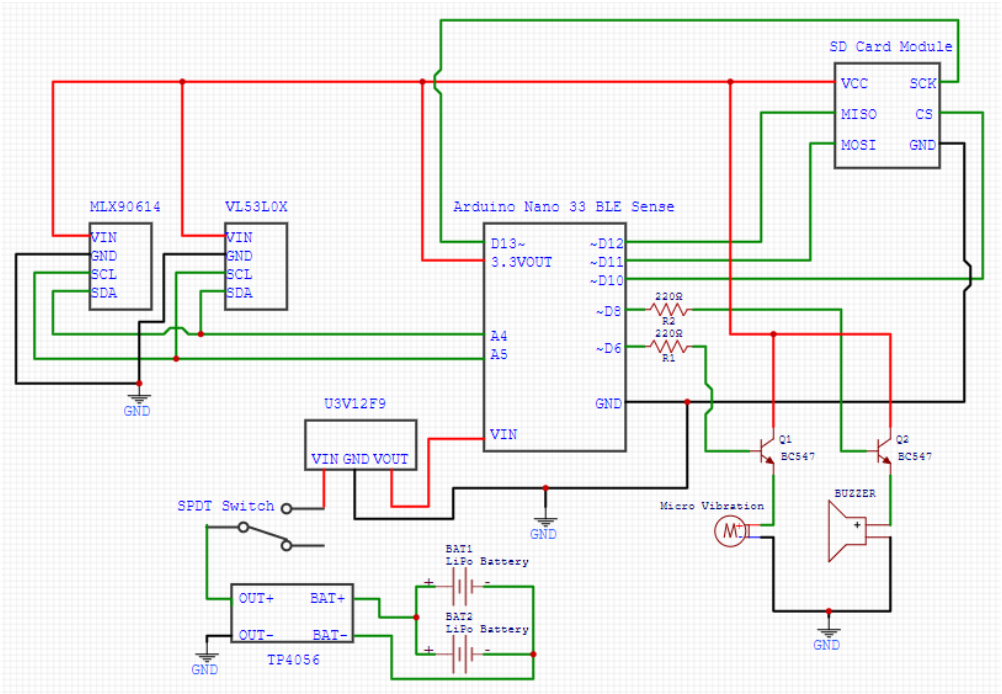


Figure 9. Schematic Diagram of Connected Components of the Wearable Device.

The SD Card Module serves as the physical storage for machine learning components and behavioral activities. To power the circuit, two (2) 3.7V 500mAh LiPo batteries are connected in parallel and linked to the TP4056 charging module, ensuring the wearable device remains rechargeable for extended use. However, the charging module alone cannot power the MCU, as the MCU requires an input voltage of 4.5 – 21V, while the charging module outputs only 3.7 – 4.2V, consistent with the LiPo batteries. To address this issue, the researcher used the U3V12F9 Step-up voltage regulator in conjunction with the charging module. This voltage regulator accepts a minimum input voltage of 2.5V and provides a constant output voltage of approximately 9V, effectively meeting the power requirements of the MCU.

Wearable Device

The components are shown in Figures 10 and 11. The researcher placed the MCU at the top of the circuit to consider the reliability of the BLE connection. The passive buzzer and the micro vibration motor are placed at the edge of the circuit. Therefore, the user makes it easier to recognize the haptic feedback. Two LiPo batteries with a total of 1000mAh are used to double the capacity for extended use. These power supplies are stacked to each other while charging port and switch are placed at the bottom to match the design of the 3D printed chassis. The charger port is exposed on the outside of the chassis. It can connect with a Micro USB Type B connector that is commonly used in the Android charger.

The temperature sensor measures the body temperature without contact in conjunction with another sensor to measure the distance between the device and the target. The purpose of placing these components externally, rather than alongside the MCU which faces outward, is to orient them toward the side of the user's palm. Thus, effectively contributes to the measurements of data for position tracking. Figure 12 shows the completely assembled wearable device. The researcher customized a

watch strap to connect the main chassis and the external chassis. It can adjust the strap size to accommodate all sizes of wrists.

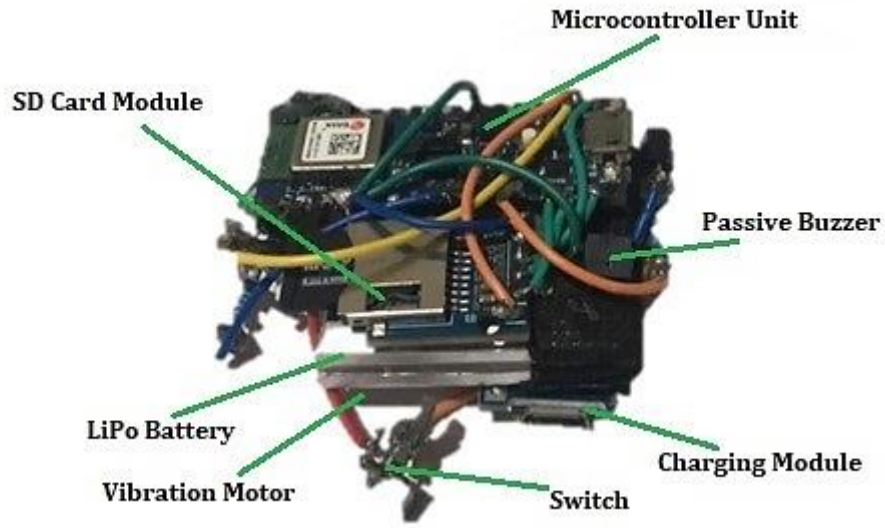


Figure 10. Components inside the main chassis.

Figure 11. External components connecting to the main chassis.

Figure 12. Completely assembled wearable device.

Device Specifications

The researchers thoroughly test the wearable device specifications. As shown in Table 2, the BLE has a maximum distance of approximately 67 feet. The microcontroller's power usage when utilizing BLE and IMU sensors amounts to 4mA, while both the distance sensor and temperature sensor consume 2mA each. During real-time operation, TensorFlow Lite significantly elevates power consumption to approximately 20mA. Additionally, the buzzer and micro vibration motor have minimal power requirements, as they are solely engaged in classification alongside TensorFlow Lite. Considering all components, the overall battery consumption is carefully managed to optimize the device's operational efficiency and longevity. With the BLE, microcontroller, sensors, TensorFlow Lite operations, and additional components combined, the overall battery consumption is approximately 30mA while having 1000 mAh battery capacitance since two (2) parallelly connected 500mAh LiPo batteries were used.

The researchers working with BLE determined that the transmission speed for their machine learning component model files was 380 bytes per second. To experiment, the researchers selected an average model file size of 70 kilobytes. Consequently, it would take approximately 3 minutes and 8 seconds to complete the transfer of these files over BLE. This estimation provides valuable insight into the time required for file transmission to the wearable device.

Table 2. Wearable Device Specs.

BLE DISTANCE (FEET)	BATTERY CAPACITANCE (mAh)	BATTERY LOAD CURRENT (mA)	FILE TRANSFER SPEED (BYTE/S)
67	1000	30	380

Software of the Wearable Device

The programming language used to create the software for the wearable is C++ using the Arduino IDE. The researchers used the library TensorFlowLite for microcontroller machine learning or TinyML. The researchers have allocated 120KB of memory for the machine learning component/model for classification, with an average model size ranging from 60KB to 90KB. The SDFat library works in conjunction with ArduinoJSON library to save the machine learning component and collects data of the occurred detection in json format. The ArduinoBLE library is important in sending sensor data and receiving commands through the mobile application. This also makes the mobile application able to send the machine learning component through BLE. The library used for reading sensor values are LSM9DS1, VL53L0X, and MLX90614 which are for 33eclerometer and gyroscope, distance, and temperature sensor. The sensor values are normalized ranging from zero to one to increase the classification performance.

The researchers used to buzz and vibrate the wearable for an interval of 5 seconds per classification of anticipating behavior. While researchers are working to test the components, they noticed that the output voltage of each digital pin of the

microcontroller is constant at 3.3V while the components require more voltage to make the vibration and buzzer effective on noticed. The passive buzzer requires to have an alternating of high and low signal to make it work but researchers noticed a decrease by half of the peak voltage which is 1.65V. Therefore, the researchers used the BC547 BJT transistor to make a common-collector transistor configuration to increase the current gain. This is also applied for the vibration motor to increase the strength of haptic feedback.

The researchers used a Generic Attribute Profile (GATT) for Bluetooth connection. It consists of service, this can have one or multiple characteristics, and each service differentiates itself from others through a distinct numerical identifier known as a Universally Unique Identifier (UUID). The researcher used a notify property to write the sensor values to the mobile application in real-time. Before the wearable continues to receive the model via BLE, the mobile application sends a calculated 32-bit cyclic redundancy check (CRC32) to detect errors in digital data transmission. It achieves this by generating a checksum, a short sequence of bits based on the data being checked, which is appended to the data. When the data is received, CRC32 is recalculated from the wearable device, and if the checksum doesn't match the one initially generated, it indicates that the data has been corrupted or altered in some way during transmission. The error notifies the mobile application that the data does not match, the model cannot initialize, and the user can resend the contents of model again. This error happens when there is noise interference. This interference can be caused by electromagnetic interference, signal attenuation, or other external factors, but this is rare to happen when used properly. The researchers used this operation to detect errors for efficiency and memory management of the wearable because instead of comparing the two large data together, they can instead compare an eight-byte character plus the two-byte prefix which significantly reduce the memory consumption of the wearable device.

Mobile Application

The researchers used the Flutter framework in the Dart programming language for building a mobile application. Although the framework is a multi-platform, the mobile application only supports Android operating system because some packages do not support iOS specially for Bluetooth operations. In Figure 13, the accounts page is shown. The researchers have integrated the HTTP Service in the client-side of the system to send requests to the server. The user must create their account on the register page on first use. Once they have created their account, they can use their credentials to login to the mobile application. The user does not need to login again their credentials to use the mobile application because once the user has logged in, their credentials including the token are saved and encrypted in the application to authenticate each time the user has opened the application.

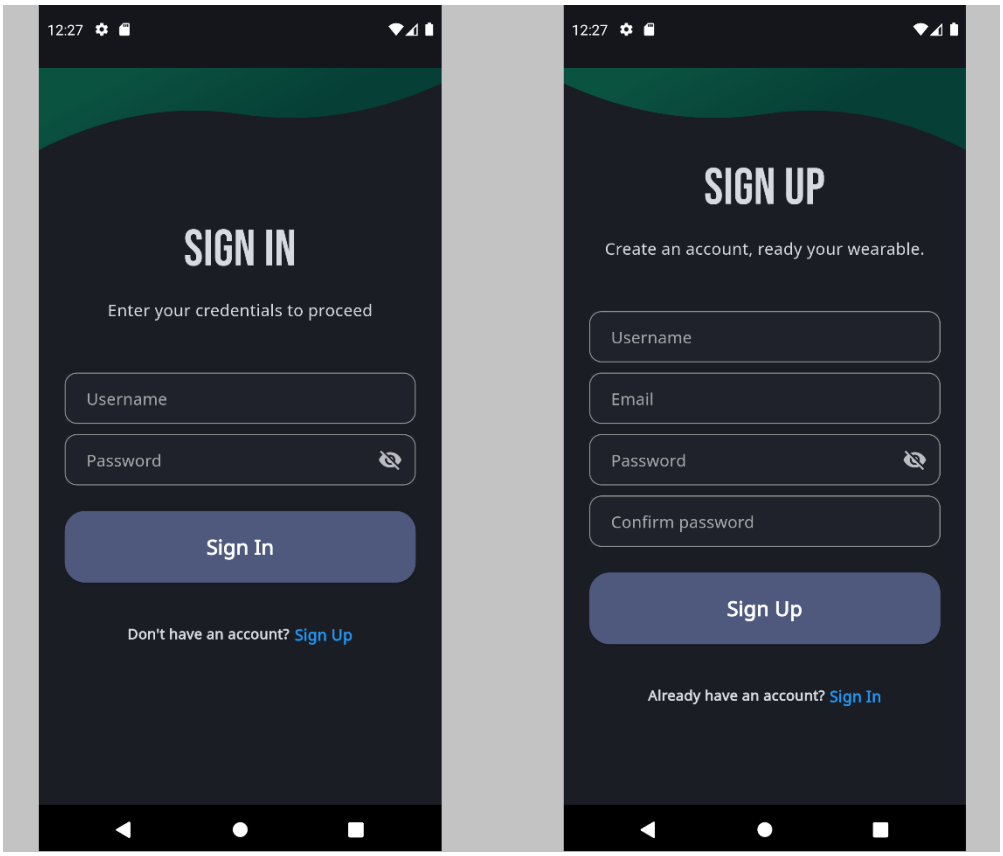


Figure 13. Accounts Page of the Mobile Application.

In Figure 14, the researchers have divided the page for Dashboard, this is the monitoring system of the application. The user can analyze its improvements over the week or month. The UI indicates if the data is improving if the trend is decreasing otherwise, not improving. The researchers have integrated calculations to analyze the trend for user experience. The improvements section in the monitoring system is calculated using the mean of the percentage change.

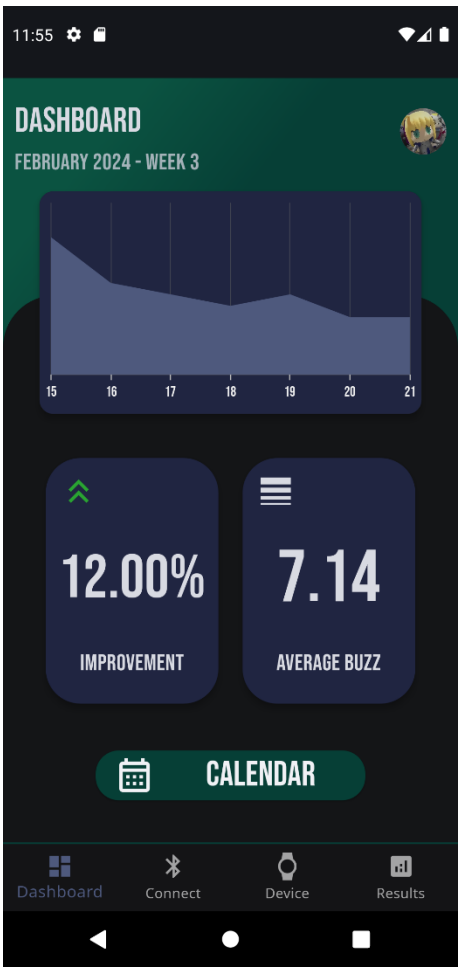


Figure 14. Monitoring System of the Mobile Application.

In Figure 15, only the button for Bluetooth connectivity is shown, this is important when the user decides to train a new model or update the dashboard to get data from the wearable device. Therefore, connecting the wearable device to the mobile application through BLE is not necessary, the user can still use the wearable

device normally. It also indicates a callback message if the device is successfully connected, or if some error has occurred during the connection.

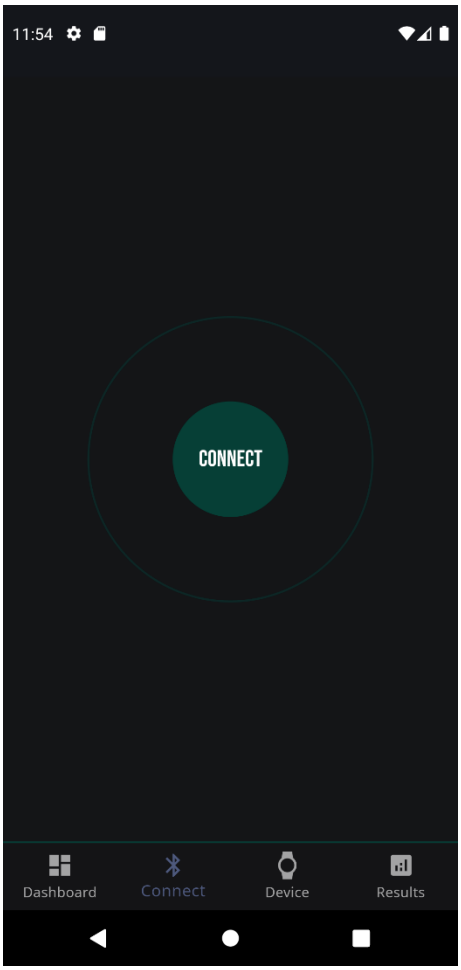


Figure 15. Bluetooth Connect Button of the Mobile Application.

The sensors page is shown in Figure 16. In this page, the user can train their new model by collecting data from the on-target and off-target. Once the collection of data is complete the user can tap the build button to train the model. The user is also notified by the message when the model is still training or completed. The purpose of adding the sensor readings in real-time is to indicate that the sensor is working properly and there are no problems of data transmission from the wearable device.

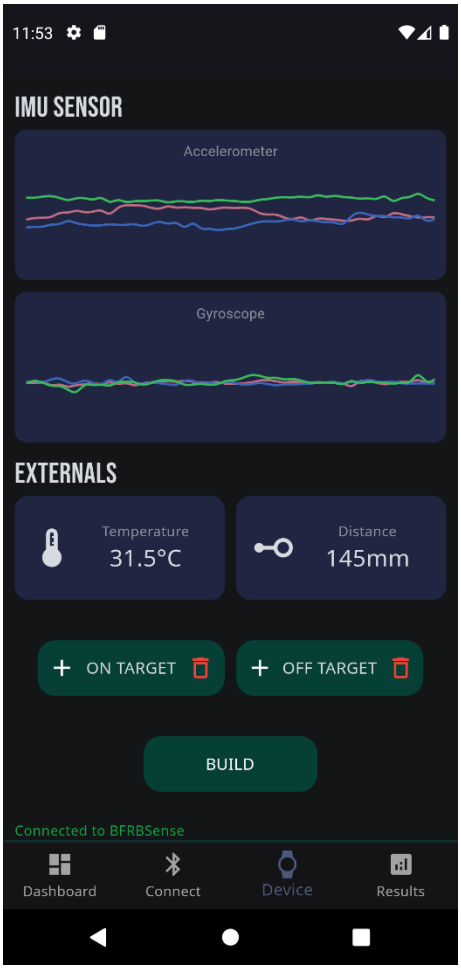


Figure 16. Sensors Data Collection Page of the Mobile Application for Training.

In Figure 17, the results page is shown. The user can monitor the model accuracy in the progress of training. This also includes the user's model saved in the web server and they can use this to update the anticipatory detection of the wearable device by sending through the BLE in the transfer section. They can also view the progress of the sent bytes and remaining bytes to the wearable.

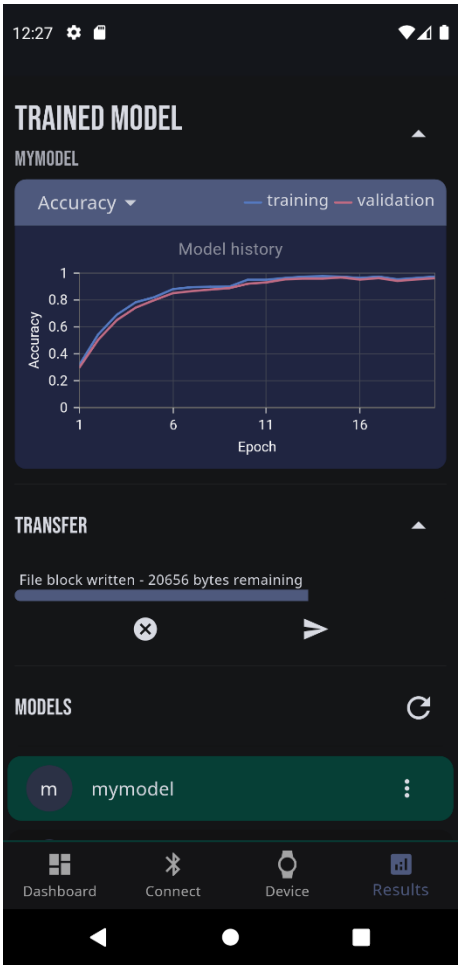


Figure 17. Results Page of the Mobile Application.

Web Server

The web server is created using the Django web framework in Python programming language. It is a Model-View-Template (MVT) architecture. The model in the architecture represents the data structure and logic while view handles user interaction and presentation logic. The template in the architecture is not widely used because the researchers have used Restful (REST) API to interact with the mobile application. Researchers also have an account management system integrated in their webserver. The accounts management system is protected using token authentication to add a layer of security to the middleware.

The primary use of the web server is to produce a machine learning component model. This is accomplished using the TensorFlow library. When the input data is

uploaded to the server, it is then normalized from zero to one value to increase the performance and accuracy of the model. The researchers used One Hot Encoding to label the output during the preprocessing because the researchers used a categorical cross entropy as their loss function. The data are augmented by adding noise to each sensor input and split in 60% for training and 40% for validation. This is due to the small size of the dataset and the simplicity of the model. The dataset comprises only 1200 samples across both classes, making it relatively small. Additionally, the model complexity is straightforward as it deals with one-dimensional time-series data.

Test and Evaluation

The researchers conducted Qualitative Field Testing across various locations in the Philippines. Respondents willingly consented to utilize wearable devices over several days for data collection. The tests were performed on individuals showing behaviors such as hair pulling, nail biting, and skin picking. The classification of anticipatory behavior was then recorded and stored in the memory storage of the wearable device. Each respondent underwent a 7-day observation period, with a total of 7 respondents, resulting in a comprehensive dataset comprising 49 samples.

Hair Pulling. In Figure 18, only one (1) respondent provided their photo for documentation purposes, with careful consideration given to safeguarding their privacy rights by excluding their faces. Through observation, it was noted that certain respondents who engaged in hair pulling showed an improvement upon using the wearable device. Feedback from respondents highlighted the effectiveness of the device, with some expressing that it served as a helpful reminder to avoid behavior. However, others noted that its effectiveness varied, particularly in cases where they were consciously engaging in extreme behaviors.

Figure 18. Respondent's Hair due to excessive pulling.

Nail Biting. In the context of nail biting, Figure 19 shows the photo of fingernail of a respondent with their consent. Respondents reported that the wearable device proved effective in reducing their behavior due to heightened sensory feedback, particularly through vibrations and audible cues. This effectiveness was attributed to the device's proximity to their faces. However, this observation posed a challenge for the researchers as they hold with the uncertainty of the device's efficacy over time. The researchers expressed concerns regarding the difficulty in assessing the device's impact after several days of use, as respondents might have already bitten their nails, eliminating the immediate trigger for the behavior.

Figure 19. Respondent's fingernail due to excessive biting.

Skin Picking. Figure 20 shows the photo provided with the respondent's consent. The respondent's skin exhibits inflammation and dark marks on the arm because of skin picking. The respondents who engaged in skin picking behavior reported a significant reduction in this habit when using the wearable device. Despite engaging in this behavior infrequently, its consequences were severe, often resulting in self-inflicted wounds. They emphasized that the effectiveness of the device's vibration and audible alerts was most effective when worn on their dominant hand, the very hand responsible for the skin picking behavior. This focused method improved how well the device reduced how often skin picking happened.

Figure 20. Respondent's skin due to excessive plucking.

As shown in Table 3, data gathered on hair pulling behavior shows a higher frequency compared to instances of nail biting and skin picking. This difference arises from the abundance of hair as a target for pulling that contrast with the limited availability of fingernails to bite and the self-inflicted pain associated with skin picking.

Table 3. Data collected from the respondents after a seven-day period.

RESPONDENT	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5	DAY 6	DAY 7	CHANGE (%)
1 (Hair)	56	60	60	51	45	40	34	7.62
2 (Hair)	80	89	88	95	101	97	98	-3.58
3 (Hair)	68	77	80	89	70	82	87	-5.05
4 (Nail)	15	11	8	12	13	9	6	9.95
5 (Nail)	10	7	5	6	8	5	6	3.79
6 (Skin)	5	7	5	8	4	3	2	6.15
7 (Skin)	10	12	10	4	7	5	5	1.71

In Table 4, the multivariate linear regression table presents the results of a trend analysis examining the performance of respondents utilizing a device designed to detect Body-Focused Repetitive Behaviors (BFRB). The analysis aimed to discern

if there was a significant trend in the number of device alerts (buzzes) over the study period, which could indicate a change in BFRB occurrences.

The Respondent 1 making it an intercept, representing the estimated number of buzzes at the baseline (the starting point of the study), was significant with a coefficient of 50.8571 ($p < 0.001$), suggesting a high initial frequency of BFRB detection by the device. The regression coefficients for the respondents represent the difference in the number of buzzes from this baseline.

Significant negative coefficients for Respondents 4 (-38.8571, $p < 0.001$), 5 (-42.7143, $p < 0.001$), 6 (-44.5714, $p < 0.001$), and 7 (-41.8571, $p < 0.001$) suggest a substantial reduction in BFRB detections compared to the baseline, indicating an improvement in managing their BFRB when using the device.

Conversely, the positive coefficients for Respondents 2 (43.1429, $p < 0.001$) and 3 (29.5714, $p < 0.001$) indicate an increase in the number of buzzes relative to the baseline, which could suggest less efficacy of the device for these individuals or an increase in BFRB frequency.

The coefficient for time ('Day') was not significant (-0.3571, $p = 0.405$), indicating no overall trend in the number of buzzes throughout the study period. This implies that the change in BFRB frequency is not associated with the duration of device usage over the days measured.

The analysis demonstrates individual variability in response to the BFRB detection device. While some respondents showed a significant decrease in detected behaviors, suggesting potential benefit from the device's use, others exhibited an increase, and no overall time trend was observed. These results underscore the importance of considering individual differences when evaluating the efficacy of intervention tools for BFRB.

Table 4. Multivariate Regression for trend analysis in respondent's performance.

VARIABLES	COEFF	STD ERR	T – VALUE	P – VALUE	REMARKS
Intercept	50.8571	2.818	18.048	0.000	Significant
Respondent 2	43.1429	3.179	13.572	0.000	Significant
Respondent 3	29.5714	3.179	9.302	0.000	Significant
Respondent 4	-38.8571	3.179	-12.223	0.000	Significant
Respondent 5	-42.7143	3.179	-13.437	0.000	Significant
Respondent 6	-44.5714	3.179	-14.021	0.000	Significant
Respondent 7	-41.8571	3.179	-13.167	0.000	Significant
Day	-0.3571	0.425	-0.841	0.405	Not significant

Significant: p – value < 0.05

Table 5 provides a summary of the results from an Ordinary Least Squares (OLS) regression analysis, which was performed to assess the trend in respondents' performance over time. The OLS model yielded an F-statistic of 240.8 with 7 degrees of freedom, resulting in a highly significant p-value of 3.33E-31. This indicates that the model as a whole has a statistically significant predictive capability at a level far below the conventional alpha threshold of 0.05.

The R-squared value of 0.976 suggests that approximately 97.6% of the variance in the respondents' performance is accounted for by the model. This high value indicates a strong association between the time and the performance of the respondents as measured by the number of buzzes detected by the BFRB device.

Given the highly significant F-statistic and the substantial R-squared value, we can conclude that the regression model provides a robust statistical representation of the trends in performance across the study period. The results suggest that changes in performance over the observed days are not due to random chance but rather can be reliably predicted by the model.

Table 5. Regression for trend analysis in respondent's performance.

MODEL	F	DF	SIG.	R-SQUARED
Ordinary Least Squares	240.8	7	3.33E-31	0.976

Reject H_0: p-value < 0.05.

The researchers also conducted a device evaluation to measure the FNN accuracy of the device for anticipatory classification. Each participant trained their own machine learning component model. To measure the accuracy, each participant intentionally placed their hand wearing the device, classifying two distinct behaviors: placing it near the target category as a proxy for anticipatory behavior and not placing it near the target category. These target samples were categorized based on hair, nail, and skin behaviors associated with BFRB.

As shown in Table 6, the device test involved ten participants, and a total of 60 samples were collected, each comprising 10 trials. The researchers collected a number of alerts or buzzes for each trial for off-target and on-target. The device should alert or buzz for on-target actions, otherwise remain silent for off-target actions.

Table 6. Occurrence of alerts in device evaluation.

	HAIR		NAIL		SKIN	
RESPONDENTS	**OFF**	**ON**	**OFF**	**ON**	**OFF**	**ON**
A	0	10	0	10	0	9
B	1	9	0	10	1	10
C	0	10	0	10	0	10
D	2	10	0	9	0	9
E	0	10	0	10	2	10
F	0	10	1	10	0	9
G	0	8	0	10	0	10
H	0	10	0	9	0	10
I	1	8	0	10	0	9
J	0	10	0	10	0	10

Table 7 displays the results of a one-way ANOVA conducted to evaluate the accuracy of the BFRB detection device across three different behaviors: Hair, Nail, and Skin. The analysis was structured to ascertain whether there were statistically significant differences in detection accuracy between these categories.

The ‘Between’ group variance, which reflects the differences in accuracy among the three BFRB types, yielded an F-statistic of 0.600 with an associated p-value of 0.558. This is below the critical F-value, and the p-value exceeds the 0.05

significance level, indicating that the variance in the accuracy of the device across the three BFRB behaviors is not statistically significant. Consequently, the null hypothesis that there is no difference in the mean accuracy among the groups cannot be rejected based on this analysis.

The 'Within' group variance represents the variability of accuracy measurements within each BFRB category, and with 27 degrees of freedom, the mean square within groups is 38.889. This suggests that the variance within each category of BFRB is considerable, but without significant between-group variance, this within-group variation does not translate to differences across the BFRB types.

Table 7. ANOVA table of accuracy detecting the BFRB.

GROUPS	SUM OF SQUARES	DF	MEAN SQUARE	F	SIG.
Between	46.667	2	23.333	0.600	0.556
Within	1050.000	27	38.889	-	-

Reject H_0: p-value < 0.05.

Table 8 presents the Welch's ANOVA results, a robust test employed when the assumption of homogeneity of variances has been violated, as indicated by a significant Levene's test. This analysis serves as an alternative to the traditional ANOVA to accommodate the observed heterogeneity in variances across the different BFRB behavior categories.

The Welch's ANOVA yielded an F-statistic of 0.714 with two degrees of freedom for the hypothesis and 17.005 degrees of freedom for the error. The associated p-value of 0.504 is above the conventional alpha level of 0.05, suggesting that the Welch's test, similar to the standard ANOVA, does not provide sufficient evidence to reject the null hypothesis. This result corroborates the findings of the standard ANOVA, indicating that there is no significant difference in the detection accuracy between the Hair, Nail, and Skin BFRB categories when variance heterogeneity is accounted for.

Both the standard and Welch's ANOVA tests indicate that the device's accuracy in detecting BFRB does not significantly differ among the types of behaviors analyzed. These findings suggest that the device performs with comparable accuracy when detecting BFRB regardless of the behavior's location (Hair, Nail, or Skin).

Table 8. Welch's ANOVA table of accuracy detecting the BFRB.

TEST	F	DF 1	DF 2	SIG.
Welch	0.714	2	17.005	0.504

Reject H_0: p-value < 0.05.

In the analysis of variance within subjects over a seven-day period, a repeated measures one-way ANOVA was conducted to examine the number of device buzzes for detecting Body-Focused Repetitive Behaviors (BFRB). As shown in Table 9, the multivariate test, specifically Wilk's Lambda, yielded an F-statistic of 4.993 with 6 degrees of freedom and an error degree of freedom of 1, resulting in a p-value of 0.330. This indicates that the mean number of buzzes did not significantly vary over the seven days at the conventional 0.05 alpha level.

Table 9. Multivariate tests table of buzzes detecting the BFRB.

TEST	VALUE	F	DF	ERROR DF	SIG.
Wilk's Lambda	0.032	4.993	6	1	0.330

Reject H_0: p-value < 0.05.

However, the assumption of sphericity, which is required for a valid one-way repeated measures ANOVA, was assessed using Mauchly's test. In Table 10, the test yielded a significant result (Chi-square = 48.844, df = 20, p = 0.002), suggesting that the assumption of sphericity had been violated. Consequently, the degrees of freedom for the F-tests were adjusted using the Greenhouse-Geisser correction to account for this violation.

Table 10. Mauchly's test of sphericity table of accuracy detecting the BFRB.

WITHIN SUBJECTS EFFECT	VALUE	CHI-SQUARE	DF	SIG.
Buzz	0.000	48.844	20	0.002

Reject H_0: p-value < 0.05.

Following this correction, in the Table 11 tests of within-subjects effects reported a Greenhouse-Geisser F-statistic of 0.431 with an adjusted degree of freedom of 1.807, leading to a non-significant p-value of 0.641.

This further supports the initial finding that there were no significant differences in the detection accuracy of the device over the seven days. Summing it up, while the device's performance did vary from day to day, these variations were not statistically significant when tested over a period of one week. The findings indicate that one week is not sufficient, thus extending the number of days for evaluating the device for an individual. The application of the Greenhouse-Geisser correction confirmed the robustness of this conclusion, despite the violation of sphericity assumptions.

Table 11. Tests of Within-Subjects effects table of accuracy detecting the BFRB.

SOURCE	TEST	F	DF	SIG.
Buzz	Greenhouse – Geisser	0.431	1.807	0.641

Reject H_0: p-value < 0.05.

Software Evaluation

Table 12 displays the evaluation results of 30 respondents who filled out a survey form in terms of software functionality, reliability, efficiency, portability, usability, information in the software, technical aspects of the software and materials, and performance. Through weighted mean scores, standard deviations, and an interpretation legend, this analysis offers valuable insights into the software's performance across these diverse criteria.

The functionality of the mobile application 49echnicalsing security measures, outcomes, and the delivery of precise information receives an excellent rating achieving a weighted mean of 4.71. This high score underscores the application's efficacy in meeting user needs and expectations, as well as its robustness in safeguarding sensitive data.

The reliability of the mobile application, including the integrity of data that is maintained all throughout its operation while showing no failures or bugs, receives an excellent rating with a weighted mean of 4.65. The high level of reliability shown by the application not only includes confidence in its users but also underscores its suitability for critical tasks, contributing to its overall effectiveness and user satisfaction.

The efficiency of the mobile application operating quickly, efficiently, and requiring less resources (memory/CPU/disk storage/internet bandwidth) receives an excellent rating with a weighted mean of 4.75. This indicates that the application delivers optimal speed and functionality while using fewer resources, enhancing user experience, and reducing strain on devices.

The portability of the mobile application requiring less effort to install and adapts to new specifications or operating environments receives an excellent rating with a weighted mean of 4.73. The high rating suggests that the application's portability enhances user convenience and accessibility, providing widespread usage and satisfaction.

The mobile application's usability whether the software is organized, clear, logical, and effective in user comprehension receives an excellent rating of 4.59. This indicates that the app is user-friendly, easy to navigate, and prioritizes user experience, resulting in enhanced satisfaction and engagement.

The clarity and conciseness of information in the mobile application for the intended user and comprehensively covering relevant content receives an excellent rating of 4.60. This indicates that the application effectively delivers relevant information in a clear and concise manner, ensuring user understanding and satisfaction.

The50echnicall aspects of the software and materials within the mobile application, including its diverse uses and capabilities such as the use of graphics and

color, receive an excellent rating of 4.73. This suggests that the application offers a rich and versatile experience, applying advanced features like graphics and color to enhance user engagement and effectiveness.

The performance of the mobile application evaluated based on its speed in loading times, response times, and stability, along with its ability to function without errors or crashes receives an excellent rating of 4.73. This indicates that the application delivers a seamless and reliable experience, ensuring prompt responses and stable operation, which enhances user satisfaction and productivity.

Table 12. Overall results from thirty respondents of software evaluation.

STATEMENT	WEIGHTED MEAN	STANDARD DEVIATION	INTERPRETATION
Functionality *The software provides appropriate functions, security measures, and shows accurate information and results.*	4.71	0.63	Excellent
Reliability *The software shows no failures or bugs, and the integrity of data is maintained all throughout its operation.*	4.65	0.65	Excellent
Efficiency *The software operates quickly, efficiently, and requires less resources (memory/CPU/disk storage/internet bandwidth).*	4.75	0.63	Excellent
Portability *The software requires less effort to install and adapts to new specifications or operating environments.*	4.73	0.51	Excellent
Usability *The software functions are organized, clear, logical, and effective making it easy for the user to understand.*	4.59	0.73	Excellent
Information in the Software *The information is clear, concise, and informative to the intended user and*	4.60	0.81	Excellent

contents are covered in a comprehensive manner.			
Technical Aspects of the Software and Materials *The software uses standard equipment that is reliable, widely available, and applicable to a variety of uses and capabilities such as graphics, color are used for appropriate instructional reasons.*	4.73	0.63	Excellent
Performance *The speed in terms of loading times and response times are fast while stability of the app, including frequency of crashes, freezes, or errors are none.*	4.73	0.63	Excellent
COMPOSITE MEAN	4.69	0.65	Excellent

Legend:

1.00 – 1.79 Poor

1.80 – 2.59 Fair

2.60 – 3.39 Satisfactory

3.40 – 4.19 Very Satisfactory

4.20 – 5.00 Excellent

Cost Analysis

The list of the materials required for the construction of the wearable device, along with their corresponding prices, is presented in Table 13. The prices of the components have been sourced from various existing product pages available on online commerce platforms, as well as international and local electronic stores. Summing it up, the total cost for all the necessary materials amounts to PHP 6,251.00.

Table 13. List of components and corresponding costs.

QUANTITY	MATERIALS	UNIT PRICE (PHP)	TOTAL PRICE (PHP)
1	Arduino Nano 33 BLE Sense	3,770.00	3,770.00
1	VL53L0X ToF Sensor	130.00	130.00
1	MLX90614 Temperature Sensor	520.00	520.00
2	LP503035 Lithium Polymer Battery	220.00	440.00
1	Micro SD Card (2 GB)	74.00	74.00
1	Micro SD Card Module	65.00	65.00

2	Carbon Film Resistor (220 ohms)	27.50	55.00
1	SPDT Slide Switch	30.00	30.00
1	Passive Buzzer	136.00	136.00
1	Micro Vibration Motor	150.00	150.00
1	Pololu U3V12F9 9V Step-Up Voltage Regulator	349.00	349.00
2	BC547 Transistor	36.00	72.00
1	Li-po Battery Charger	175.00	175.00
1	TP4056 Charger Module	120.00	120.00
20	Insulated Stranded Wire	8.25	165.00
TOTAL COST			6,251.00

SUMMARY, CONCLUSION, AND RECOMMENDATIONS

This chapter presents the researchers' summary, conclusion and recommendations based on the information obtained during the project's calculations, planning, evaluations, and interpretations.

Summary

The purpose of this study is to design and implement a microcontroller-based wearable device integrated with a monitoring system aimed at reducing Body-Focused Repetitive Behaviors (BFRB) in individuals. This device provides real-time feedback and data tracking through a mobile application, allowing users to monitor their progress over time. By using this technology, individuals can gain insights into their BFRB activities and assess the effectiveness of the device in reducing such behaviors.

The wearable device was first conceptualized and built in August 2023 in Alfonso Cavite, Philippines. Through iterative adjustments and refinements, the device reached its final development stage by February 2024. From March 3rd to April 15th, 2024, a thorough study was conducted to assess the device's effectiveness and functionality. Testing and evaluation were carried out across different locations to gather data from each participant.

The Arduino Nano 33 BLE Sense acts as the central component of the device, responsible for coordinating various functions. It facilitates connections via BLE, processes sensor data from components like the LSM9DS1 IMU sensor, VL53L0X ToF distance sensor, and MLX90614 IR temperature sensor. These sensors play a vital role in predicting user behavior. The gathered data is utilized to classify anticipated behavior, triggering actions such as vibration and passive buzzer alerts.

The SD Card Module functions as the storage for machine learning components and behavioral activities. Powering the circuit are two 3.7V 500mAh LiPo

batteries connected in parallel, managed by a TP4056 charging module for rechargeability. However, the charging module alone cannot power the MCU due to voltage differences. To resolve this, the researcher incorporated a U3V12F9 Step-up voltage regulator alongside the charging module, ensuring the MCU receives a consistent 9V output voltage, meeting its power requirements.

The Generic Attribute Profile (GATT) was used for Bluetooth connection, comprising services and characteristics distinguished by Universally Unique Identifiers (UUIDs). Sensor values were transmitted to the mobile app in real-time using the notify property. Before receiving the model via BLE, a 32-bit CRC32 was sent from the app to detect transmission errors, achieved by appending a checksum to the data. If the checksums did not match upon data reception, indicating corruption, the app notifies the user to resend the model. This error detection method efficiently manages wearable memory by comparing smaller data sets, minimizing memory consumption.

Conclusion

The Feedforward Neural Network (FNN) demonstrates proficiency in predicting the behavioral patterns associated with BFRB in individuals. This capability contributes significantly to the effectiveness of reducing BFRB tendencies over time. Moreover, the inclusion of additional parameters such as distance and temperature enhance the performance of the FNN. By adding these parameters, the predictive accuracy and overall efficacy of the FNN in identifying and addressing BFRB behaviors are further improved.

The wearable device's Bluetooth Low Energy (BLE) has a maximum distance connectivity of 67 feet, and when the microcontroller utilizes BLE and IMU sensors, it consumes 4mA of power. Additionally, the distance and temperature sensors each consume 2mA. During real-time operation with TensorFlow Lite, power consumption increases significantly to around 20mA. However, components like the buzzer and

micro vibration motor require minimal power, as they are only activated during classification alongside TensorFlow Lite. Overall, by carefully managing the power consumption of all components, including BLE, microcontroller, sensors, TensorFlow Lite operations, and additional components, the total battery consumption of the device is approximately 30mA.

The BLE transmission speed for machine learning model files was measured at 380 bytes per second. With an average model file size of 70 kilobytes, it would take approximately 3 minutes and 8 seconds to transfer these files over BLE. This estimation offers valuable understanding of the time needed for file transmission to the wearable device.

The multivariate regression analysis demonstrates diverse responses to the BFRB detection device. While some benefited with reduced behaviors, others showed an increase. This underscores the need for personalized approaches in utilizing such technology to address BFRBs effectively.

The Ordinary Least Squares (OLS) model demonstrated a highly significant predictive capability, with an F-statistic of 240.8 and a p-value of 3.33E-31, indicating statistical significance well below the conventional alpha threshold of 0.05. Additionally, The R-squared value of 0.976 indicates that 97.6% of the variance in respondents' performance is explained by the model, suggesting a strong association between time and performance as measured by the BFRB device. These results affirm the robustness of the regression model, indicating that changes in performance over time are reliably predicted by the model, rather than occurring randomly.

While the device's performance on avoiding anticipating behavior improved day by day, these changes were not statistically significant over a one-week period. This suggests that assessing the device's performance for just one week may not be enough, and extending the evaluation period for an individual is advisable. The use of

the Greenhouse-Geisser correction further supported this conclusion, even though assumptions of sphericity were violated.

Both the standard ANOVA and Welch's ANOVA tests, yielding F-statistics of 0.714 and 0.504 respectively, do not provide sufficient evidence to reject the null hypothesis. This suggests that there is no significant difference in detection accuracy between the Hair, Nail, and Skin BFRB categories when accounting for variance differences. Thus, the device's accuracy in detecting BFRB remains consistent across different behaviors, indicating comparable performance regardless of the behavior's location.

The total expenses for building the wearable device were PHP 6,251.00. Component prices were gathered from different online commerce platforms and electronic stores, both local and international. Compatibility issues arose, requiring thorough research, trial, and error to successfully assemble the device.

Recommendations

Considering the conclusions drawn above, the researchers provided the following recommendations to enhance the study:

1. The researchers recommend implementing other types of artificial neural networks to work with anticipatory behavior that can greatly reduce the memory allocation for the wearable device.
2. The researchers recommend extending the number of days for evaluation to see the significance of improvement using the wearable device.
3. The researchers recommend using alternative forms of feedback when classifying anticipatory behaviors for individuals with severe experiences, such as neuromuscular electrical stimulation, which may alter user movement.

REFERENCES

Closa, C. C., & Tambaoan, C. A. (2018). Development of a Remotely Monitored Health Status Wristband. Retrieved January 16, 2023, from Cavite State University - Main Campus Library.

Costanzo, S., & Flores, A. (2020). A non-contact integrated body-ambient temperature sensors platform to contrast COVID-19. Electronics (Switzerland), 9(10). https://doi.org/10.3390/electronics9101658

Cruz, F. R., Torres, J., Baltazar, J. M., Naungayan, K. P., Villaverde, J. F., & Linsangan, N. (2016). Body Position Detection Using Accelerometer Integration Human Belt with GPS Localization Interconnected through GSM. ASEAN Journal of Engineering Research, 5, 1 to 12. https://ejournals.ph/article.php?id=13386

Daskalos, A.-C.; Theodoropoulos, P.; Spandonidis, C.; Vordos, N. Wearable Device for Observation of Physical Activity with the Purpose of Patient Monitoring Due to COVID-19. Signals 2022, 3, 11-28. https://doi.org/10.3390/signals3010002

Deng, Z.; Guo, L.; Chen, X.; Wu, W. Smart Wearable Systems for Health Monitoring. Sensors 2023, 23, 2479. https://doi.org/10.3390/s23052479

Estrada, J. E., & Vea, L. A. (2020). Real-Time Human Sitting Position Recognition using Wireless Sensors. Proceedings of the 2020 2nd International Conference on Image, Video and Signal Processing, 133–137. https://doi.org/10.1145/3388818.3393714

Fridriksdottir, E., & Bonomi, A. G. (2020). Accelerometer-based human activity recognition for patient monitoring using a deep neural network. Sensors (Switzerland), 20(22), 1–13. https://doi.org/10.3390/s20226424

HabitAware. (2020). Keen. Habitaware. Retrieved December 12, 2023, from https://habitaware.com/.

Houghton, David C., Alexander, J. R., Bauer, C. C., & Woods, D. W. (2018). Body-focused repetitive behaviors: More prevalent than once thought? (Issue 2000, pp. 389–393). Psychiatry Research. https://doi.org/https://doi.org/10.1016/j.psychres.2018.10.002.

Hutabarat J. P., Ahmadi N., and Adiono T., "Human Activity Recognition Based on Wearable Devices and Feedforward Neural Networks," 2023 International Conference on Electrical Engineering and Informatics (ICEEI), Bandung, Indonesia, 2023, pp. 1-4, doi: 10.1109/ICEEI59426.2023.10346836.

Jalalifar, S., Kashizadeh, A., Mahmood, I., Belford, A., Drake, N., Razmjou, A., & Asadnia, M. (2022). A Smart Multi-Sensor Device to Detect Distress in Swimmers. Sensors, 22(3), 1–15. https://doi.org/10.3390/s22031059

Kok, M., Hol, J. D., & Schön, T. B. (2017). Using inertial sensors for position and orientation estimation. Foundations and Trends in Signal Processing, 11(1–2), 1–153. https://doi.org/10.1561/2000000094

Lee, D. K., & Lipner, S. R. (2022). The Potential of N-Acetylcysteine for Treatment of Trichotillomania, Excoriation Disorder, Onychophagia, and Onychotillomania: An Updated Literature Review. International Journal of Environmental Research and Public Health, 19(11). https://doi.org/10.3390/ijerph19116370

Leibinger KW, Murray E, Aschenbrenner S and Randerath J (2023) Short-term intervention complemented by wearable technology improves Trichotillomania – A naturalistic single-case report. Front. Psychol. 14:1071532. doi: 10.3389/fpsyg.2023.1071532

Liu, S., Zhang, J., Zhang, Y. et al. A wearable motion capture device able to detect dynamic motion of human limbs. Nat Commun 11, 5615 (2020). https://doi.org/10.1038/s41467-020-19424-2

Lu, L., Zhang, J., Xie, Y., Gao, F., Xu, S., Wu, X., & Ye, Z. (2020). Wearable health devices in health care: Narrative systematic review. JMIR MHealth and UHealth, 8(11). https://doi.org/10.2196/18907

Sabry F, Eltaras T, Labda W, Alzoubi K, Malluhi Q. Machine Learning for Healthcare Wearable Devices: The Big Picture. J Healthc Eng. 2022 Apr 18. doi: 10.1155/2022/4653923. PMID: 35480146; PMCID: PMC9038375.

Searle, B. L., Spathis, Di., Constantinides, M., Quercia, D., & Mascolo, C. (2021). Anticipatory Detection of Compulsive Body-focused Repetitive Behaviors with

Wearables. Proceedings of MobileHCI 2021 - ACM International Conference on Mobile Human-Computer Interaction: Mobile Apart, MobileTogether. https://doi.org/10.1145/3447526.3472061

Smitha Bhandari, M. (2020). Understanding Body-Focused Repetitive Behaviors. WebMD. https://www.webmd.com/mental-health/ss/slideshow-understanding-body-focused-repetitive-behavior

Son, J. J., Clucas, J. C., White, C., Krishnakumar, A., Vogelstein, J. T., Milham, M. P., & Klein, A. (2019). Thermal sensors improve wrist-worn position tracking. In npj Digital Medicine (Vol. 2, Issue 1). npj Digital Medicine. https://doi.org/10.1038/s41746-019-0092-2

Tanna R. & Vithalani C. (2023). IoT-based Health Monitoring of Sports Personnel through Wearables Using Machine Learning Technology. Philippine Journal of Science. ISSN 0031 – 7683

Wang, H.; Ru, B.; Miao, X.; Gao, Q.; Habib, M.; Liu, L.; Qiu, S. MEMS Devices-Based Hand Gesture Recognition via Wearable Computing. Micromachines 2023, 14, 947. https://doi.org/10.3390/mi14050947

APPENDICES

Appendix 1

Result Computations

Computations and Formulas for Mean of the Percentage Change

Respondent 1

Improvements from day to day making the first day the initial/reference day.

$$Day\ 2\text{:}\ \left(\frac{56-60}{56}\right)*100 = -7.142857143\%$$

$$Day\ 3\text{:}\ \left(\frac{60-60}{60}\right)*100 = 0.00\%$$

$$Day\ 4\text{:}\ \left(\frac{60-51}{60}\right)*100 = 15.00\%$$

$$Day\ 5\text{:}\ \left(\frac{51-45}{51}\right)*100 = 11.7647058824\%$$

$$Day\ 6\text{:}\ \left(\frac{45-40}{45}\right)*100 = 11.1111111111\%$$

$$Day\ 7\text{:}\ \left(\frac{40-34}{40}\right)*100 = 15.00\%$$

Calculating the Mean

$$\frac{-7.14+0.00+15.00+11.76+11.11+15}{6} = 7.62\%$$

$$\boldsymbol{Improvement\ over\ 7\ days} = \mathbf{7.62}\%$$

Respondent 2

Improvements from day to day making the first day the initial/reference day.

$$Day\ 2\text{:}\left(\frac{80-89}{80}\right)*100=\ 11.25\%$$

$$Day\ 3\text{:}\left(\frac{89-88}{89}\right)*100=1.12359551\%$$

$$Day\ 4\text{:}\left(\frac{88-95}{88}\right)*100=\ -7.954545\%$$

$$Day\ 5\text{:}\left(\frac{95-101}{95}\right)*100=-6.31578947\%$$

$$Day\ 6\text{:}\left(\frac{101-97}{101}\right)*100=3.96039604\%$$

$$Day\ 7\text{:}\left(\frac{97-98}{97}\right)*100=-1.03092784\%$$

Calculating the Mean

$$\frac{11.25+1.12-7.95-6.32+3.96-1.03}{6}=\ -3.58\%$$

$$\boldsymbol{Improvement\ over\ 7\ days}\ =\ \mathbf{-3.58\%}$$

Respondent 3

Improvements from day to day making the first day the initial/reference day.

$$Day\ 2\text{:}\ \left(\frac{68-77}{68}\right)*100 = -13.2352941\%$$

$$Day\ 3\text{:}\ \left(\frac{77-80}{77}\right)*100 = -3.8961039\%$$

$$Day\ 4\text{:}\ \left(\frac{80-89}{80}\right)*100 = -11.25\%$$

$$Day\ 5\text{:}\ \left(\frac{89-70}{89}\right)*100 = 21.3483146\%$$

$$Day\ 6\text{:}\ \left(\frac{70-82}{70}\right)*100 = -17.1428571\%$$

$$Day\ 7\text{:}\ \left(\frac{82-87}{82}\right)*100 = -6.09756098\%$$

Calculating the Mean

$$\frac{-13.24-3.90-11.25+21.35-17.14-6.10}{6} = -5.05\%$$

$$\boldsymbol{Improvement\ over\ 7\ days} = \mathbf{-5.05}\%$$

Respondent 4

Improvements from day to day making the first day the initial/reference day.

$$Day\ 2\text{:}\left(\frac{15-11}{15}\right)*100=\ 26.6666667\%$$

$$Day\ 3\text{:}\left(\frac{11-8}{11}\right)*100=27.2727273\%$$

$$Day\ 4\text{:}\left(\frac{8-12}{8}\right)*100=-50.00\%$$

$$Day\ 5\text{:}\left(\frac{12-13}{12}\right)*100=-8.33333333\%$$

$$Day\ 6\text{:}\left(\frac{13-9}{13}\right)*100=30.7692308\%$$

$$Day\ 7\text{:}\left(\frac{9-6}{9}\right)*100=33.33333333\%$$

Calculating the Mean

$$\frac{26.67+27.27-50.00-8.33+30.77+33.33}{6}=\ 9.95\%$$

$$\boldsymbol{Improvement\ over\ 7\ days\ =\ 9.95\%}$$

Respondent 5

Improvements from day to day making the first day the initial/reference day.

$$Day\ 2\text{:}\left(\frac{10-7}{10}\right)*100=\ 30.00\%$$

$$Day\ 3\text{:}\left(\frac{7-5}{7}\right)*100=28.5714286\%$$

$$Day\ 4\text{:}\left(\frac{5-6}{5}\right)*100=-20.00\%$$

$$Day\ 5\text{:}\left(\frac{6-8}{6}\right)*100=-33.3333333\%$$

$$Day\ 6\text{:}\left(\frac{8-5}{8}\right)*100=\ 37.50\%$$

$$Day\ 7\text{:}\left(\frac{5-6}{5}\right)*100=-20.00\%$$

Calculating the Mean

$$\frac{30.00+28.57-20.00-33.33+37.50-20.00}{6}=\ 3.79\%$$

$$\boldsymbol{Improvement\ over\ 7\ days}\ =\ \mathbf{3.79}\%$$

Respondent 6

Improvements from day to day making the first day the initial/reference day.

$$Day\ 2\text{: } \left(\frac{5-7}{5}\right) * 100 = -40.00\%$$

$$Day\ 3\text{: } \left(\frac{7-5}{7}\right) * 100 = 28.5714286\%$$

$$Day\ 4\text{: } \left(\frac{5-8}{5}\right) * 100 = -60.00\%$$

$$Day\ 5\text{: } \left(\frac{8-4}{8}\right) * 100 = 50.00\%$$

$$Day\ 6\text{: } \left(\frac{4-3}{4}\right) * 100 = 25.00\%$$

$$Day\ 7\text{: } \left(\frac{3-2}{3}\right) * 100 = 33.33333333\%$$

Calculating the Mean

$$\frac{-40.00 + 28.57 - 60.00 + 50.00 + 25.00 + 33.33}{6} = 6.15\%$$

$$\boldsymbol{Improvement\ over\ 7\ days} = \mathbf{6.15}\%$$

Respondent 7

Improvements from day to day making the first day the initial/reference day.

$$Day\ 2\text{:}\ \left(\frac{10-12}{10}\right)*100 = -20.00\%$$

$$Day\ 3\text{:}\ \left(\frac{12-10}{12}\right)*100 = 16.6666667\%$$

$$Day\ 4\text{:}\ \left(\frac{10-4}{10}\right)*100 = 60.00\%$$

$$Day\ 5\text{:}\ \left(\frac{4-7}{4}\right)*100 = -75.00\%$$

$$Day\ 6\text{:}\ \left(\frac{7-5}{7}\right)*100 = 28.5714286\%$$

$$Day\ 7\text{:}\ \left(\frac{5-5}{5}\right)*100 = 0.00\%$$

Calculating the Mean

$$\frac{-20.00 + 16.67 + 60.00 - 75.00 + 28.57 + 0.00}{6} = 1.71\%$$

$$\boldsymbol{Improvement\ over\ 7\ days} = \mathbf{1.71}\%$$

Wearable Device Battery Consumption

BLE and IMU sensors = $4mA$

Distance sensor = $2mA$

Temperature sensor = $2mA$

TensorFlow Lite operation = $20mA$

buzzer and micro vibration motor = $1.6mA$

$$sum = 4mA + 2mA + 2mA + 20mA + 1.6mA = 29.6mA$$

$$\boldsymbol{Battery\ Consumption\ =\ approx.\ 30mA}$$

Battery Longevity Calculation

$$Battery\ Life\ (HRS) = \frac{Battery\ Capacity\ (mAh)}{Load\ Current\ (mA)}$$

$$Battery\ Life = \frac{1000mAh}{30mA} = 33.333\ Hours$$

$$\boldsymbol{Estimated\ Battery\ Life\ =\ 33\ Hours\ 19\ minutes}$$

Calculations on Software Evaluation

Functionality

$$14+15+14+14+14+15+13+14+14+14+14+14+14+14+14+15+14+15+14+13+15+13+14+14+14+15+14+13+14+15+15=424$$

$$number\ of\ category = 3$$

$$Mean = \frac{424}{30} = 14.13$$

$$\boldsymbol{Weighted\ Mean = \frac{14.13}{3} = 4.711111111}$$

$$\boldsymbol{Standard\ Deviation} = \sqrt{\frac{(14-14.13)^2-(15-14.13)^2-(14-14.13)^2-\cdots}{30-1}} = \mathbf{0.63}$$

Reliability

$$9+10+10+10+8+9+10+9+10+9+10+8+10+9+8+9+9+10+9+10+9+10+10+10+9+9+9+9+9=279$$

$$number\ of\ category = 2$$

$$Mean = \frac{279}{30} = 9.3$$

$$\boldsymbol{Weighted\ Mean = \frac{9.3}{2} = 4.65}$$

$$\boldsymbol{Standard\ Deviation} = \sqrt{\frac{(9-9.3)^2-(10-9.3)^2-(10-9.3)^2-\cdots}{30-1}} = \mathbf{0.65}$$

Efficiency

$$10+9+10+10+10+10+9+9+10+10+10+10+10+10+9+10+10$$
$$+8+9+8+10+10+9+9+9+9+10+10+9+9=285$$

$$number\ of\ category = 2$$

$$Mean = \frac{285}{30} = 9.5$$

$$\boldsymbol{Weighted\ Mean = \frac{9.5}{2} = 4.75}$$

$$\boldsymbol{Standard\ Deviation = \sqrt{\frac{(10-9.5)^2-(9-9.5)^2-(10-9.5)^2-\cdots}{30-1}} = 0.63}$$

Portability

$$9+9+10+9+9+10+9+10+10+9+10+9+9+9+10+10+9+10$$
$$+10+10+9+10+9+9+10+9+9+9+10+10=284$$

$$number\ of\ category = 2$$

$$Mean = \frac{284}{30} = 9.47$$

$$\boldsymbol{Weighted\ Mean = \frac{9.47}{2} = 4.73}$$

$$\boldsymbol{Standard\ Deviation = \sqrt{\frac{(9-9.47)^2-(9-9.47)^2-(10-9.47)^2-\cdots}{30-1}} = 0.51}$$

Usability

$$14 + 13 + 14 + 14 + 14 + 14 + 14 + 15 + 13 + 15 + 13 + 14 + 13 + 14 + 14 + 13 + 14 + 12 + 13 + 13 + 14 + 14 + 15 + 14 + 14 + 15 + 14 + 14 + 13 + 13 = 413$$

$$number\ of category = 3$$

$$Mean = \frac{413}{30} = 13.77$$

$$\boldsymbol{Weighted\ Mean = \frac{13.77}{3} = 4.59}$$

$$\boldsymbol{Standard\ Deviation}$$

$$= \sqrt{\frac{(\mathbf{14 - 13.77})^2 - (\mathbf{13 - 13.77})^2 - (\mathbf{14 - 13.77})^2 - \cdots}{\mathbf{30 - 1}}} = \mathbf{0.73}$$

Information in the Software

$$15 + 15 + 13 + 14 + 14 + 15 + 14 + 14 + 15 + 13 + 14 + 14 + 14 + 14 + 13 + 14 + 14 + 14 + 12 + 14 + 13 + 14 + 13 + 14 + 12 + 13 + 13 + 15 + 14 + 14 = 414$$

$$number\ of category = 3$$

$$Mean = \frac{414}{30} = 13.8$$

$$\boldsymbol{Weighted\ Mean = \frac{13.8}{3} = 4.6}$$

$$\boldsymbol{Standard\ Deviation}$$

$$= \sqrt{\frac{(\mathbf{15 - 13.8})^2 - (\mathbf{15 - 13.8})^2 - (\mathbf{13 - 13.8})^2 - \cdots}{\mathbf{30 - 1}}} = \mathbf{0.81}$$

Technical Aspects of the Software and Materials

$$8+9+10+10+9+9+10+9+9+9+10+10+10+10+8+10+9+10$$
$$+9+10+9+9+10+9+9+10+10+10+10+10=284$$

$$number\ of category = 2$$

$$Mean = \frac{284}{30} = 9.47$$

$$\boldsymbol{Weighted\ Mean = \frac{9.47}{2} = 4.73}$$

$$\boldsymbol{Standard\ Deviation = \sqrt{\frac{(8-9.47)^2-(9-9.47)^2-(10-9.47)^2-\cdots}{30-1}} = 0.62}$$

Performance

$$10+9+10+10+10+10+9+10+10+10+10+10+10+10+9+9$$
$$+9+9+9+9+10+9+8+10+8+9+10+10+9+9$$
$$=284$$

$$number\ of category = 2$$

$$Mean = \frac{284}{30} = 9.47$$

$$\boldsymbol{Weighted\ Mean = \frac{9.47}{2} = 4.73}$$

$$\boldsymbol{Standard\ Deviation = \sqrt{\frac{(8-9.47)^2-(9-9.47)^2-(10-9.47)^2-\cdots}{30-1}} = 0.62}$$

$$\boldsymbol{Composite\ Mean = 4.71+4.65+4.75+4.73+4.59+4.6+4.73+4.73}$$
$$\boldsymbol{=4.69}$$

Appendix 2

Statistical Tabulation

Appendix Table 1. One-way ANOVA Accuracy

					95% CONFIDENCE INTERVAL FOR MEAN	
	N	MEAN	STD. DEVIATION	STD. ERROR	LOWER BOUND	UPPER BOUND
Hair	10	95.00	8.49837	2.68742	88.9206	101.0794
Nail	10	98.00	4.21637	1.33333	94.9838	101.0162
Skin	10	96.00	5.16398	1.63299	92.3059	99.6941
Total	30	96.33	6.14948	1.12274	94.0371	98.6296

Appendix Table 2. Tests of Homogeneity of Variances

	LEVENE STATISTIC	DF1	DF2	SIG.
Based on Mean	4.326	2	27	0.023
Based on Median	0.600	2	27	0.556
Based on Median and with adjusted df	0.600	2	19.621	0.559
Based on trimmed mean	3.486	2	27	0.045

Appendix Table 3. Between and Within Groups of ANOVA test.

	SUM OF SQUARES	DF	MEAN SQUARE	F	SIG.
Between Groups	46.667	2	23.333	0.600	0.556
Within Groups	1050.000	27	38.889		
Total	1096.667	29			

Appendix Table 4. ANOVA effect sizes.

		95% CONFIDENCE INTERVAL	
	POINT ESTIMATE	LOWER	UPPER
Eta-squared	0.043	0.000	0.204
Epsilon-squared	-0.028	-0.074	0.145
Omega-squared Fixed-effect	-0.027	-0.071	0.141
Omega-squared Random effect	-0.014	-0.034	0.076

a. Eta-squared and Epsilon-squared are estimated based on the fixed-effect model.

b. Negative but less biased estimates are retained, not rounded to zero.

Appendix Table 5. Robust Tests of Equality of Means.

	STATISTIC[a]	DF1	DF2	SIG.
Welch	0.714	2	17.005	0.504
Brown-Forsythe	0.600	2	19.621	0.559

a. Asymptotically F distributed.

Appendix Table 6. Post Hoc Tests of Multiple Comparisons.

	CATEGORICAL (I)	CATEGORICAL (J)	MEAN DIFFERENCE (I-J)	STD. ERROR	SIG.
Tukey HSD	Hair	Nail	-3.00000	2.78887	0.537
		Skin	-1.00000	2.78887	0.932
	Nail	Hair	3.00000	2.78887	0.537
		Skin	2.00000	2.78887	0.756
	Skin	Hair	1.00000	2.78887	0.932
		Nail	-2.00000	2.78887	0.756
Games-Howell	Hair	Nail	-3.00000	3.00000	0.590
		Skin	-1.00000	3.14466	0.946
	Nail	Hair	3.00000	3.00000	0.590
		Skin	2.00000	2.10819	0.618
	Skin	Hair	1.00000	3.14466	0.946
		Nail	-2.00000	2.10819	0.618

Appendix Table 7. Post Hoc Tests of Multiple Comparisons Confidence Interval.

			95% CONFIDENCE INTERVAL	
	CATEGORICAL (I)	CATEGORICAL (J)	LOWER BOUND	UPPER BOUND
Tukey HSD	Hair	Nail	-9.9148	3.9148
		Skin	-7.9148	5.9148
	Nail	Hair	-3.9148	9.9148
		Skin	-4.9148	8.9148
	Skin	Hair	-5.9148	7.9148
		Nail	-8.9148	4.9148
Games-Howell	Hair	Nail	-10.9081	4.9081
		Skin	-9.1770	7.1770
	Nail	Hair	-4.9081	10.9081
		Skin	-3.3993	7.3993
	Skin	Hair	-7.1770	9.1770
		Nail	-7.3993	3.3993

Appendix Table 8. Homogeneous Subsets.

			SUBSET FOR ALPHA = 0.05
	CATEGORICAL	N	1
Tukey HSD[a]	Hair	10	95.0000
	Skin	10	96.0000
	Nail	10	98.0000
	Sig.		0.537

Means for groups in homogeneous subsets are displayed.

a. Uses Harmonic Mean Sample Size = 10.000.

Appendix Table 9. Descriptive Statistics of Repeated-Measure.

	MEAN	STD. DEVIATION	N
Day 1	34.8571	31.89790	7
Day 2	37.5714	36.35866	7
Day 3	36.5714	37.84995	7
Day 4	37.8571	40.36264	7
Day 5	35.4286	37.94231	7
Day 6	34.4286	39.94937	7
Day 7	34.0000	41.50904	7

Appendix Table 10. Multivariate Tests.

EFFECT		VALUE	F	HYPOTHESIS DF	ERROR DF	SIG.
Buzz	Pillai's Trace	0.968	4.993[b]	6.000	1.000	0.330
	Wilks' Lambda	0.032	4.993[b]	6.000	1.000	0.330
	Hotelling's Trace	29.956	4.993[b]	6.000	1.000	0.330
	Roy's Largest Root	29.956	4.993[b]	6.000	1.000	0.330

EFFECT		PARTIAL ETA SQUARED
Buzz	Pillai's Trace	0.968
	Wilks' Lambda	0.968
	Hotelling's Trace	0.968
	Roy's Largest Root	0.968

a. Design: Intercept

Within Subjects Design: Buzz

b. Exact statistic

Appendix Table 11. Mauchly's Test of Sphericity.

WITHIN SUBJECTS EFFECT	MAUCHLY'S W	APPROX. CHISQUARE	DF	SIG.	EPSILON[B] GREENHOUSE GEISSER
Buzz	0.000	48.844	20	0.002	0.301

	EPSILON[b]	
Within Subjects Effect	Huynh-Feldt	Lower-bound
Buzz	0.423	0.167

Tests the null hypothesis that the error covariance matrix of the orthonormalized transformed dependent variables is proportional to an identity matrix.

a. Design: Intercept

   Within Subjects Design: Buzz

b. May be used to adjust the degrees of freedom for the averaged tests of significance. Corrected tests are displayed in the Tests of Within-Subjects Effects table.

Appendix Table 12. Tests of Within-Subjects Effects.

SOURCE		TYPE III SUM OF SQUARES	DF	MEAN SQUARE	F	SIG.
Buzz	Sphericity Assumed	98.776	6	16.463	0.431	0.854
	Greenhouse-Geisser	98.776	1.807	54.668	0.431	0.641
	Huynh-Feldt	98.776	2.539	38.899	0.431	0.703
	Lower-bound	98.776	1.000	98.776	0.431	0.536
Error (Buzz)	Sphericity Assumed	1376.367	36	38.232		
	Greenhouse-Geisser	1376.367	10.841	126.960		
	Huynh-Feldt	1376.367	15.236	90.338		
	Lower-bound	1376.367	6.000	229.395		

Appendix Table 13. Tests of Within-Subjects Contrasts.

SOURCE		TYPE III SUM OF SQUARES	DF	MEAN SQUARE	F	SIG.
Buzz	Linear	25.000	1	25.000	0.160	0.703
	Quadratic	44.633	1	44.633	3.636	0.105
	Cubic	13.714	1	13.714	0.568	0.480
	Order 4	0.134	1	0.134	0.037	0.853
	Order 5	3.000	1	3.000	0.240	0.642
	Order 6	12.295	1	12.295	0.600	0.468
Error (Buzz)	Linear	938.286	6	156.381		
	Quadratic	73.653	6	12.276		
	Cubic	144.952	6	24.159		
	Order 4	21.399	6	3.566		
	Order 5	75.048	6	12.508		
	Order 6	123.030	6	20.505		

Appendix Table 14. Tests of Between-Subjects Effects.

SOURCE	TYPE III SUM OF SQUARES	DF	MEAN SQUARE	F	SIG.	PARTIAL ETA SQUARED
Intercept	62857.653	1	62857.653	6.330	0.046	0.513
Error	59582.204	6	9930.367			

Appendix Table 15. Estimated Marginal Means.

			95% CONFIDENCE INTERVAL	
BUZZ	MEAN	STD. ERROR	LOWER BOUND	UPPER BOUND
1	34.857	12.056	5.357	64.358
2	37.571	13.742	3.945	71.198
3	36.571	14.306	1.566	71.577
4	37.857	15.256	0.528	75.186
5	35.429	14.341	0.338	70.519
6	34.429	15.099	-2.518	71.376
7	34.000	15.689	-4.389	72.389

Appendix Table 16. Pairwise Comparisons.

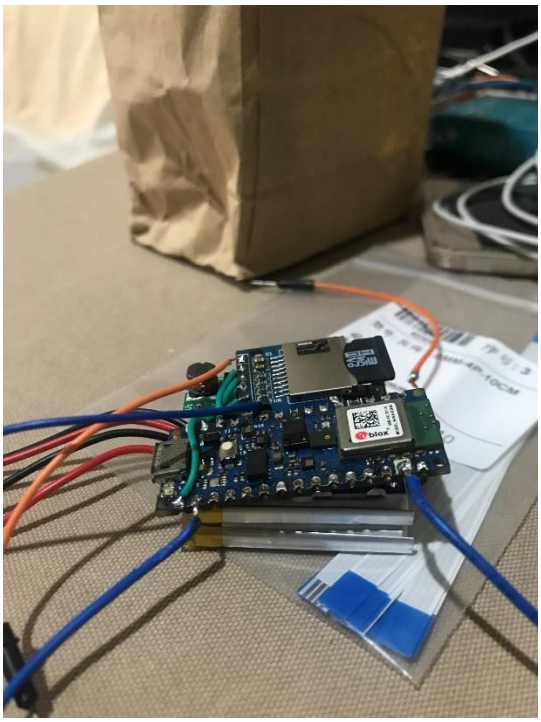
					95% CONFIDENCE INTERVAL FOR DIFFERENCE[A]	
BUZZ (I)	**BUZZ (J)**	**MEAN DIFFERENCE (I-J)**	**STD. ERROR**	**SIG.[A]**	**LOWER BOUND**	**UPPER BOUND**
1	2	-2.714	1.948	1.000	-12.514	7.085
	3	-1.714	2.570	1.000	-14.641	11.213
	4	-3.000	4.077	1.000	-23.504	17.504
	5	-0.571	3.722	1.000	-19.290	18.147
	6	0.429	4.445	1.000	-21.927	22.784
	7	0.857	5.552	1.000	-27.070	28.785
2	1	2.714	1.948	1.000	-7.085	12.514
	3	1.000	0.756	1.000	-2.802	4.802
	4	-0.286	2.792	1.000	-14.329	13.758
	5	2.143	3.181	1.000	-13.854	18.140
	6	3.143	3.426	1.000	-14.087	20.373
	7	3.571	4.545	1.000	-19.290	26.433
3	1	1.714	2.570	1.000	-11.213	14.641
	2	-1.000	0.756	1.000	-4.802	2.802
	4	-1.286	2.495	1.000	-13.834	11.263
	5	1.143	3.548	1.000	-16.705	18.991
	6	2.143	3.391	1.000	-14.911	19.197
	7	2.571	4.412	1.000	-19.622	24.765
4	1	3.000	4.077	1.000	-17.504	23.504
	2	0.286	2.792	1.000	-13.758	14.329
	3	1.286	2.495	1.000	-11.263	13.834
	5	2.429	3.169	1.000	-13.509	18.366
	6	3.429	1.744	1.000	-5.342	12.199
	7	3.857	2.539	1.000	-8.916	16.630
5	1	0.571	3.722	1.000	-18.147	19.290
	2	-2.143	3.181	1.000	-18.140	13.854
	3	-1.143	3.548	1.000	-18.991	16.705
	4	-2.429	3.169	1.000	-18.366	13.509
	6	1.000	2.225	1.000	-10.193	12.193
	7	1.429	3.330	1.000	-15.320	18.177
6	1	-0.429	4.445	1.000	-22.784	21.927
	2	-3.143	3.426	1.000	20.373	14.087
	3	-2.143	3.391	1.000	-19.197	14.911
	4	-3.429	1.744	1.000	-12.199	5.342
	5	-1.000	2.225	1.000	-12.193	10.193
	7	-0.429	1.307	1.000	-6.144	7.001
7	1	-0.857	5.552	1.000	-28.785	27.070
	2	-3.571	4.545	1.000	-26.433	19.290
	3	-2.571	4.412	1.000	-24.765	19.622
	4	-3.857	2.539	1.000	-16.630	8.916
	5	-1.429	3.330	1.000	-18.177	15.320
	6	-0.429	1.307	1.000	-7.001	6.144

Appendix Table 17. Software Evaluation Data.

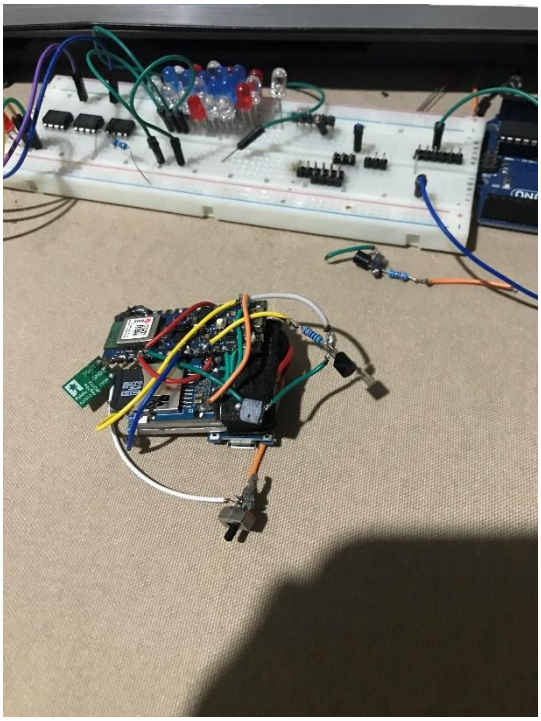
RESP.	FUNC.	RELIA.	EFFIC.	PORT.	USAB.	INFOR.	TECH.	PERF.
1	14	9	10	9	14	15	8	10
2	15	10	9	9	13	15	9	9
3	14	10	10	10	14	13	10	10
4	14	10	10	9	14	14	10	10
5	14	8	10	9	14	14	9	10
6	15	9	10	10	14	15	9	10
7	13	10	9	9	14	14	10	9
8	14	9	9	10	15	14	9	10
9	14	10	10	10	13	15	9	10
10	14	9	10	9	15	13	9	10
11	14	10	10	10	13	14	10	10
12	14	8	10	9	14	14	10	10
13	14	10	10	9	13	14	10	10
14	14	9	10	9	14	14	10	10
15	15	8	9	10	14	13	8	9
16	14	9	10	10	13	14	10	9
17	15	9	10	9	14	14	9	9
18	14	10	8	10	12	14	10	9
19	13	9	9	10	13	12	9	9
20	15	10	8	10	13	14	10	9
21	13	9	10	9	14	13	9	10
22	14	10	10	10	14	14	9	9
23	14	10	9	9	15	13	10	8
24	14	10	9	9	14	14	9	10
25	15	9	9	10	14	12	9	8
26	14	9	9	9	15	13	10	9
27	13	9	10	9	14	13	10	10
28	14	9	10	9	14	15	10	10
29	15	9	9	10	13	14	10	9
30	15	9	9	10	13	14	10	9

Appendix 3

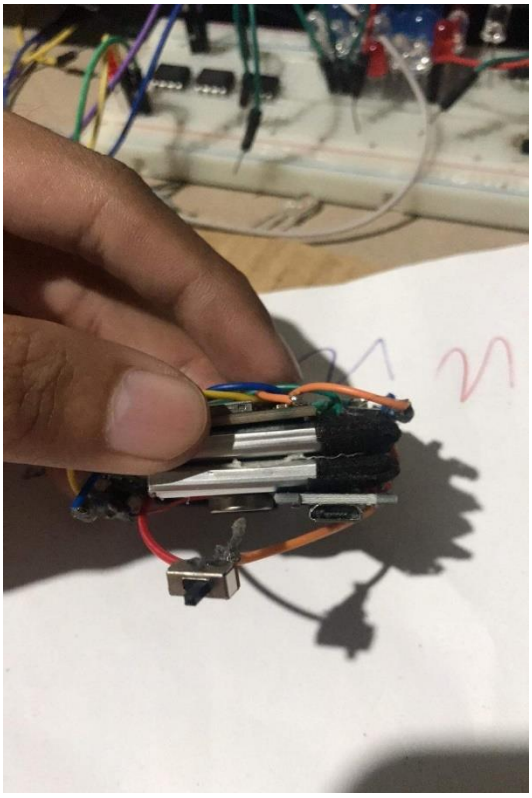
Photo Documentation



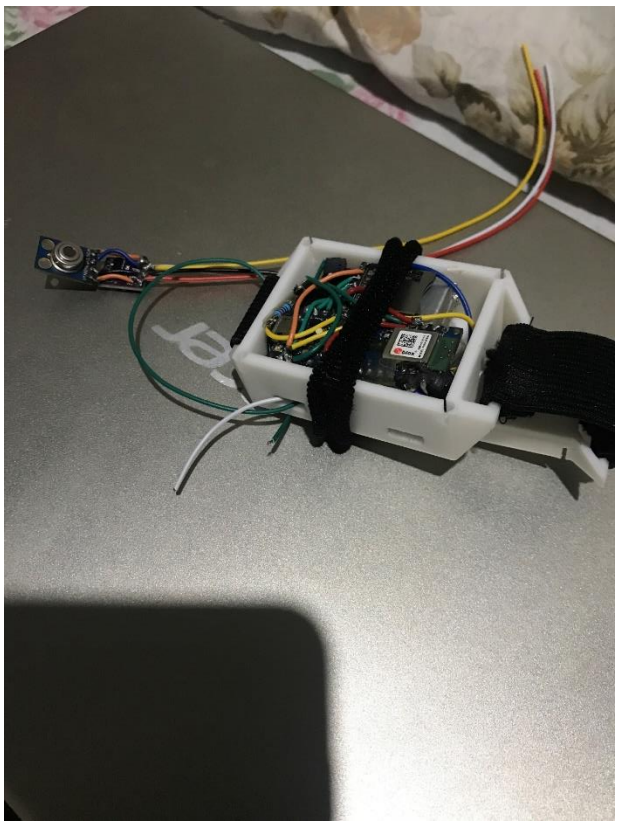
Appendix Figure 1. Planning for Development of Device.



Appendix Figure 2. Connecting the pins of Wearable Device.

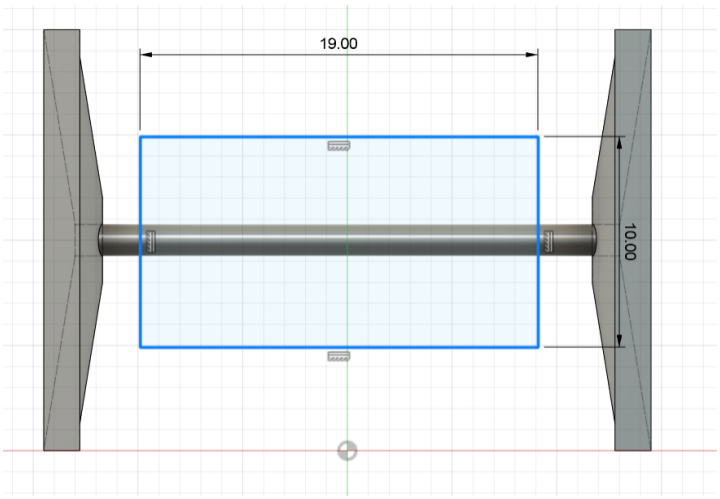


Appendix Figure 3. Measuring the Dimension of the Device.

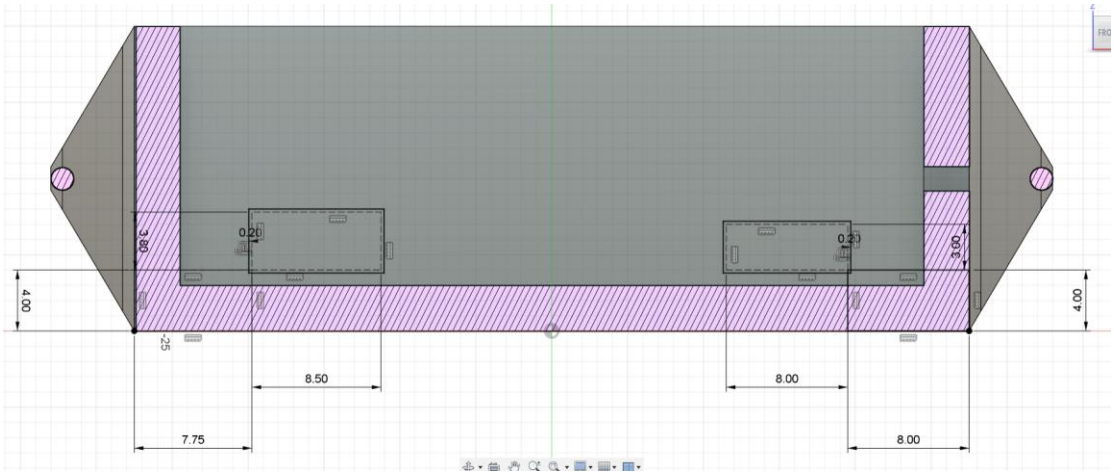


Appendix Figure 4. Assembling the Wearable Device.

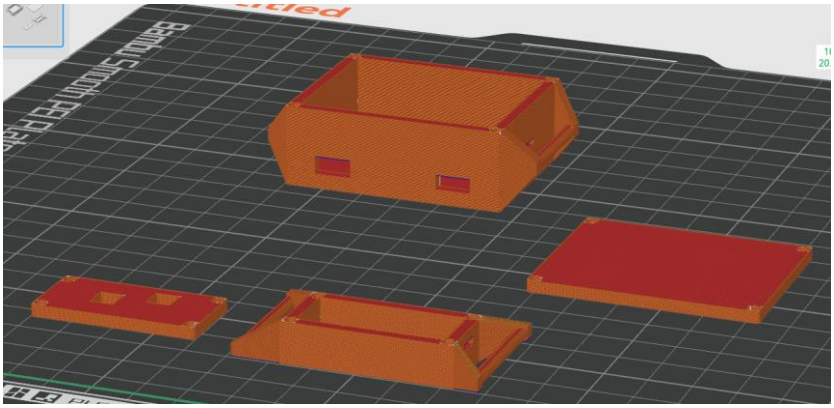
Appendix Figure 5. Mounting the Components in the Chassis.



Appendix Figure 6. Design of the Strap mount.



Appendix Figure 7. Design of the Main Chassis.



Appendix Figure 8. Preparing Chassis for 3D printing.

Appendix Figure 9. Printing the Chassis.

Appendix Figure 10. Printed Main and External Chassis.

Appendix Figure 11. Training in the ML Model for Device Evaluation.

Appendix Figure 12. Gathering Data for Device Evaluation.

Appendix 4

Program Codes

Multivariate Multiple Linear Regression Program Code for Qualitative Field Testing.

watch.ipynb was used to calculate the multivariate multiple linear regression and ordinary least squares.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
%matplotlib inline

# Read data from the spreadsheet
data = pd.read_excel("field_test.xlsx")

# Multivariate Linear Regression
# Encoding respondent_id as categorical
data['Respondents'] = data['Respondents'].astype('category')
model_mlr = smf.ols('Buzz ~ Day + Respondents', data=data).fit()
```

OLS Regression Results

Dep. Variable:	Buzz	R-squared:	0.976
Model:	OLS	Adj. R-squared:	0.972
Method:	Least Squares	F-statistic:	240.8
Date:	Sun, 31 Mar 2024	Prob (F-statistic):	3.33e-31
Time:	14:30:52	Log-Likelihood:	-152.52
No. Observations:	49	AIC:	321.0
Df Residuals:	41	BIC:	336.2
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|
Intercept	50.8571	2.818	18.048	0.000
Respondents[T.2]	43.1429	3.179	13.572	0.000
Respondents[T.3]	29.5714	3.179	9.302	0.000
Respondents[T.4]	-38.8571	3.179	-12.223	0.000
Respondents[T.5]	-42.7143	3.179	-13.437	0.000
Respondents[T.6]	-44.5714	3.179	-14.021	0.000
Respondents[T.7]	-41.8571	3.179	-13.167	0.000
Day	-0.3571	0.425	-0.841	0.405

Omnibus:	3.920	Durbin-Watson:	1.165
Prob(Omnibus):	0.141	Jarque-Bera (JB):	2.810
Skew:	-0.472	Prob(JB):	0.245
Kurtosis:	3.696	Cond. No.	34.1

Wearable Device Code

wearable.ino

```
// This is part of how to transfer small (tens of
// kilobytes) files over BLE onto an Arduino Nano BLE Sense board. Most of this
// sketch is internal implementation details of the protocol, but if you just
// want to use it you can look at the bottom of this file.
// The API is that you call setupBLEFileTransfer() in your setup() function to
// open up communication with any clients that want to send you files, and then
// onBLEFileReceived() is called when a file has been downloaded.

#include <ArduinoBLE.h>
#include <Arduino_LSM9DS1.h>
#include "Adafruit_VL53L0X.h"
#include <Adafruit_MLX90614.h>

#include <TensorFlowLite.h>
#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>

#include <SdFat.h>
#include <ArduinoJson.h>

SdFat sd;
File modelFile;
Adafruit_VL53L0X lox = Adafruit_VL53L0X();
Adafruit_MLX90614 mlx = Adafruit_MLX90614();

bool isConnected = false;
bool isBLEBusy = false;

int buzzerPin = 8;
int buzzTime = 200; // buzzer frequency
unsigned long buzzStartTime = 0;
const unsigned long buzzDuration = 5000; // buzz and vibrate for 5 seconds
int vibrationPin = 6;

// global variables used for TensorFlow Lite (Micro)
// pull in all the TFLM ops, you can remove this line and
// only pull in the TFLM ops you need, if would like to reduce
// the compiled size of the sketch.
tflite::AllOpsResolver tflOpsResolver;

const tflite::Model* tflModel = nullptr;
tflite::MicroInterpreter* tflInterpreter = nullptr;
TfLiteTensor* tflInputTensor = nullptr;
```

```
TfLiteTensor* tflOutputTensor = nullptr;

// Create a static memory buffer for TFLM, the size may need to
// be adjusted based on the model you are using
constexpr int tensorArenaSize = 8 * 1024;
byte tensorArena[tensorArenaSize] __attribute__((aligned(16)));

// array to map gesture index to a name
const char* HOTSPOT[] = {
  "off_target",
  "on_target"
};
String previous_file;
String currentDate = "";
String currentUser = "";
double on_target_threshold = 0.8;

bool isModelInitialized = false;

#define NUM_HOTSPOT (sizeof(HOTSPOT) / sizeof(HOTSPOT[0]))

// Uncomment this macro to log received data to the serial UART.
#define ENABLE_LOGGING

// Forward declare the function that will be called when data has been delivered to us.
void onBLEFileReceived(uint8_t* file_data, int file_length);

namespace {

// Controls how large a file the board can receive.
constexpr int32_t file_maximum_byte_count = (120 * 1024);

// Macro based on a master UUID that can be modified for each characteristic.
#define FILE_TRANSFER_UUID(val) ("bf88b656-" val "-4a61-86e0-769c741026c0")

BLEService service(FILE_TRANSFER_UUID("0000"));

// How big each transfer block can be. In theory this could be up to 512 bytes, but
// in practice I've found that going over 128 affects reliability of the connection.
constexpr int32_t file_block_byte_count = 128;

// Where each data block is written to during the transfer.
BLECharacteristic file_block_characteristic(FILE_TRANSFER_UUID("3000"),
BLEWrite, file_block_byte_count);

// Write the expected total length of the file in bytes to this characteristic
// before sending the command to transfer a file.
```

```
BLECharacteristic file_length_characteristic(FILE_TRANSFER_UUID("3001"),
BLERead | BLEWrite, sizeof(uint32_t));

// Read-only attribute that defines how large a file the sketch can handle.
BLECharacteristic
file_maximum_length_characteristic(FILE_TRANSFER_UUID("3002"), BLERead,
sizeof(uint32_t));

// Write the checksum that you expect for the file here before you trigger the transfer.
BLECharacteristic file_checksum_characteristic(FILE_TRANSFER_UUID("3003"),
BLERead | BLEWrite, sizeof(uint32_t));

// Writing a command of 1 starts a file transfer (the length and checksum
characteristics should already have been set).
// A command of 2 tries to cancel any pending file transfers. All other commands are
undefined.
BLECharacteristic command_characteristic(FILE_TRANSFER_UUID("3004"),
BLEWrite, sizeof(uint32_t));

// A status set to 0 means a file transfer succeeded, 1 means there was an error, and
2 means a file transfer is
// in progress.
BLECharacteristic transfer_status_characteristic(FILE_TRANSFER_UUID("3005"),
BLERead | BLENotify, sizeof(uint32_t));

// Informative text describing the most recent error, for user interface purposes.
constexpr int32_t error_message_byte_count = 128;
BLECharacteristic error_message_characteristic(FILE_TRANSFER_UUID("3006"),
BLERead | BLENotify, error_message_byte_count);

// Data parameters
constexpr int32_t data_size_count = 32;
BLECharacteristic accelerometer_characteristic(FILE_TRANSFER_UUID("3007"),
BLERead | BLENotify, data_size_count);
BLECharacteristic gyroscope_characteristic(FILE_TRANSFER_UUID("3008"),
BLERead | BLENotify, data_size_count);
BLECharacteristic
distance_temperature_characteristic(FILE_TRANSFER_UUID("3009"), BLERead |
BLENotify, data_size_count);

BLECharacteristic file_update_characteristic(FILE_TRANSFER_UUID("3010"),
BLERead | BLENotify, file_block_byte_count);

// Internal globals used for transferring the file.
uint8_t file_buffers[file_maximum_byte_count];
int finished_file_buffer_index = -1;
uint8_t* finished_file_buffer = nullptr;
int32_t finished_file_buffer_byte_count = 0;
```

```
uint8_t* in_progress_file_buffer = nullptr;
int32_t in_progress_bytes_received = 0;
int32_t in_progress_bytes_expected = 0;
uint32_t in_progress_checksum = 0;

String file_name = "";

// Training notification
uint32_t on_training = 0;

void notifyError(const String& error_message) {
  constexpr int32_t error_status_code = 1;
  transfer_status_characteristic.writeValue(error_status_code);

  const char* error_message_bytes = error_message.c_str();
  uint8_t error_message_buffer[error_message_byte_count];
  bool at_string_end = false;
  for (int i = 0; i < error_message_byte_count; ++i) {
    const bool at_last_byte = (i == (error_message_byte_count - 1));
    if (!at_string_end && !at_last_byte) {
      const char current_char = error_message_bytes[i];
      if (current_char == 0) {
        at_string_end = true;
      } else {
        error_message_buffer[i] = current_char;
      }
    }

    if (at_string_end || at_last_byte) {
      error_message_buffer[i] = 0;
    }
  }
  error_message_characteristic.writeValue(error_message_buffer,
error_message_byte_count);
}

// [send == true] : notifies app for model received
// [send == false] : notifies app for dashboard update received
void notifySuccess(bool send) {
  constexpr int32_t success_status_code = 0;
  constexpr int32_t without_message_status_code = 3;
  if (send) {
    transfer_status_characteristic.writeValue(success_status_code);
  } else {
    transfer_status_characteristic.writeValue(without_message_status_code);
  }
}
```

```
void notifyInProgress() {
  constexpr int32_t in_progress_status_code = 2;
  transfer_status_characteristic.writeValue(in_progress_status_code);
}

// See http://home.thep.lu.se/~bjorn/crc/ for more information on simple CRC32
calculations.
uint32_t crc32_for_byte(uint32_t r) {
  for (int j = 0; j < 8; ++j) {
    r = (r & 1? 0: (uint32_t)0xedb88320L) ^ r >> 1;
  }
  return r ^ (uint32_t)0xff000000L;
}

uint32_t crc32(const uint8_t* data, size_t data_length) {
  constexpr int table_size = 256;
  static uint32_t table[table_size];
  static bool is_table_initialized = false;
  if (!is_table_initialized) {
    for(size_t i = 0; i < table_size; ++i) {
      table[i] = crc32_for_byte(i);
    }
    is_table_initialized = true;
  }
  uint32_t crc = 0;
  for (size_t i = 0; i < data_length; ++i) {
    const uint8_t crc_low_byte = static_cast<uint8_t>(crc);
    const uint8_t data_byte = data[i];
    const uint8_t table_index = crc_low_byte ^ data_byte;
    crc = table[table_index] ^ (crc >> 8);
  }
  return crc;
}

void onFileTransferComplete() {
  uint32_t computed_checksum = crc32(in_progress_file_buffer,
in_progress_bytes_expected);;
  if (in_progress_checksum != computed_checksum) {
    notifyError(String("File transfer failed: Expected checksum 0x") +
String(in_progress_checksum, 16) +
      String(" but received 0x") + String(computed_checksum, 16));
    in_progress_file_buffer = nullptr;
    return;
  }

  if (finished_file_buffer_index == 0) {
    finished_file_buffer_index = 1;
  } else {
```

```
    finished_file_buffer_index = 0;
  }
  finished_file_buffer = &file_buffers[0];
  finished_file_buffer_byte_count = in_progress_bytes_expected;

  in_progress_file_buffer = nullptr;
  in_progress_bytes_received = 0;
  in_progress_bytes_expected = 0;

  onBLEFileReceived(finished_file_buffer, finished_file_buffer_byte_count);
  isBLEBusy = false;
}

void onFileBlockWritten(BLEDevice central, BLECharacteristic characteristic) {
  if (in_progress_file_buffer == nullptr) {
    notifyError("File block sent while no valid command is active");
    isBLEBusy = false;
    return;
  }
  isBLEBusy = true;
  const int32_t file_block_length = characteristic.valueLength();
  if (file_block_length > file_block_byte_count) {
    notifyError(String("Too many bytes in block: Expected ") +
String(file_block_byte_count) +
      String(" but received ") + String(file_block_length));
    in_progress_file_buffer = nullptr;
    return;
  }

  const int32_t bytes_received_after_block = in_progress_bytes_received +
file_block_length;
  if ((bytes_received_after_block > in_progress_bytes_expected) ||
    (bytes_received_after_block > file_maximum_byte_count)) {
    notifyError(String("Too many bytes: Expected ") +
String(in_progress_bytes_expected) +
      String(" but received ") + String(bytes_received_after_block));
    in_progress_file_buffer = nullptr;
    return;
  }

  uint8_t* file_block_buffer = in_progress_file_buffer + in_progress_bytes_received;
  characteristic.readValue(file_block_buffer, file_block_length);

// Enable this macro to show the data in the serial log.
#ifdef ENABLE_LOGGING
  char string_buffer[file_block_byte_count + 1];
  for (int i = 0; i < file_block_byte_count; ++i) {
    unsigned char value = file_block_buffer[i];
```

```
    if (i < file_block_length) {
      string_buffer[i] = value;
    } else {
      string_buffer[i] = 0;
    }
  }
  string_buffer[file_block_byte_count] = 0;
#endif  // ENABLE_LOGGING

  if (bytes_received_after_block == in_progress_bytes_expected) {
    onFileTransferComplete();
  } else {
    in_progress_bytes_received = bytes_received_after_block;
  }
}

void startFileTransfer() {
  if (in_progress_file_buffer != nullptr) {
    notifyError("File transfer command received while previous transfer is still in
progress");
    return;
  }

  int32_t file_length_value;
  file_length_characteristic.readValue(file_length_value);
  if (file_length_value > file_maximum_byte_count) {
    notifyError(
      String("File too large: Maximum is ") + String(file_maximum_byte_count) +
      String(" bytes but request is ") + String(file_length_value) + String(" bytes"));
    return;
  }

  file_checksum_characteristic.readValue(in_progress_checksum);

  int in_progress_file_buffer_index;
  if (finished_file_buffer_index == 0) {
    in_progress_file_buffer_index = 1;
  } else {
    in_progress_file_buffer_index = 0;
  }

  in_progress_file_buffer = &file_buffers[0];
  in_progress_bytes_received = 0;
  in_progress_bytes_expected = file_length_value;

  notifyInProgress();
}
```

```
void cancelFileTransfer() {
  if (in_progress_file_buffer != nullptr) {
    notifyError("File transfer cancelled");
    in_progress_file_buffer = nullptr;
  }
}

void onCommandWritten(BLEDevice central, BLECharacteristic characteristic) {
  int32_t command_value;
  characteristic.readValue(command_value);

  if ((command_value != 1) && (command_value != 2)) {
    notifyError(String("Bad command value: Expected 1 or 2 but received ") +
String(command_value));
    return;
  }

  if (command_value == 1) {
    startFileTransfer();
  } else if (command_value == 2) {
    cancelFileTransfer();
  }
}

// Starts the BLE handling you need to support the file transfer.
void setupBLEFileTransfer() {
  // Start the core BLE engine.
  if (!BLE.begin()) {
    while (1);
  }
  String address = BLE.address();

  // Output BLE settings over Serial.
  address.toUpperCase();
  static String device_name = "BFRB Sense";

  // Set up properties for the whole service.
  BLE.setLocalName(device_name.c_str());
  BLE.setDeviceName(device_name.c_str());
  BLE.setAdvertisedService(service);

  // Add in the characteristics we'll be making available.
  file_block_characteristic.setEventHandler(BLEWritten, onFileBlockWritten);
  service.addCharacteristic(file_block_characteristic);

  service.addCharacteristic(file_length_characteristic);

  file_maximum_length_characteristic.writeValue(file_maximum_byte_count);
```

```
  service.addCharacteristic(file_maximum_length_characteristic);

  service.addCharacteristic(file_checksum_characteristic);

  command_characteristic.setEventHandler(BLEWritten, onCommandWritten);
  service.addCharacteristic(command_characteristic);

  service.addCharacteristic(transfer_status_characteristic);
  service.addCharacteristic(error_message_characteristic);

  service.addCharacteristic(accelerometer_characteristic);
  service.addCharacteristic(gyroscope_characteristic);
  service.addCharacteristic(distance_temperature_characteristic);

  service.addCharacteristic(file_update_characteristic);

  // Start up the service itself.
  BLE.addService(service);
  BLE.advertise();
}

// Called in your loop function to handle BLE housekeeping.
void updateBLEFileTransfer() {
  BLE.poll();
  BLEDevice central = BLE.central();
  static bool was_connected_last = false;
  if (central && !was_connected_last) {
    Serial.print("Connected to central: ");
    Serial.println(central.address());
  }
  was_connected_last = central;
  isConnected = central.connected();
}
}  // namespace

void buzzVibrate(bool enable) {
  if (enable) {
    buzzStartTime = millis();
    digitalWrite(vibrationPin, HIGH);
    while (millis() - buzzStartTime < buzzDuration) {
      digitalWrite(buzzerPin, HIGH);
      delayMicroseconds(buzzTime);
      digitalWrite(buzzerPin, LOW);
      delayMicroseconds(buzzTime);
    }
    digitalWrite(buzzerPin, LOW);
    digitalWrite(vibrationPin, LOW);
  } else {
```

```
    digitalWrite(buzzerPin, LOW);
    digitalWrite(vibrationPin, LOW);
  }
}

void setup() {
  // Start serial
  Serial.begin(9600);

  // BLE setup
  setupBLEFileTransfer();

  // SD card setup
  bool sdBegin = sd.begin(10);
  while (!sdBegin) {
    sdBegin = sd.begin(10);
    delay(500);
  }

  // LSM9DS1 setup
  bool imuInit = IMU.begin();
  while (!imuInit) {
    imuInit = IMU.begin();
    delay(500);
  }


  // VL53L0X setup
  bool loxInit = lox.begin();
  while (!loxInit) {
    loxInit = lox.begin();
    delay(500);
  }
  lox.startRangeContinuous();

  // MLX90614 setup
  bool mlxInit = mlx.begin();
  while (!mlxInit) {
    mlxInit = mlx.begin();
    delay(500);
  }
  pinMode(buzzerPin, OUTPUT);
  pinMode(vibrationPin, OUTPUT);

  previous_file = getPreviousFile();
}

char *dtostrf (double val, signed char width, unsigned char prec, char *sout) {
```

```
  char fmt[9];
  sprintf(fmt, "%%%d.%df", width, prec);
  sprintf(sout, fmt, val);
  return sout;
}

char *IMU_read (float x, float y, float z, char *sout) {
  char Xreadings[9]; // buffer
  dtostrf(x, 0, 3, Xreadings);
  char Yreadings[9];
  dtostrf(y, 0, 3, Yreadings);
  char Zreadings[9];
  dtostrf(z, 0, 3, Zreadings);
  sprintf(sout, "%s,%s,%s", Xreadings, Yreadings, Zreadings);
  return sout;
}

bool isNumeric(String str) {
    unsigned int stringLength = str.length();
    if (stringLength == 0) {
        return false;
    }
    bool seenDecimal = false;
    for(unsigned int i=0; i<stringLength; ++i) {
        if (isDigit(str.charAt(i))) {
            continue;
        }
        if (str.charAt(i) == '.') {
            if (seenDecimal) {
                return false;
            }
            seenDecimal = true;
            continue;
        }
        return false;
    }
    return true;
}

void initializeTFL(uint8_t model[]){
  // get the TFL representation of the model byte array
  tflModel = tflite::GetModel(model);
  if (tflModel->version() != TFLITE_SCHEMA_VERSION) {
    while (1);
  }
  // Create an interpreter to run the model
  tflInterpreter = new tflite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena,
tensorArenaSize);
```

```
  // Allocate memory for the model's input and output tensors
  tflInterpreter->AllocateTensors();

  // Get pointers for the model's input and output tensors
  tflInputTensor = tflInterpreter->input(0);
  tflOutputTensor = tflInterpreter->output(0);
  isModelInitialized = true;
}

// void runClassification(float aX, float aY, float aZ, float gX, float gY, float gZ, uint16_t
dist, double temp) {
// return null if one of the parameters is nan or null
void runClassification(float aX, float aY, float aZ, float gX, float gY, float gZ) {
  // normalize the IMU data between 0 to 1 and store in the model's
  // input tensor
  tflInputTensor->data.f[0] = (aX + 4.0) / 8.0;
  tflInputTensor->data.f[1] = (aY + 4.0) / 8.0;
  tflInputTensor->data.f[2] = (aZ + 4.0) / 8.0;
  tflInputTensor->data.f[3] = (gX + 2000.0) / 4000.0;
  tflInputTensor->data.f[4] = (gY + 2000.0) / 4000.0;
  tflInputTensor->data.f[5] = (gZ + 2000.0) / 4000.0;


  // Run inferencing
  TfLiteStatus invokeStatus = tflInterpreter->Invoke();
  if (invokeStatus != kTfLiteOk) {
    while (1);
    return;
  }

  double onTargetPredictValue = tflOutputTensor->data.f[1];
  if (onTargetPredictValue > on_target_threshold) {
      buzzVibrate(true);
      // saveBuzz(currentDate, currentUser);
      delay(1000);
  } else {
      buzzVibrate(false);
  }
}

String getPreviousFile() {
  modelFile = sd.open("info.h");
  String file = "";
    if (modelFile) {
      while (modelFile.available()) {
        uint8_t readByte = modelFile.read();
        file += (char) readByte;
```

```
    }
   }
   modelFile.close();
   return file;
}

void setPreviousFile(String file_name) {
 modelFile = sd.open("info.h", FILE_WRITE | O_TRUNC);
 if (modelFile) modelFile.print(file_name);
 modelFile.close();
}

void saveModel(String file_name, uint8_t* model) {
 modelFile = sd.open(file_name + ".h", FILE_WRITE);
 if (modelFile) modelFile.print((char*)model);
 modelFile.close();
}

void initOldModel(String fileName) {
 if (sd.exists(fileName)) {
  modelFile = sd.open(fileName);
 } else {
  return;
 }

 if (modelFile) {
  // Convert each value separated with spaces to int using the String.toInt()
  // then assign it to the file_buffers index
  String code = "";
  uint32_t file_length = modelFile.size();
  uint32_t new_size = 0;

  for (uint32_t i=0; i<file_length; i++) {
   uint8_t readByte = modelFile.read();
   code += (char)readByte;
   // 32 = space in ASCII code
   if (readByte == 32 || i == file_length-1) {
    code.trim();
    if (isNumeric(code)) {
     file_buffers[new_size] = code.toInt();
     new_size++;
    } else {
     file_name = code;
    }
    code = "";
   }
  }
  modelFile.close();
```

```
    // Assign 0 to remaining indexes (considered as padding, does not affect the
model)
    for (size_t i=new_size; i<file_maximum_byte_count; i++) {
      file_buffers[i] = 0;
    }
  }
}

void saveBuzz(String targetDatetime, String filename) {
  // String targetDatetime = "02/25/2024";
  JsonDocument jsonDocument;
  File dashboardFile = sd.open(filename + ".json", FILE_READ);

  if (dashboardFile) {
    DeserializationError error = deserializeJson(jsonDocument, dashboardFile);
    dashboardFile.close();
    if (error) {
      return;
    }
  } else {
    return;
  }

  dashboardFile = sd.open(filename + ".json", FILE_WRITE | O_TRUNC);
  if (dashboardFile) {
    // Check if the datetime already exists in the "data" array
    bool datetimeExists = false;

    // Iterate through the "data" array
    JsonArray dataArray = jsonDocument["data"];
    for (JsonObject obj : dataArray) {
      String datetime = obj["datetime"];
      if (datetime.equals(targetDatetime)) {
        // Datetime exists, increment the "buzz" value
        obj["buzz"] = obj["buzz"].as<int>() + 1;
        serializeJson(jsonDocument, dashboardFile);
        dashboardFile.close();
        datetimeExists = true;
        break;
      }
    }

    // If the datetime does not exist, add a new object
    if (!datetimeExists) {
      JsonObject newObj = jsonDocument["data"].createNestedObject();
      newObj["datetime"] = targetDatetime;
      newObj["buzz"] = 1;
```

```
      serializeJson(jsonDocument, dashboardFile);
      dashboardFile.close();
    }
  } else {
    return;
  }
}

// Function to send data in chunks
void writeUpdatedChunks(const String& data) {
  int dataSize = data.length();
  int chunkSize = 64;

  for (int i = 0; i < dataSize; i += chunkSize) {
    String chunk = data.substring(i, min(i + chunkSize, dataSize));
    const char* chunkCStr = chunk.c_str();
    file_update_characteristic.writeValue(chunkCStr);
    isBLEBusy = true;
    delay(100);  // Adjust delay as needed based on your requirements
  }
  isBLEBusy = false;
}

void sendData(String filename) {
  File dataFile = sd.open(filename + ".json");
  JsonDocument jsonDoc;
  DeserializationError error = deserializeJson(jsonDoc, dataFile);
  // Check for parsing errors
  if (error) {
  } else {
    // Get the last two "data" array from the JSON document
    JsonArray data = jsonDoc["data"];
    JsonDocument newObj;
    int idx = 1;
    for (int i=data.size()-1; i>=data.size()-2; i--) {
      JsonObject obj = data[i];
      const char* datetime = obj["datetime"];
      int buzz = obj["buzz"].as<int>();
      newObj[idx]["datetime"] = datetime;
      newObj[idx]["buzz"] = buzz;
      idx--;
    }

    // Send the string through BLE
    String dataContents;
    serializeJson(newObj, dataContents);
    writeUpdatedChunks(dataContents);
    dataFile.close();
```

```
  }
}

void sendAllData(String filename) {
  const int bufferSize = 64;
  char buffer[bufferSize];

  File dataFile = sd.open(filename + ".json");
  if (dataFile) {
    bool isReading = true;
    while (isReading) {
      int bytesRead = dataFile.read(buffer, bufferSize);
      if (bytesRead > 0) {
        String dataChunk = "";
        for (int i = 0; i < bytesRead; i++) {
          dataChunk += (char)buffer[i];
        }
        const char* chunkCStr = dataChunk.c_str();
        file_update_characteristic.writeValue(chunkCStr);
        isBLEBusy = true;
        delay(100);
      } else {
        isReading = false;
        dataFile.close();
      }
    }
    isBLEBusy = false;
  } else {
    return;
  }
}

void onBLEFileReceived(uint8_t* file_data, int file_length) {
  // Do something here with the file data that you've received. The memory itself will
  // remain untouched until after a following onFileReceived call has completed, and
  // the BLE module retains ownership of it, so you don't need to deallocate it.

  // xupdaterequestx-<currentUser>-<request>-<currentDate>-
  //
  // UPDATE REQUEST
  // all
  //  - sends all dashboard data
  // last
  //  - sends the last two data (yesterday & today)
  String xupdaterequestx = "";
  for (uint32_t i=0; i<50; i++) {
    uint8_t dataByte = file_data[i];
    if (dataByte == 0) {
```

```
    break;
  } else {
    xupdaterequestx += (char)dataByte;
  }
}

int index = xupdaterequestx.indexOf("xupdaterequestx");
if (index != -1) {
  // Parse code to "requests"
  String requests[4];
  String code = "";
  int arraySize  = 0;
  for (int i=0; i<xupdaterequestx.length(); i++) {
    char currentChar = xupdaterequestx.charAt(i);
    if (currentChar != '-') {
      code += xupdaterequestx.charAt(i);
    } else {
      if (arraySize < sizeof(requests) / sizeof(requests[0])) {
        requests[arraySize] = code;
        arraySize++;
        code = "";
      }
    }
  }

  currentUser = requests[1];
  currentDate = requests[3];
  // Send dashboard data using BLE
  if (requests[2].equals("all")) {
    sendAllData(currentUser);
  } else if (requests[2].equals("last")) {
    sendData(currentUser);
  }

  // Initialize TFL after successful connection
  initOldModel(previous_file);
  initializeTFL(file_buffers);
  delay(100);
  notifySuccess(false);
} else {
  String code = "";
  uint32_t new_size = 0;

  for (uint32_t i=0; i<file_length; i++) {
    uint8_t dataByte = file_data[i];
    code += (char)dataByte;
    // 32 = space in ASCII code
    if (dataByte == 32 || i == file_length-1) {
```

```
        code.trim();
        if (isNumeric(code)) {
          file_buffers[new_size] = code.toInt();
          new_size++;
        } else {
          file_name = code;
        }
        code = "";
      }
    }

    // Assign 0 to remaining indexes (considered as padding, does not affect the
model)
    for (size_t i=new_size; i<file_maximum_byte_count; i++) {
      file_buffers[i] = 0;
    }

    saveModel(file_name, file_buffers);
    setPreviousFile(file_name);
    initializeTFL(file_buffers);
    delay(100);
    notifySuccess(true);
  }
}

void loop() {
  updateBLEFileTransfer(); // Keep the BLE service open
  if (isBLEBusy) return;

  // Accelerometer & Gyroscope read values
  float aX, aY, aZ, gX, gY, gZ;
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable() &&
lox.isRangeComplete()) {
    IMU.readAcceleration(aX, aY, aZ);
    char accReadings[data_size_count];
    IMU_read(aX, aY, aZ, accReadings);

    IMU.readGyroscope(gX, gY, gZ);
    char gyroReadings[data_size_count];
    IMU_read(gX, gY, gZ, gyroReadings);

    // VL53L0X & MLX90614 read values
    uint16_t lox_read = lox.readRange();
    double mlx_read = mlx.readObjectTempC();

    // Check if sensor readings are valid
    if (!isnan(lox_read) && !isnan(mlx_read)) {
      String distance = String(lox_read);
```

```
      String temperature = String(mlx_read);

      if (isConnected) {
        String distance_temperature = distance + "," + temperature;
        const char* distance_temperature_cstr = distance_temperature.c_str();
        distance_temperature_characteristic.writeValue(distance_temperature_cstr);
        accelerometer_characteristic.writeValue(accReadings);
        gyroscope_characteristic.writeValue(gyroReadings);
      }
      delay(100); // adds 0.1s for the mobile app to keep up - ( for smooth plotting )
    }
  }
  // TFLite classification
  if (isModelInitialized && !currentDate.equals("") && !currentUser.equals("")) {
    runClassification(aX, aY, aZ, gX, gY, gZ);
  }
  delay(10);
}
```

Mobile Application Code

dashboard.dart

```
part of 'page_manager.dart';

class Dashboard extends StatefulWidget {
  final String? restorationId;
  final BluetoothBuilder? ble;
  const Dashboard({super.key, this.restorationId, this.ble});

  @override
  State<Dashboard> createState() => _DashboardState();
}

class _DashboardState extends State<Dashboard> with RestorationMixin {
  List<DashboardChartData> ydata = [];
  List<List<DashboardChartData>> weeksYData = [];
  List<int> buzzValues = [];
  double? buzzMean = 0.0;
  double? meanImprovement = 0.0;
  BluetoothBuilder? ble;
  StreamSubscription? isReceivingControllerStream;
  String plotOption = "weeks";
  int weekIndex = 0;
  int userSelectedDate = 0;
  bool hasData = false;
  var rng = Random();

  late TrackballBehavior _trackballBehavior;
  List<String> monthNames = [
    'January',
    'February',
    'March',
    'April',
    'May',
    'June',
    'July',
    'August',
    'September',
    'October',
    'November',
    'December'
  ];

  @override
  void initState() {
    _trackballBehavior = TrackballBehavior(
```

```
    // Enables the trackball
    enable: true,
    // tooltipSettings: const InteractiveTooltip(enable: true, color: Colors.red),
    builder: (BuildContext context, TrackballDetails trackballDetails) {
      return Container(
        height: 50,
        width: 150,
        decoration: const BoxDecoration(
          color: Color.fromRGBO(0, 8, 22, 0.75),
          borderRadius: BorderRadius.all(Radius.circular(6.0)),
        ),
        child: Row(
          children: [
            Center(
                child: Container(
                    padding: const EdgeInsets.only(top: 11, left: 7),
                    height: 40,
                    width: 100,
                    child: Text(
                        plotOption == 'month'
                            ? '${monthNames[_selectedDate.value.month - 1]}
${trackballDetails.point!.x.toString()} : Buzz: ${trackballDetails.point!.y.toString()}'
                            : 'Buzz: ${trackballDetails.point!.y.toString()}',
                      style: const TextStyle(fontSize: 13, color: Color.fromRGBO(255, 255,
255, 1)))))
          ],
        ),
      );
    },
  );

  ble = widget.ble;
  listenReceiving();
  userSelectedDate = getDateInt(DateTime.now());
  callUpdateDashboard(userSelectedDate, plotOption);
  double x = DateTime(
    int.parse(userSelectedDate.toString().substring(0, 4)),
    int.parse(userSelectedDate.toString().substring(4, 6)) + 1,
    0,
  ).day.toDouble();
  super.initState();
 }

 int getDateInt(dynamic input) {
  if (input.runtimeType == DateTime) {
    String formattedMonth = input.month.toString().padLeft(2, '0');
    String formattedDay = input.day.toString().padLeft(2, '0');
    String formattedDate = "${input.year}$formattedMonth$formattedDay";
```

```
    return int.parse(formattedDate);
  } else if (input.runtimeType == String) {
    return int.parse(input);
  } else {
    return -1;
  }
}

void callUpdateDashboard(int date, String plotOption) async {
  bool val = await updateDashboard(date, plotOption);
  setState(() {
    hasData = val;
  });
}

@override
void dispose() {
  isReceivingControllerStream!.cancel();
  isReceivingControllerStream = null;
  super.dispose();
}

@override
String? get restorationId => widget.restorationId;

final RestorableDateTime _selectedDate = RestorableDateTime(DateTime.now());
late final RestorableRouteFuture<DateTime?> _restorableDatePickerRouteFuture =
    RestorableRouteFuture<DateTime?>(
  onComplete: _selectDate,
  onPresent: (NavigatorState navigator, Object? arguments) {
    return navigator.restorablePush(
      _datePickerRoute,
      arguments: _selectedDate.value.millisecondsSinceEpoch,
    );
  },
);

@pragma('vm:entry-point')
static Route<DateTime> _datePickerRoute(
  BuildContext context,
  Object? arguments,
) {
  return DialogRoute<DateTime>(
    context: context,
    builder: (BuildContext context) {
      return DatePickerDialog(
        restorationId: 'date_picker_dialog',
        initialEntryMode: DatePickerEntryMode.calendarOnly,
```

```
          initialDate: DateTime.fromMillisecondsSinceEpoch(arguments! as int),
          firstDate: DateTime(2024),
          lastDate: DateTime(2100),
        );
      },
    );
  }

  @override
  void restoreState(RestorationBucket? oldBucket, bool initialRestore) {
    registerForRestoration(_selectedDate, 'selected_date');
    registerForRestoration(_restorableDatePickerRouteFuture,
'date_picker_route_future');
  }

  void _selectDate(DateTime? newSelectedDate) async {
    if (newSelectedDate != null) {
      _selectedDate.value = newSelectedDate;
      int year = _selectedDate.value.year;
      String month = _selectedDate.value.month.toString().padLeft(2, '0');
      String day = _selectedDate.value.day.toString().padLeft(2, '0');
      String formattedDate = '$year$month$day';
      userSelectedDate = getDateInt(formattedDate);

      bool val = await updateDashboard(userSelectedDate, plotOption);
      setState(() {
        hasData = val;
        if (val) {
          ScaffoldMessenger.of(context).showSnackBar(SnackBar(
            content: Text('Selected: $month/$day/$year'),
          ));
        } else {
          ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
            content: Text('Empty data.'),
          ));
        }
      });
    }
  }

  void listenReceiving() {
    isReceivingControllerStream =
ble!.isReceivingController.stream.asBroadcastStream().listen((bool value) {
      // value = [String callbackMessage, double sendingProgress ,int statusCode]
      Provider.of<ConnectionProvider>(context, listen: false).setReceiving = value;
    });
  }
```

```
  void requestUpdate(String request) async {
    if (ble != null && ble!.isConnected && !ble!.isFileTransferInProgress) {
      setState(() {
        ble!.dashboardData = "";
        ble!.isReceivingController.add(true);
      });
      var user = await UserSecureStorage.getUser();
      String username = user['username'];

      // Get current date
      DateTime currentDate = DateTime.now();
      String formattedMonth = currentDate.month.toString().padLeft(2, '0');
      String formattedDay = currentDate.day.toString().padLeft(2, '0');
      String formattedDate = "${currentDate.year}$formattedMonth$formattedDay";

      var fileContents = utf8.encode('xupdaterequestx-$username-$request-
$formattedDate-') as Uint8List;
      ble?.transferFile(fileContents);
    }
  }

  List<List<DashboardChartData>> monthToPerWeek(List<DashboardChartData>
data) {
    List<List<DashboardChartData>> perWeek = [];
    List week = [];

    for (var val in data) {
      week.add(val);
      if (week.length == 7 || val == data.last) {
        perWeek.add(List.from(week));
        week.clear();
      }
    }

    if (perWeek.last.length < 4) {
      List y = perWeek.removeLast();
      for (var val in y) {
        perWeek.last.add(val);
      }
      return perWeek;
    }
    return perWeek;
  }

  void generateNullYs() {
    double maxMonthDay = DateTime(_selectedDate.value.year,
_selectedDate.value.month + 1, 0).day.toDouble();
```

```
    List<DashboardChartData> nullYValues = [];
    for (int day = 0; day < maxMonthDay; day++) {
      DashboardChartData chartData = DashboardChartData(day + 1, null);
      nullYValues.add(chartData);
    }
    setState(() {
      weeksYData.clear();
      weeksYData = monthToPerWeek(nullYValues);
      buzzMean = null;
      meanImprovement = null;
    });
  }

  Future<bool> updateDashboard(int selectedDate, String plotOption) async {
    String dir = await AppStorage.getDir();
    List<io.FileSystemEntity> dataFile = io.Directory(dir).listSync();
    // String fileRead = io.File("$dir/ryan.json").readAsStringSync();

    Map<String, dynamic> jsonData = json.decode(fileRead);
    List jsonDataList = jsonData["data"];
    List<Map<String, dynamic>> dataList = jsonDataList.cast<Map<String,
dynamic>>();

    int selectedMonth = selectedDate ~/ 100;
    int selectedDay = selectedDate % 100;

    if (plotOption == 'weeks') {
      List<DashboardChartData> monthData = [];

      for (Map<String, dynamic> item in dataList) {
        int datetime = int.parse(item["datetime"]);
        int yearMonth = datetime ~/ 100;
        int day = datetime % 100;
        if (selectedMonth == yearMonth) {
          DashboardChartData chartData = DashboardChartData(day, item["buzz"]);
          monthData.add(chartData);
        }
      }
      if (monthData.isNotEmpty) {
        setState(() {
          weeksYData = monthToPerWeek(monthData);
        });
        // find date in selected month
        for (int i = 0; i < weeksYData.length; i++) {
          for (int j = 0; j < weeksYData[i].length; j++) {
            if (weeksYData[i][j].x == selectedDay) {
              setState(() {
                weekIndex = i;
```

```
            });
            break;
          }
        }
      }
      updateImprovementAndAverage();
      return true;
    } else {
      generateNullYs();
      return false;
    }
  } else if (plotOption == 'month') {
    setState(() {
      ydata.clear();
      for (Map<String, dynamic> item in dataList) {
        int datetime = int.parse(item["datetime"]);
        int yearMonth = datetime ~/ 100;
        int day = datetime % 100;
        if (selectedMonth == yearMonth) {
          DashboardChartData chartData = DashboardChartData(day, item["buzz"]);
          ydata.add(chartData);
        }
      }

      // Improvement
      buzzValues.clear();
      buzzValues = ydata.map((data) => data.y!).toList();
      List<double> improvements = calculateImprovement(buzzValues);
      meanImprovement = calculateMean(improvements);

      // Average Buzz
      List<double> buzzValuesDouble = buzzValues.map((intNumber) =>
intNumber.toDouble()).toList();
      buzzMean = calculateMean(buzzValuesDouble);
    });
    if (ydata.isNotEmpty) {
      return true;
    } else {
      setState(() {
        buzzMean = null;
        meanImprovement = null;
      });
    }
  }
  return false;
}

void updateImprovementAndAverage() {
```

```
    // Improvement
    buzzValues.clear();
    buzzValues = weeksYData[weekIndex].map((data) => data.y!).toList();
    List<double> improvements = calculateImprovement(buzzValues);
    meanImprovement = calculateMean(improvements);

    // Average Buzz
    List<double> buzzValuesDouble = buzzValues.map((intNumber) =>
intNumber.toDouble()).toList();
    buzzMean = calculateMean(buzzValuesDouble);
  }

  Widget getIconBasedOnNumber(double? number) {
    if (number == null) {
      return const Icon(Icons.drag_handle, size: 35);
    }
    return number < 0
        ? const Icon(Icons.keyboard_double_arrow_down, color: Color.fromARGB(255,
193, 59, 49), size: 35)
        : number > 0
            ? const Icon(Icons.keyboard_double_arrow_up, color: Color.fromARGB(255,
42, 163, 50), size: 35)
            : const Icon(Icons.drag_handle, size: 35);
  }

  int getPlotOptionValue(String plotOption, int maxMonthDay) {
    if (plotOption == 'weeks') {
      return 1;
    } else if (plotOption == 'month') {
      if (maxMonthDay == 28) {
        return 3;
      } else if (maxMonthDay == 29) {
        return 4;
      } else if (maxMonthDay == 30) {
        return 5;
      } else if (maxMonthDay == 31) {
        return 5;
      }
    }
    return 0;
  }

  @override
  Widget build(BuildContext context) {
    bool isReceiving = Provider.of<ConnectionProvider>(context, listen:
true).isReceiving;
    double maxMonthDay = DateTime(_selectedDate.value.year,
_selectedDate.value.month + 1, 0).day.toDouble();
```

```
    double cardHeight = 220.0;

    return Stack(
      children: <Widget>[
        Container(
          decoration: BoxDecoration(
            gradient: LinearGradient(
                colors: [
                  Theme.of(context).colorScheme.secondaryContainer,
                  Theme.of(context).colorScheme.tertiaryContainer,
                ],
                begin: Alignment.topLeft,
                end: Alignment.bottomRight,
                stops: const [0.20, 0.45]),
          ),
          height: MediaQuery.of(context).size.height * .40,
          width: double.infinity,
          child: Container(
            margin: const EdgeInsets.all(10.0),
            child: ListTile(
              contentPadding: EdgeInsets.zero,
              title: Text(
                'Dashboard',
                textAlign: TextAlign.left,
                style: GoogleFonts.bebasNeue(fontSize: 30),
              ),
              subtitle: Text(
                plotOption == 'month'
                    ? '${monthNames[_selectedDate.value.month - 1]}
${_selectedDate.value.year}'
                    : '${monthNames[_selectedDate.value.month - 1]}
${_selectedDate.value.year} - week ${weekIndex + 1}',
                textAlign: TextAlign.left,
                style: GoogleFonts.bebasNeue(fontSize: 20),
              ),
              trailing: isReceiving
                  ? const CircularProgressIndicator()
                  : PopupMenuButton(
                      itemBuilder: (BuildContext context) => <PopupMenuEntry<String>>[
                        const PopupMenuItem<String>(
                          value: 'profile',
                          child: Text("View profile"),
                        ),
                        PopupMenuItem(
                          child: Text("View as: ${plotOption == 'month' ? 'weeks' : 'month'}"),
                          onTap: () async {
                            setState(() {
                              if (plotOption == 'month') {
```

```
                    plotOption = 'weeks';
                  } else if (plotOption == 'weeks') {
                    plotOption = 'month';
                  }
                 });
                 bool val = await updateDashboard(userSelectedDate, plotOption);
                 setState(() {
                  hasData = val;
                 });
                },
               ),
               PopupMenuItem(
                child: const Text("Update"),
                onTap: () {
                 requestUpdate("all");
                },
               ),
               PopupMenuItem(
                child: const Text("Refresh"),
                onTap: () async {
                 bool val = await updateDashboard(userSelectedDate, plotOption);
                 setState(() {
                  hasData = val;
                 });
                },
               ),
              ],
              child: const CircleAvatar(
               backgroundImage: AssetImage('images/ekusuuuu-
calibaaaaaaaaaaaaa.png'),
              ),
              onSelected: (String value) {
               if (value == 'profile') {
                Navigator.pushNamed(context, '/profile');
               }
               // Handle other menu items if needed
              },
             ),
           ),
          ),
         ),
         Positioned(
          top: MediaQuery.of(context).size.height * .24,
          bottom: 0,
          left: 0,
          right: 0,
          child: Container(
           decoration: BoxDecoration(
```

```
            color: Theme.of(context).colorScheme.background,
            borderRadius:
               const BorderRadius.only(topLeft: Radius.circular(45), topRight:
Radius.circular(45)),
          ),
        ),
      ),
      Container(
        alignment: Alignment.topCenter,
       padding: EdgeInsets.only(top: MediaQuery.of(context).size.height * .12, right:
20.0, left: 20.0),
        child: Column(
          children: [
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                SizedBox(
                  height: 200.0,
                  width: MediaQuery.of(context).size.width * 0.86,
                  child: GestureDetector(
                    onHorizontalDragEnd: plotOption == 'weeks' && hasData
                        ? (DragEndDetails details) {
                            if (details.primaryVelocity! > 0) {
                              // Swiped to the right
                              if (weekIndex > 0) {
                                setState(() {
                                  weekIndex -= 1;
                                  updateImprovementAndAverage();
                                });
                              }
                            } else if (details.primaryVelocity! < 0) {
                              // Swiped to the left
                              // weeksYData.length = 5
                              if (weekIndex < weeksYData.length - 1) {
                                setState(() {
                                  weekIndex += 1;
                                  updateImprovementAndAverage();
                                });
                              }
                            }
                          }
                        : null,
                    child: Card(
                      shadowColor: Theme.of(context).colorScheme.shadow,
                      elevation: 5.0,
                      child: SfCartesianChart(
                        trackballBehavior: _trackballBehavior,
                        plotAreaBorderWidth: 0,
```

```
                    primaryXAxis: NumericAxis(
                      isVisible: true,
                      interval: getPlotOptionValue(plotOption,
maxMonthDay.toInt()).toDouble(),
                      labelStyle: GoogleFonts.bebasNeue(fontSize: 12),
                      maximum: plotOption == 'month' ? maxMonthDay : null,
                    ),
                    primaryYAxis: NumericAxis(
                      isVisible: false,
                    ),
                    backgroundColor:
Theme.of(context).colorScheme.primaryContainer,
                    series: <ChartSeries>[
                      // Renders line chart
                      AreaSeries<DashboardChartData, int>(
                        dataSource: plotOption == 'month' ? ydata :
weeksYData[weekIndex],
                        color: Theme.of(context).colorScheme.primary,
                        xValueMapper: (DashboardChartData data, _) => data.x,
                        yValueMapper: (DashboardChartData data, _) => data.y,
                      ),
                    ],
                  ),
                ),
              ),
            ),
          ],
        ),
        const SizedBox(height: 30),
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            SizedBox(
              height: cardHeight,
              width: MediaQuery.of(context).size.width * 0.4,
              child: Card(
                shape: const RoundedRectangleBorder(
                  borderRadius: BorderRadius.all(Radius.circular(30)),
                ),
                color: Theme.of(context).colorScheme.primaryContainer,
                shadowColor: Theme.of(context).colorScheme.shadow,
                elevation: 5.0,
                child: Column(
                  children: [
                    Align(
                      alignment: Alignment.topLeft,
                      child: Padding(
```

```
                    padding: EdgeInsets.only(top: cardHeight * .05, left: cardHeight *
.05),
                    child: getIconBasedOnNumber(meanImprovement),
                  ),
                ),
                Expanded(
                  child: FittedBox(
                    child: Container(
                      margin: const EdgeInsets.all(10.0),
                      child: Text(
                        meanImprovement != null ?
'${meanImprovement!.toStringAsFixed(2)}%' : '--',
                        textAlign: TextAlign.center,
                        style: GoogleFonts.bebasNeue(fontSize: 50),
                      ),
                    ),
                  ),
                ),
                Text(
                  'Improvement',
                  textAlign: TextAlign.left,
                  style: GoogleFonts.bebasNeue(fontSize: 20),
                ),
                const SizedBox(height: 20),
              ],
            ),
          ),
        ),
        SizedBox(
          height: cardHeight,
          width: MediaQuery.of(context).size.width * 0.4,
          child: Card(
            shape: const RoundedRectangleBorder(
              borderRadius: BorderRadius.all(Radius.circular(30)),
            ),
            color: Theme.of(context).colorScheme.primaryContainer,
            shadowColor: Theme.of(context).colorScheme.shadow,
            elevation: 5.0,
            child: Column(
              children: [
                Align(
                  alignment: Alignment.topLeft,
                  child: Padding(
                    padding: EdgeInsets.only(top: cardHeight * .05, left: cardHeight *
.05),
                    child: const Icon(
                      Icons.line_weight,
                      size: 35,
```

```
              ),
            ),
          ),
          Expanded(
            child: FittedBox(
              child: Container(
                margin: const EdgeInsets.all(10.0),
                child: Text(
                  buzzMean != null ? buzzMean!.toStringAsFixed(2) : '--',
                  textAlign: TextAlign.center,
                  style: GoogleFonts.bebasNeue(fontSize: 50),
                ),
              ),
            ),
          ),
          Text(
            'Average Buzz',
            textAlign: TextAlign.left,
            style: GoogleFonts.bebasNeue(fontSize: 20),
          ),
          const SizedBox(height: 20),
        ],
      ),
    ),
  ),
),
Expanded(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    crossAxisAlignment: CrossAxisAlignment.center,
    children: [
      const Expanded(
        child: SizedBox(height: 1),
      ),
      Container(
        margin: const EdgeInsets.only(left: 60, right: 60),
        child: ElevatedButton(
          style: ElevatedButton.styleFrom(
           backgroundColor: Theme.of(context).colorScheme.tertiaryContainer,
          ),
          onPressed: () {
            _restorableDatePickerRouteFuture.present();
          },
          child: Row(
            // mainAxisSize: MainAxisSize.min,
            children: <Widget>[
              const Icon(
```

```
                        Icons.calendar_month_outlined,
                        size: 35,
                      ),
                      const SizedBox(width: 18),
                      Expanded(
                        child: Text(
                          'Calendar',
                          textAlign: TextAlign.center,
                          style: GoogleFonts.bebasNeue(fontSize: 30),
                        ),
                      ),
                    ],
                  ),
                ),
              ),
              const Expanded(
                child: SizedBox(height: 1),
              ),
            ],
          ),
        ),
      ),
    ],
  );
 }
}

class DashboardChartData {
 DashboardChartData(this.x, this.y);
 final int x;
 final int? y;
}
```

monitoring_page.dart

```
// ignore_for_file: avoid_print, non_constant_identifier_names

part of 'page_manager.dart';

class MonitoringPage extends StatefulWidget {
 final Icon navBarIcon = const Icon(Icons.monitor_heart_outlined);
 final Icon navBarIconSelected = const Icon(Icons.monitor_heart);
 final String navBarTitle = 'Monitoring App';
 final BluetoothBuilder? ble;

 const MonitoringPage({super.key, this.ble});
```

```
  @override
  State<MonitoringPage> createState() => _MonitoringPageState();
}

class _MonitoringPageState extends State<MonitoringPage> {
  BluetoothBuilder? ble;
  bool isBuildingModel = false;
  String info = '>_';
  int infoCode = 0;

  Crc32 crc = Crc32();

  Timer? timer;
  Timer? onCaptureTimer;
  Timer? offCaptureTimer;
  Timer? loadingTextTimer;

  List<_ChartData>? chartAccData;
  List<_ChartData>? chartGyroData;
  late int count;
  ChartSeriesController? axAxisController;
  ChartSeriesController? ayAxisController;
  ChartSeriesController? azAxisController;

  ChartSeriesController? gxAxisController;
  ChartSeriesController? gyAxisController;
  ChartSeriesController? gzAxisController;

  int distance = 0;
  double temperature = 0.0;

  String? onTargetText;
  String? offTargetText;

  String? accData;
  String? gyroData;

  NeuralNetworkRequestBuild buildClass = NeuralNetworkRequestBuild();
  final TextEditingController _textController = TextEditingController();

  final List<String> header = ["ax", "ay", "az", "gx", "gy", "gz", "class"];
  List<List<String>> onData = [];
  List<List<String>> offData = [];

  void setConnected(fromContext, bool value) {
    Provider.of<ConnectionProvider>(fromContext, listen: false).setConnected =
value;
```

```
  }

  bool connectionValue(fromContext) {
    return Provider.of<ConnectionProvider>(fromContext, listen: false).isConnected;
  }

  // StreamSubscription? subscription;
  StreamSubscription? subscription;
  StreamSubscription? deviceState;
  StreamSubscription? readStream;
  StreamSubscription? updateStream;
  StreamSubscription? callbackControllerStream;

  @override
  void initState() {
    super.initState();
    ble = widget.ble;
    count = 49;
    chartAccData = <_ChartData>[];
    chartGyroData = <_ChartData>[];
    listenCallback();
  }

  void listenCallback() {
    callbackControllerStream =
ble!.callbackController.stream.asBroadcastStream().listen((List value) {
      // value = [String callbackMessage, double sendingProgress ,int statusCode]
      msg(value.first, value.last);
      Provider.of<CallbackProvider>(context, listen: false).inform(value.first, value.last);
    });
  }

  /// ### [statusCode]
  /// * -2 = Crash (pink-purple)
  /// * -1 = Error (red)
  /// * 1 = Warning (yellow)
  /// * 2 = Success (green)
  /// * 3 = Info (blue)
  void msg(String m, [int statusCode = 0]) {
    setState(() {
      info = m;
      infoCode = statusCode;
    });
  }

  /* ---------------------------------------------- */
  // EVENT LISTENERS
  void _readData(BluetoothCharacteristic? characteristic) {
```

```
  readStream = characteristic!.onValueReceived.listen((value) {
    List<int> readData = List.from(value);
    String parsedData = String.fromCharCodes(readData);

    if (readData.isNotEmpty && readData != []) {
      if (characteristic.uuid.toString() == ble!.ACC_DATA_UUID) {
        accData = parsedData;
      } else if (characteristic.uuid.toString() == ble!.GYRO_DATA_UUID) {
        gyroData = parsedData;
      } else if (characteristic.uuid.toString() == ble!.DIST_TEMP_DATA_UUID) {
        setState(() {
          List<double> parsedDistTemp = dataParse(parsedData);
          distance = parsedDistTemp[0].toInt();
          temperature = parsedDistTemp[1];
        });
      }
    }
  });
}

void _readUpdateData(BluetoothCharacteristic? characteristic) {
  updateStream = characteristic!.onValueReceived.listen((value) {
    List<int> readData = List.from(value);
    String parsedData = String.fromCharCodes(readData);
    if (readData.isNotEmpty && readData != []) {
      ble!.dashboardData += parsedData;
    }
  });
}

/// Returns the realtime Cartesian line chart.
SizedBox _buildLiveAccChart(context, {double height = 130, double? width}) {
  return SizedBox(
    height: height,
    width: width,
    child: ClipRRect(
      borderRadius: BorderRadius.circular(10),
      child: SfCartesianChart(
        title: ChartTitle(
          text: 'Accelerometer',
          textStyle: TextStyle(
            fontSize: 10,
            color: Theme.of(context).colorScheme.inverseSurface.withAlpha(125),
          ),
        ),
        backgroundColor: Theme.of(context).colorScheme.primaryContainer,
        plotAreaBorderWidth: 0,
        primaryXAxis: NumericAxis(
```

```
          isVisible: false,
        ),
        primaryYAxis: NumericAxis(
          minimum: -2,
          maximum: 2,
          isVisible: false,
        ),
        series: <SplineSeries<_ChartData, int>>[
          SplineSeries<_ChartData, int>(
            name: 'x',
            onRendererCreated: (ChartSeriesController controller) {
              axAxisController = controller;
            },
            dataSource: chartAccData!,
            color: connectionValue(context) ? CustomColor.lineXColor :
CustomColor.deadLineColor,
            xValueMapper: (_ChartData data, _) => data.t,
            yValueMapper: (_ChartData data, _) => data.x,
            animationDuration: 0,
          ),
          SplineSeries<_ChartData, int>(
            name: 'y',
            onRendererCreated: (ChartSeriesController controller) {
              ayAxisController = controller;
            },
            dataSource: chartAccData!,
            color: connectionValue(context) ? CustomColor.lineYColor :
CustomColor.deadLineColor,
            xValueMapper: (_ChartData data, _) => data.t,
            yValueMapper: (_ChartData data, _) => data.y,
            animationDuration: 0,
          ),
          SplineSeries<_ChartData, int>(
            name: 'z',
            onRendererCreated: (ChartSeriesController controller) {
              azAxisController = controller;
            },
            dataSource: chartAccData!,
            color: connectionValue(context) ? CustomColor.lineZColor :
CustomColor.deadLineColor,
            xValueMapper: (_ChartData data, _) => data.t,
            yValueMapper: (_ChartData data, _) => data.z,
            animationDuration: 0,
          )
        ],
      ),
    ),
  );
```

```
  }

  SizedBox _buildLiveGyroChart(context, {double height = 130, double? width}) {
    return SizedBox(
      height: height,
      width: width,
      child: ClipRRect(
        borderRadius: BorderRadius.circular(10),
        child: SfCartesianChart(
          title: ChartTitle(
            text: 'Gyroscope',
            textStyle: TextStyle(
              fontSize: 10,
              color: Theme.of(context).colorScheme.inverseSurface.withAlpha(125),
            ),
          ),
          backgroundColor: Theme.of(context).colorScheme.primaryContainer,
          plotAreaBorderWidth: 0,
          primaryXAxis: NumericAxis(
            isVisible: false,
          ),
          primaryYAxis: NumericAxis(
            minimum: -500,
            maximum: 500,
            isVisible: false,
          ),
          series: <SplineSeries<_ChartData, int>>[
            SplineSeries<_ChartData, int>(
              name: 'x',
              onRendererCreated: (ChartSeriesController controller) {
                gxAxisController = controller;
              },
              dataSource: chartGyroData!,
              color: connectionValue(context) ? CustomColor.lineXColor :
CustomColor.deadLineColor,
              xValueMapper: (_ChartData data, _) => data.t,
              yValueMapper: (_ChartData data, _) => data.x,
              animationDuration: 0,
            ),
            SplineSeries<_ChartData, int>(
              name: 'y',
              onRendererCreated: (ChartSeriesController controller) {
                gyAxisController = controller;
              },
              dataSource: chartGyroData!,
              color: connectionValue(context) ? CustomColor.lineYColor :
CustomColor.deadLineColor,
              xValueMapper: (_ChartData data, _) => data.t,
```

```
            yValueMapper: (_ChartData data, _) => data.y,
            animationDuration: 0,
          ),
          SplineSeries<_ChartData, int>(
            name: 'z',
            onRendererCreated: (ChartSeriesController controller) {
              gzAxisController = controller;
            },
            dataSource: chartGyroData!,
            color: connectionValue(context) ? CustomColor.lineZColor :
CustomColor.deadLineColor,
            xValueMapper: (_ChartData data, _) => data.t,
            yValueMapper: (_ChartData data, _) => data.z,
            animationDuration: 0,
          )
        ],
      ),
    ),
  );
 }

 void handshake() async {
  // await Future.delayed(const Duration(milliseconds: 5000));
  if (ble != null && ble!.isConnected && !ble!.isFileTransferInProgress) {
    var fileContents = utf8.encode('xupdaterequestx-ryan-last-') as Uint8List;
    ble?.transferFile(fileContents);
  } else {
  }
 }

 void _connectFromDevice() {
  ble!.connect();

  subscription = ble!.discoverController.stream.asBroadcastStream().listen(null);
  subscription!.onData((value) {
    if (value) {
      _readData(ble!.accDataCharacteristic);
      _readData(ble!.gyroDataCharacteristic);
      _readData(ble!.distTempDataCharacteristic);
      _readUpdateData(ble!.fileUpdateCharacteristic);
      timer = Timer.periodic(const Duration(milliseconds: 100), _updateDataSource);

      // Listen from sudden disconnection
      deviceState = ble!.device!.connectionState.listen((state) async {
        if (state == BluetoothConnectionState.disconnected) {
          _disconnectFromDevice();
        }
      });
```

```
      final mtuSubscription = ble!.device!.mtu.listen((int mtu) {
        // iOS: initial value is always 23, but iOS will quickly negotiate a higher value
        // android: you must request higher mtu yourself
        Provider.of<ConnectionProvider>(context, listen: false).setMTU(mtu);
      });

      setState(() {
        setConnected(context, true);
      });
    }
  });

  // subscription!.cancel();
}

void _disconnectFromDevice() {
  ble!.disconnect();
  deviceState!.cancel();
  timer!.cancel();
  setState(() {
    setConnected(context, false);
    subscription!.cancel();
    deviceState!.cancel();
    readStream!.cancel();
    updateStream!.cancel();
    callbackControllerStream!.cancel();
    timer!.cancel();

    subscription = null;
    deviceState = null;
    readStream = null;
    updateStream = null;
    callbackControllerStream = null;
    timer = null;
  });
}

void updateControllerDataSource(listData, controller, isEdge) {
  if (isEdge) {
    controller?.updateDataSource(
      addedDataIndexes: <int>[listData!.length - 1],
      removedDataIndexes: <int>[0],
    );
  } else {
    controller?.updateDataSource(
      addedDataIndexes: <int>[listData!.length - 1],
    );
```

```
    }
  }

  // Continously updating the data source based on timer
  void _updateDataSource(Timer timer) {
    List<double> acc = dataParse(accData!);
    List<double> gyro = dataParse(gyroData!);
    chartAccData!.add(_ChartData(count, acc[0], acc[1], acc[2]));
    chartGyroData!.add(_ChartData(count, gyro[0], gyro[1], gyro[2]));

    if (chartAccData!.length == 50) {
      chartAccData!.removeAt(0);
      updateControllerDataSource(chartAccData, axAxisController, true);
      updateControllerDataSource(chartAccData, ayAxisController, true);
      updateControllerDataSource(chartAccData, azAxisController, true);
    } else {
      updateControllerDataSource(chartAccData, axAxisController, false);
      updateControllerDataSource(chartAccData, ayAxisController, false);
      updateControllerDataSource(chartAccData, azAxisController, false);
    }

    if (chartGyroData!.length == 50) {
      chartGyroData!.removeAt(0);
      updateControllerDataSource(chartGyroData, gxAxisController, true);
      updateControllerDataSource(chartGyroData, gyAxisController, true);
      updateControllerDataSource(chartGyroData, gzAxisController, true);
    } else {
      updateControllerDataSource(chartGyroData, gxAxisController, false);
      updateControllerDataSource(chartGyroData, gyAxisController, false);
      updateControllerDataSource(chartGyroData, gzAxisController, false);
    }

    count = count + 1;
  }

  bool isCapturing = false;
  void _captureData(BuildContext context, int sender) {
    //
    setState(() {
      isCapturing = true;
    });
    //
    int n = 300; // 300/200ms per data = takes 60 seconds
    onCaptureTimer = Timer.periodic(const Duration(milliseconds: 200), (Timer timer)
{
      List<double> acc = dataParse(accData!);
      List<double> gyro = dataParse(gyroData!);
```

```
    List<String> captured = [
      // Accelerometer
      acc[0].toString(),
      acc[1].toString(),
      acc[2].toString(),
      // Gyroscope
      gyro[0].toString(),
      gyro[1].toString(),
      gyro[2].toString(),
      // Label
      sender.toString(),
    ];

    if (sender == 1) {
      onData.add(captured);
    } else {
      offData.add(captured);
    }

    setState(() {
      if (sender == 1) {
        onTargetText = n.toString();
      } else {
        offTargetText = n.toString();
      }
    });

    if (n == 1) {
      timer.cancel();
      setState(() {
        isCapturing = false;
        if (sender == 1) {
          onTargetText = 'DONE';
        } else {
          offTargetText = 'DONE';
        }
      });
    }
    n -= 1;
  });
}

/* ------------------------------------------------ */
@override
Widget build(BuildContext context) {
  ValueNotifier<bool> isDialOpen = ValueNotifier(false);
  bool isNotified = Provider.of<ConnectionProvider>(context, listen: true).isNotified;
```

```
if (isNotified && !connectionValue(context)) {

  _connectFromDevice();
  Provider.of<ConnectionProvider>(context, listen: false).toggle(false);
}

if (isNotified && connectionValue(context)) {
  Provider.of<ConnectionProvider>(context, listen: false).toggle(false);
}

return Scaffold(
  backgroundColor: Theme.of(context).colorScheme.background,
  body: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      const SizedBox(height: 10),
      const ChartHeader(title: 'IMU Sensor'),
      Container(
        margin: const EdgeInsets.only(left: 10, right: 10),
        child: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              _buildLiveAccChart(context),
              const SizedBox(height: 10.0),
              _buildLiveGyroChart(context),
            ],
          ),
        ),
      ),
      const SizedBox(height: 10),
      const ChartHeader(title: 'Externals'),

      // Externals
      Container(
        margin: const EdgeInsets.only(left: 10, right: 10),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            // Temperature
            ExternalSensorWidget(
              icon: Icons.thermostat,
              title: 'Temperature',
              valueDisplay: connectionValue(context) ? '$temperature°C' : '--',
            ),
            const SizedBox(width: 10),
            // Distance
            ExternalSensorWidget(
```

```
        icon: Icons.linear_scale_rounded,
        title: 'Distance',
        valueDisplay: connectionValue(context) ? '${distance}mm' : '--',
      ),
    ],
  ),
),

Expanded(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Container(
        margin: const EdgeInsets.only(
          left: 30,
          right: 30,
          top: 0,
        ),
        child: DataButton(
          // Add
          onAddOnTarget: (!isCapturing && onData.isEmpty)
              ? () {
                  if (connectionValue(context)) _captureData(context, 1);
                }
              : () {},
          onAddOffTarget: (!isCapturing && offData.isEmpty)
              ? () {
                  if (connectionValue(context)) _captureData(context, 0);
                }
              : () {},

          // Delete
          onDeleteOnTarget: () {
            setState(() {
              onTargetText = null;
              onData.clear();
            });
          },
          onDeleteOffTarget: () {
            setState(() {
              offTargetText = null;
              offData.clear();
            });
          },

          // Label
          onTargetText: onTargetText,
          offTargetText: offTargetText,
```

```
        ),
      ),
    ],
  ),
),
Center(
  child: Container(
    width: 150,
    margin: const EdgeInsets.only(bottom: 25),
    decoration: BoxDecoration(
      color: Theme.of(context).colorScheme.tertiaryContainer,
      borderRadius: BorderRadius.circular(15),
    ),
    child: TextButton(
      style: TextButton.styleFrom(
          foregroundColor: Colors.white,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(20),
          )),
      onPressed: !isBuildingModel && !isCapturing
          ? () {
              openBuildForm();

              final snackBar = SnackBar(
                content: const Text('Build success, ready to send!'),
                action: SnackBarAction(
                  label: 'Okay',
                  onPressed: () {
                    // Some code to undo the change.
                  },
                ),
              );

              // Find the ScaffoldMessenger in the widget tree
              // and use it to show a SnackBar.
              ScaffoldMessenger.of(context).showSnackBar(snackBar);
            }
          : () {},
      child: const Text('BUILD'),
    ),
  ),
),
Container(
  margin: const EdgeInsets.only(left: 10, right: 10),
  child: Align(
    alignment: Alignment.bottomLeft,
    child: textInfo(info, infoCode),
  ),
```

```
          )
        ],
      ),
    );
  }

  startSpinningBar() {
    int i = 0;
    List<String> m = ['|', '/', '-', '\\'];
    loadingTextTimer = Timer.periodic(const Duration(milliseconds: 100), (Timer timer)
{
      if (i == m.length) i = 0;
      msg("Building model...       ${m[i]}");
      i += 1;
    });
  }

  Future<Widget> buildPageAsync([ble]) async {
    return Future.microtask(() {
      return ResultsPage(ble: ble);
    });
  }

  void viewResults([ble]) async {
    var page = await buildPageAsync(ble);
    var route = MaterialPageRoute(builder: (_) => page);
    if (!mounted) return;
    Navigator.push(context, route);
  }

  Future openBuildForm() {
    final formKey = GlobalKey<FormState>();
    _textController.value = TextEditingValue.empty;
    return showDialog(
        context: context,
        builder: (context) {
          return AlertDialog(
            title: const Text('Build'),
            content: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                Form(
                  key: formKey,
                  child: TextFormField(
                    autovalidateMode: AutovalidateMode.onUserInteraction,
                    validator: (value) => _textController.text != '' ? null : 'Cannot be empty',
                    autofocus: true,
                    decoration: const InputDecoration(hintText: 'Enter model name'),
```

```
        controller: _textController,
      ),
    ),
  ],
),
actions: [
  TextButton(
    onPressed: () async {
      if (formKey.currentState!.validate()) {
        setState(() {
          isBuildingModel = true;
        });

        Navigator.of(context).pop();
        startSpinningBar();

        String fileName = _textController.text;
        var token = await UserSecureStorage.getToken();

        String dir = await AppStorage.getDir();
        String fileInputPath = "$dir/$fileName/${fileName}_input.csv";
        String fileModelPath = "$dir/$fileName/${fileName}_model.h";
        String fileCallbackPath = "$dir/$fileName/${fileName}_callback.csv";
        String fileInfoPath = "$dir/$fileName/${fileName}_info.json";

        await AppStorage.writeCsv(
          data: [header, ...onData, ...offData],
          filePath: fileInputPath,
        );

        buildClass.sendInput(
          filePath: fileInputPath,
          modelName: _textController.text,
          userToken: token,
        );

        Future<TrainedModels> model = buildClass.model;
        model.then((value) async {
          int errorsCount = 0;
          var response = value.toJson();

          // save response as json
          await AppStorage.writeJson(data: response, filePath: fileInfoPath);

          msg('Downloading your model, please wait.');
          // MODEL
          await buildClass
              .downloadFile(fileUrl: response['file'], location: fileModelPath)
```

```
                    .then((value) {
                  msg(value);
                }).onError((error, _) {
                  errorsCount += 1;
                });

                // CALLBACK
                await buildClass
                    .downloadFile(fileUrl: response['callback_file'], location:
fileCallbackPath)
                    .then((value) {
                  msg(value);
                }).onError((error, _) {
                  errorsCount += 1;
                });

                if (errorsCount == 0) {
                  msg('Build success, ready to send!', 2);
                } else {
                  msg('Error building the model', -1);
                }
              }).onError((error, _) {
                msg(error.toString(), -1);
              }).whenComplete(() {
                setState(() {
                  isBuildingModel = false;
                });
                loadingTextTimer!.cancel();
              });
            }
          },
          child: const Text('Submit'),
        )
      ],
    );
  });
 }
}

/// Private calss for storing the chart series data points.
class _ChartData {
 _ChartData(this.t, [this.x = 0, this.y = 0, this.z = 0]);
 final int t;
 final num x;
 final num y;
 final num z;
}
```

Web Server Code

views.py – program for training the Feedforward Neural Network model.

```
import base64
from django.shortcuts import render
from django.http import JsonResponse
from rest_framework.response import Response
from rest_framework.decorators import api_view
from rest_framework.views import APIView
from .models import Item, TrainedModel

from .serializers import (ItemSerializer, TrainedModelSerializer, UserSerializer,
    RegisterSerializer, LoginSerializer)

from rest_framework.permissions import IsAuthenticated
from rest_framework import status, generics
from rest_framework.authentication import TokenAuthentication,
SessionAuthentication, BasicAuthentication
from rest_framework.authtoken.models import Token
from .utils import *
from django.conf import settings
import os
import json

from knox.models import AuthToken
from knox.views import LoginView as KnoxLoginView

from rest_framework.permissions import AllowAny
from rest_framework.authtoken.serializers import AuthTokenSerializer
from django.contrib.auth import login
from django.forms.models import model_to_dict

from django.core.files.base import ContentFile
from .train import BFRBNeuralNetwork
import zipfile
from io import BytesIO
from django.http import HttpResponse

#
import numpy as np
import csv
import tensorflow as tf


PARAMS = ['ax', 'ay', 'az', 'gx', 'gy', 'gz', 'dist', 'temp','class']
SAMPLES_PER_HOTSPOT = 1
```

```
def home(request):
    return render(request, 'apis/home.html', {})

class NeuralNetworkBuilder(APIView):
    permission_classes = (IsAuthenticated,)

    def get(self, request, format=None):
        items = Item.objects.all()
        serializer = ItemSerializer(items, many=True)
        return Response(serializer.data)

    def post(self, request, format=None):
        data = request.data.get('data')
        named_model = request.data.get('model_name')
        file = request.data.get("file")
        fileName = request.data.get("fileName")

        if named_model == '':
            named_model = 'NO_NAME_MODEL'

        df = pd.read_csv(file)
        df.to_csv('xtnctxData.csv', index=False)
        off_target = pd.DataFrame(df[df['class']==0].values.tolist(), columns=PARAMS)
        on_target = pd.DataFrame(df[df['class']==1].values.tolist(), columns=PARAMS)

        HOTSPOT = [off_target, on_target] # known location / class
        NUM_HOTSPOT = len(HOTSPOT)
        ONE_HOT_ENCODED_HOTSPOT = np.eye(NUM_HOTSPOT)
        N_EPOCH = 100

        inputs = []
        outputs = []

        for hotspot_index in range(NUM_HOTSPOT):

            target = HOTSPOT[hotspot_index]

            num_recordings = int(target.shape[0] / SAMPLES_PER_HOTSPOT)

            output = ONE_HOT_ENCODED_HOTSPOT[hotspot_index]

            print(f"\tThere are {num_recordings} recordings.")

            for i in range(num_recordings):
                tensor = []
                noise_tensor = []
                for j in range(SAMPLES_PER_HOTSPOT):
                    index = i * SAMPLES_PER_HOTSPOT + j
```

```
            # normalize the input data, between 0 to 1:
            # - acceleration is between: -4 to +4
            # - gyroscope is between: -2000 to +2000
            tensor += [
                (float(target['ax'][index]) + 4) / 8,
                (float(target['ay'][index]) + 4) / 8,
                (float(target['az'][index]) + 4) / 8,
                (float(target['gx'][index]) + 2000) / 4000,
                (float(target['gy'][index]) + 2000) / 4000,
                (float(target['gz'][index]) + 2000) / 4000
            ]
        inputs.append(tensor)
        outputs.append(output)


    # convert the list to numpy array
    inputs = np.array(inputs)
    outputs = np.array(outputs)


    # Randomize the order of the inputs, so they can be evenly distributed for
training, testing, and validation
    # https://stackoverflow.com/a/37710486/2020087
    num_inputs = len(inputs)
    randomize = np.arange(num_inputs)
    np.random.shuffle(randomize)

    # Swap the consecutive indexes (0, 1, 2, etc) with the randomized indexes
    inputs = inputs[randomize]
    outputs = outputs[randomize]

    # Split the recordings (group of samples) into three sets: training, testing and
validation
    TRAIN_SPLIT = int(0.6 * num_inputs)
    TEST_SPLIT = int(0.2 * num_inputs + TRAIN_SPLIT)

    inputs_train, inputs_test, inputs_validate = np.split(inputs, [TRAIN_SPLIT,
TEST_SPLIT])
    outputs_train, outputs_test, outputs_validate = np.split(outputs, [TRAIN_SPLIT,
TEST_SPLIT])

    # build the model and train it
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(50, activation='relu'))
```

```
        model.add(tf.keras.layers.Dropout(0.3))
        model.add(tf.keras.layers.Dense(15, activation='relu'))
        model.add(tf.keras.layers.Dense(NUM_HOTSPOT, activation='softmax'))
        model.compile(
            optimizer=tf.keras.optimizers.Adam(),
            loss=tf.keras.losses.CategoricalCrossentropy(),
            metrics=['accuracy']
        )

        history = model.fit(inputs_train, outputs_train, epochs=N_EPOCH,
batch_size=1,
                            validation_data=(inputs_validate, outputs_validate))


        # Convert the model to the TensorFlow Lite format without quantization
        converter = tf.lite.TFLiteConverter.from_keras_model(model)
        converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
        tflite_model = converter.convert()

        owner_file = str(named_model).replace(" ", "_") + f'--{request.user.username}'
        data = {
            'owner': request.user.id,
            'model_name': named_model,
            'file': ContentFile(bytes(model_to_databytes(tflite_model), 'utf-8'),
                                name=owner_file + '_model.h'
                                ),
            'callback_file': ContentFile(bytes(callback_string(history), 'utf-8'),
                                name=owner_file + '_callback.csv'),
        }

        serializer = TrainedModelSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

class UserFiles(APIView):
    permission_classes = (IsAuthenticated,)

    def get(self, request, format=None):
        # Files (local path) to put in the .zip
        # FIXME: Change this (get paths from DB etc)
        userTrainedModels = TrainedModel.objects.filter(owner=request.user)

        # Folder name in ZIP archive which contains the above files
        # E.g [thearchive.zip]/somefiles/file2.txt
        # FIXME: Set this to something better
        zip_subdir = request.user.username
```

```
        zip_filename = "%s.zip" % zip_subdir

        # Open StringIO to grab in-memory ZIP contents
        s = BytesIO()

        # The zip compressor
        zf = zipfile.ZipFile(s, "w")

        for model in userTrainedModels:
            # for fpath in filenames:
            # Calculate path for file in zip
            fpath = f'./media/{model.file.name}'
            fdir, fname = os.path.split(fpath)
            # TrainedModels
            zip_path = os.path.join(zip_subdir, f'{model.model_name}/{fname}')
            zf.write(fpath, zip_path)

            # Callbacks
            fpath = f'./media/{model.callback_file.name}'
            fdir, fname = os.path.split(fpath)
            zip_path = os.path.join(zip_subdir, f'{model.model_name}/{fname}')
            zf.write(fpath, zip_path)

            # Info
            data = model_to_dict(model)
            data['file'] = str(data['file'])
            data['callback_file'] = str(data['callback_file'])
            json_data = json.dumps(data, indent=4)
            zip_path = os.path.join(zip_subdir,
f'{model.model_name}/{model.model_name}_info.json')
            zf.writestr(zip_path, json_data)
            s.seek(0)

        # Must close zip for all contents to be written
        zf.close()

        # Grab ZIP file from in-memory, make response with correct MIME-type
        resp = HttpResponse(s.getvalue(), content_type="application/x-zip-
compressed")
        # ..and correct content-disposition
        resp['Content-Disposition'] = 'attachment; filename=%s' % zip_filename
        return resp
```

```
class RegisterAPI(generics.GenericAPIView):
    serializer_class = RegisterSerializer

    def post(self, request, *args, **kwargs):
       serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        user = serializer.save()
        _, token = AuthToken.objects.create(user)
        return Response({
            "user": UserSerializer(user, context=self.get_serializer_context()).data,
            "token": token
        })



class LoginAPI(generics.GenericAPIView):
    serializer_class = LoginSerializer
    permission_classes = ()

    def post(self, request, *args, **kwargs):
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        user = serializer.validated_data
        _, token = AuthToken.objects.create(user)
        return Response({
            "user": UserSerializer(user, context=self.get_serializer_context()).data,
            "token": token
        })


class UserAPI(generics.RetrieveAPIView):
  permission_classes = [IsAuthenticated]
  serializer_class = UserSerializer

  def get_object(self):
    return self.request.user
```

Full project: https://github.com/xtnctx/bfrbsys

Appendix 5

Letters and Certifications

Appendix 6

Forms