A System for Sitting Detection Based on TinyML and Mobile Application

Machine Learning with TensorFlow Lite on Arduino and Android Application Development

By

CHUN-YU CHEN

Department of Computer Science
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of MASTER OF SCIENCE within the School of Computer Science, Electrical and Electronic Engineering, and Engineering Maths (SCEEM).

SEPTEMBER 2021

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and the requirements of the degree of Master of Science in the Faculty of Engineering. Except where indicated by specific reference in the text, the work is entirely the author's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: Chun-Tu Chen DATE: 11/09/2021

Executive Summary

This project proposes a system which helps people to self-monitor their sedentary habits. Nowadays, people spend a considerable amount of time in front of their desks working and studying. Growing evidence has shown that long sedentary time may pose a risk to our health. Therefore, the system aims to provide a way to help people understand how much time they have spent sitting in the past time period. By reviewing the matrices, people can respond positively, such as reducing sitting time or interrupting sitting with physical activities.

The system consists of an Arduino device and an Android application. Unlike other products on the market, the Arduino device with a camera module is placed on a desk, and faces the seat of a user. A computer vision model is the core component that makes the device distinguish between a person sitting or an empty chair. To deploy the model on a source-constrained device, Tiny Machine Learning techniques are applied in the development. The workflow includes training the model by TensorFlow, shrinking it down to small size by TensorFlow Lite and TensorFlow Lite for microcontrollers. The model performance is evaluated to ensure it can successfully identify between objects and is not biased toward edge cases. The binary outcome is produced by the model, which indicates whether a person is sitting on a chair or not. The Arduino device used in this project supports Bluetooth Low Energy (BLE) connection. The BLE connection is used for syncing data between the Arduino, mobile phone, and the cloud database.

Data originates from the Arduino is received by the user's mobile phone. Flutter is the framework for building an application in the project. The application has three main pages, which are the time tracker page, the log page, and the manually registration page. The time tracker page shows today's total sitting time of the user and displays a time series plot of the daily sitting time of the past five days. The log page shows a table of the raw data for making the plot. The manual registration page lets the user add or remove data manually. All users' data is kept on the Firebase. Therefore, several internal functions have been developed for communicating data between the device, the application, and the cloud.

The biggest limitation of the system is that the application can not run in the background. To support the background process, writing a native code in Java or Kotlin is required. Due to the time limit, the development can not be finished in this project, but it will be a top priority in future work. In addition, a more comprehensive training dataset can be collected to refine the model. In general, this project creates a prototype that serves as a great foundation for the development of related products in the future. The added value of the project is a large improvement in my personal development. I learned Sketch programming on Arduino, TinyML workflow, and Android development.

Dedication and acknowledgements

I would like to thank my supervisor, Dr. David Bernhard, for providing me with invaluable guidance throughout the development of my project and thesis. A special appreciation to several friends in Harbour Court who generously contributed their time to assist me in testing the system. Finally, I want to express my gratitude to my parents and girlfriend for their unwavering support during my time in Bristol.

TABLE OF CONTENTS

List of Tables

Table **Page**

List of Figures

CHAPTER 1

INTRODUCTION

This project works on a system which is able to detect if a person is sitting on a chair. The system encompasses hardware and software parts. A reliable and robust hardware part can continuously monitor the occupation condition of a chair. On the other hand, a mobile application acts as a software part and provides total sitting time for the user. The two parts will work together as an integrated system to give users insightful results.

1.1 Overview

Today, people spend a considerable amount of time sitting in front of their desks working or studying. Several reports have suggested that sitting down too much can be a risk to our health. Therefore, the project works toward the development of an intelligent system for monitoring users' sitting conditions. The system uses Tiny Machine Learning (TinyML) technique to detect if a person is sitting on a chair. It is an example of applying deep learning to computer vision. The raw data is processed in the mobile application, which provides a time tracking system for users. Upon completion, the users can benefit from using the system by adjusting their working or studying habits to reduce the time spent sitting.

Current methods for tracking user habits are mainly platform-based. People can track their activities or manage their time through their personal computer or phone. However, there are only limited ways to directly measure the time that people spend sitting. This system may provide a more effective and simpler approach to tracking users' daily sitting time. This project focuses on hardware and software development. It does not aim to conduct a research on improving people's sitting habits with the assistance of technology. This project has three goals:

1. Develop an integrated system which has a hardware device and software application.
2. The hardware device uses a machine learning model to detect if a person is sitting on a chair.
3. The software application can run on the mobile phone, receive data from the hardware device, and generate useful information for the users.

To achieve the goals, a number of objectives can be set in further detail:

1. An Arduino board with a camera module will be used as the hardware device in this project.
2. The device should be simple, portable, and energy-efficient.
3. The device is able to periodically transmit data to the mobile phone.
4. The Android application with Bluetooth connecting capability will be served as the software application in this project.
5. The mobile application provides a user interface for users to review the data.
6. Summary statistics of the data can be produced and shown in the application.

1.2 Personal Development

To successfully develop the whole system, I have studied several new topics in computer science. For hardware development, there are two parts. Firstly, I learned how to deploy a machine learning model on an Arduino device. The process included data collection, model training, model optimization, and model deployment. I have no relevant experience with machine learning or deep learning. Therefore, I studied lots of online material to get familiar with related concepts and techniques. Secondly, Sketch programming or programming an Arduino device is critical to handling data within the board and communicating with its outer counterparts. C++ is a programming language that is used. For software development, Flutter is my choice of the framework to build applications for Android. Flutter is based on the programming language Dart, which features powerful object-oriented programming characteristics. I have limited knowledge of mobile application development, so I have learned new skills and technologies during doing the project. With the knowledge I gained from the MSc course modules, I am not learning all the required skills from scratch. For instance, C++ and Dart have similar features to C and Java respectively, which are both quite familiar to me. In my opinion, this project not only has practical usage but also is a great practice and extension to my coursework.

CHAPTER 2

BACKGROUND

Because this project is for a tool that will be used to measure the sitting time of a user, a review of the current literature was undertaken to understand how researchers and developers have conducted previous studies and what conclusions they have reached. Existing tools have been scrutinised in particular, with a focus on those that use different methods from this project.

2.1 Move More, Sit Less

According to the National Health Service [48], many adults in the United Kingdom (UK) sit for approximately 9 hours every day. Although there is not enough evidence to limit how much time people should sit each day. There is increasing evidence that sitting down too much can be a risk to our health. A high amount of total inactive time is linked to an elevated risk of type 2 diabetes, cardiovascular disease (CVD), and all-cause and CVD mortality, according to observational evidence [6, 14, 53, 56]. Despite the fact that high levels of physical exercise may mitigate the effects of long sitting, there is a clear connection between sedentary time and mortality risk [22]. Additionally, short-term intervention studies have indicated that interspersing periods of sedentary behavior with times of modest activity or standing improves cardiometabolic health [13, 29]. Adults can significantly reduce their sedentary behaviour through the intervention of physical activity [39, 52, 57].

2.2 Devices for Self-Monitoring Sedentary Time

In recent years, more and more commercial models of pedometer have been introduced to the market. Using a real-time feedback device such as a pedometer, it has been proven that people can be intervened from sitting and increase their physical activity levels [8, 28, 40, 55]. However, there are few effective intervention tools to help people reduce their sedentary habits. Sanders *et al* concluded that at the time they were doing the investigation, most of the devices had been designed for monitoring physical activity rather than tracking sedentary behaviour [56]. Also, devices can have different forms and placements depending on their usage cases. For example, most of the devices are wearable, but there are some models that need to be placed on the seat.

2.2.1 Definition of Self-Monitoring

To understand whether devices can help self-monitoring sedentary time, the clear definition of "self-monitoring" is crucial. Self-monitoring is defined as "a person closely and actively watches their own behaviour" [31, 41] and "enabling the alteration of their behaviour to meet preset goals or results" [24]. The users of the devices anticipate that they can rely on the machines, get insightful outcomes, and change their sitting habits. In other words, users do not passively get the results and wait for guidance from the services or other people. According to the Control Theory, a cycle of steps, including "self-monitoring of behaviour, setting goals, receiving feedback, and revisiting goals with feedback, are essential to self-management and behavioural control" [12, 40]. Self-monitoring has been shown to be a successful behaviour change method for a range of activities, including smoking, food, and physical activity, and it is regarded as the basis of lifestyle behaviour modification programs [9, 40].

2.2.2 Technologies of Devices

In the report from Sanders *et al*, they investigated 82 devices which were identified as having the function of self-monitoring sedentary time and/or physical activity [56]. Nine of the devices are particularly used for monitoring users' sitting conditions. Sedentary time is typically measured in two ways by these devices. Firstly, posture sensors use an accelerometer which is combined with gravitational components and proprietary algorithms to measure sedentary time, or they use the changes in pelvic alignment to measure different postures and calculate the sitting time further. Secondly, pressure sensors are used to detect the users' sitting condition. Pressure sensors can be placed in a sock, a shoe, or a chair. When inserted into a sock or shoe, the pressure may identify whether the person is standing, sitting or reclining depending on the amount of pressure on the sensor. The pressure sensor on a chair has a simpler outcome: when it is active, the user is sitting; otherwise, there is no sitting behaviour in that place.

Devices usually have feedback mechanisms to interact with users and enhance the user ex-

perience. Vibratory feedback and screen display are common examples. In addition, a mobile application is often used as an interface to get feedback. For sedentary time, the feedback usually comes in the form of time spent sitting. The design of the mobile applications also embedded some common functionalities, including real-time feedback, self-improvement functions, and push notifications. The connection between devices and mobile applications usually uses Bluetooth Low Energy (BLE). In some cases, USB can also be used to connect with mobile phones and communicate with applications [56].

2.2.3 Example: VitaBit Sit–Stand Tracker

The VitaBit device (VitaBit Software International B.V., Eindhoven, The Netherlands) is a small, cuboid accelerometer that is worn on the thigh (Figure 2.1) [32]. Unlike other monitors that focus on physical activity rather than detection of sitting and standing, the group from VitaBit states that their model is a specific and precise tool to measure if a user is sitting, standing, or walking. According to the report, the proprietary algorithm calculates pedometer data to determine whether the output for a 30-s window is classified as walking [5]. If the output is not walking, the algorithm distinguishes between sitting and standing [5]. The device can keep the user's data for at least 30 days. Once users connect the device to the mobile application via BLE, data will be sent to a server. After data is processed and analyzed instantaneously, it will be safely stored in a database. Users can review their activity profiles either through the mobile application or the web-based analytics portal.

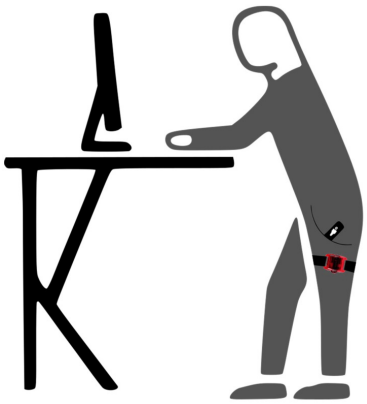


FIGURE 2.1. Wearing locations of VitaBit. Figure reproduced from [5].

The design of the mobile application is another critical part of product development. The app is the main user interface, so the appearance and design can significantly affect the user experience. Based on the description and screenshots from the Play Store, the VitaBit app contains four main sections which users can switch between by using the bottom navigation bar. These sections are "Dashboard", "Device", "Profile", and "Settings". Inside the Dashboard, users can review their daily total time spent sitting and standing (Figure 2.2(a)). As for walking, it calculates the

total steps that users have walked every day. Another feature is the daily average of times and steps that are estimated from a certain time period. The Dashboard also provides detailed daily activity profiles in the form of stacked bar charts (Figure 2.2(b)). Users can select different time scales: 3, 8, and 24 hours to observe their sedentary patterns. The Device section contains basic information about the device, including battery level, MAC address, and firmware version (Figure 2.2(c)). The Profile section shows the user's profile, including name, gender, birthday, height, and weight (Figure 2.2(d)). The Settings section can let users log out of their accounts (Figure 2.2(e)).

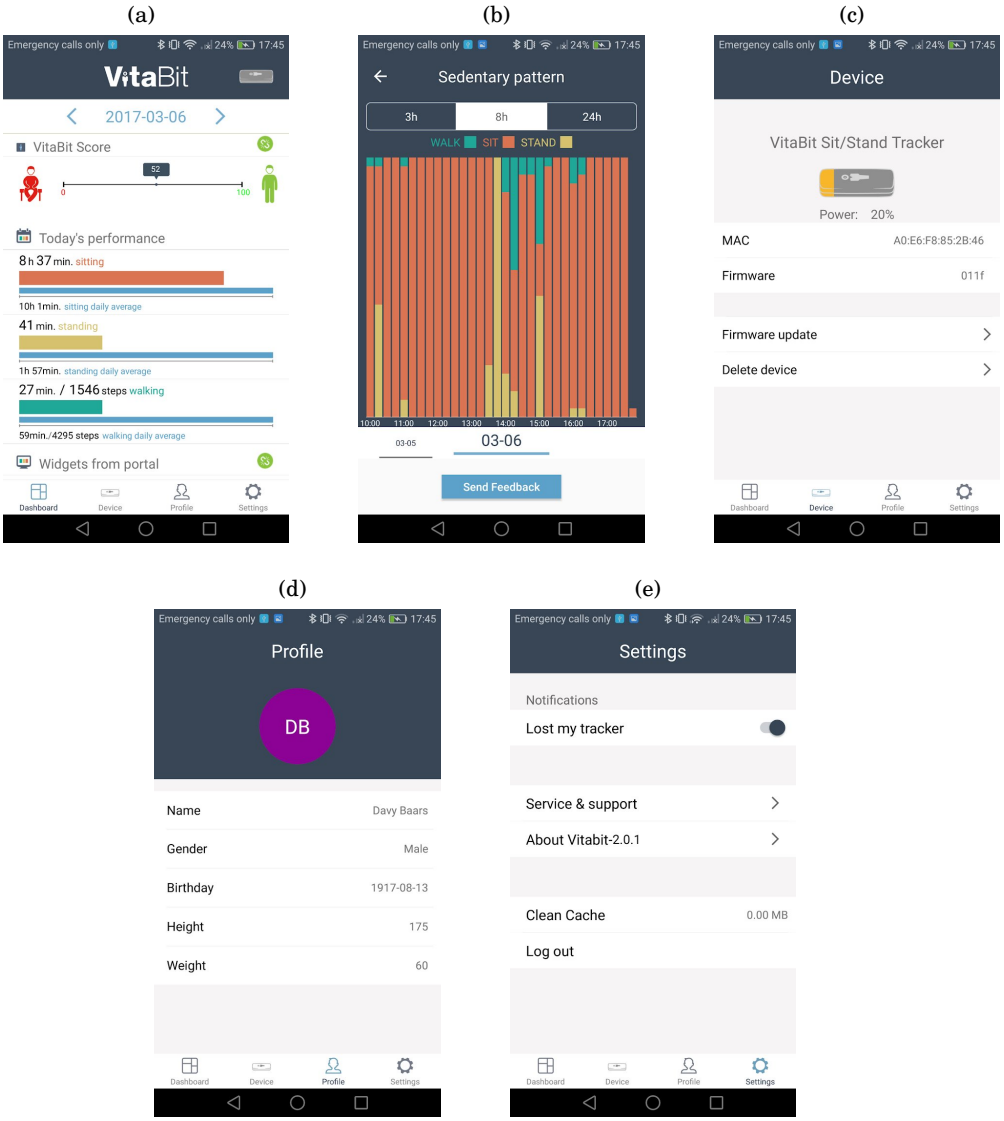


FIGURE 2.2. Screenshots of VitaBit app. (a) Dashboard page. (b) Sedentary pattern page. (c) Devices page. (d) Profile page. (e) Settings page. Figure reproduced from [10].

2.3 Person Detection in TinyML

TinyML is a new field that applies machine learning's revolutionary power to the performance- and power-constrained world of embedded devices. It makes the embedded system deploy machine learning algorithm on the chip in real-time without connecting to the cloud. For example, the OK Google team successfully ran neural networks that were just 14 kilobytes on Android phone. The main CPU was power-off without connecting to the internet, and specialised chips use only very little energy to run the model.

To "see" if a person is sitting, vision is the most direct and easiest way for humans. However, machines did not have access to vision until recently. Visual data was just too messy, unstructured, and very hard to decipher. It's gotten much easier to create programmes that can "see" because of the advancements in convolutional neural networks (CNNs). CNNs learn to understand our visual world by filtering an excessively complex input into a map of known patterns and shapes, which was inspired by the structure of the visual cortex [16]. Algorithms smartly put all the pieces together and reveal the entities present in a digital image. However, the privacy issue is a big concern when we are applying vision models. Users can be worried about their actions being recorded by the camera on the device, and the data might stream to the cloud. Actually, it is how ML inference is done traditionally. TinyML is a game changer which is powered by a tiny microcontroller but without any internet connection. The device can make an inference on the chip without causing privacy issues.

With the aid of the TinyML technique, the power of vision can be equipped to the device. The binary outputs can be generated from the sensor, which a 1 if a target is in view and a 0 if it is not. The application that was developed in this project uses a person-sitting-detection mode, running on a microcontroller with a camera attached, to know when an individual is sitting on a chair.

2.3.1 The Machine Learning Paradigm

In traditional programming, the programmer comes up with rules and turns them into code that acts on the data to give the answers (Figure 2.3(a)). The programmer takes care of every detail of logic inside the code; for example, different condition statements that can handle any potential scenarios. However, sometimes it is almost impossible to write down all the possible situations or rules in traditional programming because rules are the most complicated part of the whole workflow. Instead of figuring out the rules by the programmers themselves, we let the computers handle the rules. The machine learning paradigm allows a computer to find out the parameters that fit a function (Figure 2.3(b)). The basic idea of the algorithm is a repeating cycle in which the computer guesses the answer based on the first version of the rules, then it measures accuracy,

and optimises the guesses by tuning the parameters inside the rules to make an updated version. After a number of cycles, we can choose an appropriate rule (not necessarily the one that has the highest accuracy) to recognise or categorise future inputs.

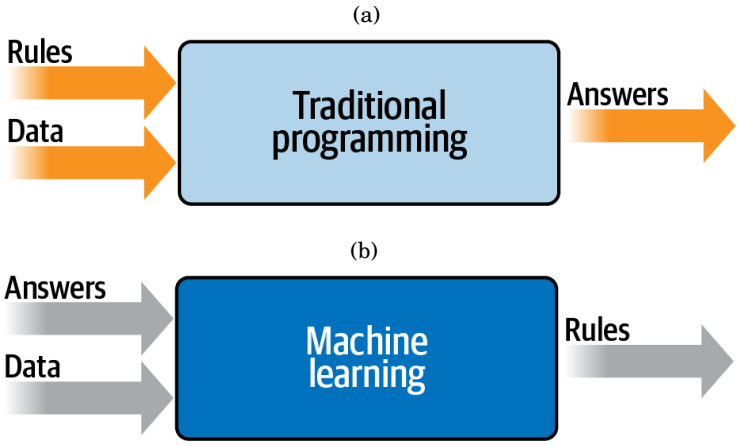


FIGURE 2.3. Workflows of traditional programming and machine learning paradigm. (a) Workflow of traditional programming. (b) Workflow of machine learning paradigm. Figures reproduced from [43].

2.3.2 TensorFlow Family

TensorFlow is an open source framework created by Google to let users readily create and use machine learning models. Today, TensorFlow has grown into an ecosystem that supports different training aspects and deployment of models to a variety of platforms (the web, mobile, and embedded systems). As Figure 2.4 shows, the left side of the diagram presents the architecture and APIs that can be used for training models. TensorFlow provides several ways to design the model. However, using Kersa, a high-level API that already prepares common machine learning algorithms in code, is the most convenient way to go. In the center of the diagram is the interface for saving a model, called TensorFlow SavedModel. On the right are the ways that models can be deployed. The standard TensorFlow is useful for training different deep learning models. Its core library is about 400 MB in size, but it can use a considerable amount of memory (> 1 GB) even running a small model. As a result, a more lightweight framework, TensorFlow Lite, has been developed. Its binary file is approximately 1 MB in size, which is ideal for running on smaller systems such as Android, iOS, and single-board computers like the Raspberry Pi. To fit the model on the microcontroller, which is often equipped with less than 1 MB of storage and 256 KB of RAM, the TensorFlow micro is used. The framework compresses the TensorFlow library to the extreme, removing all but essential functionality. Its core runtime library only uses 16 KB, which can work on microcontrollers such as the Arduino Nano.

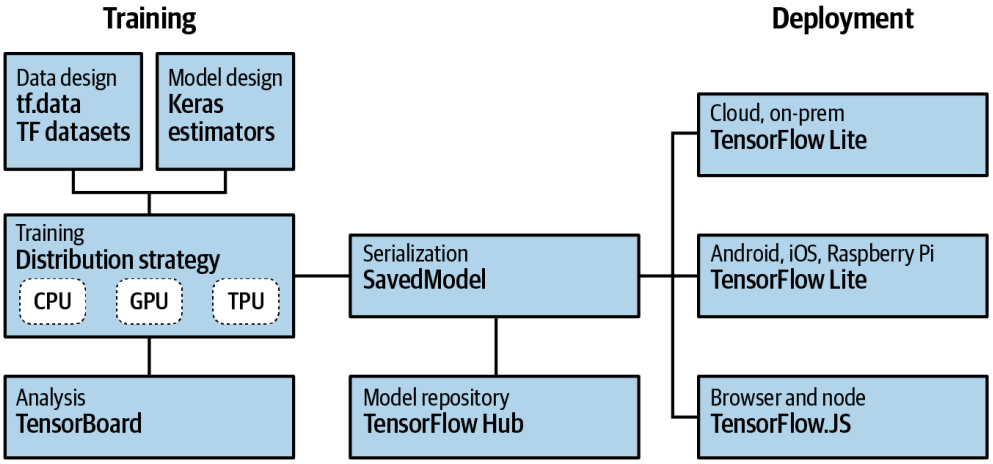


FIGURE 2.4. TensorFlow high-level architecture. Caption and figure reproduced from [43].

2.3.3 The Building Blocks of Deep Learning

Machine learning is a subfield of artificial intelligence (AI) focused on developing algorithms that learn to solve problems by analyzing data for patterns. Deep learning is part of both AI and machine learning that leverages neural networks and big data [45]. In recent years, deep learning has become one of the most popular approaches to machine learning. In deep learning, a conceptual network is often used to represent the model. Each circle in Figure 2.5 represents a neuron, and each column of circles represents a layer in a neural network. As a result, there are three layers, including five neurons in the first, four in the second, and two in the third. In the language of data science, inputs and outputs of deep learning models are in the form of tensors. A tensor is fundamentally an array that can include either numbers or other tensors.

Different arrangements of neurons or shapes of networks can be designed to fit different tasks. In Figure 2.5, for instance, there are numerous layers of tightly coupled neurons, with each neuron connected to the following layer's neuron. This "Deep Neural Network" (DNN) structure is the most prevalent form of layer. Designing model architecture is a major field of research. Developers often find existing models for many common problems. However, to deploy deep learning models on tiny devices, we should always consider the size of the model. The number of neurons in a model, as well as how those neurons are linked, determine its size [62].

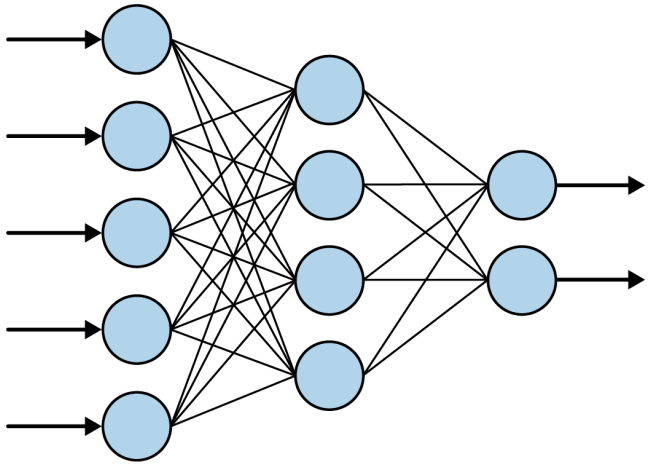


FIGURE 2.5. A typical neural network. Caption and figure reproduced from [43].

2.3.4 Building a Computer Vision Model

To identify a person or a chair in this project, a computer vision model is needed. A colour image in the computer world is represented in a three-dimensional matrix consisting of height, width, and the number of channels. The three overlapping matrices provide information about three colours respectively: red, green, and blue (RGB). For a black-and-white image, there is only one matrix or channel to control the colour. Each matrix which keeps RGB information becomes a basic unit of image in the digital world, called a pixel. Since pixels are essentially number matrices, they can be easily fed into a neural network as numeric inputs. However, a neural network which is made of dense layers like DNN, can only work successfully on images that are similar to those used for training. That is, the model performs poorly when the original image resizes, partially or completely rotates [46].

To take the model to the next level, researchers again borrowed ideas from the natural world. In 1962, an experiment was carried out by Nobel Laureates David Hunter Hubel and Torsten Wiesel, in which they found only specific neurons in the brain activate when the eye perceives certain patterns, such as horizontal, vertical, or diagonal edges. In other words, the neurons in the human vision cortex tend to capture important "features" in vision. Instead of keeping raw pixels in the images, the model can filter the images down to constituent elements. A convolution are filters that reduces the complexity of each pixel in the image and generates a new image with essential features. A neural network with a convolution layer is a convolutional neural network (CNN). In Figure 2.6, a convolution kernal (a small matrix) is applied to the input image. Within the convolution window, every value of pixel is multiplied by the corresponding number inside the kernal, and the summation of all values becomes the new value for the selected pixel. For example, if the selected pixel is the central pixel in Figure 2.6, it has an original value of 1. After

multiplying all the 9 values inside the window and the summation, the updated value of the selected pixel is -2 [46]. The same process will repeat over all the pixels of the input image.

$$(0 \times 1) + (0 \times 0) + (0 \times 0) + (0 \times 0) + (1 \times 0) + (1 \times 0) + (0 \times 0) + (1 \times 0) + (2 \times -1) = -2 \tag{2.1}$$

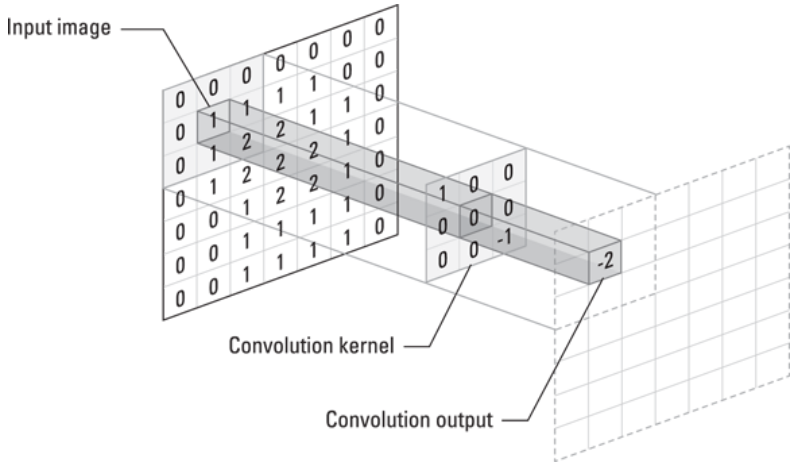


FIGURE 2.6. A convolution processes a chunk of an image by matrix multiplication. Caption and figure reproduced from [46].

Different convolutions can generate different filtered images (Figure 2.7). A small change in the convolution matrix can produce different features in images. A useful aspect of using CNN is that users do not need to design the convolution matrices from scratch. Each kernel matrix element is treated as a parameter in the training process, so the optimum filters to match inputs to outputs will be learned over time [44]. When a network consists of multiple and various kinds of filtering layers, the complexity of the model grows as well. Pooling is the process of removing pixels from images but keeping representative ones. In Figure 2.8, a two-by-two pooling layer is introduced to the network. Because the input image is a four-by-four matrix (16 pixels), the pooling layer can group the input into four two-by-two matrices. The pooling layer takes the highest value from each groups, and puts them together into a new image [44]. The number of pixels has been reduced from 16 to 4 in this example, which shrinks the size by 75%. With the combination of convolution and pooling layers, an image could be compressed and could have its features enhanced.

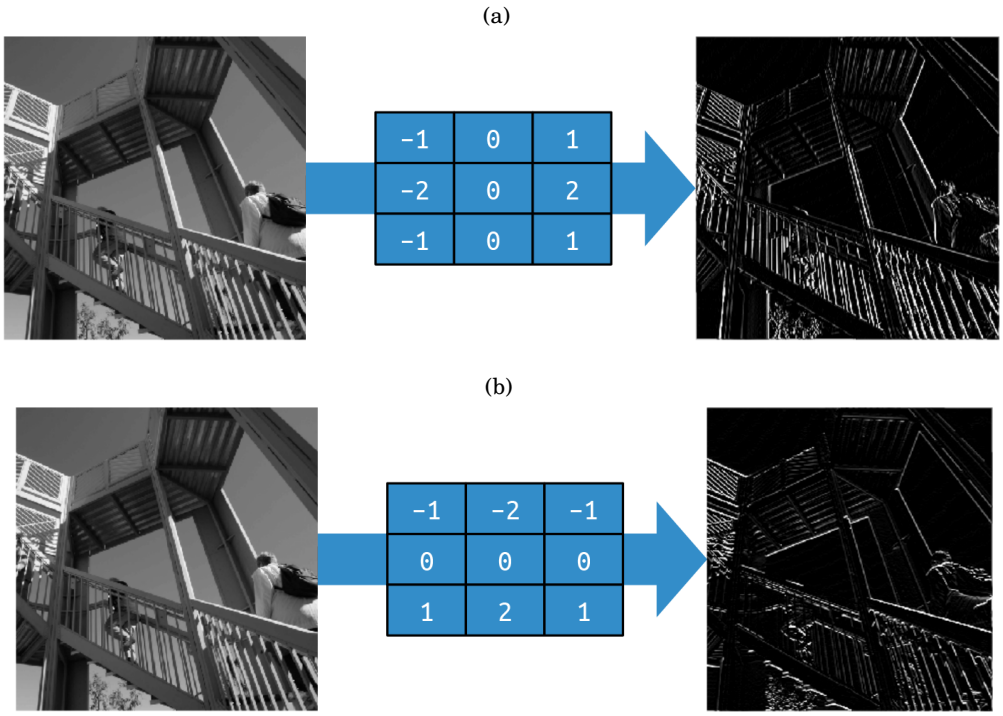


FIGURE 2.7. Two filters emphasise on different features. (a) Using a filter to get vertical lines. (b) Using a filter to get horizontal lines. Caption and figure reproduced from [44].

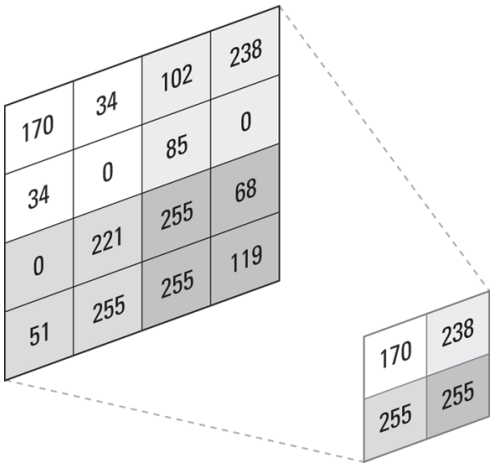


FIGURE 2.8. Demonstrating max pooling. Figure reproduced from [46].

2.4 Understand Electronic Devices Market

Just like all the self-monitoring devices on the market, a complete system needs both hardware and software components. Today, mobile phones are equipped with powerful processors and many other decent gadgets, which makes their computational capacity very close to some personal computers. To develop an application that will be accessed by most target users, a market survey should be conducted to decide the platform that should run the application. Smartphones and laptops are among the most widely owned gadgets in households in the UK (Figure 2.9). When we compare penetration rates between desktops and smartphones in the UK, a vivid trend can easily be spotted, which shows that the popularity of using smartphones has overwhelmed desktops (Figure 2.10). Between 2009 and 2020, the proportion of UK homes with a desktop computer or an iMac dropped significantly. Only 24% of UK households had such a device in 2020, compared to 56% in 2009. The similar trend can be found in the forecast of computing device shipments in the last couple of years (Figure 2.11).

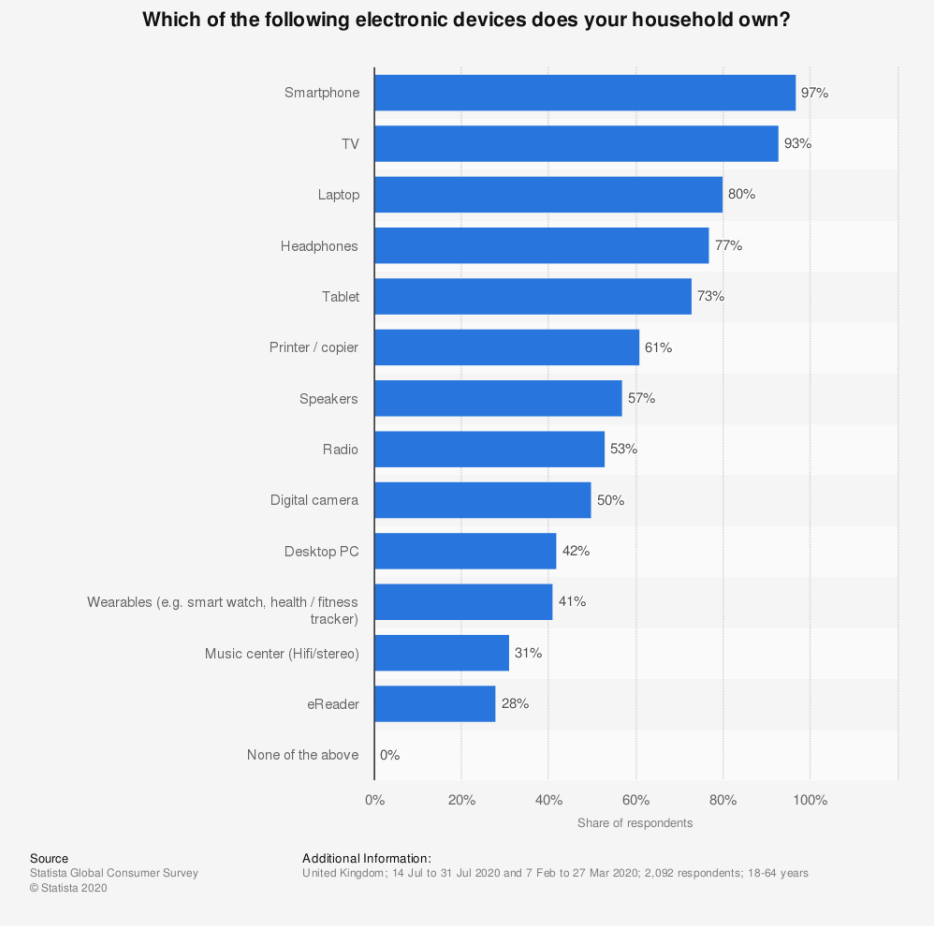
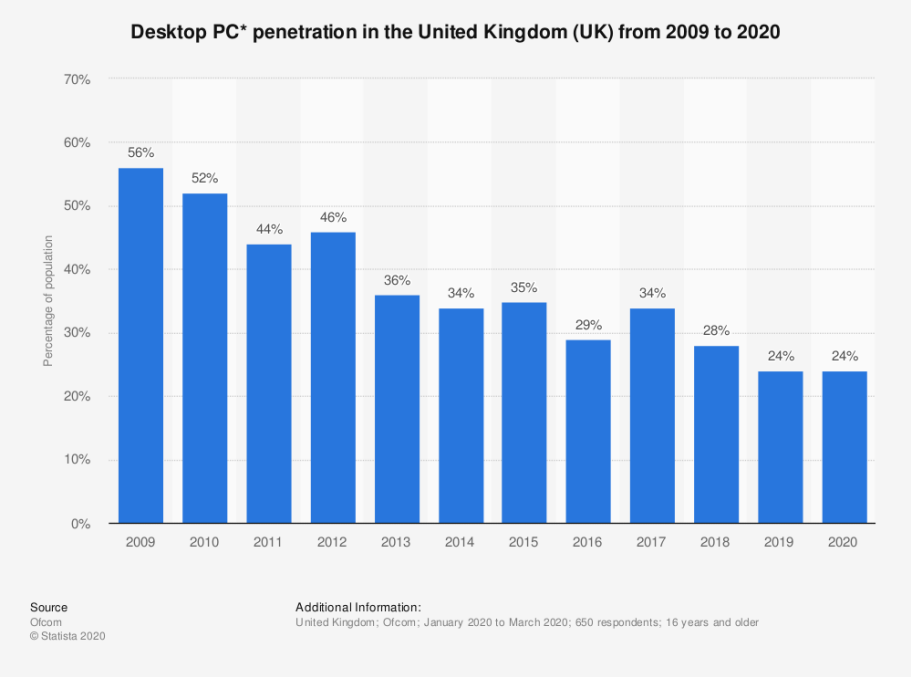


FIGURE 2.9. Consumer electronics ownership in the UK 2020. Caption and figure reproduced from [35].

(a)



(b)

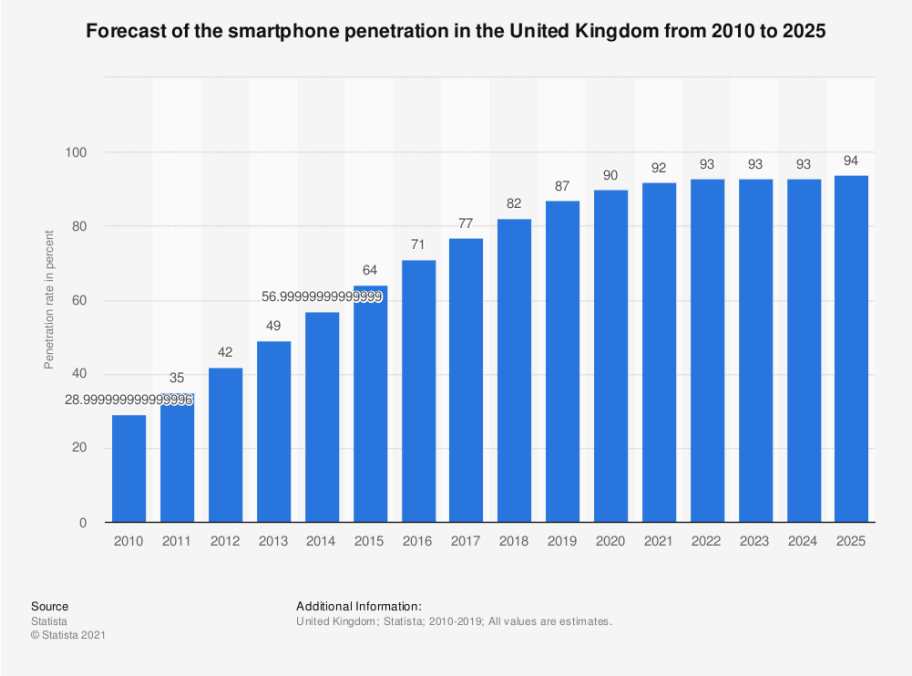


FIGURE 2.10. Comparison between penetration rate between desktop and smartphone in the UK. (a) Desktop PC penetration in the UK 2009-2020. (b) Smartphone penetration forecast in the UK 2010-2025. Caption and figure reproduced from [3, 17].

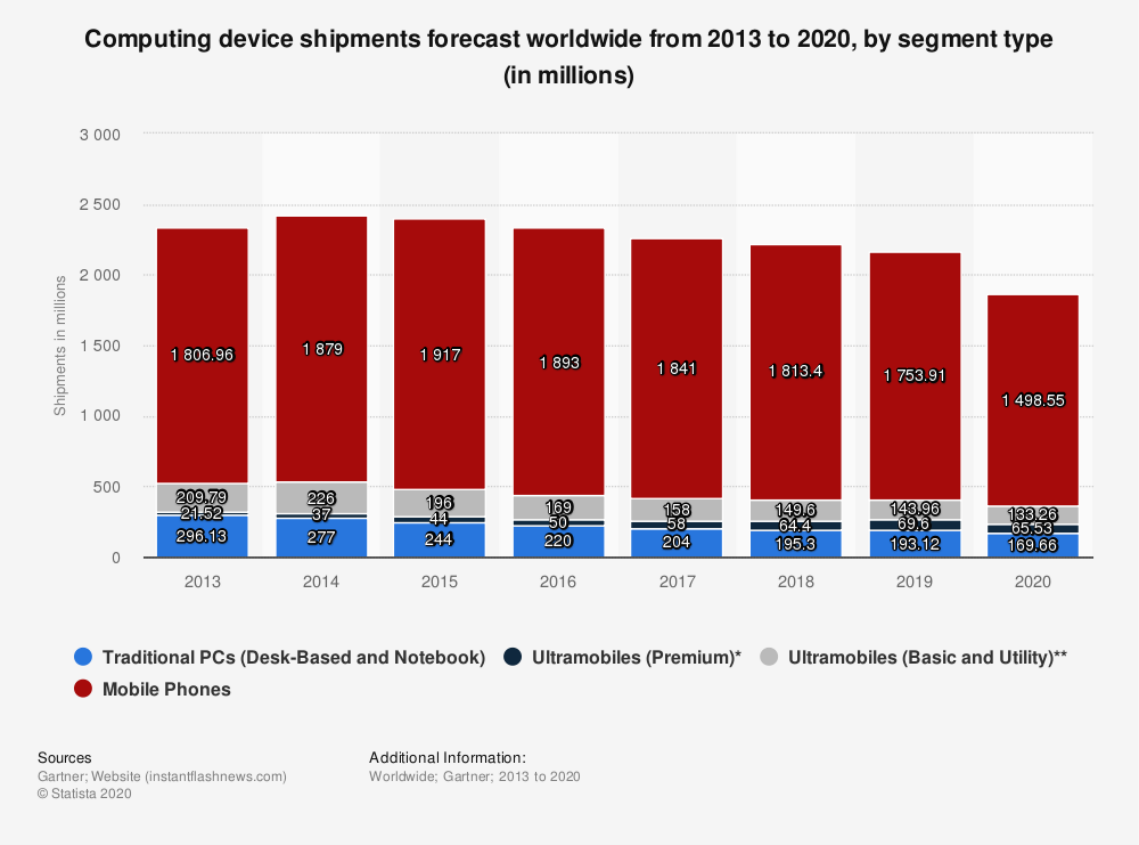


FIGURE 2.11. PCs, tablets, ultra mobiles, mobile phones global shipments forecast 2013-2020. Caption and figure reproduced from [4].

Based on the survey, smartphones have surpassed computers as the most popular device for accessing the internet in the UK by 2020, accounting for almost half of all internet users (Figure 2.12). After researching the market for different platforms, it is apparent that developing applications for smartphones can reach most users. Further investigation can be carried out to choose between two popular smartphone operating systems (OSs), Android and iOS. Although the majority of smartphones run on the Android system (Figure 2.13), sometimes it is a personal preference or organization strategy to determine which OS to develop. Furthermore, developers today can use cross-platform toolkits to develop applications that are able to run on both OSs.

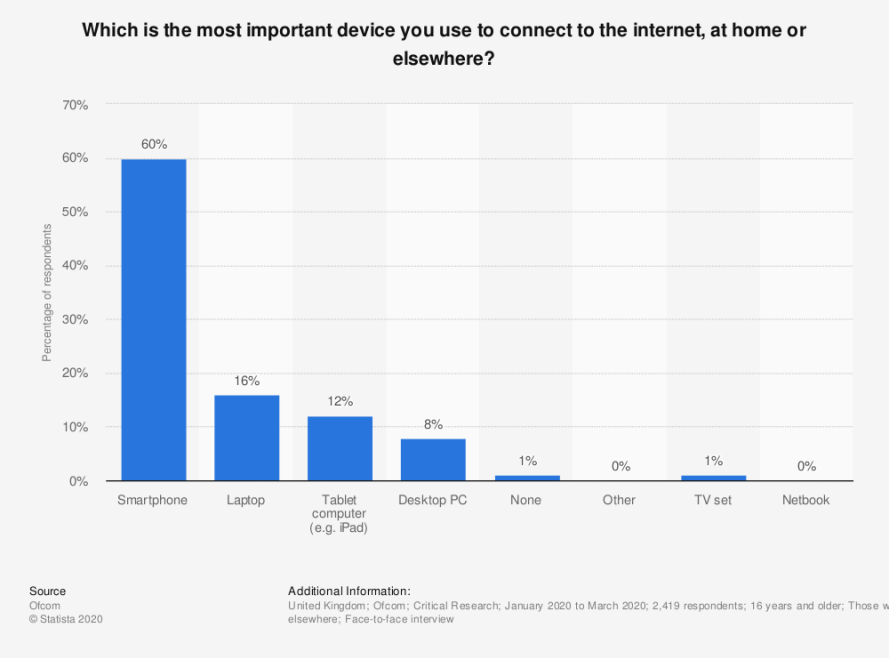


FIGURE 2.12. Devices used to access the internet in the UK in from 2020. Caption and figure reproduced from [49].

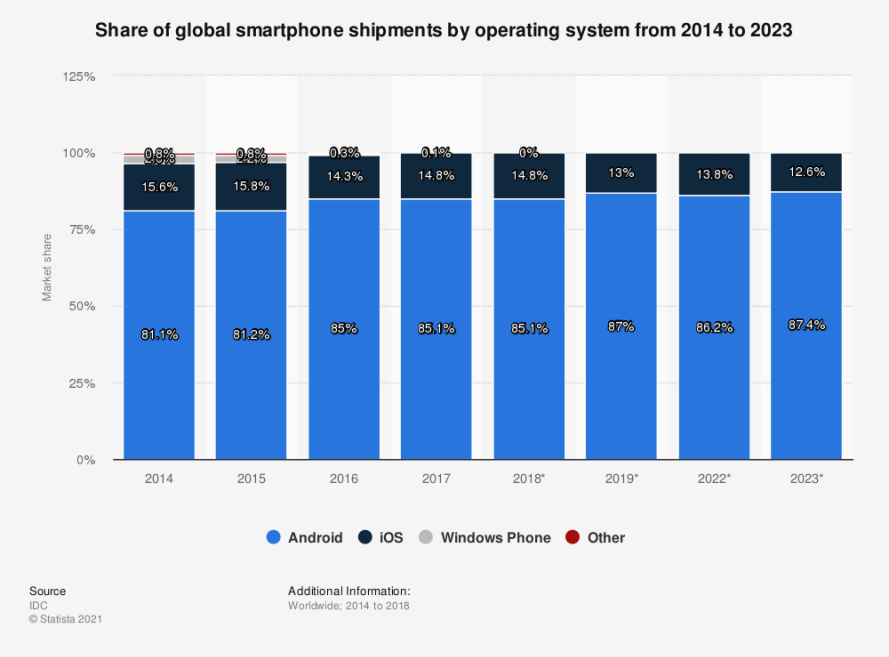


FIGURE 2.13. Global market share smartphone operating systems of unit shipments 2014-2023. Caption and figure reproduced from [50].

CHAPTER 3

DESIGN AND ARCHITECTURE

The system I planned to build for this project has two components: hardware and software. The hardware part is an Arduino board with a camera module; the software part is an Android application that was developed using Flutter. In this chapter, I will outline the methodology chosen for this project and illustrate the details of the design and architecture of the whole system.

3.1 Arduino TinyML Kit

The Arduino TinyML kit is a tool set that helps beginning learners get into the field of TinyML. The kit consists of Arduino's TinyML Shield, or PCB cradle, for the Nano 33 BLE Sense AI-enabled microcontroller board that can attach to external components (Figure 3.1). The Nano board is equipped with a microcontroller and a series of embedded sensors (Figure 3.2). The kit also includes a camera module (OV7675) that can connect with the Nano board via the Shield. The Arduino Nano 33 BLE Sense uses the nRF52840 from Nordic Semiconductors, a 32-bit ARM Cortex-M4 CPU running at 64 MHz, as the processor [1]. The processor also carries 1MB of flash memory and 25 KB of SRAM. These tech specs are important information for TinyML developers, which reflects the constrained memory space for the models.

FIGURE 3.1. Arduino Tiny Machine Learning Kit unbox. Figure reproduced from [2].

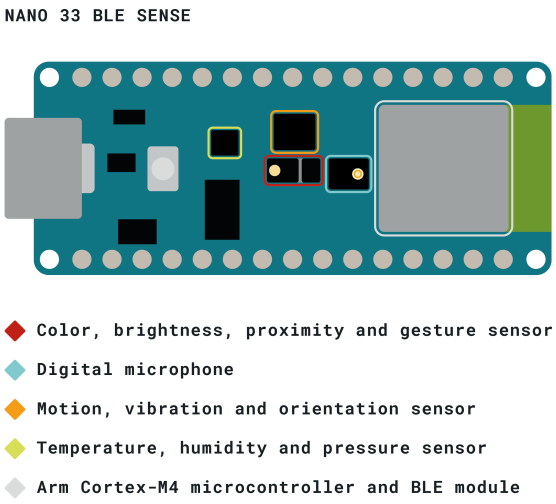


FIGURE 3.2. Embedded sensors on Nano 33 BLE Sense. Figure reproduced from [1].

3.2 Teachable Machine

The Teachable machine is a web-based tool that helps users collect and train their models without specialized knowledge of machine learning [11]. It was built by teams from Google and its first version was released in 2017. In the updated version, the platform supports three common applications in machine learning: images, sounds, and poses. Users can gather the data, train the models, and export their models all by using the web browser (Figure 3.3). Moreover, the platform

supports TinyML applications, so users can export the models in formats that can directly run on the devices. I used Teachable Machine to generate the model for sitting detection.

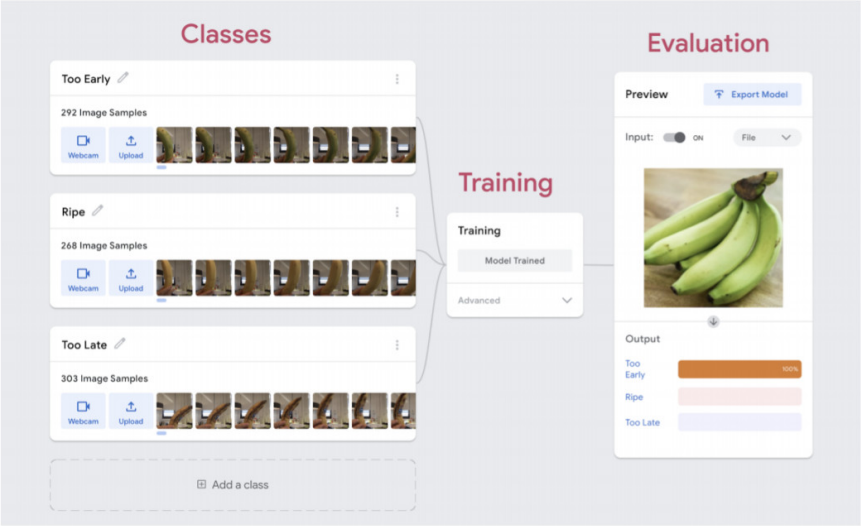


FIGURE 3.3. Teachable Machine interface. Figure reproduced from [11].

3.3 TinyML Application Pipeline

In this project, I followed a standard workflow for building the TinyML application. There are four main steps in the TinyML workflow [62]:

1. Data engineering: collect and preprocess data
2. Model engineering: design and train a model
3. Model deployment: evaluate, optimize, convert and deploy model
4. Make inference

3.4 Data Engineering

Data engineering is a critical element of supervised learning. The model's performance heavily depends on the quality of the data. In addition, data for training the models needs to include enough salient features and other conditions and events that can occur in the system [62]. Data might come from a variety of sources, such as sensors, product users, or contributors. In the process of collecting data, it is important to look at potential licensing or privacy regulations. For my sitting-detection model, I collected two groups of data, which are a person and a chair. The camera is expected to be placed on a table or desk and face a chair. Because the window size of the

camera is small, the person can not fully fit into it. As a result, the image data will be expected to merely contain a person's chest or hand on the table. To make the data more similar to the real scenarios, I collected the images with the same camera that will be used in the system. The Aruduino camera module not only has a limited window size but also takes pictures in grayscale.

To make the data collection process easier, I used the Teachable Machine platform for collecting and labelling the data. It also supports Arduino devices to directly collect the data. This is a key feature for Arduino users, because the devices alone usually do not have screens for users to review the images. I have collected 350 images of an empty chair and 337 images of a person sitting. The dataset is then split into training and testing datasets, which each take up 85% and 15% of the total images, respectively..

3.5 Model Engineering

TinyML is a novel technique for running machine learning models on microcontrollers. An intuitive question about this new approach is how we can fit the sophisticated model onto such a source-limited device. With the progress of the machine learning framework, some useful methods have been proposed to reduce the size of models, and are wrapped in a user-friendly interface for both software and hardware engineers. TensorFlow ecosystem plays a key roles in creating TinyML projects. But still, some challenging issues need to be taken care of in the whole process, including optimize latency, energy usage, and model size. With the help of Teachable Machine, building the model will be much simpler for developers.

3.5.1 Transfer Learning

The Teachable Machine does not build users' models from scratch. It leverages the power of transfer learning. The concept underlying transfer learning is straightforward: a network can borrow some filters from other pre-trained models [44]. For example, the Teachable Machine uses a set of layers from MobileNet v2 [26]. MobileNets is a family of architectures for mobile and embedded vision applications [30]. The first version of the model was released in 2017. The model was trained on the ImageNet dataset, a large dataset that contains 1.4 million images and 1000 different categories [27]. The team from Teachable Machine reused most of the layers from the MobileNet model, except the layers for making classifications. Since our model still needs to do classification between our objects (person and chair), the classification layers or top layers will be left for us. One of the advantages of using MobileNet is that it uses depthwise separable convolution. This mutated version of classic 2D convolutions works in a much more efficient way. To optimize the model size further, Teachable Machine specified a width multiplier of 0.25. The setting will reduce the number of channels in each activation layer by 75% compared to the standard model (width multiplier = 1).

3.5.2 Quantization

What I have illustrated so far is run by standard TensorFlow. After the model finishes training, the Teachable Machine provides the options for users to convert their models into tflite models or models that work on microcontrollers. To better understand what is happening behind the screen, I exported both formats of the model. The Tflite file is an output file after converting the model by using TensorFlow Lite, which fully encapsulates the model and its saved weights. During the model conversion, quantization is applied. Quantization is an optimization that reduces the precision of the numeric values used to represent a model's weights and biases, which by default are stored as 32-bit floating-point numbers [59]. Quantization reduces the precision of these numbers to fit into 8-bit integers, reducing their size by four times. As Figure 3.4 shows, by dividing the 0 to 1 range (in 32-bit numbers) into 256 intervals, the original numbers can be fit into the new scale. During the inference stage, 8-bit numbers can also be turned back to a floating-point representation by multiplying 1/256. Certainly, there might be some numbers which can not be converted back to their original values, reflecting the fact that precision is decreasing. However, because neural networks are primarily driven by matrix arithmetic, the process tends to tolerate small differences in values and suffer no noticeable loss in the output [34]. Furthermore, utilizing an 8-bit representation minimises the amount of computing necessary to conduct model inference, resulting in decreased latency. [59].

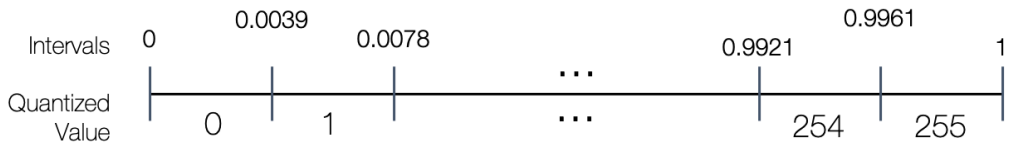


FIGURE 3.4. Quantizing from a 0 to 1 32-bit floating-point range down to an 8-bit integer range. Caption and figure reproduced from [34].

3.6 Model Deployment

The Teachable Machine enables users to directly convert their models into the format of TensorFlow Lite for microcontrollers. As described in the last chapter, TensorFlow Lite for microcontrollers shrinks the model to make it executable on microcontrollers. There are many files which have been generated in the output folder. These files play different roles in running the model on Arduino. The most important one is the model itself. After the conversion of the model data, the model was stored in a C source file. Because most embedded devices do not have a filesystem, the model in a C data array is easier for the hardware to execute and store in flash.

Figure 3.5 shows the overall structure of the sitting detection application that will run on the

Arudino board. In the Arduino world, the main program that runs on the board is called "Sketch", which uses the file extension "ino". Sketch programming uses the C++ dialect, so experienced programmers in C or C++ can quickly get familiar with the syntax and grammar [36]. Basically, an Arduino program performs tasks continuously. Inside a typical Sketch program, two functions, setup() and loop() can be found. When the Arduino board is turned on, the setup() method is called once, whereas the loop() function is executed repeatedly, which acts like an infinite loop. The initialization stage in the figure represents the steps at the top of Sketch and those inside the setup functions, which prepare all the required ingredients to run the loop() function. The steps include: defining a device name for BLE connection, declaring variables, allocating memory space, and loading the model.

Moving on to the main loop. The image provider captures image data from the camera, crop and convert data into size of 96×96, and feeds it to the input tensor. The TensorFlow Lite interpreter runs the model that is stored in a C source file and outputs a set of probabilities. One inference will be made every one second. The detection responder takes probabilities that are produced by the model and makes different responses according to the probabilities. In this project, the responses are the different colour lights of the LED on the Arduino board. If a person is detected, the colour of the light will be green; otherwise, the light will be red. In the next two sections, I will introduce TensorFlow Lite for microcontrollers and BLE.

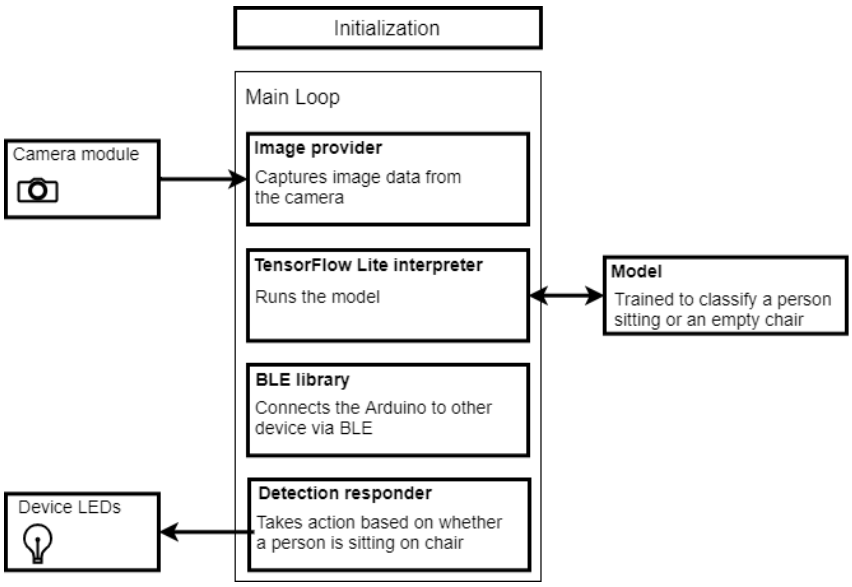


FIGURE 3.5. Overall hardware structure of sitting detection application. Figure revised from [66].

3.6.1 TensorFlow Lite for Microcontrollers

TensorFlow Lite for microcontrollers aims to build models to fit on resource-constrained devices. In other words, there are a number of limitations that need to be overcome, including no operating system dependencies, no floating-point arithmetic support, no dependencies on complex parts of the standard C/C++ libraries, and no dynamic memory allocation [61]. Before the model can be used, the model that is stored in the C source file should be imported via a method named GetModel(). To use the model in the Sketch program, a basic workflow will be implemented (Figure 3.6).

Firstly, the OpResolver allows the TensorFlow Lite for microcontrollers interpreter to access operations. Developers can pull in only the operation implementations they need. Loading only the operations that are required is a way to conserve memory usage. The interpreter is a critical part of running the model. TensorFlow Lite for microcontrollers uses an interpreter design, which stores the model as data and loops through its operations at runtime. The advantages are that developers can change the model without recompiling the code, and the same operator code can be used across multiple different models in the system [64]. The memory area for keeping all the operations in the model is called the "Tensor Arena". The Tensor arena is a piece of contiguous memory that ensures memory allocation will not end up fragmented. That is, the framework guarantees that it will not allocate from this area after initialization. Therefore, the size of the tensor arena should be specified at the beginning, and users should also create enough space to run the model. After initializing the interpreter, the AllowcateTensors() method iterates over all of the model's tensors, allocating memory from the tensor arena to each one. To this point, all the preparation work has been done. To run an inference, the Invoke() method is called on the interpreter. Finally, the output data can be collected and the outcomes can be sent to the responder for making responses.

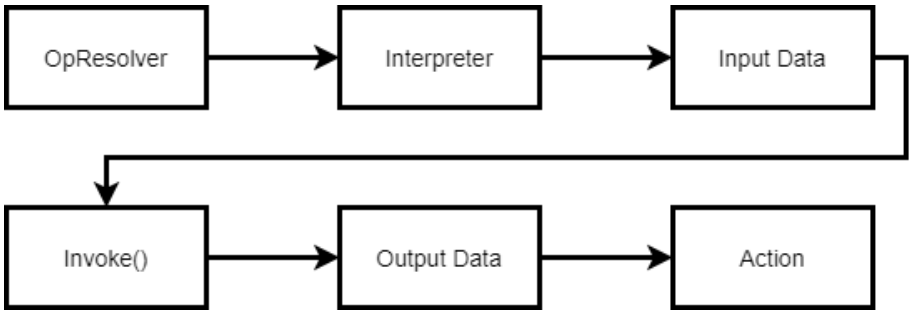


FIGURE 3.6. Workflow of TensorFlow Lite for microcontrollers in Sketch program.

3.6.2 Bluetooth Low Energy

The Arduino board used in this project has a BLE radio module. The BLE is a light-weight version of classic Bluetooth, which was introduced in 2011. Thanks to the explosive growth of mobile devices, BLE has gradually become part of the standard wireless connection methods that are equipped on modern devices. Instead of a continuous stream, BLE only transmits data as needed, which remarkably saves energy. To understand and utilise the BLE module in Arduino, the Generic Access Profile (GAP) and Generic Attribute Profile (GATT) layers of the BLE stack will be introduced here. GAP controls connections and advertising in Bluetooth [60]. It also defines different roles for devices in a connected network. Peripheral devices periodically broadcast advertisements that contain high-level objects transacted within the communication. On the other hand, central devices act like listeners to the peripherals, which can connect to one or more peripherals and receive the data. In this project, the Arduino device is a peripheral which periodically sends model detected results to the central, which will be a mobile phone in my case. The high-level, nested objects that are transmitted between the devices are service and characteristics. These objects are defined under the GATT [60]. A service is a collection of characteristics. For example, sitting detection will be a service in my project. By accessing the target service, I can get the characteristics that are the raw data sent from the peripheral. Typically, a characteristic contains a value, properties, and configuration information. For instance, the binary outputs of the model are stored as characteristics of 0 or 1. To distinguish between different services, each service should specify a 128-bit universally unique identification number (UUID). This ID can be customised and will be defined at the top of the Sketch program [37]. Therefore, when the connection is built between the Arduino and a mobile phone, the software side (mobile app) can specifically find the target device, service, and characteristics.

3.7 Software Architecture Design

Software architecture is an idea that has no an unified definition across the industry [54]. Here, I would like to use the simpler definition, which defines software architecture as the blueprint of the system. Therefore, this section will describe in what way each component will contribute to and meet my project requirements, including solutions and methods to achieve that.

3.7.1 Design

The application is designed to receive data from the Arduino device and provide a user-friendly interface that helps users to review their sitting habits. To achieve the project's goals, I proposed three elements to compose the system: 1) user authentication, 2) data receiving and sending mechanism, and 3) data summarising method. Firstly, user authentication ensures different users can use the same device and application. Different users are able to review their own activity history. Secondly, data communication between the Arduino device, mobile phone, and database

in the cloud is critical. Instead of saving users' data locally, I tend to keep the data on the cloud. As a result, raw data is generated from the device, then sent to the phone as an intermediate step, and uploaded to the cloud in final. When users review their data, the online database will send it back and summarise in the phone application. Finally, data can be summarised in many meaningful ways to indicate users' sitting habits. I think the total time that users have spent on the chair is an intuitive way. Consequently, the application can show users their daily and 5-day sitting time. As Figure 3.7 shows, the mobile application is a core part in the whole system. The application shall provide BLE interface, communication mechanism to the database, and simple interface for the users.

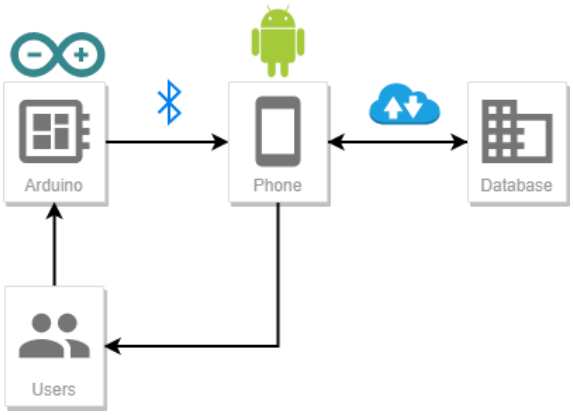


FIGURE 3.7. High level overview of the whole system.

3.7.2 Language, Framework, and Environment

Flutter is the framework that I used for building the Android application in this project. Google announced Flutter 1.0 for cross-platform development in 2018. With a cross-platform framework, a developer writes one program that can run on both iOS and Android. In contrast, native code is written for either the Android or iOS API, meaning that software written for one platform is not compatible with other platforms. However, native development gives the programmer maximum control and makes the app have potentially better performance. Frameworks like Flutter act as a bridge between a developer and another API. We do not write the code specifically for Android or iOS when we write a Flutter program. In fact, we create code that can be converted into API calls for either system. Flutter is built on top of the programming language Dart. Dart is an object-oriented language which is also maintained by Google. The core idea in Flutter is that everything is a widget. In Figure 3.8, every UI component, such as buttons, text, and images, is controlled by a small widget. The nested widget structure is how Flutter organizes the UI. Just like many other modern frameworks, there is also a big developer community behind Flutter. Namely, lots of functions might already have been developed by others, so we can search for libraries in advance and utilize them. For example, I used a BLE library for Flutter in my project

to easily handle BLE connections. Flutter version 2.2.3 is used in this project.

Android Studio Arctic Fox is used as an IDE for developing Flutter in this project. Additionally, I used my phone to test the app. The phone is a Google Pixel 3a, which has a processor of Qualcomm Snapdragon 670 and 4 GB of memory. It also has WiFi and Bluetooth connecting capabilities, which I can connect with my Arduino board.

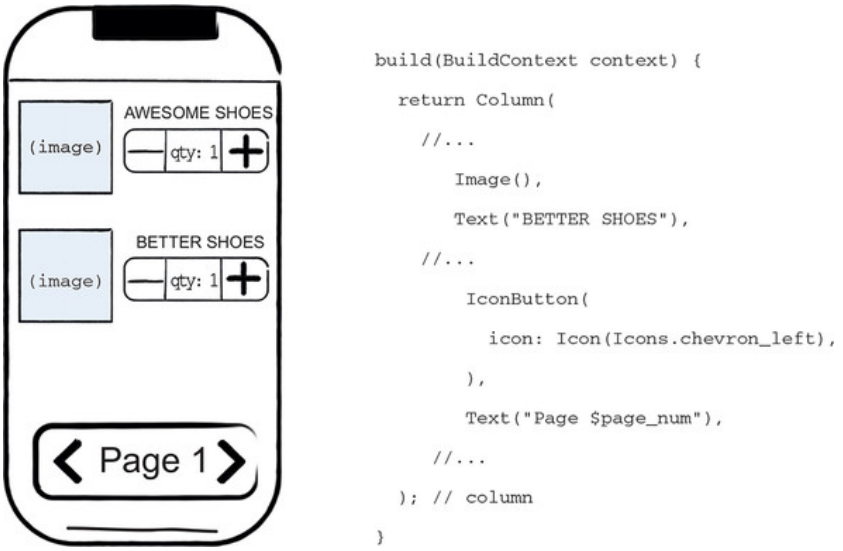


FIGURE 3.8. Examples of common layout widgets in Flutter. Caption and figure reproduced from [68].

3.7.3 User Interface

Designing a good user interface is a critical step in developing a new product. In particular, I would like to build a self-monitoring system, which means users can actively and progressively improve their sitting habits by using the system. A well-designed user interface can enhance the user experience and encourage users to use the application regularly. From the VitaBit example in Chapter 2, they have built a nice and intuitive application from which I could extract some useful features for my project. In Figure 3.9(a), a simple welcome page is prepared for the user to choose to log in or register. Log in and registration pages look similar, which let the user enter their e-mail as a user name and password (Figure 3.9(b)). Figure 3.9(c) is the first version of the time tracker page, which is also the home page that the user will encounter after entering the app. Basically, this page shows today's total sitting time of the user and provides a basic chart for the user to compare the times in the past five days. Another feature is the bottom navigator, which serves as a menu, letting the user switch between the sections. In the first edition, I planned to create a calendar page for searching for the sitting time of a specific date, a summery page for building a dashboard to include more charts, and a setting page for controlling the user's

settings. The top bar has a Bluetooth icon which will be an entry to scan for the Arduino device and connect with it. The back arrow at the beginning of the top bar is a button to leave the page and send the user back to the welcome page, which means the user will be logged out. The top bar and bottom navigator will persist on the screen wherever the user changes the page. The design of the user interface will be very likely changed during the development stage. However, the core functions will be kept in the app and represented in alternative ways.

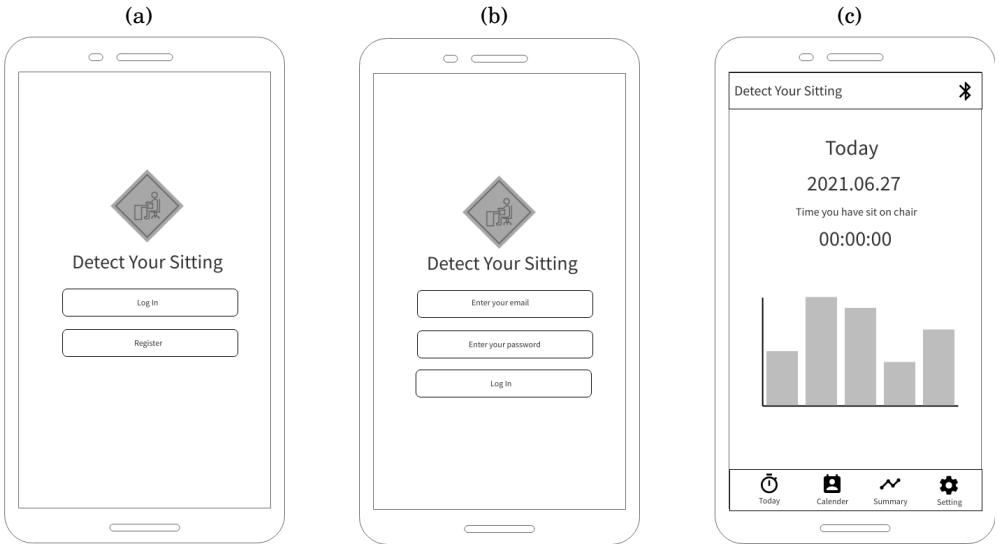


FIGURE 3.9. Wireframe of different pages in the app. (a) Welcome page. (b) Login page. (c) Time tracker page.

3.7.4 Cloud Database

As mentioned in 3.7.1, I will store the user data in a cloud database. The user data includes user name (e-mail), user password, and user sitting activity. A user name and password are used for authentication, which ensures a specific user only gets his/her own data. The cloud database I chose for this project is Firebase. Just like Flutter, Firebase is also a product of Google, so it comes with well-written online documentation that illustrates how to use it with Flutter. Firebase is free for low volumes, including 20K writes, 50K reads, and 20K deletes per day [25]. In this project, I will use two tools from Firebase. Firstly, Firestore is a database with an API to read and write data [51]. Many built-in methods from the library can directly be called in Flutter, which significantly minimizes the communication process between the front and back. Next, utilizing Firebase authentication, users can safely log in to the app using their social accounts or a pair of user name and password. Traditionally, setting up an authentication database is a tedious process because developers must consider dozens of security issues, such as authentication logic, password hashing, and coping with forgotten passwords [51]. Another key feature of Firebase

is setting the internal rules for the database. Developers can specify different permissions for the user with or without verification. For testing and learning purposes in this project, the free volume is enough for me. Even so, if one day the project needs to scale up the usage, I can update the plan, and everything will be production-ready.

3.8 Revisit the System

The design plan for the whole system includes an Arduino device, a mobile app, and a cloud database. The main data that streams within this system is a binary value that represents whether the user is sitting on a chair or not. This data stream starts from the Arduino device, which is equipped with a machine learning model, and passes to the mobile app. Although the Arduino will continuously detect the object after the power turns on, the data will only start to sync with the cloud database when the device is connected to the phone via BLE. In addition, the device sends data every 15 seconds. After the BLE connection is established, the fronted and backend can begin to communicate. A number of methods have been developed to read and write from/to the Firebase. The summarising function is used for determining how many positive data points within a minute will make the minute count. The main pages of the app are stateful widgets, which can re-render and dynamically change the content when new data streams in.

3.9 User Testing

User testing aims to test if the system is working as expected according to the design plan. A number of users will be invited to use the system, and they will be asked to give feedback in the form of a survey. There are two levels of testing. Firstly, testing the machine learning model on the Arduino. Because the model is trained on a limited training dataset that only collected images of myself, I would like to know if the model still performs well when different people are shown in the images. Also, there is mainly one chair that was collected in the training dataset, so a different chair will be tested as well. Secondly, users will test the whole system, including both the Arduino device and the mobile application. Users will be asked to register and log into the system and will be monitored by the system. The total time that users have seated will be measured by the system and by manual estimation. The test approximates lasting about 10 minutes per user, which includes sign up an account, sitting for two minutes with different poses, and answering the survey.

3.9.1 Setup

The system that will be tested includes an Arduino board with a camera module, which connects with my personal laptop via USB. Connecting with the computer not only provides power to the device, but also helps the developer to monitor the live data output in the Arduino IDE. I had

signed up accounts for all the test users before the testing began. Therefore, all test users can login to the app and start to use the app. I prepared a room that was different from the one I had used for collecting the training dataset. That is, the height of the desk, the appearance of the chair, and even the light intensity of the room were all different from the training set. To ensure the camera is placed appropriately, I downloaded the Sketch programs for the Arduino and my computer from the official GitHub page of Teachable Machine [7]. The Sketch programs generate a window on the computer that displays the view of the camera on an Arduino. With the help of the programs, I adjusted the place for the camera, making sure that the test user and the chair were included in the camera's view.

After the hardware was ready, I introduced the app to all test users. As a result, users were able to use the app by themselves and review their sitting activities. The testing lasted about 2 minutes per user, and each user was tested one by one. In the first minute, the user was confined to changing three specific poses. These three poses were users putting their hands on the keyboard, resting their chin on their hand, and leaving their hands from the desk. The postures were those I had mainly added to the training set, so the model supposed to perform well on identifying them. The test users could change their poses without constraint during the second minute. After the testing is finished, users can review the sitting time, which should be 2 minutes. If the time is less than two minutes, false negative results were generated. The final step of testing was asking users to complete the survey on Google Forms.

3.9.2 Test Users

The system I built targets any users who are interested in changing their sitting habits. I invited four participants to attend the test. Three of them were postgraduate students from the University of Bristol and one was working in the hospitality industry. The three test users from the university stated that they have spent considerable time sitting in front of their desks since the beginning of online learning. Therefore, I think they might show more interest in using the system. The test user who works in the hospitality industry described her work generally not requiring long time sitting. However, she still thought the project was useful for helping users understand their sitting activities. All the users were made anonymous in the testing (Table 3.1).

Test User	Job
User A	Student
User B	Student
User C	Student
User D	Hospitality Industry

Table 3.1: Test users.

Chapter 4

Results

This section covers the final product of the whole system, including hardware and software parts. The key feature will focus on how the Arduino communicates with a mobile phone and how to use Firebase as the backend database for managing different users' data. In addition, the detailed information about the critical functionalities will also be illustrated in this section.

4.1 Hardware Development

As described in the last chapter, building TinyML applications is a complicated process. Developers not only need to get familiar with the machine learning paradigm, but also do some Sketch programming on Arduino. To facilitate work in the pipeline, I decided to use Teachable Machine from Google to collect data and build the model. After training the model, I revised the output files from Teachable Machine and integrated the BLE library to implement wireless connection capability.

4.1.1 Building Model

To train the model, I collected 337 images of a person sitting on a chair and 350 images of an empty chair. The person is myself and the chair is the one in front of my desk. As Figure 4.1 shows, because the camera's view is limited, only a part of the upper body can be fit into the images. In addition, I wore different tops to increase the diversity of the data and intentionally stood behind the chair to train the model that will not be biased toward the interference. Because the Arudino device is expected to be placed on a desk, I also considered if there are some obstacles blocking the view of the camera. I labeled a number of blank and full black images as a person to mimic the

blocking conditions. As a result, the model might misclassify an obstacle as a person, which causes false positive results. Therefore, users should be careful not to accidentally put any obstacles between the camera and the seat. Also, I found that light intensity will particularly affect grayscale images, where strong light intensity will make images look overexposed. Consequently, I added the images that were taken under different light intensities to reduce the effects.

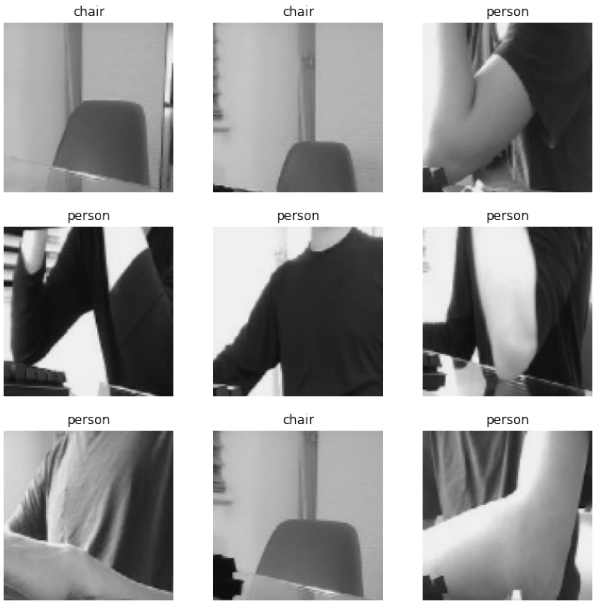


FIGURE 4.1. Sample images in training dataset. There are two classes in the dataset: person and chair.

Before starting to train the model, there are three adjustable parameters on the Teachable Machine platform. Firstly, epochs determine the number of times that training data will be run through the network during training [63]. A common pitfall of setting the epochs is that too many rounds of training on the same dataset might give rise to overfit. Additionally, the network will stop improving at some points even if no overfitting occurs. I set the epochs to 20 and observed the model performance. The second parameter is batch size, which I set to 16. A batch is a set of samples used in one round of training before measuring the model's accuracy [63]. For example, I have 337 images of a person sitting for training, so after dividing by the batch size, there are 22 batches that can fully cover all the data. In every iteration of the 20 epochs, the system will randomly choose 16 images, run inference on them, estimate the loss, and update the network once per batch. According to the Teachable Machine, users typically do not need to change the size of the batch to get well-performed models. The last parameter is the learning rate, which I left with a default value of 0.001. With a high learning rate, the weights and biases are tuned at a high frequency in each iteration, resulting in convergence happening fast. However, fast hopping means the model might skip some ideal values. In contrast, the model might achieve a better result in the end with a lower learning rate. To get the best model performance, trial and

error on these three parameters is needed [65].

The Teachable Machine splits users' samples into training and test datasets. The training dataset accounts for 85% of the samples and is used to train the model. The test dataset uses the rest of 15% of the samples that were never presented in training. By using the test dataset, Teachable Machine can show us some metrics to help us evaluate the performance of the model. In table 4.1, I got both 98% accuracy in predicting a person and an empty chair. If a person sitting is a true positive result, we can find only one false negative and one false positive in prediction based on the confusion matrix (Table 4.2). Additionally, the accuracy during the training gradually increased (Figure 4.2(a)). Finally, the loss significantly dropped in the first five epochs and stayed low till the end of the training (Figure 4.2(b)).

Class	Accuracy	# Samples
person	0.98	51
chair	0.98	53

Table 4.1: Accuracy per class.

		Prediction outcome	
		person	**chair**
Actual value		50	1
		1	52

Table 4.2: Confusion matrix.

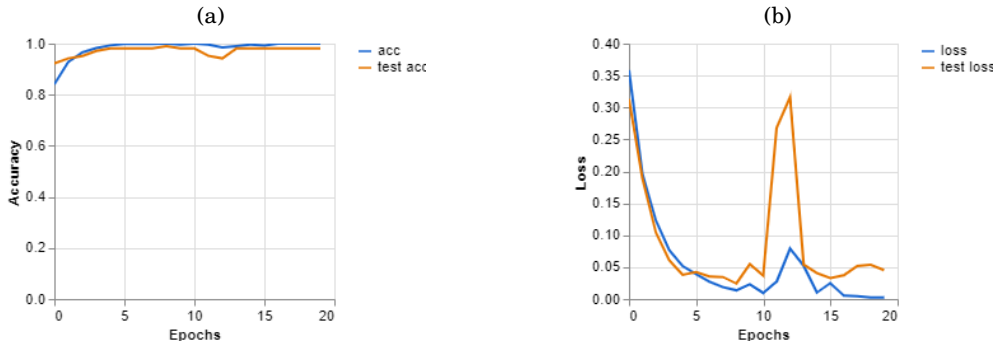


FIGURE 4.2. Model performance evaluation. (a) Accuracy per epoch. (b) Loss per epoch.

4.1.2 Deploying Model

The overall performance of the model seems to be superior. The model is stored in a C source file and in the specialised format of FlatBuffers. FlatBuffers is able to be embedded directly into flash memory and accessed immediately, meaning no parsing or copying is required. The format is compatible with the hardware because it stores an array of bytes that holds all of the information about the model, including operators and weights (Figure 4.3). FlatBuffers is decoded by a schema file that defines the data structure, working along with a compiler that turns the schema into native C++, and generates code to access the information in the model byte array.

```
alignas(8) const unsigned char g_person_detect_model_data[] = {

  0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
  0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00,
  0x03, 0x00, 0x00, 0x00, 0x24, 0x1c, 0x08, 0x00, 0x94, 0xe0, 0x06, 0x00,
  0x7c, 0xe0, 0x06, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
```

FIGURE 4.3. First five lines of byte array of the model. The byte array is generated by TensorFlow Lite.

In the setup() function of my Sketch program, I kept the common operations for the model that was generated by Teachable Machine. As mentioned in the last chapter, TensorFlow Lite for microcontrollers lets users pull in the desired operations rather than all the available ones, which will help to reduce the working space. The end result of the model is the output of the softmax layer. These are two numbers, one for each of "person" and "chair". The numbers are the scores for each class, and the one with the larger score is the model's prediction. After the model is prepared, the BLE module should be set up. The Arduino device acts as a peripheral in my system, so I named the device "SittingDetection" to let the central device (the mobile phone) discover it. Before finishing all the setting process, the device will start to advertise its service and wait for the connections.

The loop() function of the Sketch program is the place to run the model. The device makes an inference in every iteration of the loop even if there is no BLE connection. The loop runs every second, so the program can have time to initialize the interpreter, make an inference, and make responses. The prediction results are stored as scores and assigned to two variables. The scores are actually the probabilities in signed 8-bit representation. In the last chapter, I mentioned TensorFlow Lite applied quantization to simplify the calculation and reduce the storage space. A normal probability is presented in a range of 0 to 1, but the range will be divided into 256 intervals after quantization, which has a minimum of -128 and a maximum of 127. In other words, -128 equals a probability of 0, whereas 127 equals a probability of 1. The scores are then sent to the responder. Inside the responder, a Boolean function is defined to check if the score of the

person sitting is larger than the score of an empty chair. The binary outputs are the arguments to the function that can change LED light colours to reflect the results (Figure 4.4). Finally, the binary results are written as a characteristic that will be delivered to the central device. In Figure 4.5, the serial monitor in the Arduino Editor shows messages for device connecting and model predicting status. If the detection result is 0, it means no one was detected. In contrast, if the result is 1, it means a person has been detected.

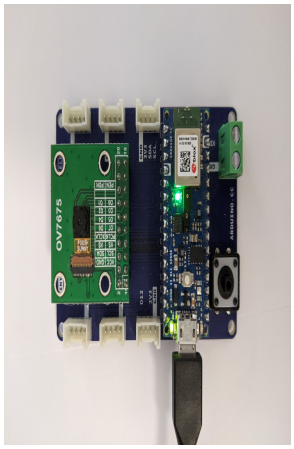


FIGURE 4.4. Light of the LED on the Arduino Nano board. When a person is sitting on a chair, the light will turn into green. Otherwise, it will be red.

```
Bluetooth device active, waiting for
connections...
person : 127
chair : -128
person : 126
chair : -126
person : 111
chair : -111
Person is detected: 1
person : 111
chair : -111
Person is detected: 1
```

FIGURE 4.5. Serial monitor shows messages from the device. After the device is connected, "Person detected: " message starts to show up.

4.2 Software Development

In the last chapter, three fundamental pieces for building my application on the Android phone were proposed: 1) user authentication, 2) data receiving and sending mechanism, and 3) data summarising method. Apparently, these processes need data communication methods between the frontend and backend. In this section, I will start by introducing page widgets, which represent

every single page in the application. Next, I will focus on the BLE widgets that make BLE work between the Arduino and the app. In particular, I used the example project from a third-party plugin with some modifications to facilitate the development. Finally, I will illustrate the detailed structures and methods that make different parts of a system work together.

4.2.1 Page Widgets

As mentioned in the last chapter, everything in Flutter is a widget. It is common to organise each screen that is shown in the app as a big widget (Figure 4.6). In my app, users first arrive at the welcome screen, which has two options: logging in or registration (Figure 4.7(a)). Each button will bring the users to the corresponding page. A new user can register an account by filling in his/her e-mail and password. Firebase limits the password length to at least 6 characters. After the account is successfully created, a snack bar widget will show up at the bottom of the screen to notice the user (Figure 4.7(b)). The screen will then jump to the log in page, asking the user to log in again. Alternatively, a registered user can log in to the application via the log in screen. The system will check if the user has entered the correct user name and password. The different message dialogues will pop-up and ask the user to retry according to different types of error; namely, incorrect user name or password (Figure 4.7(c)(d)). After users finish logging in, the app shows the time tracker screen with an app bar on the top and a navigator at the bottom. The top and bottom bars are controlled by the BottomNavigator. No matter which pages that users have browsed within the app, those bars will stay at the top and bottom. The top and bottom bars were implemented by following the design plan in the last chapter. For the top bar, the back arrow will log out the current user and bring the user back to the welcome page. The alert screen will pop-up to make sure if the user really wants to log out (Figure 4.7(e)). At the other end of the bar, there is a Bluetooth icon, which is controlled by BleScreen. The BleScreen is a page for users to scan, connect with the Arduino, and sync their data with the cloud. The page can also jump to another page, BleLogScreen, which synchronizes with the database and serves as a log record. This log page can let the user review the raw data in the cloud and provide an interface that helps the developer to inspect the data stream. The detailed methods that are used for BLE connection will describe in next section.

Users switch between the main three pages: the TimeTracker Screen, LogScreen, and ManualScreen through the bottom navigator. These three pages are the places for users to review their daily and 5-day sitting conditions and manipulate their data. Furthermore, two internal data structures take care of user data, which are the UserData class and the TimeSeriesDaily class (Figure 4.6). The UserData class matches the storing format in the cloud database, including three data fields: value, time, and user. The TimeSeriesDaily class is an entity with two fields: date and total time, which is used for storing the total time that a user has sat on a specific date. The time tracker screen shows today's total sitting time of the user. There are two buttons: Fetch

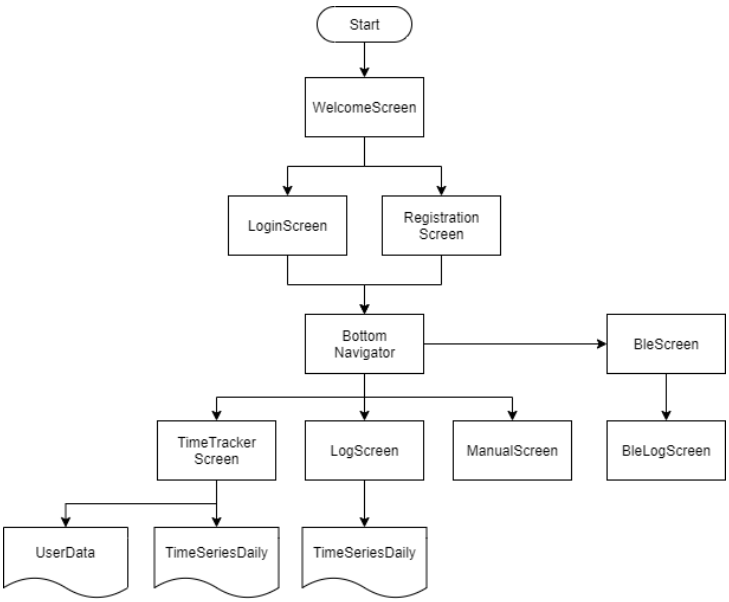


FIGURE 4.6. Page widgets in the app.

User Data and Summary Plot (Figure 4.8(a)). Users can get their data from the cloud database by pressing Fetch User Data, and a progress indicator will appear to let users know the data is under fetching. After the data is fetched, the time will automatically update. The Summary Plot is another button that asks the database to send the data and triggers summary functions to calculate the 5-day users' sitting profile. A time series plot will be shown after the summarising process is finished. The plot is generated by using the plugin from the Flutter library of Charts. The Charts library provides various types of charts that developers can craft in their apps. It is interesting that both buttons on the time tracker screen change the page dynamically. This mutable behaviour is controlled by the StatefulWidget class in Flutter. Stateful widgets come in handy when developers want some compositions within the interface to change dynamically, e.g. due to depending on some system state. In my case, the state has changed while users' data arrives from the cloud.

The log screen is a second page in the bottom navigator bar. The page shows a simple data table with three columns: weekday, date, and time (Figure 4.8(b)). Essentially, it can be viewed as the raw data of the chart on the time tracker page. Users can review their sitting time over the past four days and today. Unliking users need to press the buttons to refresh the states on the time tracker page, the log screen uses the RefreshIndicator widget to support scrolling down the screen to refresh. In order to aggregate users' 5-day sitting data into a table format, I used the DataTable widget and arranged the data into columns. A useful feature of the table is sorting the data by the daily sitting time. In other words, users can easily sort the table into ascending order based on their sitting time.

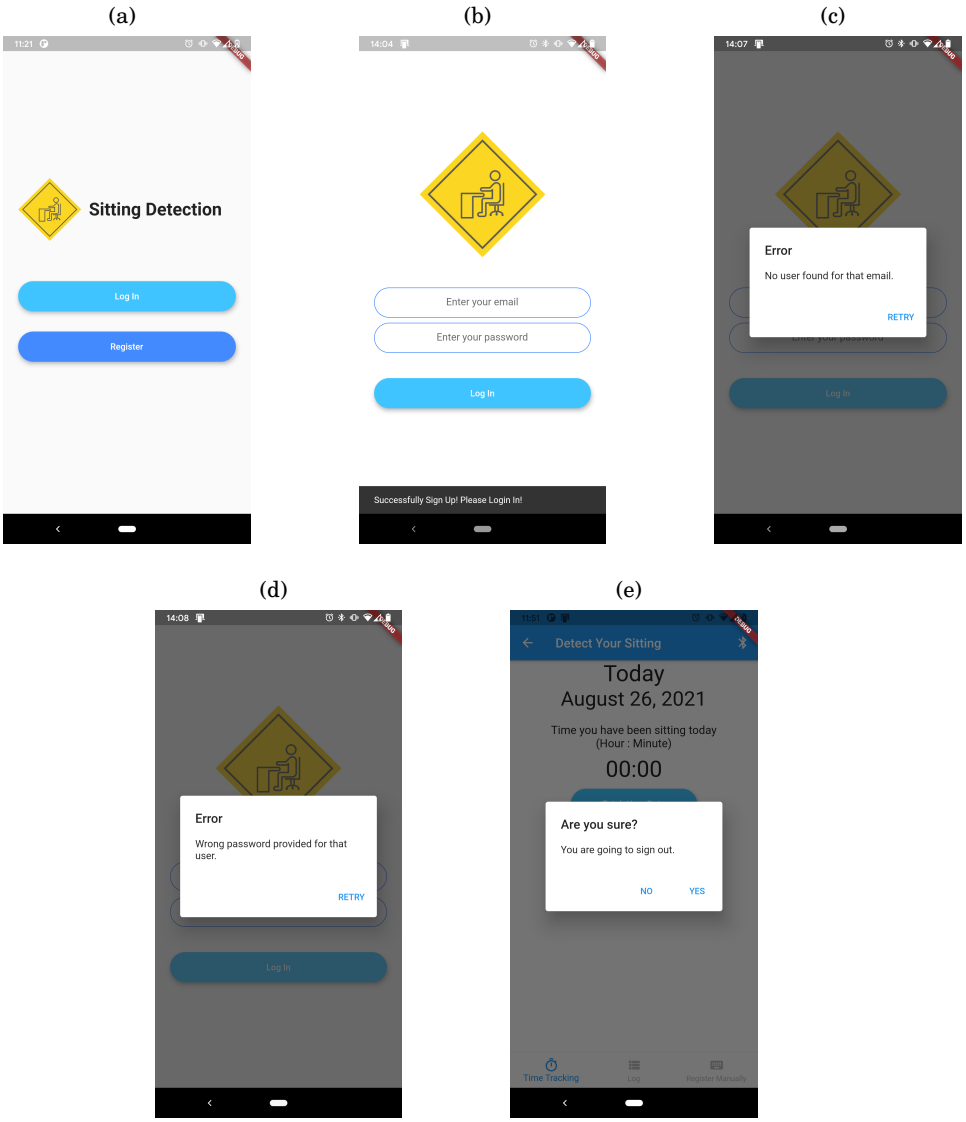


FIGURE 4.7. Different widgets are used under page widget. (a) Welcome page. (b) Snack bar widget shows the successfully sign up message. (c) Alert dialog widget shows error message of incorrect user name. (d) Alert dialog widget shows error message of incorrect password. (e) Alert dialog widget confirms if the user want to log out.

The manual screen is a third page in the bottom navigator bar. This page provides options that users can manually add, modify, and remove the sitting records (Figure 4.8(c)). To add the time to a date, users select the date via the DatePicker widget. The DatePicker is a Flutter material widget that appears in a dialogue with a Material Design date picker when an inbuilt function (showDatePicker) is called. After selecting the date, users choose a time that they have spent on a chair on that specific date. The minimum time that users can choose is an hour and up to 12

hours. Next, users press the "ADD" button to add the time. If there is already a time on the date, the system will throw a warning message, reminding users that they are going to overwrite the existing data. If users choose the "yes" option in the warning dialogue, the data will be updated. In contrast, choosing the "no" option will not change the data. To remove the data from a certain data, users first pick a date from the DatePicker widget and then press the remove button. If there is no data available on the date, a snack bar will show up, indicating nothing has been removed.

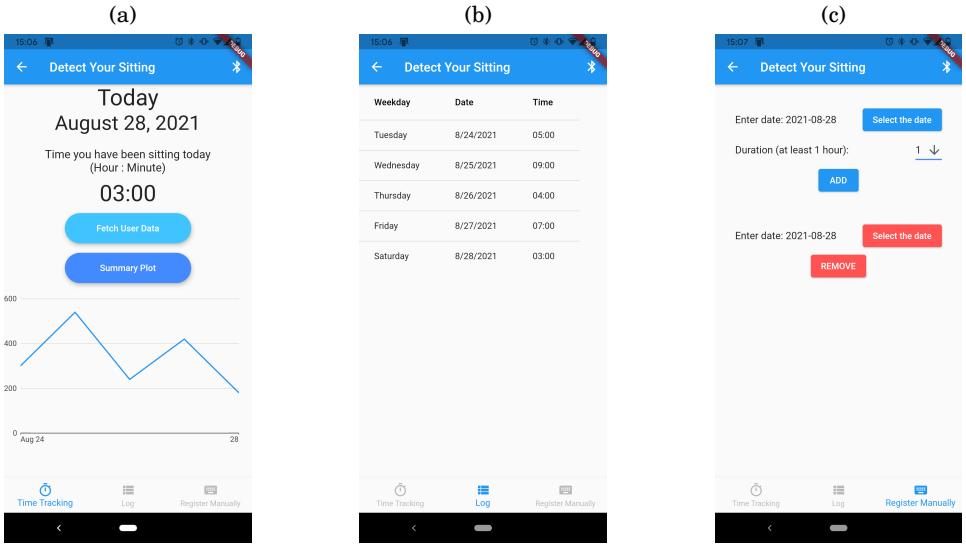


FIGURE 4.8. Three main pages in the app. (a) Time tracker screen shows the time and the chart. (b) Log screen shows 5-day sitting times. (c) Register manually screen shows options for adding and removing the data.

4.2.2 BLE Widgets

I mentioned BleScreen in the last section, which is the place for the app to connect with the Arduino device through BLE. To make a connection through BLE, I used a library called FlutterBlue in Flutter. I also borrowed the example from the plugin's official GitHub as a template to build my BleScreen widget [18]. The plugin has several useful APIs that are associated with different items in the BLE hierarchy and can intuitively be implemented in Flutter. For example, the BluetoothDevice class represents the nearby device that users want to scan for and connect to, and it encompasses several methods, such as connect, disconnect, state, etc. Before the system can start scanning for the device, a StreamBuilder widget should be created and listen to the state of the Bluetooth. The StreamBuilder is an important component in asynchronous programming in Flutter. As its name indicates, StreamBuilder keeps listening to a data stream, and updates and re-renders the widgets when the stream receives new information. If the Bluetooth state is

not on, a BluetoothOffScreen will show, reminding users to turn on their phone's Bluetooth signal.

To start scanning for the device, users press the floating button in the right bottom corner and wait for the update of the device list. Users can also scroll down the page to search for the new devices. Since the name of my Arduino device has been set as "SittingDetection", users can connect to the device once the name shows up (Figure 4.9(a)). After the connection is established, the device will appear at the top of the page and show an "OPEN" button (Figure 4.9(b)). The button will bring users to the device page. Inside the device page, users should see the device is connected and has a refresh button beside the device name. The refresh button is for discovering the services. As described in the previous chapter, a peripheral device can send multiple services to central devices, and every service can also have multiple characteristics. The BluetoothService class is an object that represents the BLE service in the FlutterBlue plugin. The app can precisely find the target service which contains our sitting data, because I have specified the UUID of the service in Sketch programming. Therefore, only the target service will appear as a tile on the page. It might take a few seconds to discover the service. Users then expand the service tile and prepare to sync the detected data from the Arduino. Similarly, each characteristic can specify an ID, so the app will search for the preset ID of the target characteristic. The last step to syncing the data with the cloud database is that users need to press the sync button under the characteristic tile (Figure 4.9(c)). This is an important step towards using the whole system, because only if the data can be sent to the cloud, can the data be kept. The Arduino device itself does not store any temporary data, so if the sync process does not activate, there will be no data reserved. Obviously, it is not intuitive enough for users, so I will discuss the possible solutions in the next chapter. After the app has synchronised with the cloud database, a characteristic can be automatically notified by using the setNotifyValue() method. This method is a built-in function in FlutterBlue, which is designed for a BluetoothCharacteristic object. As a result, if there is any new value from the device, BLE will notify the app.

4.2.3 Frontend and Backend

Until now, I have covered the screen and BLE widgets. However, to make the whole system work (Arduino, mobile app, and cloud database), the methods for communicating with the frontend and backend should be implemented. Figure 4.10 shows the overall structure of how data is read and written from/to the Firebase. Two collections were built in my Firestore database, which are bluetooth_data and manual_data. The former one stores the sync data from the Arduino device, and the latter one keeps the manually register data. After the app syncs with the Arduino, the data will be sent to the Firestore every 15 seconds. Every new added data is a document under the collection, and every document has its own unique auto-generated document ID and three fields: person, time, and user. All three fields are stored in the format of strings. The person field is the detected value from the Arduino, which will be either 0 or 1, representing without and with

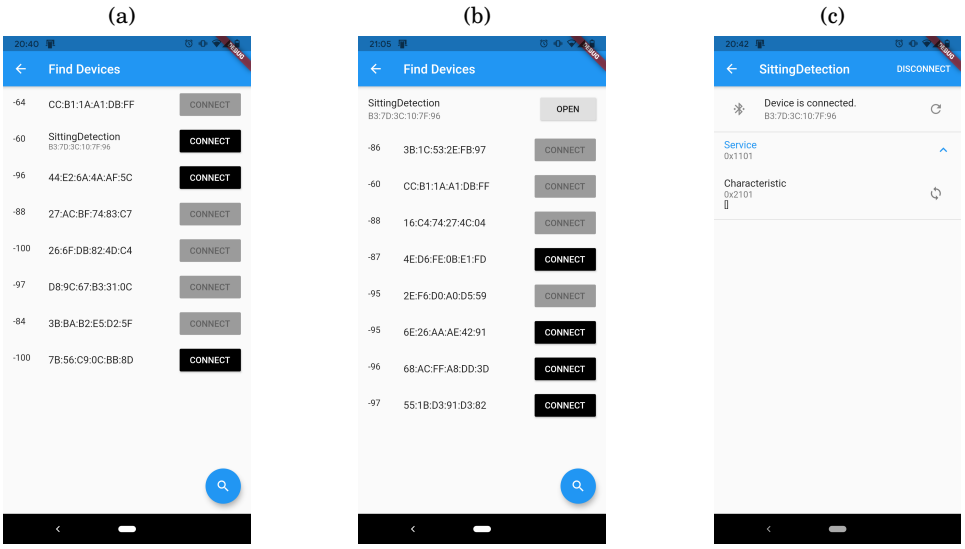


FIGURE 4.9. Using an example project from the official GitHub page of FlutterBlue [18]. (a) Scanning for the device of SittingDetection. (b) SittingDetection device is connected. (c) Characteristics tile is used for syncing the data with the cloud database.

a person detected respectively. The time field contains a string of date and time, which time is recorded in the format of "hour:minute:second". The user field is the user's e-mail, which is used for authentication. As for manual_data, every document is a record when users manually add them through the manual screen. Each document has a special ID, which is composed of data and user e-mail. This unique ID composition is useful for querying the database. Similarly, every document under manual_data has three fields: date, duration, and user. The date field stores the date that users have picked through a date picker. The duration field is the hour that users have selected. The user field is the same as the one under bluetooth_data.

It is clear now that there are two kinds of collections stored in the Firestore. One represents the data collected from the device, the other comes from the users. Therefore, a straightforward question is how the app decides which one to use when the date exists in both collections. The decision-making statement resides in the method of checkManualRecord, and it will always favour the manually registered data. That is, users should add their manual data with caution, because the manual data will mask the data from the device. The checkManualRecord returns a Boolean value, so if it returns true, it means there is a manual data existed on that day. Accordingly, it can call the method of getManualRecordData to get the data from the collection of manual_data. Otherwise, it can call the method logStream to collect the data from the collection of bluetooth_data. These two methods use query commands from Firebase. The getManual-RecordData directly searches for a document ID, because the ID is a special combination of data

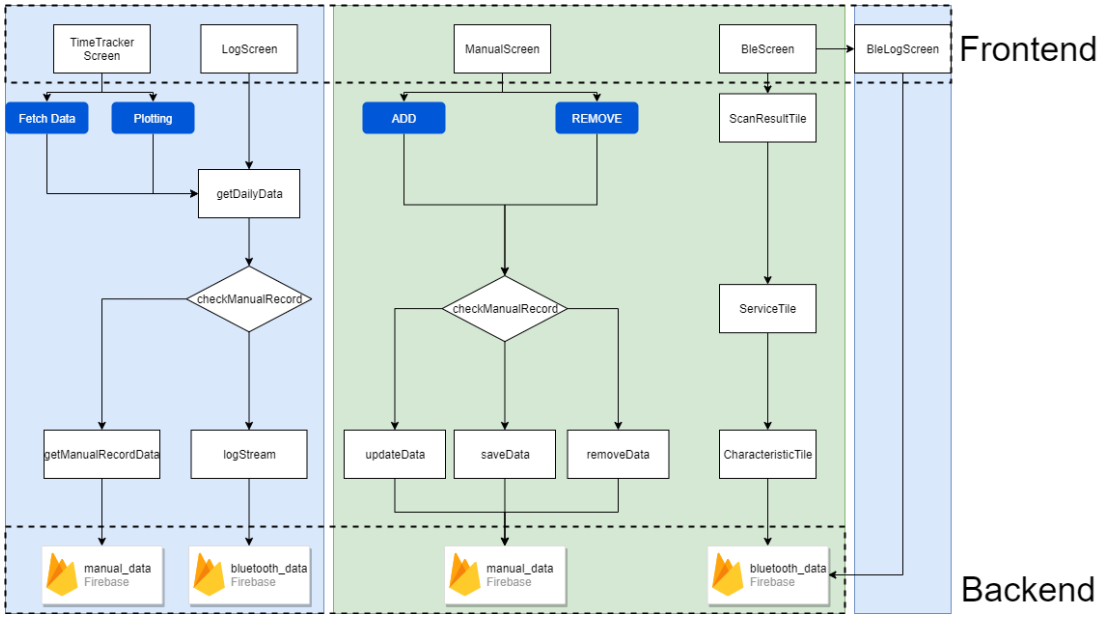


FIGURE 4.10. Key methods used in communicating between frontend and backend. Blue background represents the frontend reads data from the backend. Green background represents the frontend writes or deletes data to/from the backend.

and user e-mail, which can be utilized for querying. The return value is the duration in string format. On the other hand, logStream can not search for a document ID because the ID was auto-generated. It finds the target date by filtering the "date" field, and only keeps the target user's data with the value true. The return value is a UserData object that I introduced in section 4.2.1. To sum up, logStream is designed to collect all the data points on a specific day of a user.

In Figure 4.10, the plotting function in TimeTrackerScreen and LogScreen use the method getDailyData to summarise the total time of a day that the user has seated. This method is the place where the outputs of getManualRecordData and logStream are turned into an object called TimeSeriesDaily that I have described in section 4.2.1. Summarising the daily data from manual sources is a relatively simple task because it contains the total sitting time. As for summarising the data from the device, the method of updateTotalTime is applied to the output of logStream. Due to the Arduino sending data to the cloud every 15 seconds, it is reasonable that there are about 3 to 5 data points per minute. Therefore, if there are at least two data points within a minute classified as true value, which means a person is sitting, that minute will be counted as sitting. The updateTotalTime calculates the number of valid minutes by filtering every document in the collection, and generates the total time that the user has seated in a day. In summary, getDailyData wraps up the methods of checkManualRecord, getManualRecordData, and logStream. Thus, it can return the total sitting time of a day from the appropriate dataset in Firestore.

Compared to the TimeTrackerScreen and LogScreen, the ManualScreen has a simpler data processing mechanism. Because this page only focuses on manual registered data, it can directly send the commands to operate the data in Firestore. I encapsulated three common data operations into three methods, which are updateData, saveData, and removeData. These three methods are based on the built-in functions of Firestore, which are able to locate the target document and perform the actions. The BleScreen used many BLE widgets that I introduced in the last section. The page only needs to interact with the data that comes from the device. Finally, the BleLogScreen reads data from the collection of bluetooth_data and sorts the log data by time in ascending order.

4.3 User Testing Result

The testing aims to find out if the machine learning on the Arudino is adaptive enough to recognise a person other than myself, and if the whole system works according to the design plan.

4.3.1 Test Machine Learning Model

As described in the testing plan (section 3.9), four test users were asked to sit for two minutes under the detection of the system. Because the Arduino sends data every 15 seconds, there were at least 8 data points in two minutes. In other words, the model's performance was determined by those 8 data points. The number of data points might not exactly equal 8 because of the latency of the syncing process between the device and the app. Table 4.3 shows the testing results. If we observe every single data point that is written into the database, there is still the possibility of 20% to 40% of false negative results, which means there is a person on a chair but the model predicts there is no one. I think the result is not so bad because the model was solely built on the images of myself, but it seems to recognise other people. Additionally, if a person is moving when the camera captures the image, the image could be blurred, which may bias the prediction. Therefore, I set the criteria for identifying a person sitting within a given minute is that there are at least two data points in a minute. This setting can tolerate some false negative results during detection. From the results of total time, we can find that although there were some missing detection, the app still correctly returned two minutes of the time. When we look at the false positive results in the first and second minute, we can see there are no differences in the results of users A, B, and C (Table 4.4). Only the results from user D show that the model performed better in the first minute. It might still be a better idea that I need to collect more images with different poses.

Test User	Data Points	True Positive	False Negative	Total Time (minute)
User A	10	6	4	2
User B	10	8	2	2
User C	7	5	2	2
User D	8	6	2	2

Table 4.3: Model's performance based on user test.

Test User	Data Points	Total FN	FN in 1st min.	FN in 2nd min.
User A	10	4	2	2
User B	10	2	1	1
User C	7	2	1	1
User D	8	2	0	2

Table 4.4: False positive results in two minute testing. (FN = false negative; min. = minute)

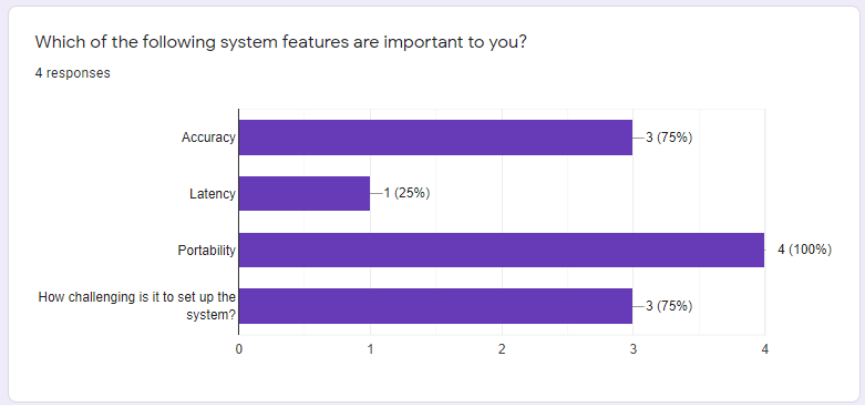
4.3.2 Test Whole System

All of the four test users were independently operating the system, and there were no technical issues. The users tested the connection between the Arduino and the app, syncing the data to the cloud, and switching between the screens to review the results. After the users finished testing, they were asked to complete a survey about the user experience and their thoughts about the system. There are 8 questions in the survey. Based on the results of the first three questions, I find that every user cares about their sitting habits and is willing to improve them by using technology. A technology in the context of the question is not necessary to be my system. We might not be surprised by the results, because people do indeed spend too much time on chairs these days, and people really get used to utilizing new technologies to solve their problems. It is a positive sign that all users thought that this project could help them track their sitting activities. The results support the trend that there are more and more self-monitoring devices coming on the market, which I have introduced in Chapter 2. The responses to the next question indicate that some people might feel uncomfortable with facing a camera. Therefore, it is not surprising that some people will prefer to wear a device rather than be monitored by a camera.

The final two questions are multiple choice, which means the users can select as many options as they like if they agree to them (Figure 4.11). The responses to the first question show that all the users considered portability to be the most significant feature of the system (Figure 4.11(a)). With high portability, the device can be carried to different places and continuously monitor users.

Accuracy and how challenging it is to use the system are also marked as important features for users. The second question is the most critical one for the app developer (Figure 4.11(b)). All of the users would like to have a push notification from the app. This function can notice users if they have been seated for a long time. In addition, most test users want to have a search interface for finding the sitting time for a specific date, and a decent dashboard with more charts and matrices so that they can have a more comprehensive view of their sitting habits.

(a)



(b)

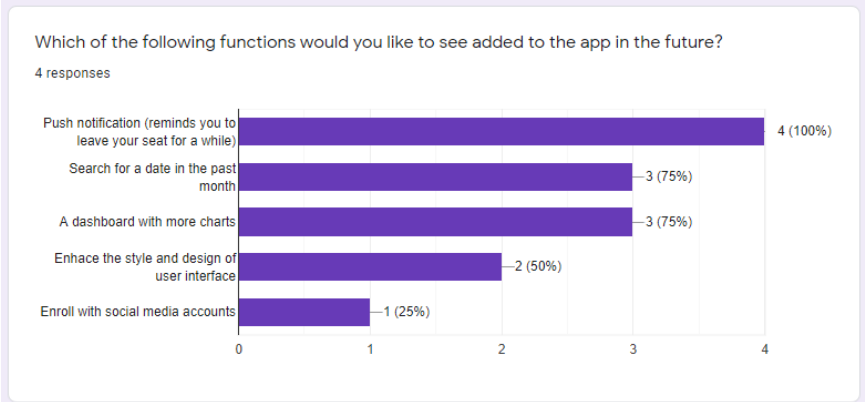


FIGURE 4.11. Results of two multiple choice questions in the user testing survey. (a) Which of the following system features are important to you? (b) Which of the following functions would you like to see added to the app in the future?

CHAPTER 5

DISCUSSION

The discussion starts with the limitations of the system, including both hardware and software parts. Then, the future work of the project is discussed, which suggests reasonable and achievable goals for development in the near future. Finally, the analysis of the project's achievements is covered. It concludes the project outcomes with the project's aims.

5.1 System Limitations

There are a number of limitations that exist in the system. Those shortages make the system still a prototype. In my opinion, different factors affect the progress of building a complete product. The technical skills needed to do the project are one of the biggest limitations. As mentioned in Chapter 1, most of the skills involved in this project, which included both hardware and software programming, were learned by myself from the beginning. After building the prototype, I still do not have expertise in the related fields, but I found many possibilities and directions for future development. This section covers the limitations of hardware and software that I have encountered during development, and the potential solutions I have tried.

5.1.1 Hardware Limitation

The Arduino TinyML Kit was used as the hardware part of the system. The Arduino board is the Arduino Nano, which supports BLE connection. The BLE connection was the way that I used to transmit data between the Arduino and my mobile phone. After several rounds of testing by myself, I found there were some issues related to BLE connection. Firstly, it is more complicated to setup a Bluetooth connection than connecting to WiFi. In my case, users need to make sure they have connected the device to the phone, because only if the connection exists, then the data can

be sent. There are several steps that are involved in BLE connection, including scanning for the device, selecting the target service and characteristics, and syncing the data. Additionally, I found if the device disconnected from the phone, there was no easy way to reconnect, and the connection entered a "zombie" status. The zombie status was that the Arduino showed it was connected, whereas no device was shown on the phone. For now, I will restart the Arduino by pushing the reset button on the board, and reconnect to the phone. The BLE is unlike a classic Bluetooth connection that can pair with the phone, which causes some instabilities in the connection. The BLE connection also confines the distance between peripheral and central devices. The connection is more unstable with the larger distance between devices. I think using WiFi could reduce the complexity of setup and transmit the data in a more stable way. The possible solution is to replace the Arduino board with an Arduino Nano IOT, which supports both WiFi and BLE connections, and it almost has the same specs as the current board. With the WiFi connection, the data can be directly uploaded to the cloud database by the Arduino, and the app can download the data from the cloud without making a connection to the device.

The camera module poses another limitation on the system. The module has a very small view, so it is difficult to find an appropriate position to include a meaningful scene. Consequently, the device can be out of function when users accidentally move the camera. Any slight changes in position might make the machine learning model perform poorly. This drawback could be annoying to users. After all, users can not see what the camera is "seeing". Moreover, I found that the camera was quite sensitive to the light intensity of the surroundings. The reason is that the camera only has grayscale input. The brighter parts will become white and the darker parts will be black in the images. This issue can be overcome by collecting more images under different light intensities. Thus, I collected the images at different times of the day. A much better solution might be to change the camera model to one that has a wider view angle.

In general, the TinyML model on the microcontroller of the devise performs well. However, there are some circumstances that might bias the model. For instance, if there are any obstacles in the view of the camera, it is hard to predict how the model will identify the objects. Another scenario is if the chair can not show its complete outward appearance, such as if some pieces of clothing are hanging on it or someone puts their hands on it. In my training dataset, I intentionally added some images of me standing behind the chair and putting my hands on it. Due to the limited development time, I could not collect a comprehensive enough dataset for training the model. I think the dataset can be improved by adding more people and chairs. In addition, I should add more pictures facing different perspectives of the chair.

5.1.2 Software Limitation

The application can not run in the background, which is the biggest limitation to this prototype. The background process means the application can run tasks in the background. For example, when users return to the home page of their phone by pressing the home button, or when users switch to another app. These actions will send the app to the background rather than terminate the app. Many common scenarios in today's apps utilize the background process. For instance, an app with a messaging function can push a notification when a new message comes in even though users are not using the app. However, the background process is not an easy task in Flutter. Flutter is a framework that helps developers build cross-platform apps, especially developing user interfaces. The background process is handled very differently on Android and iOS, which means Flutter can not provide a universal solution to implement the function. To achieve a background process in Android, I need to write native code like Java or Kotlin, and use Flutter's Platform Channels to call native platform APIs. The platform channels provide an interface to call methods and pass data between codes written in the native language (Java/Kotlin) and Flutter [58]. Alternatively, I can build the whole app in Java or Kotlin. Learning any new language during the development stage is not practical due to the time limit, so I have tried some potential solutions to fix the problem.

One way I have tried is to use a library called "workmanager". This plugin can periodically execute tasks when the app is in the background [23]. To reduce the use of the battery and enhance the performance of the phone, Android limits the minimum time for periodic work to every 15 minutes [19]. The most important part of my app that needs to run in the background is syncing the data to the cloud. As described before, the syncing process is tied to the BLE connection. Therefore, I moved the BLE connection to the callback under workmanager to fire those functions periodically. Unfortunately, the callback did not work as expected. There is no BLE connection being executed in the background. After struggling to find a solution for days, I posted the question on Stack Overflow, but by the time I was writing, I had not received any response [15]. Some similar questions can be found on Stack Overflow, but neither of them can be answered positively [20, 21]. I plan to write native code and use Flutter's platform channels after finishing writing the thesis.

The next limitation to the app is that the UI will overflow when users rotate their phone's screen. Unless users have set their phones to always display in a portrait orientation, it is a common feature that the phones will automatically change the orientation to landscape when users rotate the phone. The UI designed in the portrait orientation, which is the default setting, can not fit the length and width in landscape mode. Flutter has two built-in classes that can deal with different screen orientations: LayoutBuilder and OrientationBuilder. The LayoutBuilder provides a parameter called constraints for developers to specify layout constraints. For example, I can

define the constraint for width at 600 and prepare two versions of the UI design to fit the width over/below 600. The OrientationBuilder works similar to the LayoutBuilder but lets developers directly apply different layouts to portrait and landscape orientations. In my project, I assume users will use the app in a portrait orientation, but it is better to implement different layouts to make the design responsive.

5.1.3 Testing Limitation

To fully test the system, it is better to invite more users to join the test. However, due to an ongoing pandemic of COVID-19 at the time the project was conducted, I could only find a limited number of test users to participate in the test. Because the testing required different users to sit on a chair and use the same computer, it could not be conducted remotely, which made the testing much more inaccessible. The testing should invite more users and be planned in a more comprehensive way to truly understand the performance of the model and the whole system in the futrue.

5.2 Future Work

A number of limitations in the system have been discussed in the last section, and some possible solutions have also been proposed. In addition to the limitations, some nice-to-have features collected from the test users will also be incorporated into planning. The future work will primarily focus on software development, because most hardware development is limited by current hardware.

Running the app in the background is the project's biggest limitation. Only one official article from Flutter illustrates how to run the Dart in the background [33]. The article goes much deeper into building a custom Flutter plugin than the official document has introduced [58]. Due to the very limited resources that can be studied to solve the problem, I tend to re-develop the app by writing native code such as Kotlin. Although Flutter can make building the user interface much easier, it is still not practical for the app without supporting the background process. After all, users can not always keep the app in the foreground without using other apps.

To enhance the user experience, the feedback from the test users about new features for the app is being discussed. The most user-wanted function is that the app can push a notification if users have been sitting too long. To implement this function, the system should keep connecting to the Arduino and the cloud database. Furthermore, a new summarising method will be needed to periodically check the sitting time. Under the current data model, every new record in the database has a timestamp, so just like the method that summarises daily total sitting time, an hourly based method can be added. If a method finds that users have been sitting for the past

hour, it can push a notification to remind them. The assumptions are that the connection is steady enough and the background issue can be rectified.

Users would like to have an interface for searching for data on a specific date. This function can be achieved by using the querying methods in Firestore. Actually, the functions for manually adding and removing data require querying the data first. However, the developer may restrict the time period that users can trace back. The next feature is a dashboard with more charts. This feature will be implemented by using Flutter's Charts library and new summarise functions. I used the Charts library to plot the time series plot in the app. There are a variety of plots that can be used if the related methods are ready. For example, if I want to reproduce the stacked bar plot in Figure 2.2(b), I need to develop new methods for summing up sitting times every 3, 8, and 24 hours. The last feature I would like to discuss is letting users enroll in the app with their social media accounts. Surprisingly, only one test user would like to have this function. Logging in with a social media account not only simplifies the enrollment process but also provides social interaction with others. It is a common property in today's fitness apps. People can compete with their friends and encourage themselves to move more. Firebase provides user authentication via social media accounts. As for building a portal that lets users connect their social networks, more studies are needed to implement the function.

5.3 Conclusion

The prototype developed in this project successfully achieved the project's goals. By leveraging the power of TinyML, a simple device can identify a person and a chair. It provides a new way to detect users' sedentary behaviour, which is different from the products on the market that use accelerometers. The prototype has a hardware device and a software application that runs on a mobile phone. The app shows the total time that users have spent sitting every day for the last five days. The users' data is stored on Firebase by Google and is synced with the Arduino every 15 seconds through a BLE connection. The three core components inside the app are 1) user authentication, 2) data communication methods, and 3) functions for summarising sitting time, which positively completes part of the project's aims and objectives. The added value of completing this project is making significant progress in my personal development. I did not have any background knowledge of programming with Arduino, TinyML, and Flutter. By doing the project, I realised the machine learning paradigm and its basic workflow. Furthermore, learning to train the model and deploy it to a microcontroller. I also stepped into the world of TensorFlow, especially studying TensorFlow Lite and TensorFlow Lite for microcontrollers. Flutter might not be the best tool to develop my app due to the background process issues. However, it is an easy-to-use framework for a beginner in Android development. Above all, building a usable system from scratch greatly makes me feel a sense of accomplishment.

There are a number of things that can be refined in the future. Obviously, the background issue will be a top priority. To solve the problem, I will start to learn Kotlin to build a native app for Android. In addition, the user interface can be enhanced to a great extent. Although the model performed well in the testing, it could still be improved by including a larger and more diverse training dataset. I would also like to rebuild the system by using a new Arduino device, which supports WiFi connection. If an Arduino can periodically send the data through the WiFi connection, it is not necessary to run the app in the background.

APPENDIX A

MAIN SOURCE CODE

A.1 Sketch Program for Running the Arduino

```
// The code is adapted from the template that generated
// from Teachable Machine. I added the BLE connection
// and control the data writing interval sections.

// ArduinoBLE - Version: Latest
#include <ArduinoBLE.h>
#include <TensorFlowLite.h>

#include "main_functions.h"
#include "image_provider.h"
#include "model_settings.h"
#include "sitting_detect_model_data.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
#include "detection_responder.h"

// Setting UUID
BLEService personDetectionService("00001101-0000-1000-8000-00805f9b34fb");
BLEUnsignedCharCharacteristic personDetectionStatus("2101", BLERead | BLENotify
    );

// Setting time interval (every 15 seconds)
unsigned long previousMillis = 0;
```

```
26 const long interval = 15000;
27
28 namespace {
29 tflite::ErrorReporter* error_reporter = nullptr;
30 const tflite::Model* model = nullptr;
31 tflite::MicroInterpreter* interpreter = nullptr;
32 TfLiteTensor* input = nullptr;
33
34 // Setting TensorArea size
35 constexpr int kTensorArenaSize = 136 * 1024;
36 static uint8_t tensor_arena[kTensorArenaSize];
37 }
38
39 void setup() {
40   static tflite::MicroErrorReporter micro_error_reporter;
41   error_reporter = &micro_error_reporter;
42
43   // Import the model
44   model = tflite::GetModel(g_person_detect_model_data);
45   if (model->version() != TFLITE_SCHEMA_VERSION) {
46     TF_LITE_REPORT_ERROR(error_reporter,
47                          "Model provided is schema version %d not equal "
48                          "to supported version %d.",
49                          model->version(), TFLITE_SCHEMA_VERSION);
50     return;
51   }
52
53   // Only include the required operators to the network
54   static tflite::MicroMutableOpResolver<6> micro_op_resolver;
55   micro_op_resolver.AddAveragePool2D();
56   micro_op_resolver.AddConv2D();
57   micro_op_resolver.AddDepthwiseConv2D();
58   micro_op_resolver.AddReshape();
59   micro_op_resolver.AddSoftmax();
60   micro_op_resolver.AddFullyConnected();
61
62   static tflite::MicroInterpreter static_interpreter(
63       model, micro_op_resolver, tensor_arena, kTensorArenaSize, error_reporter)
     ;
64   interpreter = &static_interpreter;
65
66   // Allocate memory from the Tensor Arena
67   TfLiteStatus allocate_status = interpreter->AllocateTensors();
68   if (allocate_status != kTfLiteOk) {
69     TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed");
70     return;
71   }
```

```
72
73    // Prepare information of Tensor Arena to the model's input
74    input = interpreter->input(0);
75
76    // BLE section
77    Serial.begin(9600);
78    while (!Serial);
79    if (!BLE.begin()) {
80      Serial.println("starting BLE failed!");
81      while (1);
82    }
83
84    BLE.setLocalName("SittingDetection");
85    BLE.setAdvertisedService(personDetectionService);
86    personDetectionService.addCharacteristic(personDetectionStatus);
87    BLE.addService(personDetectionService);
88    BLE.advertise();
89    Serial.println("Bluetooth device active, waiting for connections...");
90  }
91
92  void loop() {
93    unsigned long currentMillis = millis();
94
95    // Get image from provider.
96    if (kTfLiteOk != GetImage(error_reporter, kNumCols, kNumRows, kNumChannels,
97                              input->data.int8)) {
98      TF_LITE_REPORT_ERROR(error_reporter, "Image capture failed.");
99    }
100
101   // Run the model
102   if (kTfLiteOk != interpreter->Invoke()) {
103     TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed.");
104   }
105
106   TfLiteTensor* output = interpreter->output(0);
107
108   for (int i = 0; i < kCategoryCount; i++) {
109     int8_t curr_category_score = output->data.uint8[i];
110     const char* currCategory = kCategoryLabels[i];
111     TF_LITE_REPORT_ERROR(error_reporter, "%s : %d", currCategory,
      curr_category_score);
112   }
113
114   // Transfer data through BLE
115   BLEDevice central = BLE.central();
116
117   // Write the data to phone every 15 seconds
```

```
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    if(central.connected()) {

    // Process the inference results
    int8_t person_score = output->data.uint8[kPersonIndex];
    int8_t no_person_score = output->data.uint8[kNotAPersonIndex];
    RespondToDetection(error_reporter, person_score, no_person_score);
    bool detect_result = RespondInBool(person_score, no_person_score);

    Serial.print("Person is detected: ");
    Serial.println(detect_result);
    personDetectionStatus.writeValue(detect_result);
    }
  }
}
```

A.2 Functions for Data Communication in the App

```
import 'package:cloud_firestore/cloud_firestore.dart';

import 'user_data.dart';
import 'package:intl/intl.dart';
import '../util/time_series.dart';

final _firestore = FirebaseFirestore.instance;

Future<bool> checkManualRecord(String date, String currentUser) async {
  bool result = false;

  await _firestore
      .collection('manual_data')
      .where('user', isEqualTo: currentUser)
      .where('date', isEqualTo: date)
      .get()
      .then((QuerySnapshot querySnapshot) {
        if (querySnapshot.size > 0) {
          result = true;
        } else {
          result = false;
        }
      });
  return result;
}

```

```
Future<String> getManualRecordData(DateTime date, String currentUser) async {
  String duration = '';
  String formatDate = DateFormat("yyyy-MM-dd").format(date).toString();
  await _firestore
      .collection('manual_data')
      .doc(formatDate + '_' + currentUser)
      .get()
      .then((DocumentSnapshot documentSnapshot) {
        if (documentSnapshot['duration'] == '') {
          duration = '0';
        } else {
          duration = documentSnapshot['duration'];
        }
  });
  return duration;
}

Future<List<UserData>> logStream(String date, String currentUser) async {
  List<UserData> userDataList = [];

  await _firestore
      .collection('bluetooth_data')
      .orderBy('time')
      .startAt([date])
      .endAt([date + '\uf8ff'])
      .get()
      .then((QuerySnapshot querySnapshot) {
    querySnapshot.docs.forEach((tmpData) {
      UserData userData =
      UserData(tmpData['person'], tmpData['time'], tmpData['user']);
      if (currentUser == userData.getUser()) {
        // Only include the data that a person sitting
        if (userData.getValue()) {
          userDataList.add(userData);
        }
      }
    });
  });
  return userDataList;
}

bool checkTimeCount(int numTrue, List<UserData> dataList) {
  if (dataList.length >= numTrue) {
    return true;
  }
  return false;
}
```

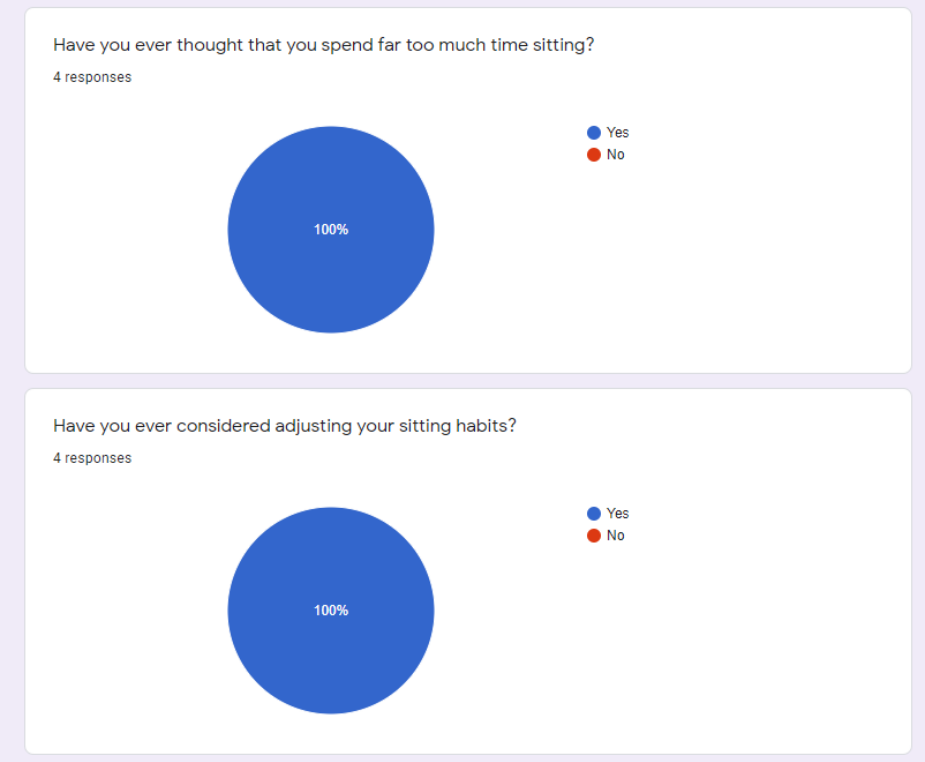
```
74
75 DateTime updateTotalTime(List<UserData> userDataList) {
76   DateTime time = DateFormat("H:m:s").parse('00:00:00');
77
78   // No user data in this date
79   if (userDataList.length == 0) {
80     return time;
81   }
82
83   List<UserData> tmpList = [];
84   for (UserData userData in userDataList) {
85     if (tmpList.length == 0) {
86       tmpList.add(userData);
87     } else {
88       if (userData.getMinute() == tmpList.last.getMinute()) {
89         tmpList.add(userData);
90       } else {
91         if (checkTimeCount(2, tmpList)) {
92           time = time.add(const Duration(minutes: 1));
93         }
94         tmpList.clear();
95         tmpList.add(userData);
96       }
97     }
98   }
99   // Check for the last minute
100  // If the last minute should be added, there is no chance
101  // for the last minute to enter the if-statement in the for-loop
102  if (checkTimeCount(2, tmpList)) {
103    time = time.add(const Duration(minutes: 1));
104  }
105  return time;
106 }
107
108 Future<TimeSeriesDaily> getDailyData(int day, String currentUser) async {
109   DateTime targetDate = DateTime.now().subtract(Duration(days: day));
110   List<UserData> tmpUserDataList = [];
111   int totalTime = 0;
112   TimeSeriesDaily timeSeriesDaily =
113   TimeSeriesDaily(DateFormat('Hms').parse('00:00:00'), 0);
114   String dayStr = DateFormat.yMd()
115       .format(targetDate)
116       .toString();
117
118   // Check if user add the data manually
119   await checkManualRecord(dayStr, currentUser).then((value) async {
120     if (value) {
```

```
121       await getManualRecordData(targetDate, currentUser).then((value) {
122         totalTime = int.parse(value) * 60;
123         timeSeriesDaily = TimeSeriesDaily(targetDate, totalTime);
124       });
125     } else {
126       await logStream(dayStr, currentUser).then((value) {
127         tmpUserDataList = value;
128         DateTime tmpTotalTime = updateTotalTime(tmpUserDataList);
129         totalTime = tmpTotalTime.hour * 60 + tmpTotalTime.minute;
130         timeSeriesDaily = TimeSeriesDaily(targetDate, totalTime);
131       });
132     }
133   });
134   return timeSeriesDaily;
135 }
```

APPENDIX B

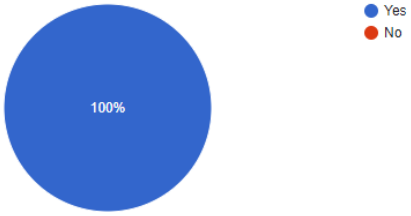
SURVEY DATA

B.1 Test Users' Responses



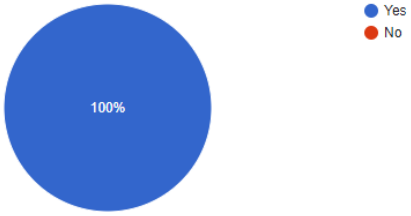
Would you like to adjust your sitting habits with the help of a novel technology?

4 responses



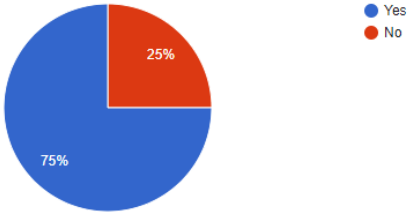
Do you think the sitting detection system can help you keep track of how much time you spend sitting?

4 responses



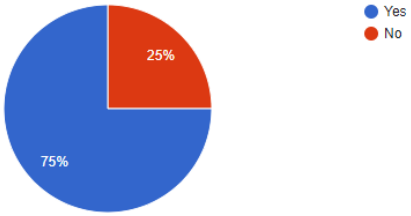
Would you be concerned if the camera continued to recognise you despite the fact that no images were stored on the device or in the cloud?

4 responses



Do you believe using a camera to identify if you are seated is a better approach than using a device that requires you to wear it?

4 responses



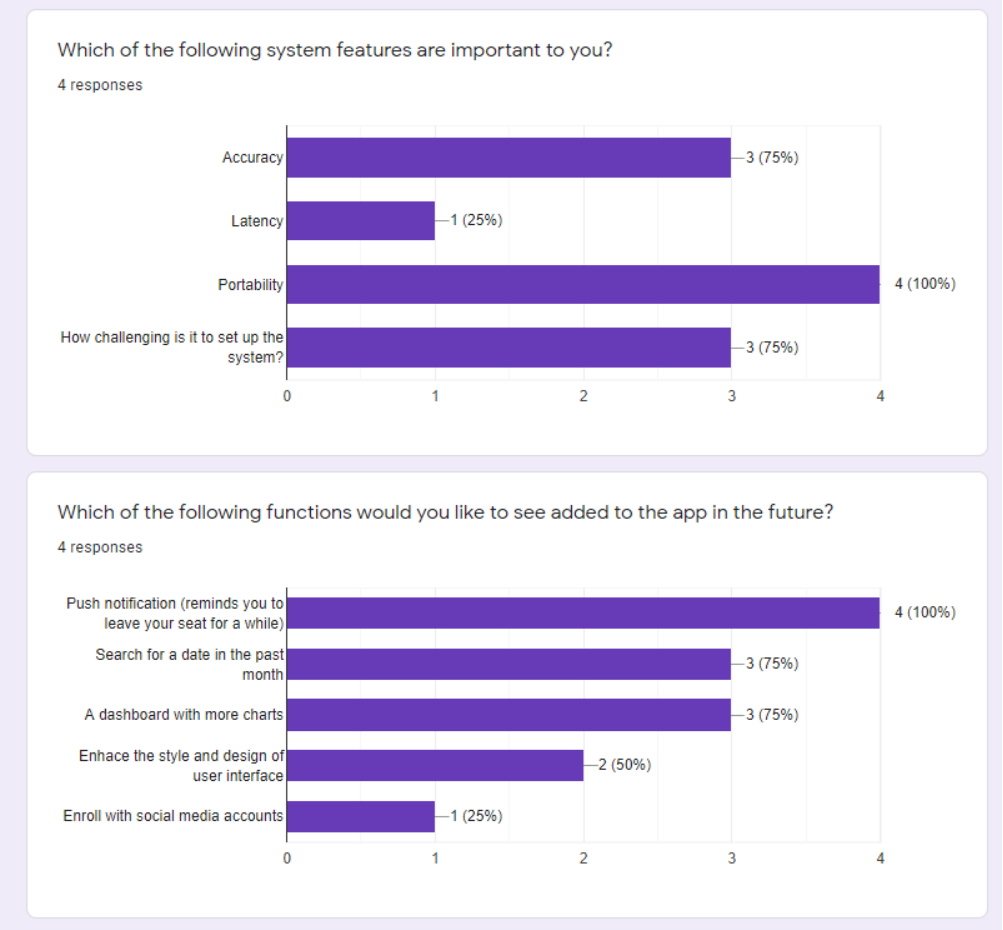


FIGURE B.1. Screenshots from Google Forms of the survey.

BIBLIOGRAPHY

[1] *Arduino nano 33 ble sense information on official website*. https://store.arduino.cc/arduino-nano-33-ble-sense.

[2] *Arduino tiny machine learning kit information on official website*. https://store.arduino.cc/tiny-machine-learning-kit.

[3] T. ALSOP, *Desktop computer penetration 2009-2020*, Jul 2020. https://www.statista.com/statistics/274164/pc-penetration-in-the-united-kingdom-uk-since-2009/.

[4] ——, *Global device shipments 2013-2020*, May 2020. https://www.statista.com/statistics/265878/global-shipments-of-pcs-tablets-ultra-mobiles-mobile-phones/.

[5] N. M. BERNINGER, G. A. TEN HOOR, AND G. PLASQUI, *Validation of the vitabit sit–stand tracker: Detecting sitting, standing, and activity patterns*, Sensors, 18 (2018), p. 877.

[6] A. BISWAS, P. I. OH, G. E. FAULKNER, R. R. BAJAJ, M. A. SILVER, M. S. MITCHELL, AND D. A. ALTER, *Sedentary time and its association with risk for disease incidence, mortality, and hospitalization in adults: a systematic review and meta-analysis*, Annals of internal medicine, 162 (2015), pp. 123–132.

[7] G. BOSE, *Getting started with teachable machine's embedded model*, 2021. https://github.com/googlecreativelab/teachablemachine-community/blob/master/snippets/markdown/tiny_image/GettingStarted.md.

[8] D. M. BRAVATA, C. SMITH-SPANGLER, V. SUNDARAM, A. L. GIENGER, N. LIN, R. LEWIS, C. D. STAVE, I. OLKIN, AND J. R. SIRARD, *Using pedometers to increase physical activity and improve health: a systematic review*, Jama, 298 (2007), pp. 2296–2304.

[9] L. E. BURKE, J. WANG, AND M. A. SEVICK, *Self-monitoring in weight loss: a systematic review of the literature*, Journal of the American Dietetic Association, 111 (2011), pp. 92–102.

[10] V. S. I. B.V., *Vitabit app on google play store*, 2021.
https://play.google.com/store/apps/details?id=com.VitaBit.VitaBit&hl=en_GB&gl=US.

[11] M. CARNEY, B. WEBSTER, I. ALVARADO, K. PHILLIPS, N. HOWELL, J. GRIFFITH, J. JONGEJAN, A. PITARU, AND A. CHEN, *Teachable machine: Approachable web-based tool for exploring machine learning classification*, in Extended abstracts of the 2020 CHI conference on human factors in computing systems, 2020, pp. 1–8.

[12] C. S. CARVER AND M. F. SCHEIER, *Control theory: A useful conceptual framework for personality–social, clinical, and health psychology.*, Psychological bulletin, 92 (1982), p. 111.

[13] S. F. CHASTIN, T. EGERTON, C. LEASK, AND E. STAMATAKIS, *Meta-analysis of the relationship between breaks in sedentary behavior and cardiometabolic health*, Obesity, 23 (2015), pp. 1800–1810.

[14] J. Y. CHAU, A. C. GRUNSEIT, T. CHEY, E. STAMATAKIS, W. J. BROWN, C. E. MATTHEWS, A. E. BAUMAN, AND H. P. VAN DER PLOEG, *Daily sitting time and all-cause mortality: a meta-analysis*, PloS one, 8 (2013), p. e80000.

[15] C.-Y. CHEN, *Cannot use the flutter_reactive_ble plugin inside the workmanager*, 2021.
https://stackoverflow.com/questions/68514562/cannot-use-the-flutter-reactive-ble-plugin-inside-the-workmanager.

[16] D. D. COX AND T. DEAN, *Neural networks and neuroscience-inspired computer vision*, Current Biology, 24 (2014), pp. R921–R929.

[17] J. DEGENHARD, *Smartphone penetration in the united kingdom 2025*, Jul 2021.
https://www.statista.com/forecasts/1147023/smartphone-penetration-forecast-in-the-united-kingdom.

[18] P. DEMARCO, *Flutterblue example on official github page*, 2021.
https://github.com/pauldemarco/flutter_blue/tree/master/example.

[19] A. DEVELOPERS, *Optimize for doze and app standby*, 2021.
https://developer.android.com/training/monitoring-device-state/doze-standby#restrictions.

[20] DJANKO, *How to run bluetooth and location code in background?*, 2019.
https://stackoverflow.com/questions/59253028/how-to-run-bluetooth-and-location-code-in-background.

[21] A. DUBEY, *How to do flutter ble device background communication?*, 2021. https://stackoverflow.com/questions/68126409/how-to-do-flutter-ble-device-background-communication.

[22] U. EKELUND, J. STEENE-JOHANNESSEN, W. J. BROWN, M. W. FAGERLAND, N. OWEN, K. E. POWELL, A. BAUMAN, I.-M. LEE, L. P. A. SERIES, L. S. B. W. GROUP, ET AL., *Does physical activity attenuate, or even eliminate, the detrimental association of sitting time with mortality? a harmonised meta-analysis of data from more than 1 million men and women*, The Lancet, 388 (2016), pp. 1302–1310.

[23] FLUTTERCOMMUNITY.DEV, *workmanager v0.4.1*, 2021. https://pub.dev/packages/workmanager.

[24] B. J. FOGG, *Persuasive technology: using computers to change what we think and do*, Ubiquity, 2002 (2002), p. 2.

[25] GOOGLE, *Firebase pricing*, 2021. https://firebase.google.com/pricing.

[26] GOOGLECREATIVELAB, *Teachable machine github page*. https://github.com/googlecreativelab/teachablemachine-community/tree/master/libraries.

[27] A. GULLI, A. KAPOOR, AND S. PAL, *Advanced convolutional neural networks*, in Deep learning with TensorFlow 2 and Keras: REGRESSION, ConvNets, GANS, RNNs, NLP, and more with TensorFlow 2 and the Keras API, Packt Publishing Ltd, 2019.

[28] G. W. HEATH, D. C. PARRA, O. L. SARMIENTO, L. B. ANDERSEN, N. OWEN, S. GOENKA, F. MONTES, R. C. BROWNSON, L. P. A. S. W. GROUP, ET AL., *Evidence-based intervention in physical activity: lessons from around the world*, The lancet, 380 (2012), pp. 272–281.

[29] J. HENSON, M. J. DAVIES, D. H. BODICOAT, C. L. EDWARDSON, J. M. GILL, D. J. STENSEL, K. TOLFREY, D. W. DUNSTAN, K. KHUNTI, AND T. YATES, *Breaking up prolonged sitting with standing or walking attenuates the postprandial metabolic response in postmenopausal women: a randomized acute study*, Diabetes care, 39 (2016), pp. 130–138.

[30] A. G. HOWARD, M. ZHU, B. CHEN, D. KALENICHENKO, W. WANG, T. WEYAND, M. ANDREETTO, AND H. ADAM, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, arXiv preprint arXiv:1704.04861, (2017).

[31] F. H. KANFER AND A. P. GOLDSTEIN, *Helping people change: A textbook of methods*, Pergamon Press, 1991.

[32] V. KETTLE, *Sedentary behaviour in office workers: correlates and interventions*, PhD thesis, Loughborough University, 2019.

[33] B. KONYI, *Executing dart in the background with flutter plugins and geofencing*, 2020. https://medium.com/flutter/executing-dart-in-the-background-with-flutter-plugins-and-geofencing-2b3e40a1a124.

[34] A. KOUL, S. GANJU, AND M. KASAM, *Chapter 6: Maximizing speed and performance of tensorflow: A handy checklist*, in Practical deep learning for cloud, mobile, and Edge: Real-world AI and computer-vision projects using python, Keras, and TensorFlow, O'Reilly Media Inc., 2019.

[35] A. KUNST, *Consumer electronics ownership in the uk 2020*, Nov 2020. https://www.statista.com/forecasts/997925/consumer-electronics-ownership-in-the-uk.

[36] A. KURNIAWAN, *Chapter 2: Arduino nano 33 ble sense board development*, in IoT projects with Arduino Nano 33 BLE sense: Step-by-step projects for beginners [38].

[37] ——, *Chapter 4: Bluetooth low energy*, in IoT projects with Arduino Nano 33 BLE sense: Step-by-step projects for beginners [38].

[38] ——, *IoT projects with Arduino Nano 33 BLE sense: Step-by-step projects for beginners*, Apress, 2021.

[39] A. MARTIN, C. FITZSIMONS, R. JEPSON, D. H. SAUNDERS, H. P. VAN DER PLOEG, P. J. TEIXEIRA, C. M. GRAY, AND N. MUTRIE, *Interventions with potential to reduce sedentary time in adults: systematic review and meta-analysis*, British journal of sports medicine, 49 (2015), pp. 1056–1063.

[40] S. MICHIE, C. ABRAHAM, C. WHITTINGTON, J. MCATEER, AND S. GUPTA, *Effective techniques in healthy eating and physical activity interventions: a meta-regression.*, Health psychology, 28 (2009), p. 690.

[41] S. MICHIE, M. RICHARDSON, M. JOHNSTON, C. ABRAHAM, J. FRANCIS, W. HARDEMAN, M. P. ECCLES, J. CANE, AND C. E. WOOD, *The behavior change technique taxonomy (v1) of 93 hierarchically clustered techniques: building an international consensus for the reporting of behavior change interventions*, Annals of behavioral medicine, 46 (2013), pp. 81–95.

[42] L. MORONEY, *AI and machine learning for coders a programmer's guide to artificial intelligence*, O'Reilly Media Inc., 2021.

[43] ——, *Chapter 1: Introduction to tensorflow*, in AI and machine learning for coders a programmer's guide to artificial intelligence [42].

[44] ——, *Chapter 3: Going beyond the basics: Detecting features in images*, in AI and machine learning for coders a programmer's guide to artificial intelligence [42].

[45] J. MUELLER AND L. MASSARON, *Chapter 1: Introducing deep learning*, in Deep learning for dummies [47].

[46] ——, *Chapter 10: Explaining convolutional neural networks*, in Deep learning for dummies [47].

[47] ——, *Deep learning for dummies*, John Wiley & Sons, Inc., 2019.

[48] NHS, *Why we should sit less*, 2019. https://www.nhs.uk/live-well/exercise/why-sitting-too-much-is-bad-for-us/.

[49] S. O'DEA, *Devices used to access the internet uk 2020*, May 2020. https://www.statista.com/statistics/387447/consumer-electronic-devices-by-internet-access-in-the-uk/.

[50] ——, *Android vs ios market share 2023*, Mar 2021. https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems/.

[51] R. PAYNE, *Chapter 12: Using firebase with flutter*, in Beginning app development with Flutter: Create cross-platform mobile apps, Apress, 2019.

[52] S. PRINCE, T. SAUNDERS, K. GRESTY, AND R. REID, *A comparison of the effectiveness of physical activity and sedentary behaviour interventions in reducing sedentary time in adults: a systematic review and meta-analysis of controlled trials*, Obesity Reviews, 15 (2014), pp. 905–919.

[53] K. I. PROPER, A. S. SINGH, W. VAN MECHELEN, AND M. J. CHINAPAW, *Sedentary behaviors and health outcomes among adults: a systematic review of prospective studies*, American journal of preventive medicine, 40 (2011), pp. 174–182.

[54] M. RICHARDS, *Chapter 1: Introduction*, in Fundamentals of software architecture: An engineering approach, O'Reilly Media, 2020.

[55] C. R. RICHARDSON, T. L. NEWTON, J. J. ABRAHAM, A. SEN, M. JIMBO, AND A. M. SWARTZ, *A meta-analysis of pedometer-based walking interventions and weight loss*, The Annals of Family Medicine, 6 (2008), pp. 69–77.

[56] J. P. SANDERS, A. LOVEDAY, N. PEARSON, C. EDWARDSON, T. YATES, S. J. BIDDLE, AND D. W. ESLIGER, *Devices for self-monitoring sedentary time or physical activity: a scoping review*, Journal of medical Internet research, 18 (2016), p. e90.

[57] N. SHRESTHA, V. HERMANS, K. K. KUKKONEN-HARJULA, S. IJAZ, AND S. BHAUMIK, *Are workplace interventions for reducing sitting at work effective?*, in PREMUS2016: Preventing work-related musculoskeletal disorders in a global economy, Institute for Work & Health, 2016, pp. 76–76.

[58] F. TEAM, *Writing custom platform-specific code*, 2021. `https://flutter.dev/docs/development/platform-integration/platform-channels?tab=android-channel-kotlin-tab`.

[59] T. TEAM, *Model optimization*, 2021. `https://www.tensorflow.org/lite/performance/model_optimization`.

[60] B. A. M. USAMA, *Chapter 2: Ble and the internet of things*, in Building Bluetooth low Energy SYSTEMS: Take your first steps in IoT, Packt Publishing, 2017.

[61] P. WARDEN AND D. SITUNAYAKE, *Chapter 13: Tensorflow lite for microcontrollers*, in TinyML: Machine learning with TensorFlow lite on Arduino and ultra-low-power microcontrollers [67].

[62] ——, *Chapter 3: Getting up to speed on machine learning*, in TinyML: Machine learning with TensorFlow lite on Arduino and ultra-low-power microcontrollers [67].

[63] ——, *Chapter 4: The "hello world" of tinyml: Building and training a model*, in TinyML: Machine learning with TensorFlow lite on Arduino and ultra-low-power microcontrollers [67].

[64] ——, *Chapter 5: The "hello world" of tinyml: Building an application*, in TinyML: Machine learning with TensorFlow lite on Arduino and ultra-low-power microcontrollers [67].

[65] ——, *Chapter 8: Wake-word detection: Training a model*, in TinyML: Machine learning with TensorFlow lite on Arduino and ultra-low-power microcontrollers [67].

[66] ——, *Chapter 9: Person detection: Building an application*, in TinyML: Machine learning with TensorFlow lite on Arduino and ultra-low-power microcontrollers [67].

[67] ——, *TinyML: Machine learning with TensorFlow lite on Arduino and ultra-low power microcontrollers*, O'Reilly Media Inc., 2020.

[68] E. WINDMILL AND R. RISCHPATER, *Chapter 1: Meet flutter*, in Flutter in action, Manning Publications Co., 2020.