

**HUMAN ACTIVITY RECOGNITION VIA ACCELEROMETER AND GYRO
SENSORS**

**BY
TEE JIA LIN**

**A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)**

MAY 2023

REPORT STATUS DECLARATION FORM

Title: Human Activity Recognition Via Accelerometer and Gyro Sensors

Academic Session: May 2023

I TEE JIA LIN

(CAPITAL LETTER)

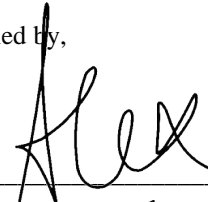
declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.



(Author's signature)

Verified by,



(Supervisor's signature)

Address:

2356, Jalan Seksyen 2/10,
Taman Bandar Baru,
31900 Kampar, Perak

Ts Dr Ooi Boon Yaik _____

Supervisor's name

Date: 15 September 2023

Date: 15 September 2023

Universiti Tunku Abdul Rahman			
Form Title : Sample of Submission Sheet for FYP/Dissertation/Thesis			
Form Number: FM-IAD-004	Rev No.: 0	Effective Date: 21 JUNE 2011	Page No.: 1 of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN

Date: 15 September 2023

SUBMISSION OF FINAL YEAR PROJECT

It is hereby certified that **Tee Jia Lin** (ID No: **20ACB05546**) has completed this final year project entitled “Human Activity Recognition Via Accelerometer and Gyro Sensors” under the supervision of Ts Dr Ooi Boon Yaik (Supervisor) from the Department of Computer Science, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,



(Tee Jia Lin)

DECLARATION OF ORIGINALITY

I declare that this report entitled “**HUMAN ACTIVITY RECOGNITION VIA ACCELEROMETER AND GYRO SENSORS**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.



Signature : _____

Name : Tee Jia Lin

Date : 15 September 2023

ACKNOWLEDGEMENTS

I would like to convey my gratitude to those who have contributed to the successful completion of my Final Year Project 2.

First and foremost, I would like to express my sincere appreciation to Ts Dr Ooi Boon Yaik, my supervisor, and my moderator Dr Sarah A'fifah Binti Abdullah Sani, who have given me this golden opportunity to involve in the field of Human Activity Recognition using smartphone embedded accelerometer and gyroscope. A million thanks for their guidance and motivation throughout the project.

Other than that, I would like to say thanks to my parents and friends for their love and support, which kept me motivated throughout the project.

ABSTRACT

In recent years, the scholarly community has shown great interest in Human Activity Recognition (HAR) as a result of its wide applications and growing significance spanning different domains. While much research has been conducted, focusing on the development of algorithms and techniques for HAR, less emphasis was placed on the improvement of HAR research's efficiency in terms of sensor data collection, annotation, and storage, resulting in the use of incomplete, inefficient, and time-consuming data engineering systems.

This project aims to address the issue of inefficient data engineering infrastructure being used in current HAR research by developing an efficient, comprehensive, and user-friendly data engineering system for data collection, annotation, and storage. To implement the data engineering system proposed, two mobile applications, SensorData and SensorDataLogger with user-friendly interfaces and intuitive functionalities are developed using Java programming language and Android Studio. The dataset created using the proposed data engineering system is then used to train unidirectional Long Short Term Memory (LSTM) model to evaluate the efficiency of proposed system in terms of accuracy and generalization capabilities. In other words, if the dataset created using the proposed system can achieve good accuracy and generalization during training and testing, it means that the proposed data engineering system is effective. To prevent overfitting, early stopping is used to monitor validation loss during training and dropout rate of 0.3 are applied. This project proves that the proposed data engineering system is efficient, which is able to achieve an accuracy of 96.57%.

In conclusion, this project will be a significant contribution to the development of HAR in multiple aspects. Firstly, it advances the domain enhancing the data engineering system's efficiency. Next, it improves the accuracy and reliability of HAR research by allowing the researchers to produce dataset of high-quality. Furthermore, it improves scalability and reproducibility by allowing researchers to expand projects to large scope or reproduce other research with least effort. Moreover, it reduces the barriers of entry for non-technical researchers to engage in HAR research. Lastly, the project paves the way for establishment of standardized dataset, with streamlined data collection, data annotation and data storage, and allow comparative research and benchmarking.

TABLE OF CONTENTS

TITLE PAGE	i
REPORT STATUS DECLARATION FORM	ii
FYP THESIS SUBMISSION FORM	iii
DECLARATION OF ORIGINALITY	iv
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	ix
LIST OF TABLES	xiii
LIST OF ABBREVIATIONS	xiv
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement and Motivation	1
1.2 Objectives	3
1.3 Project Scope and Direction	5
1.4 Contributions	6
1.5 Report Organization	8
CHAPTER 2 LITERATURE REVIEW	10
2.1 Review of the Technologies	10
2.2 Review of the Existing Systems	27
2.3 Proposed System	40
CHAPTER 3 SYSTEM METHODOLOGY/APPROACH (FOR DEVELOPMENT-BASED PROJECT)	42
3.1 System Design Diagram	42
3.1.1 System Architecture Diagram	42
3.1.2 Use Case Diagram and Description	44
3.1.3 Activity Diagram	58

CHAPTER 4 SYSTEM DESIGN	61
4.1 System Block Diagram	61
4.2 System Components Specifications	62
CHAPTER 5 SYSTEM IMPLEMENTATION (FOR DEVELOPMENT- BASED PROJECT)	73
5.1 Hardware Setup	73
5.2 Software Setup	74
5.3 Cloud Setup	76
5.4 Setting and Configuration	77
5.5 System Operation (with Screenshot)	78
5.6 Implementation Issues and Challenges	117
5.7 Concluding Remark	118
CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION	119
6.1 Model Testing and Performance Metrics	119
6.2 Testing Setup and Result	120
6.3 Project Challenges	121
6.4 Objectives Evaluation	122
6.5 Concluding Remark	124
CHAPTER 7 CONCLUSION AND RECOMMENDATION	125
7.1 Conclusion	125
7.2 Recommendation	126
REFERENCES	127
APPENDIX	131
WEEKLY LOG	145
POSTER	152
PLAGIARISM CHECK RESULT	153
FYP2 CHECKLIST	167

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1	RNN model in [1]	23
Figure 2.2	LSTM neuron in [1]	24
Figure 2.3	Accuracy for individual classifiers for different dataset in [4]	29
Figure 2.4	Accuracy for combined classifiers for different dataset in [4]	29
Figure 2.5	Variation in acceleration signals when smartphones' positions change in [4]	30
Figure 2.6	Accuracy of SVM change when number of features used change in [10]	31
Figure 2.7	Accuracy of RF change when number of features used change in [10]	32
Figure 2.8	Accuracy in [12]	33
Figure 2.9	System architecture diagram in [14]	33
Figure 2.10	Vanilla LSTM model in [14]	35
Figure 2.11	2-Stacked LSTM model in [14]	35
Figure 2.12	3-Stacked LSTM model in [14]	35
Figure 2.13	CNN-LSTM model in [14]	36
Figure 2.14	4-layered CNN-LSTM model in [14]	36
Figure 3.1	System Architecture Diagram	42
Figure 3.2	Use Case Diagram of HAR system	44
Figure 3.3	Activity Diagram	58
Figure 4.1	Block diagram for smartphone based HAR system	61
Figure 4.2	Flowchart for data collection in smartphone based HAR system	62
Figure 4.3	Flowchart for dataset creation in smartphone based HAR system	64

Figure 4.4	Flowchart for data preprocessing in smartphone based HAR system	65
Figure 4.5	Flowchart for data exploration in smartphone based HAR system	66
Figure 4.6	Flowchart for data segmentation in smartphone based HAR system	67
Figure 4.7	Flowchart for model training and evaluation in smartphone based HAR system	68
Figure 4.8	Architecture of LSTM model proposed	68
Figure 4.9	Flowchart for hyperparameter tuning in smartphone based HAR system	70
Figure 4.10	Flowchart for model testing in smartphone based HAR system	71
Figure 5.1	Android Studio Arctic Fox, 2020.3.1, Patch 4	75
Figure 5.2	Android emulators used	75
Figure 5.3	Google Colab interface	76
Figure 5.4	Google Cloud Storage bucket used to store the log files.	76
Figure 5.5	User Interface of SensorData	79
Figure 5.6	User Interface of SensorDataLogger	80
Figure 5.7	Dialog box asking for Bluetooth permission	81
Figure 5.8	Dialog box asking for storage permission	81
Figure 5.9	Dialog box asking to turn on Bluetooth in SensorDataLogger	82
Figure 5.10	Dialog box asking to turn on Bluetooth in SensorData	83
Figure 5.11	Service account created	84
Figure 5.12	Bucket <i>sensordatalogger-logfiles.appspot.com</i> created	84
Figure 5.13	Google account sign-in page	85
Figure 5.14	Dialog box to add activity buttons in SensorData	86
Figure 5.15	Swipe to the left to delete activity button	87
Figure 5.16	Real-time accelerometer and gyroscope data in SensorDataLogger	88
Figure 5.17	List of activity buttons in SensorData	89
Figure 5.18	The devices show status 'Not connected'	90

Figure 5.19	The device displays list of paired devices	91
Figure 5.20	SensorData initiate Bluetooth connection to SensorDataLogger, status showing ‘Connecting...’	92
Figure 5.21	The devices show status ‘Connected to Redmi Note 11 Pro 5G’ and ‘Connected to Samsung Galaxy A8’.	93
Figure 5.22	Click activity button ‘WALK’ via SensorDataLogger	94
Figure 5.23	A dialog asking to set duration. After that, click ‘START’	94
Figure 5.24	A timer starts at SensorData	94
Figure 5.25	A timer starts at SensorDataLogger	95
Figure 5.26	Timer stops. SensorDataLogger stops to collect sensor signals for ‘WALK’ activity	96
Figure 5.27	Click ‘Save Log File’ button	97
Figure 5.28	Log files saved with specified filename	97
Figure 5.29	Log files content with automated annotation	98
Figure 5.30	Click ‘Upload Log File’ button	99
Figure 5.31	Upload log files to the bucket	100
Figure 5.32	Individual log files created and stored in Google Cloud Storage	102
Figure 5.33	Retrieve the list of CSV file paths stored in GCS	102
Figure 5.34	Format the filename of CSV file	103
Figure 5.35	Storage spaces needed for CSV log files (LHS) compared to Parquet log files (RHS).	103
Figure 5.36	Convert CSV files to Parquet files. Save back the Parquet files to GCS bucket inside folder ParquetFile/	104
Figure 5.37	Retrieve the list of Parquet file paths stored in GCS	104
Figure 5.38	Format the filename of Parquet file	105
Figure 5.39	Preprocessing	105
Figure 5.40	Some part of the final dataset, <i>har.parquet</i>	106
Figure 5.41	Data type of each attribute in the dataset	108
Figure 5.42	Statistical summary of dataset	108
Figure 5.43	Null values of dataset for each attributes	108
Figure 5.44	Infinity values of dataset	108
Figure 5.45	Number of samples for each activity	109

Figure 5.46	Bar graph showing number of samples for each activity	109
Figure 5.47	Correlation matrix	110
Figure 5.48	Libraries or packages used	112
Figure 5.49	LSTM model architecture	113
Figure 5.50	Results of model training	114
Figure 5.51	A graph plotted on the changes of train loss, train accuracy, validation loss and validation accuracy	114
Figure 5.52	Accuracy, loss and RMSE of training	114
Figure 5.53	Confusion matrix	115
Figure 5.54	Libraries or packages used	116
Figure 5.55	param_grid	116
Figure 5.56	Best combination of hyperparameters and accuracy	116
Figure 5.57	Accuracy, loss and RMSE of testing	117
Figure 5.58	Confusion matrix	117
Figure 6.1	Confusion matrix of model testing	120
Figure 6.2	Accuracy, loss and RMSE of model testing	121

LIST OF TABLES

Table Number	Title	Page
Table 2.1	Comparison between smartphones and smartwatches	14
Table 2.2	Comparison between existing systems and proposed system	37
Table 3.1	Use Case Legend	45
Table 5.1	Specifications of laptop	73
Table 5.2	Specifications of smartphone	73
Table 5.3	Specifications of tablet	74
Table 6.1	Objectives evaluation	122

LIST OF ABBREVIATIONS

<i>LSTM</i>	Long Short Term Memory
<i>CNN</i>	Convolutional Neural Network
<i>RNN</i>	Recurrent Neural Network
<i>HAR</i>	Human Activity Recognition
<i>SVM</i>	Support Vector Machine
<i>RF</i>	Random Forest
<i>DT</i>	Decision Tree
<i>SGD</i>	Stochastic Gradient Descent
<i>LG</i>	Logistic Regression
<i>k-NN</i>	k-nearest neighbors
<i>GPS</i>	Global Positioning System
<i>MSE</i>	Mean squared error
<i>RMSE</i>	Root mean squared error
<i>GCS</i>	Google Cloud Storage
<i>FFT</i>	Fast Fourier Transform
<i>DFT</i>	Discrete Fourier Transform
<i>APF</i>	average of peak frequency
<i>BPTT</i>	backpropagation through time
<i>PPV</i>	positive predictive value
<i>TPR</i>	true positive rate
<i>BLE</i>	Bluetooth Low Energy
<i>IDE</i>	Integrated Development Environment

Chapter 1

Introduction

1.1 Problem Statement and Motivation

Human activity recognition (HAR) refers to classification of human activities or actions carried out by one or more individuals by using signals data recorded from responsive sensors when they make movements [3]. In recent years, the scholarly community has shown great interest in Human Activity Recognition (HAR) as a result of its wide applications and growing significance spanning different domains. [8] indicates that there are approximately 120 publications in the field of HAR in 2019, about 130 publications in 2020, followed by 80 publications in 2021. Theoretically, this trend has been motivated by a surge in the development of smart systems and environments that are progressively revolutionizing our daily lives over the past few years [14]. The smart systems and intelligent environments being built cover various aspects of human lives, including health monitoring, elderly care, surveillance system, disaster occurrence detection, interactive gaming, etc. Besides, the proliferation of advanced sensors, particularly in electronic devices and the prevalence of these devices have enabled the collection of rich and diverse data about human movements [5]. On the other hand, the improvement in Machine Learning and Deep Learning algorithm helps to further advance the area of HAR by allowing the development of sophisticated algorithms to accurately classify human activities [5]. This could be seen in [5], where researchers achieved an average recognition accuracy of 93.0% and 92.2% by using Deep Learning and Machine Learning algorithms, respectively.

Despite the extensive research done on the development of HAR algorithms and techniques to improve the recognition accuracy, less emphasis was placed on the improvement of HAR research's efficiency in terms of sensor data collection, annotation, and storage. The relevant process remains time-consuming, inefficient, and complicated. For instance, the studies in [1] and [3] did not specify the data collection, annotation, and storage procedure. There were, however, researches who did pay attention to the data engineering process. For example, the authors in [2] clearly showed the data collection, annotation, and storage strategy. However, the methodologies used were quite inefficient. The data subjects in [2] were required to wear 3-D accelerometers on wrist, hip and hold a rucksack with GPS sensor and a flash-

Chapter 1

card-memory-based 19-channel recorder to store the data. The data collected was then annotated manually by supervisors or data subjects. A better example was [8], which gave clear descriptions and used effective solutions for data collection and storage. The data subjects in [8] were required to wear accelerometer and heart rate sensors on wrists. The data obtained from these sensors were sent to an Android phone via BLE and stored in MySQL database located on a server in cloud. However, the data collected still required manual labelling [8]. The inefficient data engineering system being used in HAR research represents a significant challenge and there is a pressing need to address the inefficiencies to enhance the overall effectiveness and performance of HAR research.

The motivation behind this HAR research stems from the critical need to propel the field forward while maximizing its impact. The current inefficiencies in the data engineering process of HAR researches bring the problems of scalability, reproducibility, and comparability. For example, the inefficiency in sensor data collection, annotation and storage used in HAR research hampers the project scalability since it is difficult and time-consuming for researchers to expand projects to larger scopes as large and reliable datasets are required. Besides, the inefficient and non-standardized data engineering system being employed affects the reproducibility of the project for future researchers. This will limit the potential and achievements of HAR research. Not only that, the data engineering system in HAR which lacks efficiency and standardization can affect the comparability of studies within the field. This is because inefficient methodology causes inaccurate data, noise and inconsistent quality of data being collected. When the annotations are done manually without standardization, there is a risk of inconsistencies in labelling human activities, which leads to discrepancies in labeled ground truth. Parka et al. [15] pointed out that one of the factors which led to misclassification of human activities was due to the lack of synchronization between activity performances and data annotation. Using inefficient strategy will also result in the waste of resources including time, money, and manpower. Moreover, non-standardized and inefficient data engineering system presents a challenge in establishing benchmark dataset that serves as a common ground for HAR algorithm's performance evaluation.

Hence, a comprehensive and effective data engineering infrastructure is required to address the issues by streamlining data collection methods, implementing more effective annotation procedures, and optimizing storage strategies. By doing that, researchers can undertake larger-scale studies with reduced time and resource constraints. Moreover, the efficient data engineering ensures the accurate, reliable, and high quality of sensor data being

collected, labelled, and stored, which further improve the accuracy and reliability in HAR models. This will allow future researchers to be able to reproduce researches done to further enhance them and improve the overall performance of HAR field. Furthermore, efficient data engineering enables comparability of studies, and this will contribute to the development of standardized benchmarks for evaluating HAR algorithms. Last but not least, the proposed data engineering system will remove the barriers for those researchers who do not have strong background in data engineering.

1.2 Objectives

1.2.1 To address the issue of inefficient data engineering infrastructure being used in current Human Activity Recognition (HAR) research.

As mentioned earlier, much emphasis was given to development of algorithms and technologies in HAR, rather than on improving the efficiency of data engineering infrastructure used to collect, annotate, and store data. Consequently, most of the current HAR research still employed inefficient methodologies, resulting in inaccurate or unreliable results, affects the scalability, reproducibility, comparability and limits the progress of HAR area. The primary objective of this project is to address the issue of inefficient data engineering infrastructure existing in most of the HAR research nowadays. This project aims to improve the efficiency, accuracy, and reliability of HAR research by designing and constructing an efficient and comprehensive data engineering model for the process of data collection, data annotation and data storage. Achieving this objective can help other researchers to overcome the challenges associated with inefficient data engineering infrastructure and to advance the state of the art in the area of Human Activity Recognition.

1.2.2 To develop a comprehensive, efficient, and user-friendly data engineering system for data collection, annotation, and storage.

Another project's objective is to develop a comprehensive, efficient, and user-friendly data engineering system for data collection, annotation, and storage. The data engineering system proposed can be achieved by developing streamlined data collection techniques, automated annotation strategies, and efficient data storage solutions. The implementation of the system will help to simplify the process of data collection, annotation, and storage. This will directly improve the performance and productivity of subsequent HAR research, which is important to achieve greater results and breakthroughs in HAR. This project will also focus on

developing user-friendly system that is easy to use, especially to those researchers with do not have technical background in data engineering. The system should also be flexible enough to be adapted for use in a variety of research with minimal modifications.

1.2.3 To design and develop mobile applications, SensorData and SensorDataLogger

Another project objective is to design and develop mobile applications used in the data engineering system, which are SensorData and SensorDataLogger. The applications are developed to achieve the efficient data engineering system proposed. The applications are developed using Java programming language and Android Studio as IDE.

The applications are connected using Bluetooth to send and receive instructions. Users can give instructions via SensorData while SensorDataLogger is responsible to receive and execute instructions. In SensorData, users can add the activities they want to collect data for by defining the activities' names and relevant IDs. The IDs should be unique among the activities being created. Then, users can start the data recording process by clicking at the activity created and set the durations they want to collect data for. After users click the 'START' button, SensorData starts a timer. When the time is up, SensorData will automatically send instructions to SensorDataLogger to stop the recording process. When users click the 'Save Log File' button, SensorData will send instructions to SensorDataLogger with the specific filename, which is the ID of the activity. When users click 'Upload Log File', SensorData will send instructions to SensorDataLogger to upload log files to Google Cloud Storage.

SensorDataLogger contains functions such as showing real-time data of accelerometer and gyroscope to monitor the data collection process. Not only that, SensorDataLogger can start the sensor data recording process for a particular physical activity for a specified period of time and stop automatically when the time is up, upon the instructions from SensorData. Besides, when the recording process stops and users click the 'Save Log File' button through SensorData, SensorDataLogger will start the process of saving log file locally into mobile phone with specified filename. Furthermore, SensorDataLogger can upload the log files to Google Cloud Storage server for later processing, upon the instructions from SensorData.

This project will not focus on developing applications for iOS as most of the researchers use Android phone to conduct the research.

1.2.4 To evaluate the efficiency of proposed data engineering system on HAR algorithm.

Another project objective is to evaluate the efficiency of proposed data engineering system on HAR algorithm by looking at the performance of the HAR algorithms. This assessment will encompass performance metrics such as accuracy and the generalization capabilities of HAR algorithms. These metrics enable a quantitative assessment of how the HAR algorithms perform when the proposed data engineering system is being implemented. By systematically gauging the efficiency of the proposed data engineering system, this objective inherently addresses the practicality and feasibility of integrating the proposed system into real-world HAR applications. Besides, this project also wants to demonstrate that the proposed data engineering system can improve the scalability, reproducibility, and comparability of project without compromising recognition accuracy or efficiency.

1.3 Project Scope and Direction

The scope of this project revolves around addressing the issue of inefficient data engineering system within current Human Activity Recognition (HAR) researches. As stated in the problem statement, much of the emphasis in HAR has been directed towards the development of algorithms and techniques used to increase the activity recognition accuracy, overshadowing the optimization of data engineering infrastructure, in terms of data collection, data annotation and data storage. This imbalanced effort has resulted in the employment of inefficient data engineering solutions in current HAR researches, leading to inaccuracies, unreliability, and limitations in scalability, reproducibility, and comparability of HAR studies. Henceforth, the scope of this research is to develop a comprehensive, efficient, and user-friendly data engineering system. The proposed system can be achieved by developing streamlined data collection techniques, automated annotation strategies, and efficient data storage solutions. With this system being constructed and utilized, sensor data can be accurately collected, labelled, and stored, which would bring efficiency to subsequent HAR research. The system designed should be user-friendly and straightforward to use, especially to those researchers who do not have technical background in data engineering.

To achieve the goal, the project covers the design and development of two mobile applications, SensorData and SensorDataLogger. The applications are developed using Java programming language and Android Studio as IDE. Both applications are connected via Bluetooth. Users can give instructions via SensorData while SensorDataLogger is responsible to receive and execute instructions. In SensorData, users can add the activities they want to collect data for by defining the activities' names and relevant IDs. The IDs should be unique

Chapter 1

among the activities being created. Then, users can start the data recording process by clicking at the activity created and set the durations they want to collect data for. After users click the ‘START’ button, SensorData starts a timer. When the time is up, SensorData will automatically send instructions to SensorDataLogger to stop the recording process. When users click the ‘Save Log File’ button, SensorData will send instructions to SensorDataLogger with the specific filename, which is the ID of the activity. When users click ‘Upload Log File’, SensorData will send instructions to SensorDataLogger to upload log files to Google Cloud Storage. SensorDataLogger contains functions such as showing real-time data of accelerometer and gyroscope to monitor the data collection process. Not only that, SensorDataLogger can start the sensor data recording process for a particular physical activity for a specified period of time and stop automatically when the time is up, upon the instructions from SensorData. Besides, when the recording process stops and users click the ‘Save Log File’ button through SensorData, SensorDataLogger will start the process of saving log file locally into mobile phone with specified filename. Furthermore, SensorDataLogger can upload the log files to Google Cloud Storage server for later processing, upon the instructions from SensorData.

Since most of the research were done using Android device, therefore this project will not focus on developing mobile applications for iOS device.

1.4 Contributions

1.4.1 Advancement of Data Engineering System in HAR research

This project helps to advance the area of Human Activity Recognition (HAR) by addressing the often-overlooked issue related to inefficient data engineering system being used to collect, label, and store data. Inefficient data engineering system being used to handle data will limit the potential progress and achievements of researchers in the area of HAR research. This project aims to remove this barrier by developing a comprehensive and efficient data engineering system to provide researchers with necessary tools to achieve more in the field. By devising and implementing an efficient data engineering system for data collection, annotation, and storage, this research fosters a new paradigm in HAR research.

1.4.2 Improvement of Accuracy and Reliability of HAR research

Furthermore, this project helps to improve the accuracy and reliability of HAR research. Sensor data is one of the important components in HAR research. The accuracy and reliability of the data being collected and processed will have a significant impact on the outcome of

subsequent HAR research. Hence, it is crucial that the data engineering system used to handle data is accurate and reliable. The proposed system directly addresses the challenges associated with inefficient data handling practices, paving the way for more accurate and reliable HAR studies.

1.4.3 Improvement of Scalability and Reproducibility of HAR research

By implementing the efficient data engineering system, the optimized system facilitates the handling of larger datasets without compromising accuracy, thereby enabling researchers to explore broader contexts and scenarios. Additionally, the standardized data engineering procedures ensure greater reproducibility of experiments across different studies.

1.4.4 Foundation and Impetus for Future Researches

Besides, this project also contributes a foundation and reference for future research. The data engineering system being designed and developed in this project helps to facilitate future research, as it can be adapted to be used in other research, either within the same area or different area of study with only minimal modifications. The system proposed will have long-term significance and benefits to all the research that relies on data collection, annotation, and storage. The proposed system is also expected to stimulate further innovation and exploration within HAR domain.

1.4.5 Practical Implementation for Non-technical Researchers

Moreover, this project will also benefit those researchers who do not have technical background in data engineering. By offering a user-friendly interface and intuitive functionalities, the proposed data engineering system allows more researchers to engage in HAR research without grappling with complex technical intricacies. This helps to reduce the barriers to entry for new researchers and increase the overall performance of the related field.

1.4.6 Comparative Research and Benchmarking

The proposed data engineering system allows the establishment of more standardized dataset, with streamlined data collection, data annotation and data storage. This allows for meaningful comparative research and benchmarking in the realm of HAR. Researchers can now compare the performance of various HAR algorithms and techniques on standardized

Chapter 1

datasets, eliminating the confounding effects of inefficient data engineering practices. This contributes to a more accurate assessment of different HAR approaches.

1.5 Report Organization

This research is organized into six chapters, which includes: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Methodology/Approach, Chapter 4 System Design, Chapter 5 System Implementation, Chapter 6 System Evaluation and Discussion and Chapter 7 Conclusion.

Chapter 1 serves as the introduction of this research, describing background of the project, problem statements discovered in the current HAR researches, motivation of implementing the project, project scope, project objectives, contributions of the project and project report structure.

Chapter 2 entails a comprehensive exploration of existing Human Activity Recognition researches conducted using smartphone embedded sensors. The evaluations of strengths and weaknesses of the researches have been done.

Next, Chapter 3 delves into the overall system design and proposed methodology adopted in this project. System architecture diagram, use case diagram and activity diagram along with the detailed descriptions are included in this chapter.

Chapter 4 includes a system block diagram, which shows the components of the proposed system, including data collection, dataset creation, data exploration, data segmentation, model training and evaluation, hyperparameter tuning and model testing. We also include the flowchart explaining the process and implementation of each component in the proposed system.

Additionally, Chapter 5 discusses on the system implementation, which includes the details of the hardware, software and cloud used in this system. We also include the configurations and settings needed for the hardware, software, and cloud. Then, we discuss the partial outcome of data engineering system, which is the SensorData and SensorDataLogger applications. After that, we discuss on coding and partial results of each process in the machine learning pipeline. We also talk about some implementation issues and challenges faced and how we overcome them.

After that, Chapter 6 presents the outcome of our project. Firstly, we show and explain the coding on how we perform model testing and evaluate their performance. Then, we demonstrate the confusion matrix and accuracy, loss, RMSE values we are able to achieve from

Chapter 1

model testing. Moreover, we review some of the project challenges and our solutions to tackle them. We also had an objective evaluation to show the current progress of this project.

Lastly, Chapter 7 includes the conclusion which discuss about our current accomplishments and summary of the project. Recommendations are also given on the improvements we can make to further enhance the performance of this research.

Chapter 2

Literature Review

2 Previous works on Human Activity Recognition (HAR)

Human Activity Recognition (HAR) stands as a central focus of this research, poised at the intersection of technology and human behavior. As our lives become intensively interlaced with digital devices, the ability to understand and predict human activities from sensor data holds tremendous value. Researchers have ardently explored the area of HAR due to its growing significances and its wide applications in our daily life that span healthcare, surveillance, fitness tracking, etc. This literature review will delve into the essential elements that constitute the foundation of HAR based on inertial sensors, which are sensor dataset creation, signals preprocessing, model training and model evaluation.

2.1 Review of the Technologies

2.1.1 Dataset

The first step in Human Activity Recognition (HAR) research based on inertial sensors is dataset preparation. This research will capture the sensor signals from accelerometer and gyroscope. Accelerometer captures the linear acceleration and gravitational forces acting on it, where the processed signals can be used to identify sudden changes in movements [3]. Gyroscope records the angular velocity, capturing the rotational motion around each axis, where the processed data can be used to identify the position and alignment of the devices [3].

2.1.1.1 Data Source

Most of the previous research conducted using inertial sensors involved implementing the data collection process and creating an independent, bespoke datasets. For instance, the authors in [1] collected accelerometer and gyroscope sensor data for the activities of stop, walking, running, and running helter-skelter, aiming to identify occurrences of disasters such as fire, tsunamis, etc. Besides, in [2], the authors collect sensor data in supervised and unsupervised settings by allowing data subjects wearing 3-D accelerometers on wrist, hip, and by holding a rucksack with GPS sensor. Moreover, the authors in [4] collected the accelerometer data using mobile phone by asking the data subjects to hold the smartphones in

Chapter 2

front of the chests or inside pockets while performing activities such as slow walking, fast walking, running, aerobic dancing, climbing upstairs and climbing downstairs. Furthermore, the data subjects in [6] carried the smartphones in their leg pockets to detect actions such as standing, sitting, fallen down, squatting in toilet, etc. The sensor suite in [6] includes accelerometer, gyroscope, humidity, and temperature sensors.

Creating an independent dataset allows the researchers to have full control over the data quality, ensuring the data collected is consistent, and of high quality. This helps to reduce the noises and biases in the dataset. Besides, researchers can also customize the data collection to match the specific requirements of the research, focusing on the types of activities, scenarios or contexts that are most relevant. For instance, in [1], [4] and [6], the authors included the actions which are not commonly found in existing open datasets, such as running helter-skelter, aerobic dancing and squatting in toilet. Hence, the authors created independent datasets for their own usages. [2] emphasizes the significance of using out-of-laboratory sensor data for the development of HAR, which is not commonly seen in existing open datasets. The authors in [6] also collected humidity data and temperature data via smartphone, which are rarely found in public datasets. Most of the existing datasets are largely based on accelerometer, gyroscope, and magnetometer [5].

However, creating an independent dataset can introduce limitations such as inaccurate measurement due to different ways of sensor placement and attachment on data subjects. For example, the authors in [4] found that the acceleration data showed significant variation for the same physical activity when data subjects shifted the position of smartphones from holding in hands in front of chests to putting smartphones inside pockets. [2] also found that sensor location, attachment, and other factors such as body shape of data subjects and the clothes wore by the data subjects would affect the accuracy of acceleration data collected to differentiate sitting and standing activities. The body shape of data subjects might have impact on the sensor orientation on the waist [2]. The data subjects in [2] wore sport clothes, which caused the unsteady position of accelerometers.

There were also researchers who opted for open existing datasets instead of creating on their own. For example, the authors in [11], chose to use Opportunity dataset which collects data from 23 body sensors, 12 object sensors and 21 ambient sensors regarding the human daily morning activities in a room. Similarly, [21] use KU-HAR dataset, which collects data from accelerometer and gyroscope through smartphone placed in a waist bag, due to its wide variations of activities included and it is currently the largest datasets available for

Chapter 2

accelerometer and gyroscope data. Furthermore, [20] used HHAR dataset which contains signals collected from both smartphones and smartwatches. Other existing dataset includes WISDM, REALDISP, PAPAM2 and others [5].

Using public datasets helps to reduce biases that may be introduced during data collection. Besides, employing public datasets help to solve the problem of recognizing human movement patterns among different individuals since human actions have large intra-subject and inter-subject variability [5]. Researchers faced difficulties in developing a methodology that can recognize human movement among all subject [5]. Hence, using publicly available datasets offer a benchmark for evaluating and fine-tuning models.

However, existing datasets may have limited relevance with your research purpose and context and limited diversity in terms of activities type and sensor type. [5]. In short, it is nearly impossible to find a public dataset which aligns with your research goals in terms of activities type, sensor type, annotations, etc. This will limit the applicability and accuracy of your research.

2.1.1.2 Smartphone Embedded Sensors

The prior researches of HAR based on inertial sensors can be classified into two categories in terms of data collection device, which includes smart devices and wearable, standalone sensors. The smart devices include smartphones and smartwatches. This research will concentrate on collecting sensor signals using embedded sensors within smartphone.

The widespread adoption of smart devices, especially smartphones and smartwatches has led to a surge in HAR research done using these technologies [5]. For instance, the author in [4] and [7] proposed a human action recognition system leveraging acceleration data from only single triaxial accelerometer embedded in smartphone. Han et al. [1] devised a human action recognition model using accelerometer and gyroscope embedded in smartphones to detect occurrences of disasters. Similarly, Bulbul et al. [3] used smartphone embedded accelerometer and gyroscope sensors to collect signals to recognize six activities. Moreover, Masum et al. [6], proposed a reliable data gathering process by using accelerometer sensor, gyroscope sensor, temperature sensor and humidity sensor to gather data.

There were also researches done based on smartwatches. For example, Kheirkhahan et al. [18] developed a real-time health monitoring framework using smartwatches. The authors proposed a ROAMM framework which involves continuous sensor data gathering at high frequency to monitor and evaluate physical activity, along with patient-reported outcomes [18].

Chapter 2

An activity recognition system based on smartwatch that utilizes active learning was suggested in [19] and was able to obtain a 93.3% of recognition accuracy. The authors in [20] conducted HAR research by using acceleration signals from both smartphones and smartwatches. The authors then analyzed and compared the outputs from both devices [20].

The growing diffusion of smartphones and smartwatches into humans' daily lives renders them to become the daily demands of users nowadays and hence, they can be an effective tool used to gather data about users' physical movement while performing daily activities [3] [6]. Some researcher argued that smartwatches are convenient to wear and have high user acceptability [18], which positions smartwatches as the optimal device for continuous data collection and monitoring, compared with smartphones [18]. This is because smartphones are bulkier and less portable than smartwatches and smartwatches can be easily worn at any time, such as during exercise or sleep time [19]. Both smartphones and smartwatches feature a wide array of embedded sensors, allowing different types of sensor signals being collected, including accelerometer, gyroscope, GPS sensor, etc. [3] [18]. This helps to extend the research done to a larger scope by incorporating sensor data received from various sensors to identify more complicated situations and activities [3]. Both smartphones and smartwatches offer interactivity. The researchers can implement the research without direct contact with data subjects and devices by giving feedbacks through the screens [18] [19]. Smartwatches allow the data subjects to be easily notified by the messages or warnings in terms of texts, sounds and vibrations from researchers due to the close proximity to their visual sights compared to smartphones [19].

The main difference between smartphones and smartwatches is their battery life. Smartwatches has limited battery life, especially when all embedded sensors are collecting data concurrently at high sampling rate [18]. The study in [18] shows that accelerometer consumes the least battery but would also exhausts the battery after collecting data continuously for a short period of time. Next, smartwatches have limited computational capabilities in terms of memory and CPU compared with smartphones [19]. In contrast, smartphones nowadays have better computational capabilities in terms of memory, battery, and CPU [5] [17]. For instance, researchers can execute HAR models on smartphones to classify human activity based on sensor data, which means smartphones allow real-time activity monitoring and feedback in online mode [5] [17]. Furthermore, another difference is that smartwatch records both body and arm movements, introducing higher variability [20]. There are instances when sometimes the arm movements are not associated to the corresponding human activity [20]. This will lead

to outliers in the data collected. In contrast, smartphones capture only the whole-body movement [20].

Smartphone-based HAR also introduce some limitations. For instance, smartphone based HAR poses a challenge of recognizing the location and orientation of smartphone if the data is collected in a free-living environment [16]. Different users have different ways and positions of carrying smartphones with them in real-world scenarios [16]. [4] showed that the accelerometer data exhibited significant variation for the same physical activity when data subjects shifted the position of smartphone from holding it in hand in front of chest to putting smartphone inside pocket. This makes the task of recognizing human activity to be more challenging, as changes in smartphone position will result in the decrease of recognition accuracy [17]. Besides, smartphone based HAR cannot overcome the problem common to all HAR researches, which is sensor data used to recognize human activity must be collected in controlled environment [16]. In [2], the authors showed that training using supervised data and validation using unsupervised data would result in a significant decrease of recognition accuracy, meaning the recognition algorithm was not feasible in a free-living environment. Moreover, another limitation is lack of personalization, meaning that an offline model shows lower accuracy rate of recognizing an individual's actions if the individual's data has not been used to train the model [16].

	Smartphone	Smartwatch
Daily demand	✓	✓
Wide array of sensor types	✓	✓
Weight	Bulkier than smartwatch	Lighter than smartphone
Interactivity	✓	✓
Battery life	Short	Long
Computational capabilities	Low	High
Variability	High	Low
The position of device	Various position	Always on wrist

Table 2.1: Comparison between smartphones and smartwatches

2.1.1.3 Data Annotation for Smartphone Based HAR

The dataset collected needs to be annotated to be used to perform supervised learning, which will then be used to classify human activities. Difficulty in data annotation remains as

Chapter 2

one of the open challenges for HAR, since it involves large amount of real-time data [5]. Generating ground truth annotation is a labor-intensive and time-consuming process that presents a challenge to all supervised learning [16].

Some researches applied purely human supervision to annotate the dataset collected via smartphone embedded sensors. The data subjects in [4] carried the smartphone in hands in front of the chests or inside pockets and perform the activities, with the smartphones continuously collect data. The data subjects were asked to pause and wait for a few seconds before proceeding with the subsequent activity [4]. This is used as an indication to the researcher who will perform the manual annotation later, to avoid mislabeling [4]. Manual annotation for supervised learning is time-consuming since substantial amounts of sensor signals are involved [16]. Manual annotation also poses a problem of mislabeling or misinterpretation due to human error or bias.

There are also researches perform data labelling with the help of devices. The authors in [24] developed a tool called VoiceLabel, where the data subjects can use interaction voice recognition to annotate the data. The mobile device will start or stop the recording process if the data subjects said keywords like 'Start Recording' [24]. Then, the data subjects can label the activities performed by saying the name of activities, like 'Walking', 'Standing', etc [24].

Besides, crowd labelling approach was proposed to facilitate the manual annotation process [22]. The authors in [22] used AR module to detect the physical activity of an individual. Upon detecting a shift from physical activity to standing still activity, the PLM module would then prompt the users to indicate the label of the previous physical activities [22]. The limitation of this approach is that the process was time-consuming and interrupting from the users' perspective, as they were required to take out the phone to provide the labels [22]. Frequent prompting may result in depletion of battery power within a short period of time [22].

To deal with the problem of manual labelling, development of automated annotation tools to facilitate the annotation process is needed. However, the tools developed can only be used in offline situations, meaning the data collection and data annotation process are performed asynchronously [16].

Next, [16] proposed an automatic labelling approach to address the issue of data annotation. The authors in [16] presented a heuristic function that integrates GPS sensor data and step count information from step detector. The step count information was employed as a metric to detect different physical actions, based on the step / minutes (spm) rate [16]. For

example, walking activities normally correspond to a rate between 90-110 spm [16]. However, the accuracy of the proposed heuristic function fluctuated according to the type of activities [16]. The heuristic function worked better for activities with lower variability such as long walking, compared with activities with high variability [3].

2.1.1.4 Data Storage for Smartphone Based HAR

Data storage methods depend on the type of device used to collect the data. For data collection using smartphones, the dataset created will be stored somewhere before further processing. Most of the research done using smartphones will store the collected sensor data locally inside mobile phones, since mobile phones nowadays have large memory space. This could be seen in [1], [3], [4], [7] etc. However, they did not specify how the stored data would be extracted out for processing and activity recognition. In [21] and [22], the authors specified that the data would be stored inside mobile phone storage and uploaded to cloud for later processing and storage.

2.1.1.5 Sampling frequency

Sampling frequency or also known as sampling rate, plays a vital role in human activity recognition research to capture all the necessary frequencies [17]. Sampling frequency design choice can be influenced by factors such as resource availability, desired precision, and the specific data features used for activity recognition [17]. A high sampling rate can capture sensor signals with high accuracy, but at the same time it would increase the rate of battery consumption [17]. A low sampling rate consumes less power but would introduce impulses and decrease recognition accuracy for particular activities such as foot hitting the ground during running or during Nordic walking [2] [17]. Besides, a high sampling rate should be used if only frequency domain features are considered to recognize human activities [17]. Studies in [2], [8], [9] and [17] suggested that the range of sampling frequency appropriate to record daily activities of human is between 20Hz to 30Hz. This prevents the problem of under sampling and high battery consumption [16].

In [3], the signals were recorded via smartphone embedded accelerometer and gyroscope at a sampling rate of 50Hz. The authors in [4] used a sampling frequency of 100Hz to collect acceleration signals. Next, the authors in [6] collected signals from accelerometer, gyroscope, temperature sensor and humidity sensor at a sampling rate of 1Hz. In [9], the sampling frequency being employed was 25Hz. In [16], the accelerometer signals were

collected via smartphones at 30Hz. In [25], the authors recorded acceleration signals via smartphone embedded accelerometer by using a sampling rate of 125Hz. Overall, a sampling frequency range of 1Hz to 125Hz was observed for smartphone based HAR. There may be a tradeoff between accuracy and resource consumption [17].

2.1.2 Signals Preprocessing

The time-series sensor signals collected using smartphone embedded sensors are preprocessed using various ways before they are used to train the activity recognition models.

2.1.2.1 Noise Reduction and Label Encoding

The raw sensor data collected exhibits inherent noises [5]. Preprocessing steps are required to eliminate noises in the data to produce high-quality data suitable to be fed to the activity recognition models [5].

For example, the study in [3] collected time-series signals for each dimension from smartphone embedded accelerometer and gyroscope. To eliminate noises, the signals were first subjected to median and High-pass Butterworth filter (20Hz) [3]. Then, the second filtering was performed using Low-pass Butterworth filter (3Hz) to negate the influences of gravity in accelerometer data [3]. After that, normalization was performed to keep the signals' values within the interval of $(-1, 1)$ [3]. The magnitudes of Euclidean values across three dimensions were computed to combine the three-dimensional signal into single dataset [3]. Moreover, the authors in [7] who collected data from smartphone embedded accelerometer implemented error handling as the first step of data preprocessing to remove the rows which have additional attributes or null values. Then, normalization or known as min-max scaling was performed to keep the data within the range of 0 to 1 [7].

There are also researches who do not perform any noise reduction or removal. For instance, the authors in [4] used the smartphone embedded triaxial accelerometer to capture the acceleration along the x-axis, y-axis, and z-axis. The authors chose not to perform any noise cancellation on the raw data since the data subjects were required to hold the smartphones in hands in front of the chest along the way while they performed all the six activities required [4]. The smartphone position and orientation did not change and hence, no noise was introduced. Similarly, in [6], the authors proposed that preprocessing was not necessary since the data collected were free from outliers and missing data.

Next, label encoding plays a pivotal role in supervised learning. Label encoding transforms the text data (activity name) into numerical data (activity code) to prepare the data to be input to activity recognition model [7]. In [3], activity codes were appended at the end of each row after noise removal step. Similarly, in [7], label encoding is performed by importing the class Label Encoder from sklearn library.

2.1.2.2 Feature Engineering

Feature engineering need be implemented to produce informative data representation and increase the classification accuracy [4] [10]. Raw data cannot be fed directly to activity classification model [4]. Feature extraction normally starts with 5ation of time-series sensor signals in fixed window size using window overlapping technique. Then, the desirable features will be extracted from fixed windows [10]. The features that can be extracted include time domain features, frequency domain features and discrete domain features [5] [10]. Time domain features include the mean, standard deviation, median, etc. while frequency domain features include Fast Fourier Transform (FFT) coefficients, Discrete Fourier Transform (DFT), entropy spectrum, etc. [5].

[5] states that researches involving classification using deep learning models like CNN, LSTM, Transformer Model, etc. do not implement feature extraction step. This is because deep learning techniques can generate optimal features automatically from raw data [5]. On the other hand, studies involving classification using classic machine learning models like Random Forests, Decision Tree, k-Nearest Neighbors, etc. required the implementation of feature extraction.

For instance, in [4], feature creation involved applying low pass filter and high pass filter to separate gravitational signals and body acceleration signals for each time series in different directions by setting the cutoff frequency at 0.25Hz. After that, window overlapping was performed by breaking down the datasets into smaller segments and applying window for each segment [4]. A window size of 128 samples, corresponding to 1.28 seconds of accelerometer data, is utilized for extracting features from each time series [4]. This size is well-suited for capturing various activity cycles [4]. The authors in [4] also allowed 50% of overlap between consecutive windows to minimize information loss at the edges of windows. After that, the features were extracted from each window, which include mean value, average of peak frequency (APF), variance of APF, etc. [4]. The extracted features were used for subsequent classification.

Chapter 2

Similarly, in [10], the sensor signals were transformed into fixed windows with size of 2.56 seconds each. 50% overlap was allowed [10]. Then, time domain features and frequency domain features were generated from the fixed windows of data [10]. A 561-column feature vector was generated. Next, [10] implemented the permutation importance feature selection method to assess the significance of each feature in the given set and their impacts on the classification model. Values were assigned to each feature in the feature set by using permutation importance feature selection method [10]. The 561-column feature vector in [10] represented variable-based features extracted from signals. The number of features for each signals was computed and sorted in descending order. After that, various number of features would be selected from each signals to be used as input to activity recognition model in each iteration [10]. The accuracy of activity recognition for each iteration was computed and compared with the accuracy of activity recognition when all the features were inputted [10].

[16] suggested that the window size span from 1 second to 10 seconds, highlighting that the optimal window size for common activity recognition tasks is 1 second. The authors performed segmentation of accelerometer signals by using sliding window techniques with a 50% overlap and 1 second of window size [16]. Then, the author came out with features set consisting of both time-domain features and frequency-domain features, including mean, variance, skewness, location of high peaks, etc. [16].

Furthermore, feature selection was employed in [6] to distinguish between transitional and non-transitional human activities by using accelerometer and gyroscope data. Moreover, [7] implemented feature scaling to facilitate the learning process of algorithms. The authors in [7] employed standard scaler provided in sklearn library to perform feature scaling for both training and testing datasets.

There is also research whereby feature engineering was not implemented, such as in [1] [12] and [14], whereby they involved classification using LSTM.

2.1.3 Machine Learning Algorithm

The features extracted from previous step are being used as input to different classification model.

2.1.3.1 Classic Machine Learning

Classic Machine Learning models include k-Nearest Neighbours (k-NN), Random Forests (RF), Decision Trees (DT), k-Means Clustering, Support Vector Machine (SVM), etc.

Chapter 2

Classic Machine Learning models are more suitable to be used when the input dataset has small size and lower dimensions, or when fast training is required [5]. Besides, Classic Machine Learning models can generate high accuracy with low computational requirement [5]. Classic Machine Learning models come with certain constraints. For instance, the features used in training the Classic Machine Learning models are often selected and engineered by human based on their understanding of the problem [14]. This approach can be ineffective, time-consuming and may not always result in the best possible model [14].

One of the Classic Machine Learning models used is Support Vector Machine (SVM). SVM is a supervised machine learning algorithm that can be used for both classification and regression tasks. It works by finding a hyperplane that best separates different classes in a dataset and ignoring outliers. This hyperplane aims to maximize the margin between the classes, which is the distance between the hyperplane and the nearest data points from each class. The data points that are closest to the hyperplane and influence its position are called support vectors. SVM is effective for handling both linearly separable and non-linearly separable data by using different types of kernel functions, such as linear, polynomial, and radial basis function (RBF) kernels. These kernels allow SVM to map the data into a space where it's easier to find a separating hyperplane. SVM algorithm was used in [3], [4], [6], [7], [16] and [10]. In [3], the authors used supervised SVM with a cubic polynomial kernel to recognize six activities and achieved a success rate of 99.4%. Besides, [4] used SVM to classify six activities for two datasets, one with smartphone in hand and one with smartphone in pocket and achieved accuracies of 88.76% and 72.27% respectively. [6] implemented SVM with RBF kernel and achieved 98.90% of accuracy. [16] indicates that SVM have demonstrated good accuracy outcomes and being computationally affordable, especially during the prediction phase, although it may not be the best solution when dealing with inaccuracies in labelling.

Besides, another classic models being utilized is k-Nearest Neighbors (k-NN). k-NN algorithm functions by identifying the 'k' closest data points in the training set to a new, unseen data point and then making predictions based on the labels or values of those neighboring points. The choice of 'k' value influences the algorithm's sensitivity to noise and the smoothness of the decision boundary [3]. Small 'k' value may lead to noisy predictions, high 'k' values may provide smoother but biased predictions [3]. k-NN algorithm is non-parametric, making it suitable to be used for predictions for various types of data [6]. However, k-NN can be computationally expensive for large datasets, as it requires calculating distances between

Chapter 2

the new data point and all training data points. k-NN model was being employed in [3], [6], [16] and [10].

Another two Classic Machine Learning models being employed are Decision Tree (DT) and Random Forests (RF). Both DT and RF algorithms are non-parametric supervised learning methods used for both classification and regression tasks. DT model functions by breaking down a dataset into smaller subsets while making decisions based on feature values. These decisions create a tree-like structure, consisting of a root node, branches, internal nodes, and leaf nodes. The root node presents at the beginning of the tree and the segmentation of dataset according to various features starts from here. The internal node represents a feature while the branch corresponds to a possible feature value. The leaf node represents a predicted outcome. The process of constructing a DT involves selecting the best features to split the dataset at each internal node. The feature is chosen based on criteria like Information Gain, Gain Ratio or Gini Index. The algorithm continues recursively, creating nodes and branches until a stopping condition is met. DT is easy to use and interpret. It is also computationally less expensive [16]. However, DT can be prone to overfitting, especially when the tree is deep and capture noises in the data. The RF algorithm, which is an ensemble learning method, can improve the performance of DT, by combining the predictions of multiple DT [6] [13]. The process begins with creating a collection of DTs, each trained on a different subset of the data obtained through random sampling with replacement, by using bagging or boosting technique. These techniques help to reduce overfitting and enhance generalization. Besides, only a random subset of features is considered for each split during the construction of individual DT within the RF. This introduces further diversity among the trees and prevents them from becoming overly correlated. Once the ensemble of DT is built, the algorithm combines their predictions, by using majority voting for classification tasks or averaging for regression tasks, to arrive at a final prediction. RF can prevent the overfitting problem and manage outliers or noises well by due to the introduced randomness [13]. However, RF can use large space and time when there are many DT [13]. The DT algorithm was used in [3] and [10] while the RF algorithm was used in [4], [6], [7], [16], [20] and [10]. RF model can achieve better accuracy compared to DT model, as in [3] where a binary DT achieved 53.1% of success rate while RF model with 20 subtrees achieved 91.7% of success rate.

2.1.3.2 Deep Learning

Chapter 2

Deep Learning models include Convolutional Neural Network (CNN), Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), Transformer model, etc. Deep Learning models possess the capability of generating optimized features automatically and spontaneously from raw input sensor data [5]. This automation, through multiple hidden layers, facilitates the discovery of previously unknown patterns [5]. Deep Learning models can recognize complicated activities while achieving high accuracy [5]. Deep Learning models come with certain limitations. For instance, deep learning models which have complicated structures present a challenge in interpreting and understanding how the models work [5]. Besides, deep learning models are more suitable to be used when the datasets have large sizes, to ensure that the models have effective performance [5]. Deep Learning models require a high computational burden, demanding significant processing resources for training and inference procedures [5].

Convolutional Neural Network (CNN) is a feedforward neural network primarily designed for tasks involving visual data, such as image recognition and computer vision. Each neuron in CNN processes data in its receptive field only. A typical CNN model consists of three layers: convolutional layer, pooling layer, and fully connected layer. The convolutional layer serves as the fundamental component of CNN model, responsible for the main computational tasks. The layer first scan through the input image by using small kernels to detect specific features such as edges, corners, textures, etc. After adding bias and applying activation function, the convolution features are passed to next layer. The pooling layers are placed between convolutional layers. They are often used to reduce spatial dimensions, retaining the most important information while decreasing computational load. Some pooling that are normally used are Max Pooling and Average Pooling. CNN also incorporate fully connected layers to make predictions on the extracted features. CNN introduces some challenges such as large dataset and high computing power are required for model training. CNN algorithm was used in [7], [23] and [26]. The authors in [7] created a dataset consisting of six activities by using smartphone-embedded accelerometers. The training dataset that was being preprocessed was used to train CNN model by using Softmax activation function, applying dropout technique with probability of 0.5, using Adam as the optimizer and sparse categorical entropy for multiclass classification [7]. The CNN model was able to achieve an accuracy of 99% [7]. [26] employed a two-convolutional-layered and two-dropout-layered CNN model to prevent overfitting problem. The model achieved 91% accuracy [26].

Another deep learning algorithm is Recurrent Neural Network (RNN). RNN is a class of artificial neural networks designed for sequential data processing, such as natural language processing, time-series forecasting, etc. According to Figure 2.1, the input data at time t goes through computation steps in hidden layer and produce an output [1]. The output from time t , along with the input data from time $t+1$ become the input for the hidden layer at time $t+1$ [1]. This is one of the key features of RNNs, which is recurrent connections, allowing information to flow from one time step to the next. This looping connection allows the network to maintain memory and capture dependencies in sequential data. RNNs are trained using backpropagation through time (BPTT), which will adjust its weights and biases during training to minimize the difference between its predictions and the actual target values, thereby learning to model sequential patterns. Traditional RNNs introduce long-term dependency problem, where the network struggles to capture and learn relationships between data points that are separated by a significant number of time steps [1]. Besides, RNN also introduces vanishing gradient or exploding gradient problem, where the gradients used in the backpropagation algorithm become extremely small or extremely large. This is primarily due to the repeated multiplication of weight matrices in the network during backpropagation.

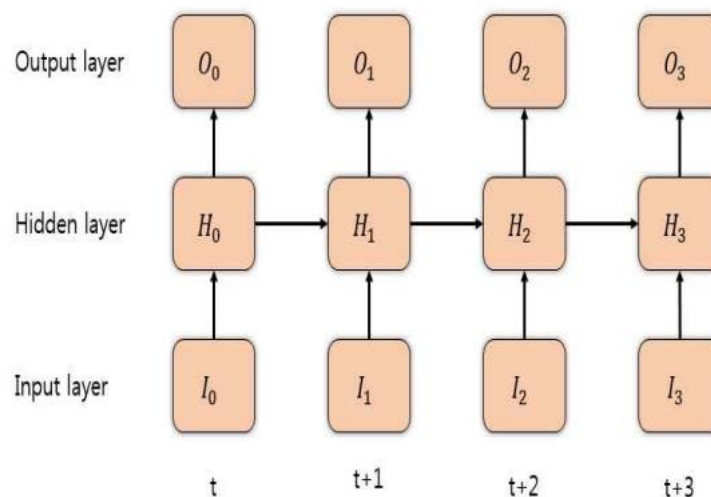


Figure 2.1: RNN model in [1]

The next deep learning algorithm being commonly used is Long Short-Term Memory Network (LSTM). LSTM is advanced version of RNN architecture which is designed to solve the problem of vanishing gradient or exploding gradient [23]. LSTM networks consist of LSTM cells, which are connected in a sequential manner, allowing them to process data over multiple time steps. Each LSTM cell contains several crucial components, including input gate,

output gate, forget gate and memory cell [1]. The input gate determines which parts of the new input data are important and should be incorporated into the cell's memory. The forget gate decides what information from the previous cell state should be retained or discarded by using sigmoid activation function [12]. The function will output value between 0 and 1, where 0 represents to discard the data while 1 represents to retain the data [12]. The output gate regulates the information that will be passed to the next time step as the output. The memory cell can store and carry data over long sequences, making it capable of capturing long-range dependencies in the data. Similar with RNN, LSTMs are trained using BPTT. LSTM was being used in [1], [12], [14], [23] and [26]. To prevent overfitting, the authors in [12] applied dropout techniques to randomly deactivate certain neurons in a layer by setting their values to zero during training. Fully operate neural network was used during testing [12].

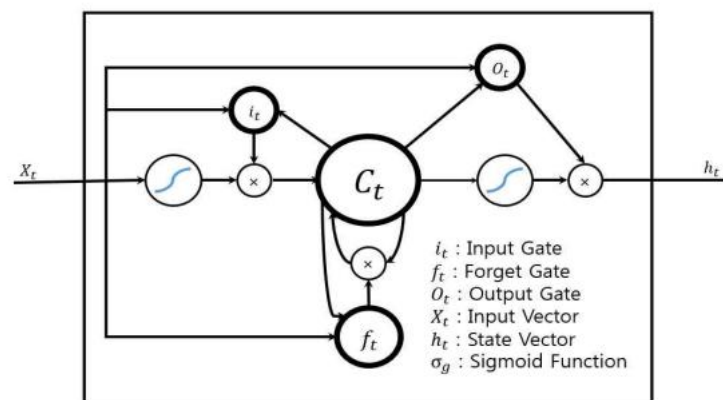


Figure 2.2: LSTM neuron in [1]

2.1.4 Performance Evaluation

In HAR, the evaluation of activity recognition model's performance is of paramount importance to assess the effectiveness of classification algorithms. Performance metric serves as a quantitative measure to determine the accuracy and reliability of activity recognition models. The performance metrics will provide insights into how well a model generalizes to new, unseen data and its ability to correctly recognize different activities.

Some of the commonly utilized performance metrics in HAR include accuracy, precision, recall and f1-score. For example, studies in [1], [4], [6], [7], [16], [25] and [12] used accuracy as a performance metrics. Accuracy measures the percentage of overall correct predictions. Besides, researches in [6] [16] involved the calculation of precision, recall and f1-score. Precision, or positive predictive value (PPV) measures the accuracy of positive prediction. Recall, or true positive rate (TPR) measures the ratio of positive samples that are

correctly detected. F1-score combines the precision and recall into a single score and strikes a balance between them. Furthermore, researches in [3], [7] and [12] presented a comprehensive view of activity recognition model classification by using confusion matrices, detailing true positives, true negatives, false positives, and false negatives.

2.1.5 Summary of the Technologies Review

In summary, the choice of data source in smartphone based HAR research plays a crucial role in shaping the quality and relevance of the collected data. Researchers have predominantly pursued two main approaches: creating independent, customized datasets and utilizing existing public datasets. Creating an independent dataset allows the researchers to have full control over the data quality, ensuring the data collected is consistent, and of high quality. Besides, researchers can also customize the data collection to match the specific requirements of the research, focusing on the types of activities, scenarios or contexts that are most relevant. However, inaccurate sensor data may be collected due to different ways of sensor placement and attachment on data subjects. Using existing dataset may solve the problem of data accuracy, but it can lead to limited applicability to real-world scenarios, as it is difficult to find an existing dataset which perfectly fits the research purpose and scope.

Next, in terms of data collection device, most researchers chose to use smart devices as the data collection device, including smartphones and smartwatches. This is due to the growing diffusion of smartphones and smartwatches into humans' daily lives renders them to become the daily demands of users nowadays, which make them suitable devices to collect sensor signals regarding human daily activities. Besides, both smartphones and smartwatches feature a wide array of embedded sensors, allowing different types of sensor signals being collected. Both smartphones and smartwatches offer interactivity. Smartwatches are lighter in weight compared to smartphones. However, smartwatches have limited battery life and computational capabilities in terms of CPU and memory compared to smartphones. Additionally, smartwatch records both body and arm movements, introducing higher variability. In contrast, smartphones capture only the whole-body movement.

Another important element in data collection is the sampling frequency. Sampling frequency design choice can be influenced by factors such as resource availability, desired precision, and the specific data features used for activity recognition. A high sampling rate can capture sensor signals with high accuracy, but at the same time it would increase the rate of battery consumption [17]. A low sampling rate consumes less power but would introduce

Chapter 2

impulses and decrease recognition accuracy for particular activities. The range of sampling frequency to record daily activities of human is suggested to be between 20Hz and 30Hz, which caters both the under sampling and battery consumption problems. However, a sampling frequency range of 1Hz to 125Hz was observed for smartphone based HAR research reviewed. There may be a tradeoff between accuracy and resource consumption [17].

In terms of data annotation, most of the reviewed system still required manual annotation, which was tedious and time-consuming since it involved a large amount of sensor data. Manual annotation also prone to human mistake and mislabeling because the data subjects perform the physical activity continuously. There is research which perform data labelling with the help of devices. The authors used voice recognition technique to perform data annotation. Besides, there is also research done on using crowd labelling approach to facilitate the manual annotation process, which prompt the data subjects to provide the labels of the activities. Moreover, there is also research proposing a heuristic function to label the data automatically, which is then proved to have fluctuated accuracy according to the type of activities.

In terms of data storage, the data storage methods depend on the type of data collection device. The research based on smart devices allow the data collected to be stored temporarily inside the local storage and uploaded later to cloud server for further processing.

Signals preprocessing in smartphone-based Human Activity Recognition (HAR) involves essential steps to prepare raw sensor data for training activity recognition models. This process primarily includes noise reduction and label encoding, as well as feature engineering. In many HAR studies, the raw sensor data collected from smartphones inherently contain noise, necessitating preprocessing steps to enhance data quality. Techniques like median and Butterworth filters are employed to eliminate noise [3]. Additionally, label encoding plays a pivotal role in supervised learning by converting activity names into numerical codes [7]. Some studies choose not to perform noise reduction when sensor positions and orientations remain stable [4]. Similarly, data sets free from outliers and missing data may not require extensive preprocessing [6]. Next, feature engineering is a critical step to create informative data representations and enhance classification accuracy. The process involves segmenting time-series sensor signals into fixed windows and extracting relevant features, including time and frequency domain features [4]. Deep learning models like CNN and LSTM can automatically generate optimal features from raw data, reducing the need for explicit feature extraction [5]. However, classic machine learning models often require feature extraction [5]. Most of the research allows the consecutive segmented windows to have 50%

overlap to minimize information loss at the edges of windows, as in [4], [10] and [16]. The window size needs to be carefully decided to capture various activity cycles. The suggested window size range between 1 second to 10 seconds [16].

The machine learning algorithms used for classification can be categorized into two main groups: Classic Machine Learning and Deep Learning models. The choice between these two categories depends on the size of the dataset, computational resources, and the complexity of the problem. Classic Machine Learning models, including k-Nearest Neighbors (k-NN), Random Forests (RF), Decision Trees (DT), k-Means Clustering, and Support Vector Machine (SVM), are preferred when dealing with smaller datasets or when rapid model training is required. These models are known for their ability to achieve high accuracy with relatively low computational demands. However, they often rely on manually engineered features, which can be time-consuming and may not always lead to the best model performance. Deep Learning models, such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), and Transformer models, are particularly adept at automatically generating optimized features from raw sensor data. They excel in recognizing complex activities and achieving high accuracy but require large datasets and substantial computational resources. Knowledge to interpret deep learning model is also one of the challenges.

Last but not least, the evaluation of activity recognition models in Human Activity Recognition (HAR) research is a critical step to assess the effectiveness and reliability of classification algorithms. Performance metrics serve as quantitative measures that gauge how accurately activity recognition models can identify different activities and generalize to new, unseen data. Commonly utilized performance metrics in HAR include accuracy, precision, recall, f1-score, and confusion matrix. The choice of performance metrics or combination of performance metrics depends on the nature of the applications and objectives of the activity recognition tasks.

2.2 Review of the Existing Systems

2.2.1 A Study on Human Activity Recognition Using Accelerometer Data from Smartphones [4]

The authors in [4] collected acceleration data from smartphone-embedded accelerometer by using a sampling frequency of 100Hz. Two datasets were created, collecting acceleration signals for six activities including running, slow-walk, fast-walk, aerobic-dancing,

Chapter 2

walking upstairs and walking downstairs. The authors included action which is not commonly found in public datasets such as aerobic dancing [4]. The first dataset was asking the data subjects to hold the smartphones in hands in front of the chests while the second dataset was asking the data subjects to put the smartphones in their pockets while performing activities. The collected data were stored locally inside smartphone. However, the authors did not specify how the stored data would be extracted out for processing and activity recognition. Manual annotation was used to label the datasets. The smartphone recorded signals continuously while data subjects performed the six activities. The data subjects were asked to wait for a few seconds after completing one activity before starting the next activity [4]. This is used as an indication to the researcher who will perform the manual annotation later, to avoid mislabeling [4]. There were in total 79573 samples of triaxial accelerometer being collected.

For data preprocessing, [4] did not perform any noise cancellation on the raw data. This is because the data subjects held the smartphones in hands in front of the chest or inside pockets along the way while they performed all the six activities required [4]. The smartphone position and orientation did not change and hence, no noise was introduced. On the other hand, [4] indicates that feature engineering need be implemented to produce informative data representation and increase the classification accuracy. Raw acceleration data cannot be fed directly to activity classification model [4]. In [4], feature creation involved applying low pass filter and high pass filter to separate gravitational signals and body acceleration signals for each time series in different directions by setting the cutoff frequency at 0.25Hz. After that, feature extraction was performed by first implementing window overlapping [4]. The datasets were broken down into smaller segments and applying window for each segment [4]. A window size of 128 samples, corresponding to 1.28 seconds of accelerometer data, is utilized for extracting features from each time series [4]. This size is well-suited for capturing various activity cycles [4]. The authors in [4] also allowed 50% of overlap between consecutive windows to minimize information loss at the edges of windows. After that, the features were extracted from each window, which include mean value, min value, max value, average of peak frequency (APF), variance of APF, etc. [4]. To find the most representative features in the feature vector, the authors in [4] applied clustering-based evaluation approach. As a result, they narrowed down their original 24-dimensional feature vector to 18 relevant features. The extracted features were used for subsequent classification.

Chapter 2

Next, the authors in [4] performed classification for the two datasets using individual classifiers like Multilayer Perceptron, RF, LMT, SVM, Simple Logistic and LogitBoost. 10-fold cross validation was used.

Classifier	Accuracy (in-hand)	Accuracy (in-pocket)
Multilayer Perceptron	89.48%	89.72%
SVM	88.76%	72.27%
Random Forest	87.55%	85.15%
LMT	85.89%	85.04%
Simple Logistic	85.41%	85.05%
Logit Boost	82.54%	82.24%

Figure 2.3: Accuracy for individual classifiers for different dataset in [4]

Besides, the authors also performed classification using combined classifiers. The authors proposed that various classifiers might provide varied insights into the patterns that need classification, potentially leading to increased accuracy, efficiency, and robustness in the classification process [4]. The classification using combined classifiers applied average of probabilities to achieve final predictions [4].

IN-HAND		IN-POCKET	
Classifier	Accuracy	Classifier	Accuracy
MP, LogitBoost, SVM	91.15%	MP, Random Forest, SimpleLogistic	90.34%
MP, LogitBoost, SVM, LMT	90.90%	MP, LogitBoost, SimpleLogistic, Random Forest	90.34%
MP, LogitBoost, SVM, Random Forest	90.90%	MP, LogitBoost, SimpleLogistic	88.47%
MP, LogitBoost	88.51%	MP, Random Forest, LMT, SimpleLogistic	88.16%
MP, SVM	88.27%	MP, LMT, SimpleLogistic	86.92%
MP, LogitBoost, SVM, Random Forest, LMT	81.10%	MP, LogitBoost, SimpleLogistic, Random Forest, LMT	83.18%

Figure 2.4: Accuracy for combined classifiers for different dataset in [4]

The authors in [4] found that the acceleration data showed significant variation for the same physical activity when data subjects shifted the position of smartphones from holding in hands in front of chests to putting smartphones inside pockets. Besides, [4] found that the combined classifiers using the first three individual classifiers that had good performance outperformed the individual classifiers, in terms of accuracy. [4] suggested that the HAR architecture they proposed can accurately recognize human activities, without concerning about the smartphone's position.

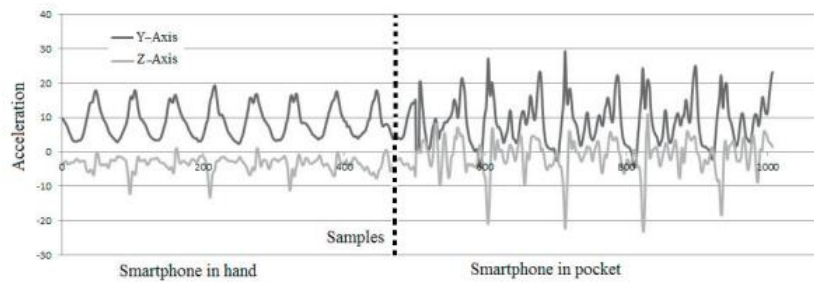


Figure 2.5: Variation in acceleration signals when smartphones' positions change in [4]

2.2.2 Assessment of Human Activity Recognition based on Impact of Feature Extraction Prediction Accuracy [10]

In [10], UCI HAR dataset was used for activity recognition task. UCI HAR dataset consists of signals collected at 50Hz for six activities, including walking, walking upstairs, walking downstairs, sitting, standing, and laying, via smartphone-embedded accelerometer and gyroscope.

Data preprocessing was done by the authors of the dataset. The data was labeled manually. Then, the data were preprocessed by first applying noise filters. Next, the data were segmented using window sliding technique. The data was segmented using fixed width sliding windows of 2.56 seconds with a 50% overlap, resulting in 128 readings per window. To isolate the sensor acceleration signal, which encompasses both gravitational and body motion elements, a Butterworth low-pass filter was employed. The intention was to differentiate body acceleration from gravitational force. The author of the dataset assumed that gravitational force consists of low-frequency components. Hence, a filter with a cutoff frequency of 0.3 Hz was applied. Subsequently, time domain features and frequency domain features were generated from the fixed windows of data [10]. The features generated include mean, standard deviation, max value, min value, etc. A 561-column feature vector was generated.

The authors in [10] proposed that using all the features in the feature vector for activity recognition was not necessary. Additionally, the features in the feature vector do not have equal weightage. Therefore, [10] implemented the permutation importance feature selection method to assess the significance of each feature in the given set and their influences on the classification model. Values were assigned to each feature in the feature set by using permutation importance feature selection method [10]. The 561-column feature vector in [10] represented variable-based features extracted from signals. The number of features for each signals was computed and sorted in descending order. After that, various number of features

Chapter 2

would be selected from each signals to be used as input to activity recognition model in each iteration [10].

The authors employed seven machine learning algorithm for classification, including Neural Network, SVM, DT, RF, k-NN, SGD and LG. The accuracy of activity recognition for each iteration was computed and compared with the accuracy of activity recognition when all the features were inputted [10].

The authors in [10] found that by incorporating more features with the highest permutation values at each iteration, the classifier's accuracy approaches very closely to that achieved when utilizing all available features. The classifier's accuracy only experiences a slight decrease even when using just single feature [10]. Besides, the results also showed that the accuracy of the models increased exponentially at the first few iterations. This is because the features with highest value were selected in the first few iterations, and they have significant influence on the overall accuracy [10]. After a few iterations, the less significant features were selected and hence, they have less influence on the overall accuracy [10]. The less influential features can then be ignored to reduce the dataset's size and computation resources needed [10]. The authors conclude that high accuracy of classification with reduced size of datasets can be achieved by using permutation importance feature selection method proposed [10].

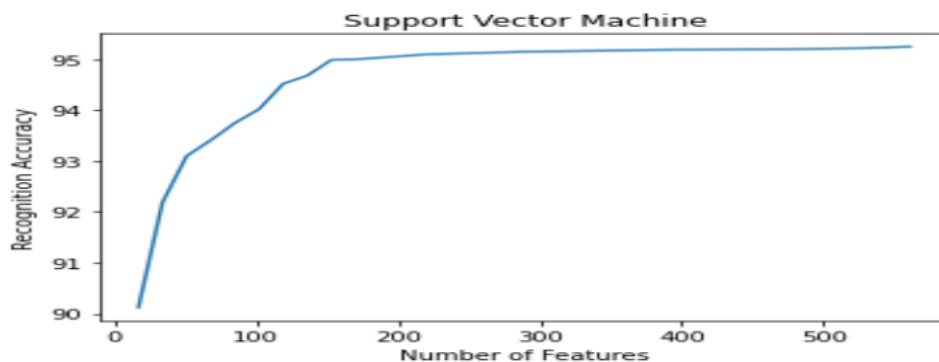


Figure 2.6: Accuracy of SVM change when number of features used change in [10]

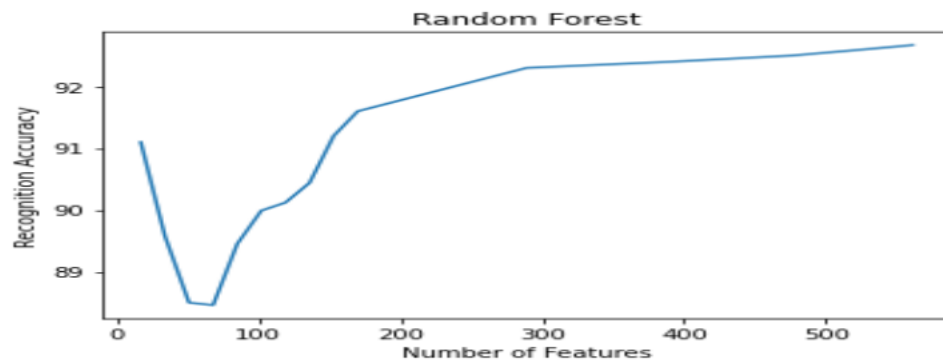


Figure 2.7: Accuracy of RF change when number of features used change in [10]

2.2.3 Human Activity Recognition Using LSTM Network with Dropout Technique [12]

In [12], the authors chose to use public datasets for model training and evaluation. No preprocessing was mentioned in [12].

The authors indicated that RNN introduced long-term dependencies problem. Hence, a LSTM network where each neuron encompasses four components was being employed. The first component is the input gate, which receives input from current state and sends output to next layer. The next element is the forget gate, deciding what information from the previous cell state should be retained or discarded by using sigmoid activation function [12]. The function will output value between 0 and 1, where 0 represents to discard the feature while 1 represents to retain the feature [12]. The following component is the update gate, where new features are introduced after the removal of unwanted ones. This is achieved through vector addition, allowing the network to update its information. The final component is the output gate, which regulates the information that will be passed to the next module by processing through a sigmoid function and a tanh function. The authors in [12] suggested that the dropout technique should be used to prevent overfitting of data. The dropout technique will randomly deactivate certain neurons in a layer during training of the model. This makes the layer appear and function as if it had a different number of nodes and connections than the preceding layer. During training, every update to a layer incorporates a fresh perspective of the current layer, which introduces randomness and helps prevent overfitting. During testing, the entire neural network was considered, without dropout [12].

For the activity recognition purpose, the authors proposed a simple unidirectional LSTM network. The network consists of an input layer, a hidden layer with 32 neurons, a layer with sigmoid activation function and an output layer. The authors first split the dataset into training and testing dataset. The authors then divided the large training dataset into smaller

Bachelor of Computer Science (Honours)
Faculty of Information and Communication Technology (Kampar Campus), UTAR

batches to enhance the performance. The authors used a constant batch size of 4 and perform iterations over a specified number of epochs. This is done because smaller dataset will save computational resources and execute faster. Besides, the model also used backpropagation to adjust the weights and biases. Adam optimizer was applied to adjust the model’s learning rate and to locate the global minimum or the loss value for each iteration. Categorical cross-entropy was used to calculate the loss when there are inaccurate predictions.

The result showed that the proposed LSTM model achieved an accuracy of 92.67% of recognizing activities such as laying, sitting, standing, walking, walking downstairs, and walking upstairs [12].

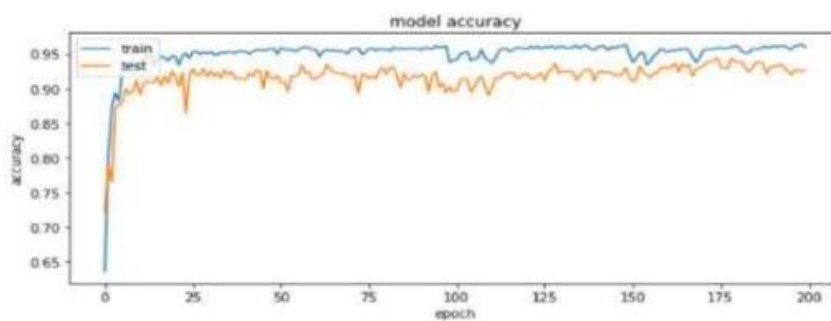


Figure 2.8: Accuracy in [12]

2.2.4 LSTM Networks Using Smartphone Data for Sensor-Based Human Activity Recognition in Smart Homes [14]

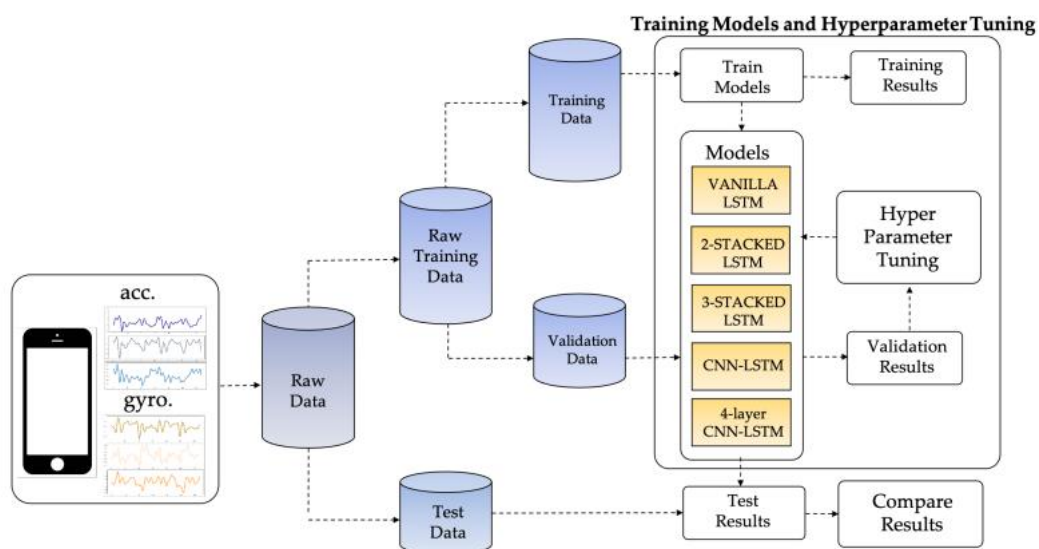


Figure 2.9: System architecture diagram in [14]

In [14], UCI HAR dataset was being used for activity recognition task. The datasets collected signals via smartphone embedded accelerometer and gyroscopes at a constant rate of 50 Hz. The data subjects were required to hold the smartphone at waist level while performing six activities, i.e., walking, walking upstairs, walking downstairs, sitting, standing, and lying down.

The data preprocessing was done by the authors of the dataset, instead of authors in [14]. The data was labeled manually. Then, the data were preprocessed by first applying noise filters. To capture the human body motion, a third-order Butterworth low-pass filter was employed with a cutoff frequency of 20 Hz [14]. The frequency was chosen because 99% of human activities' energy were below 15 Hz [14]. Next, the data were segmented using window sliding technique. The data was segmented using fixed width sliding windows of 2.56 seconds with a 50% overlap [14]. There are a few considerations of segmenting the dataset into 2.56 seconds of windows. Firstly, it accounts for the typical walking rate of an average individual, which falls within the range of 90 to 130 steps per minute, equivalent to a minimum of 1.5 steps per second. Secondly, the chosen window size ensures that each segmented window encompasses at least one complete walking cycle, which consists of two steps. Thirdly, this approach accommodates individuals with slower walking cadences, including the elderly and those with disabilities, by assuming a minimum speed equal to 50% of the average human cadence. After that, the data went through normalization process, showing a mean value of 0 and a variance value of 1 [14]. At the first stage, the dataset was split into training dataset (71.39%) and testing dataset (28.61%). Next, during the second stage where the model was trained and hyperparameter tuning was performed, the training dataset was split again into 75% of training datasets and 25% of validating datasets.

For activity recognition using LSTM networks, the authors proposed five LSTM network architectures, which includes Vanilla LSTM, 2-Stacked LSTM, 3-Stacked LSTM, CNN-LSTM, and 4-layer CNN-LSTM [14]. The Vanilla LSTM model, or the original LSTM model consists of a single hidden layer and a feedforward output layer [14].

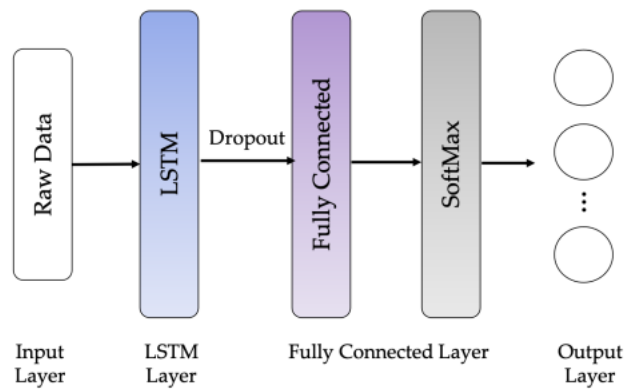


Figure 2.10: Vanilla LSTM model in [14]

The 2-Stacked LSTM model comes with two hidden layers while the 3-Stacked LSTM model comes with three hidden layers. Each hidden layer associates with multiple memory cells.

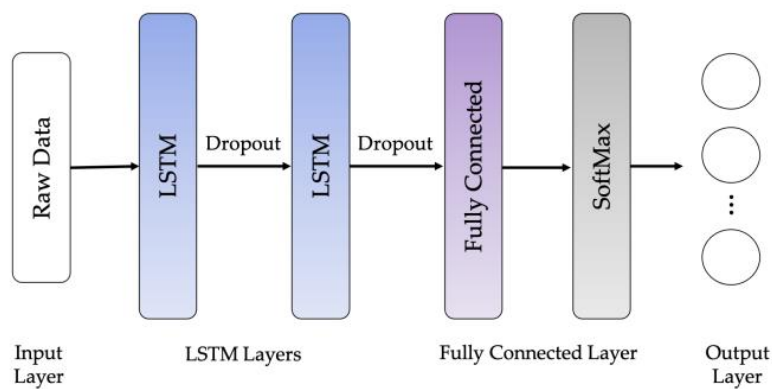


Figure 2.11: 2-Stacked LSTM model in [14]

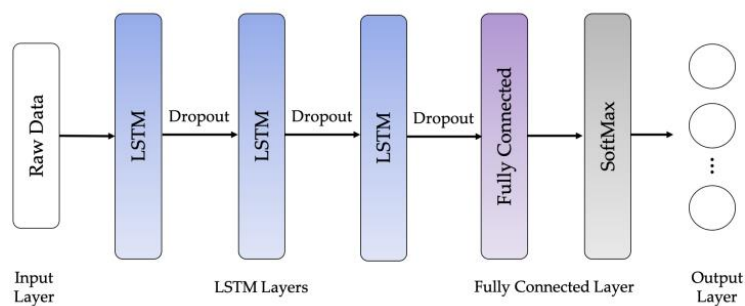


Figure 2.12: 3-Stacked LSTM model in [14]

The CNN-LSTM network incorporates CNN layers to extract features from raw input data. The extracted features were then integrated with LSTM layer to facilitate sequence prediction.

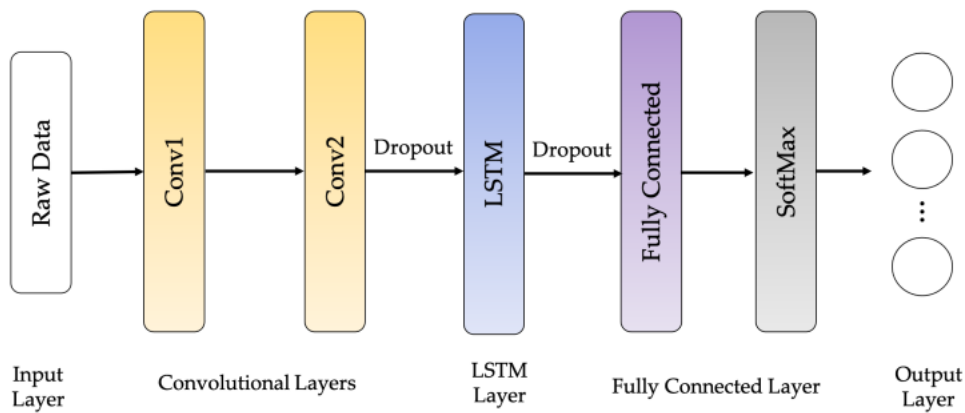


Figure 2.13: CNN-LSTM model in [14]

To improve activity recognition accuracy, the authors in [14] proposed a 4-layer CNN-LSTM network. Four 1-D CNN layers were used to extract features from raw input data by using ReLU activation functions. After features extraction, a max pooling layer was implemented to summarize the feature maps generated by the previous CNN layers and to reduce computational complexities. Multiple dropout layers were added on top of pooling layer to prevent overfitting. The dimensions of the feature maps need to be reduced by using a flattened layer to transform feature map in matrix representation to vector representation. This allows LSTM to be able to process the feature maps. The output then went through the LSTM layer to model temporal dynamics and to activate the feature maps. The final layer consists of a fully connected layer and a Softmax layer for activity classification purpose. The optimal hyperparameters for the 4-layer CNN-LSTM network was determined by using Bayesian Optimization.

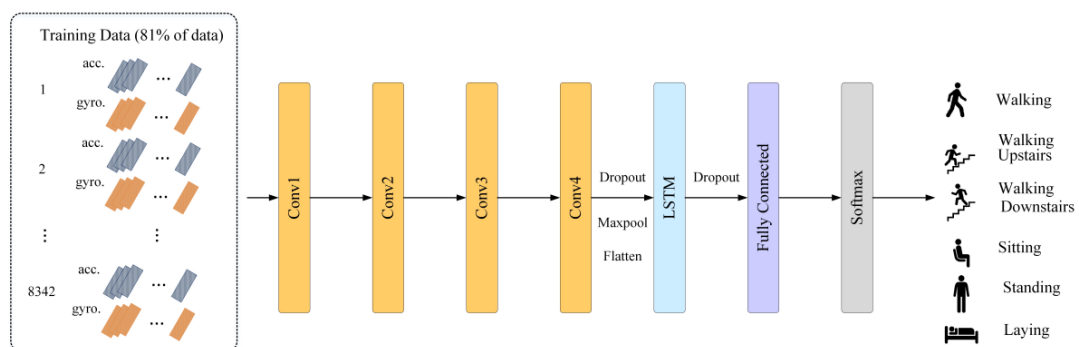


Figure 2.14: 4-layered CNN-LSTM model in [14]

The result shows that the 4-layer CNN-LSTM model achieved an accuracy of 99.39%, outperforming other models. The proposed 4-layer CNN-LSTM model introduced two benefits. One of them is the CNN layers in the proposed network can directly map the spatial representation of raw sensor data for feature extraction. The next benefit is that the LSTM layer excels at capturing and leveraging temporal dependencies within the data and use this information to improve feature extraction for HAR.

2.2.5 Summary of the Existing Systems

	[4]	[10]	[12]	[14]	Proposed System
Create own datasets	✓				✓
Data collection using smartphone	✓				✓
Automated annotation					✓
Save log file into local storage	✓				✓
Upload log file to cloud					✓
Noise management		✓		✓	Drop rows with null values. Drop rows when timestamp = 0. Sort in ascending order of timestamp.
Data segmentation	✓	✓	✓	✓	✓
Feature Engineering	✓	✓			✓
Classification model	Classic Machine Learning algorithm	Classic Machine Learning algorithm	Deep Learning algorithm	Deep Learning algorithm	LSTM
Performance metric	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy

Table 2.2: Comparison between existing systems and proposed system

Chapter 2

According to the Table 2.2, only [4] chose to create dataset by collecting signals at 100Hz using smartphone embedded accelerometer. [10], [12] and [14] chose to use public datasets, where [10] and [14] used UCI HAR dataset. The UCI HAR dataset was constructed by using signals collected from smartphone embedded accelerometer and gyroscope at 50Hz. The dataset used in [12] did not have any extra information. [4] annotate the dataset manually and store the dataset inside smartphone local storage. However, the authors in [4] did not specify how to extract the dataset out for further processing and analysis. For UCI HAR dataset, the authors also annotated the dataset manually.

For data preprocessing, only [10] and [14] applied noise filters for noise management since they used similar public datasets. [4] did not perform any noise cancellation since the authors suggested that no noise was being introduced into the dataset they created. [12] did not specify anything about noise management. For feature engineering, [4] applied low pass filter and high pass filter to separate gravitational signals and body acceleration signals. Similarly, [10] and [14] applied Butterworth low pass filter to differentiate body acceleration from gravitational force. Next, [4], [10] and [14] also performed data segmentation using window sliding techniques with 50% overlap between consecutive windows to minimize information loss at the edges of windows. After that, only [4] and [10] implemented feature extraction from raw data since [4] and [10] performed classification using classic machine learning algorithms while [12] and [14] used deep learning algorithms which can automatically extract features from raw data. Both [4] and [10] wanted to find the most representative features in the feature vector generated. [4] applied clustering-based evaluation approach while [10] applied permutation importance feature selection method.

For activity recognition, [4] and [10] employed classic machine learning algorithms, including SVM, RF, DT, k-NN, etc. [4] also proposed the combination of classifiers to improve the performance of activity recognition. On the other hand, [12] and [14] employed deep learning algorithms. [12] implemented a simple, unidirectional LSTM network, which consists of an input layer, a hidden layer with 32 neurons and an output layer. The authors in [12] divided the large training datasets into batches to improve performance and to save computational resources. [12] also applied dropout techniques to prevent overfitting. [14] implemented original LSTM network, stacked LSTM network and CNN-LSTM network. The authors in [14] performed Bayesian Optimization for hyperparameter tuning.

All the four studies used accuracy as the performance metrics. [4] concludes that the combined classifier using SVM, Multilayer Perceptron and LogitBoost achieved the highest

accuracy at 91.15%. The authors also proposed that their recognition method is robust enough to recognize human activities without concerning about smartphone's position. [10] suggested that high accuracy of classification with reduced size of datasets can be achieved by using permutation importance feature selection method proposed. The proposed simple unidirectional LSTM network with dropout technique in [12] achieved an accuracy of 92.67%. Whereas the proposed 4-layer CNN-LSTM architecture in [14] achieved an accuracy of 99.39%.

2.2.5.1 Strengths

One of the strengths in [4] is the authors chose to create dataset by themselves. Creating an independent dataset allows the researchers to have full control over the data quality, ensuring the data collected is consistent, and of high quality. This helps to reduce the noises and biases in the dataset. Besides, researchers can also customize the data collection to match the specific requirements of the research, focusing on the types of activities, scenarios or contexts that are most relevant. For instance, [4] included the actions which are not commonly found in existing open datasets, such as aerobic dancing.

Besides, another strength found in [10] is the implementation of feature engineering and permutation importance features selection method. Feature engineering need be implemented to produce informative data representation and increase the classification accuracy [10]. The data were segmented using window sliding technique and features were extracted from fixed windows of data, including both time-domain features and frequency-domain features. Then, the permutation importance feature selection method was implemented to assess the significance of each feature in the given set and their influences on the classification model. By doing this, the size of the feature vector used to train machine learning model can be significantly reduced by removing the least influential features from the vector.

Another strength can be found in [12], where the authors used dropout techniques to prevent the problem of overfitting of LSTM model. The dropout technique will randomly deactivate certain neurons in a layer during training of the LSTM model. This makes the layer appear and function as if it had a different number of nodes and connections than the preceding layer. During training, every update to a layer incorporates a fresh perspective of the current layer, which introduces randomness and helps prevent overfitting. Besides, another strength is [12] divided the large training dataset into smaller batches to enhance the performance. The authors used a constant batch size of 4 and perform iterations over a specified number of

epochs. This is done because smaller dataset will save computational resources and execute faster.

2.2.5.2 Weaknesses

One of the weaknesses in [4] is the inefficient data engineering system used to collect, annotate, and store sensor signals. The authors in [4] collected signals using smartphone. The signals were stored inside mobile phone locally. After the data collection process is done, the data were then extracted out and manually annotated. The authors in [4] did not specify how the data was going to be extracted out.

Another common weakness in [10], [12] and [14] is the usage of public datasets. Public datasets only cover common human activities, such as walking, standing, climbing upstairs, etc. They may have limited relevance with your research purpose and context and limited diversity in terms of activities type and sensor type. [5]. This will limit the applicability and accuracy of the researches. The authors in [10], [12] and [14] chose to use public datasets, possibly for the sake of convenience. Constructing a dataset from scratch requires substantial efforts, time, and techniques, especially when considering the necessity for large datasets to train deep learning models. It saves them a lot of efforts to use preprocessed and well-defined public datasets.

2.3 Proposed System

The proposed system will implement smartphone based human activity recognition system. Firstly, the data engineering system proposed will be used for data collection, data annotation and data storage by using SensorData application and SensorDataLogger application developed. Two Android devices will be installed with the applications, whereby the device installed with SensorDataLogger will be carried by data subjects in their pockets while they perform the activities. The device installed with SensorData will be carried in hand by the researcher or the data subject himself or herself. The two applications are connected via Bluetooth to send and receive instructions. The researcher will collect signals from accelerometer and gyroscope embedded in smartphone at 100Hz of sampling frequency for four activities: standing, walking, climbing upstairs and climbing downstairs. The log files collected for each activity will be annotated automatically by the data engineering system proposed and saved inside the local storage. After that, the log files can be uploaded to Google Cloud Storage for further processing.

Chapter 2

Next, the log files collected will be extracted out from Google Cloud Storage by using Google Colab. The log files will be combined into single dataset. After that, common data preprocessing will be implemented on dataset, such as dropping rows with null values, dropping rows when timestamp equals to zero and sorting the dataset in ascending order of timestamp. Subsequently, data exploration will be performed to understand better the dataset's characteristics, structure, quality and to identify any data preparation steps needed.

Then, data segmentation will be performed, by sliding the dataset into windows of size 0.5 seconds each. We allow 50% overlap between consecutive windows to minimize information loss. The windows of data will be divided into 80% of training dataset and 20% of testing dataset. We then proceed to LSTM model training and evaluation using the windows of data. Hyperparameter tuning is also performed to further enhance the performance of LSTM model by identifying the combination of hyperparameters that work best. Finally, we will implement model testing using dedicated testing dataset to test the model's generalization capability on new, unseen data.

Chapter 3

System Methodology/Approach

3.1 System Design Diagram

3.1.1 System Architecture Diagram

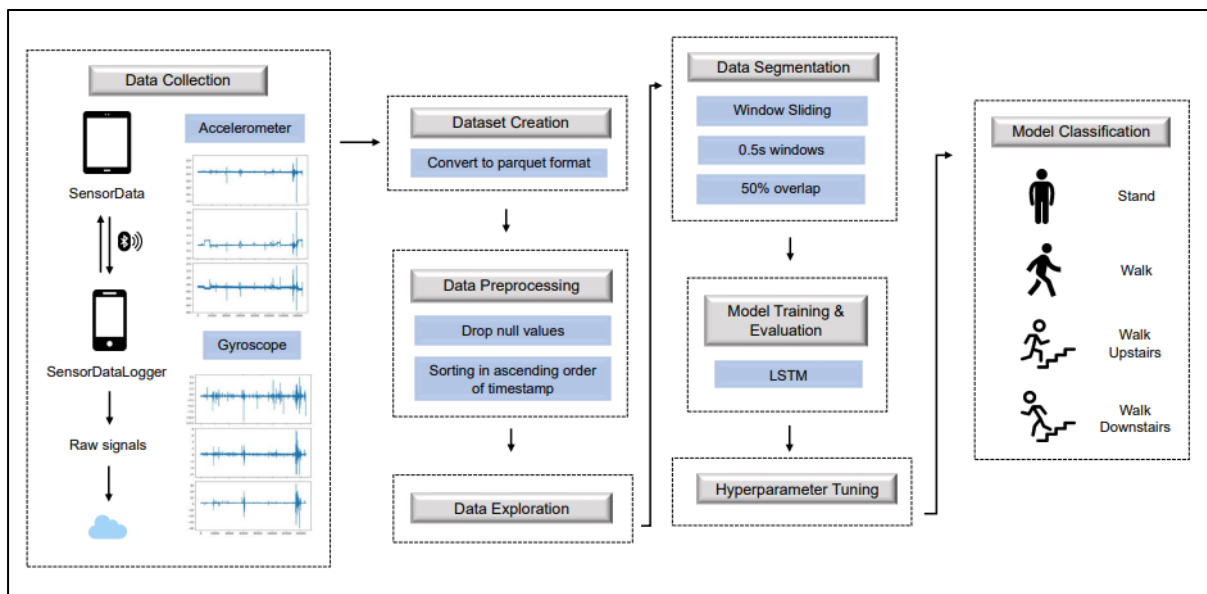


Figure 3.1: System Architecture Diagram

Figure 3.1 shows the system architecture diagram of the proposed HAR system in this research.

Firstly, the data engineering system proposed in this research is used for data collection. The data engineering system proposed comprising hardware, software, and cloud. The hardware used in this system consists of two Android devices installed with the applications, SensorData and SensorDataLogger. The applications are connected via Bluetooth to send and receive instructions. The data subjects are required to carry the smartphones with SensorDataLogger applications installed inside their pockets and perform each activity for one minute. The device installed with SensorData application will be held in hand by data subject or researcher. After collecting sensor signals from accelerometer and gyroscope embedded in smartphone installed with SensorDataLogger, the log files are annotated automatically and saved into local storage with specified filename, which is the activity ID specified by user.

Chapter 3

Then, upon the instructions from SensorData, SensorDataLogger uploads all the log files in the specified path to Google Cloud Storage server for storage and further processing, via Wi-Fi.

After that, the log files stored in the Google Cloud Storage are extracted out for dataset creation purpose by using Google Colab. Since there are many individual log files created during data collection, therefore we need to combine all the log files to become one single dataset. The original CSV log files are first converted to Parquet log files. Parquet files are used instead of CSV files since Parquet is a columnar storage file format designed for efficient compression and storage. It can significantly reduce the amount of space required to store the dataset compared to CSV files. Besides, Parquet files store data in a columnar format, making it easier to skip over irrelevant columns during data retrieval. This can lead to faster query and analysis performance, which is important in HAR research as we need to process and analyze data frequently. After that, the Parquet log files are combined into single dataset.

Next, the dataset goes through the data preprocessing. For example, dropping rows with missing values in certain attributes, dropping rows when timestamp is zero and sorting the dataset in ascending order of timestamp.

After data preprocessing, data exploration is carried out to gain insights into the dataset's characteristics, structure, and quality. This step is performed to identify missing values or outliers in the dataset, obtain summary information regarding the dataset, data visualization, explore relationships between the variables, etc. The findings from data exploration can identify the subsequent data preparation steps that need to be performed.

Subsequently, the dataset will go through data segmentation by using window sliding techniques. This is because the raw time-series data cannot be used directly to train the machine learning or deep learning models. The dataset is divided into smaller segments, called windows. Each window consists of 50 rows or 0.5 seconds of accelerometer and gyroscope signals, with 50% overlap between consecutive windows to minimize information loss at the edge of windows. After data segmentation, we need to split the windows into 80% of training dataset and 20% of testing dataset.

Then, we can proceed with the training of LSTM model by using the windows of data produced in previous step. After training, the LSTM model's performance is evaluated by computing confusion matrix and accuracy.

After model training and evaluation, hyperparameter tuning will be performed to further enhance the performance of the models. During this iterative optimization process,

various hyperparameters, such as learning rates, batch sizes, layer sizes, and dropout rates are fine-tuned to find the combination that produces the best model performance.

Once the optimal hyperparameters are identified, model testing and evaluation are performed using the identified hyperparameters. The fine-tuned models are tested on a dedicated testing dataset to measure the models' real-world performance and generalization capability on new, unseen data. This step helps to validate the practical utility of the proposed smartphone based HAR system and the reliability of the system in real-world applications.

3.1.2 Use Case Diagram and Description

3.1.2.1 Use Case Diagram

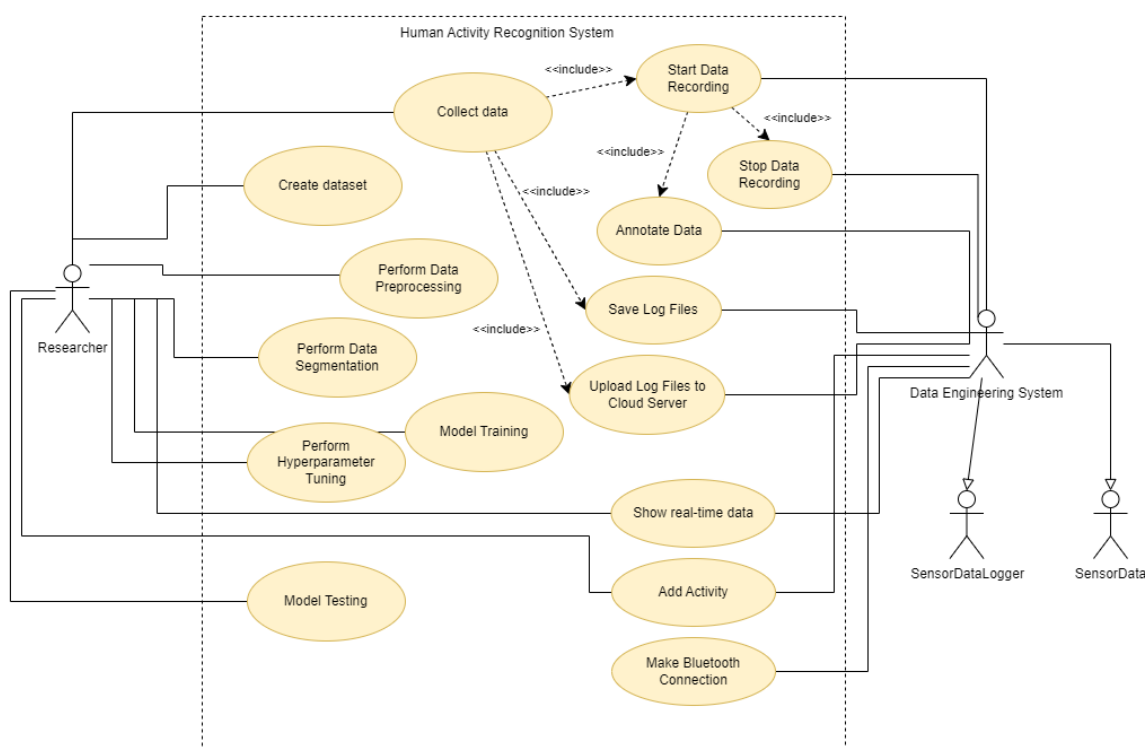



Figure 3.2: Use Case Diagram of HAR system

Figure	Name	Explanation
 Researcher	Researcher	It represents the people who perform HAR research



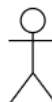
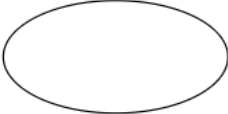

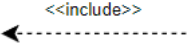

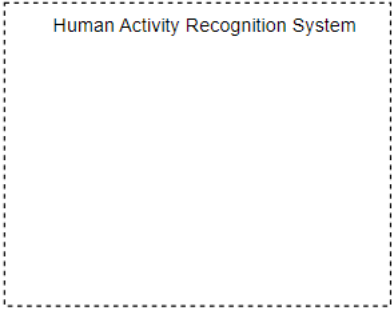
 <p>Data engineering system</p>	<p>Data engineering system</p>	<p>It represents the data engineering system involved in the HAR research.</p>
 <p>SensorData</p>	<p>SensorData application</p>	<p>It represents the SensorData application used in data engineering system.</p>
 <p>SensorDataLogger</p>	<p>SensorDataLogger application</p>	<p>It represents the SensorDataLogger application used in data engineering system.</p>
	<p>Use Case</p>	<p>It presents a system or a process that interacts with the researchers or system.</p>
	<p>Association relationship</p>	<p>It connects use cases and actors if there is interaction between them.</p>
	<p>Include relationship</p>	<p>It is used to indicate that one use case includes the functionality of another use case.</p>
	<p>Generalization relationship</p>	<p>It means one actor can inherit the role of another actor.</p>
 <p>Human Activity Recognition System</p>	<p>System and Boundary</p>	<p>It represents the scope or limits of the HAR system.</p>

Table 3.1: Use Case Legend

3.1.2.2 Use Case Description

Use Case Name	Show Real-time data	
Use Case ID	UC001	
Description	This use case allows data engineering system to show real-time data of accelerometer and gyroscope through SensorDataLogger application.	
Primary Actor	Data Engineering System	
Secondary Actor	Researcher	
Trigger	Researcher opens the SensorDataLogger application.	
Precondition	<ul style="list-style-type: none"> • Researcher has installed the SensorDataLogger application in his or her device. • Researcher has not opened the SensorDataLogger application. 	
Scenario Name	Step	Action
Main Flow	1	Researcher clicks the SensorDataLogger icon through his or her device.
	2	The SensorDataLogger executes.
	3	The user interface of SensorDataLogger shows real-time data of accelerometer and gyroscope.
Alternate Flow – Application Crash	2.1	The application window displays application crash error.
Alternate Flow – Absence / Non-functional Sensors	3.1	The user interface of SensorDataLogger shows only the real-time data of sensor that exists.
Rules	<ul style="list-style-type: none"> • The SensorDataLogger application must be opened through Android device. • The device installed with SensorDataLogger must consists of functional accelerometer and gyroscope. 	
Author	Tee Jia Lin	

Use Case Name	Add Activity
Use Case ID	UC002
Description	This use case allows researchers to add activity button that they want to collect data for through SensorData application.
Primary Actor	Researcher
Secondary Actor	Data Engineering System
Trigger	Researcher clicks the ‘Add Activity’ button through the user interface of SensorData application.

Precondition	Researcher has opened the SensorDataLogger application and can access the user interface of the application.	
Scenario Name	Step	Action
Main Flow	1	Researcher clicks the ‘Add Activity’ button via SensorDataLogger user interface.
	2	A dialog pops out. Researcher fills in the name of activity and class ID of activity.
	3	Researcher clicks the ‘CREATE ACTION’ button.
	4	The application checks the validity of the activity name and class ID.
	5	The activity button created has been shown in the list in the user interface of the application.
Alternate Flow – Invalid activity name or class ID	4.1	The dialog will prompt an error.
Rules	The activity name and Class ID must be unique among all the activities.	
Author	Tee Jia Lin	

Use Case Name	Make Bluetooth Connection	
Use Case ID	UC003	
Description	This use case allows researchers to connect SensorData and SensorDataLogger via Bluetooth.	
Primary Actor	Researcher	
Secondary Actor	Data Engineering System	
Trigger	Researcher clicks the ‘Connect to device’ button, through SensorData application	
Precondition	<ul style="list-style-type: none"> • Researcher has given permissions for both applications to access Bluetooth. • Researcher has enabled the Bluetooth connection of both devices. 	
Scenario Name	Step	Action
Main Flow	1	The Bluetooth connection status bar showing ‘Not connected’. Researcher clicks the kebab menu button on the upper corner of SensorData’s user interface.
	2	Researcher clicks the ‘Connect to device’ option.

	3	A dialog pops out, showing the list of devices connected before.
	4	Researcher clicks the device name where SensorDataLogger is installed on. The Bluetooth connection status bar showing ‘Connecting...’.
	5	SensorData application initiates the Bluetooth connection.
	6	The Bluetooth connection status bar will change to ‘Connected to x’, where x represents the device name of SensorDataLogger.
Alternate Flow – Device name does not exist	3.1	Researcher clicks the ‘SCAN FOR DEVICE’ option to scan for nearby devices.
	3.2	The device name where SensorDataLogger is installed shows in the list.
	3.3	Back to Main Flow step 4 to continue the connection process.
Alternate Flow – Fail to connect to SensorDataLogger	5.1	A toast message pops out, indicating ‘Unable to connect to device’.
	5.2	Back to Main Flow step 1 to reinitiate the Bluetooth connection.
Rules	The devices stay within the acceptable range.	
Author	Tee Jia Lin	

Use Case Name	Start Data Recording
Use Case ID	UC004
Description	This use case allows researchers to start data recording process.
Primary Actor	Researcher
Secondary Actor	Data Engineering System
Trigger	Researchers click the ‘START’ button through user interface of SensorData application.
Precondition	<ul style="list-style-type: none"> • SensorData and SensorDataLogger are connected via Bluetooth. • The device installed with SensorDataLogger is put inside data subject’s pocket. • The activity that researcher wants to collect data for has been created via SensorData user interface.

Include Use Case:	<ul style="list-style-type: none"> • Stop Data Recording • Annotate Data 	
Scenario Name	Step	Action
Main Flow	1	Researcher clicks the activity button that he or she wants to collect data for via SensorData.
	2	Researcher set the duration of how long he or she wants to collect data.
	3	Researcher clicks 'START' button.
	4	System starts a timer and start to collect data.
Alternate Flow – Fail to connect to SensorDataLogger	5.1	A toast message pops out, indicating 'Unable to connect to device'.
	5.2	Back to Main Flow step 1 to reinitiate the Bluetooth connection.
Rules	The Bluetooth connection between two devices stay connected.	
Author	Tee Jia Lin	

Use Case Name	Stop Data Recording	
Use Case ID	UC005	
Description	This use case allows system to stop data recording process.	
Primary Actor	Data Engineering System	
Trigger	The timer for the data recording process has stopped.	
Precondition	<ul style="list-style-type: none"> • Researcher has started the data collection process. • A timer is counting down on both SensorData and SensorDataLogger. 	
Scenario Name	Step	Action
Main Flow	1	Time is up. The timer for the data recording process has stopped on both SensorData and SensorDataLogger.
	2	SensorData sends instructions to SensorDataLogger to stop data recording via Bluetooth.
	3	SensorDataLogger stops recording data.

	4	A toast message pops out, indicating ‘Stop recording’.
Alternate Flow – Fail to stop data recording	2.1	SensorDataLogger fails to stop data recording.
	2.2	Use case terminates.
Rules	The Bluetooth connection between two devices stay connected.	
Author	Tee Jia Lin	

Use Case Name	Save Log Files	
Use Case ID	UC006	
Description	This use case allows researchers to save collected data locally inside smartphone storage.	
Primary Actor	Researcher	
Secondary Actor	Data Engineering System	
Trigger	The researcher clicks the ‘Save Log File’ button through user interface of SensorData application.	
Precondition	<ul style="list-style-type: none"> • The data recording process has stopped. • A dataset is available to be saved. 	
Scenario Name	Step	Action
Main Flow	1	Researcher clicks the ‘Save Log File’ button through the user interface of SensorData.
	2	SensorDataLogger saves the log file in the specified location.
	3	A toast message pops out, indicating ‘Log file saved’.
Alternate Flow – The dataset is unavailable	2.1	SensorDataLogger fails to save log file
	2.2	Use case terminates.
Rules	The Bluetooth connection between two devices stay connected.	
Author	Tee Jia Lin	

Use Case Name	Upload Log Files to Cloud Server
Use Case ID	UC007

Description	This use case allows researchers to upload log files stored inside smartphone local storage to Google Cloud Storage for further processing through SensorDataLogger.	
Primary Actor	Researcher	
Secondary Actor	Data Engineering System	
Trigger	The researcher clicks the 'Upload Log File' button through user interface of SensorData application.	
Precondition	The datasets to be uploaded are stored inside local storage of the device.	
Scenario Name	Step	Action
Main Flow	1	Researcher clicks the 'Upload Log File' button through user interface of SensorData.
	2	SensorDataLogger upload all the log files in the specified path to Google Cloud Storage.
	3	A toast message pops out, indicating 'Uploaded x files successfully', where x represents the number of files uploaded.
	4	SensorDataLogger deletes the log files uploaded in the local storage.
Alternate Flow – The dataset is unavailable	2.1	System fails to upload log file
	2.2	Use case terminates.
Rules	The Bluetooth connection between two devices stay connected.	
Author	Tee Jia Lin	

Use Case Name	Annotate Data	
Use Case ID	UC008	
Description	This use case allows system to annotate data.	
Primary Actor	Data Engineering System	
Trigger	SensorData sends the instruction to start the data collection process.	
Precondition	<ul style="list-style-type: none"> • Researcher has started the data collection process. • Instruction that contains the class ID has been sent from SensorData to SensorDataLogger. 	
Scenario Name	Step	Action

Main Flow	1	The SensorDataLogger has started the data collection process.
	2	SensorDataLogger identifies the class ID sent from SensorData.
	3	SensorDataLogger appends the class ID at the end of each row of data.
Alternate Flow – Class ID is unavailable	2.1	SensorDataLogger fails to recognize the class ID from the instructions sent.
	2.2	Use case terminates.
Rules	The Bluetooth connection between two devices stay connected.	
Author	Tee Jia Lin	

Use Case Name	Collect data	
Use Case ID	UC009	
Description	This use case allows researchers to perform data collection	
Primary Actor	Researcher	
Trigger	Researcher wants to implement a Human Activity Recognition research and does not want to use public datasets.	
Precondition	<ul style="list-style-type: none"> • Data sources, sensors, or mechanisms for data collection are available and properly configured. • The purpose and requirements of the dataset are defined. 	
Include Use Case:	<ul style="list-style-type: none"> • Start Data Recording • Save Log Files • Upload Log Files to Cloud Server 	
Scenario Name	Step	Action
Main Flow	1	Researcher determines the activities he or she wants to collect data for.
	2	Researcher determines the duration for each activity.
Rules	-	
Author	Tee Jia Lin	

Use Case Name	Collect data	
Use Case ID	UC009	
Description	This use case allows researchers to perform data collection	
Primary Actor	Researcher	
Trigger	Researcher wants to implement a Human Activity Recognition research and does not want to use public datasets.	
Precondition	<ul style="list-style-type: none"> • Data sources, sensors, or mechanisms for data collection are available and properly configured. • The purpose and requirements of the dataset are defined. 	
Include Use Case:	<ul style="list-style-type: none"> • Start Data Recording • Save Log Files • Upload Log Files to Cloud Server 	
Scenario Name	Step	Action
Main Flow	1	Researcher determines the activities he or she wants to collect data for.
	2	Researcher determines the duration for each activity.
Rules	-	
Author	Tee Jia Lin	

Use Case Name	Perform Data Preprocessing	
Use Case ID	UC011	
Description	This use case allows researchers to preprocess dataset to make it suitable for subsequent classification.	
Primary Actor	Researcher	
Trigger	Researchers extract the datasets out from Google Cloud Storage to Google Colab.	
Precondition	<ul style="list-style-type: none"> • The dataset created is available in Google Cloud Storage. • The dataset is in .parquet format. • The researcher has opened the Google Colab. 	
Scenario Name	Step	Action
Main Flow	1	Researcher opens the Google Colab, imports necessary libraries, and makes configuration.

	2	Researcher extracts the .parquet dataset out from GCS in Google Colab.
	3	Researcher performs data preprocessing steps such as removing null values, sorting in ascending order of timestamp, etc.
	4	Researcher stores the pre-processed dataset back to GCS.
Alternate Flow – Failed to extract the datasets	1.1	Researcher failed to read the .parquet dataset in Google Colab.
	1.2	Back to Main Flow step 1 to re-extract the datasets.
Rules	The kernel of the Google Colab stays opened.	
Author	Tee Jia Lin	

Use Case Name	Perform Data Segmentation	
Use Case ID	UC012	
Description	This use case allows researchers to break dataset into smaller segments to make it suitable for subsequent classification.	
Primary Actor	Researcher	
Trigger	Researchers extract the pre-processed datasets out from Google Cloud Storage to Google Colab.	
Precondition	<ul style="list-style-type: none"> • Researcher has pre-processed the dataset. • The dataset is in .parquet format. 	
Scenario Name	Step	Action
Main Flow	1	Researcher extracts the pre-processed dataset in .parquet format from GCS to Google Colab.
	2	Researcher perform window sliding, with window size of 50 rows of data and 50% overlap between consecutive windows.
	3	Researcher splits the windows of data into 80% of training dataset and 20% of testing dataset.
	4	Researcher stores the segmented dataset back to GCS.
Alternate Flow – Failed to extract the datasets	1.1	Researcher failed to read the .parquet datasets in Google Colab.
	1.2	Back to Main Flow step 1 to re-extract the datasets.
Rules	The kernel of the Google Colab stays opened.	

Author	Tee Jia Lin	
Use Case Name	Model Training	
Use Case ID	UC013	
Description	This use case allows researchers to perform model training using labelled dataset.	
Primary Actor	Researcher	
Trigger	Researchers extract dataset out from Google Cloud Storage to Google Colab.	
Precondition	<ul style="list-style-type: none"> • The dataset is in .parquet format and ready to be used to train models. • The dataset is labelled and pre-processed. 	
Scenario Name	Step	Action
Main Flow	1	Researcher extracts the labelled dataset in .parquet format from GCS to Google Colab.
	2	Researcher builds the model architecture.
	2	Researcher trains the models with labelled training dataset.
	3	Researcher evaluates the model performance.
Alternate Flow – Failed to extract the datasets	1.1	Researcher failed to read the .parquet datasets in Google Colab.
	1.2	Back to Main Flow step 1 to re-extract the datasets.
Rules	The kernel of the Google Colab stays opened.	
Author	Tee Jia Lin	

Use Case Name	Perform Hyperparameter Tuning
Use Case ID	UC014
Description	This use case allows researchers to perform hyperparameter tuning on selected models to optimize model performance.
Primary Actor	Researcher
Trigger	The selected models achieve low performance during training or the researcher wants to enhance the performance of the model.
Precondition	Models have been selected for tuning.

Scenario Name	Step	Action
Main Flow	1	Researcher selects the model that requires hyperparameter tuning.
	2	Researcher identifies the hyperparameters that need to be tuned for the selected model.
	3	For each hyperparameter, researcher defines the search space.
	4	Researcher identifies the hyperparameter tuning strategy that he or she is going to use.
	5	Researcher implements the hyperparameter tuning.
	6	System evaluates the model performance after hyperparameter tuning.
	7	System identifies the combination of hyperparameters that resulted in the best performance of model.
	8	Researcher selects and identifies the hyperparameters for the final model.
Alternate Flow – Tuning failure	5.1	Hyperparameter tuning process encounters error.
	5.2	Researcher needs to debug the errors.
Alternate Flow – Hyperparameter tuning is not effective	7.1	The hyperparameter tuning process does not improve the model performance.
	7.2	Back to Main Flow step 2.
Rules	The kernel of the Google Colab stays opened.	
Author	Tee Jia Lin	

Use Case Name	Model Testing
Use Case ID	UC015
Description	This use case allows researchers to perform model testing to evaluate a model's performance on new data.
Primary Actor	Researcher
Trigger	Combinations of hyperparameter are identified.

Precondition	<ul style="list-style-type: none"> • A trained model is available for testing. • A labelled testing dataset which contains new data and pre-processed is available. • The labelled testing dataset has been pre-processed. 	
Scenario Name	Step	Action
Main Flow	1	Researcher gets the testing dataset.
	2	Researcher gets the trained model to be tested.
	3	Researcher implements the model testing.
	4	Researcher evaluates the model performance on testing dataset.
Alternate Flow – Test dataset unavailable	1.1	Researcher fails to read the testing dataset.
	1.2	Use case terminates.
Rules	The kernel of the Google Colab stays opened.	
Author	Tee Jia Lin	

3.1.3 Activity Diagram

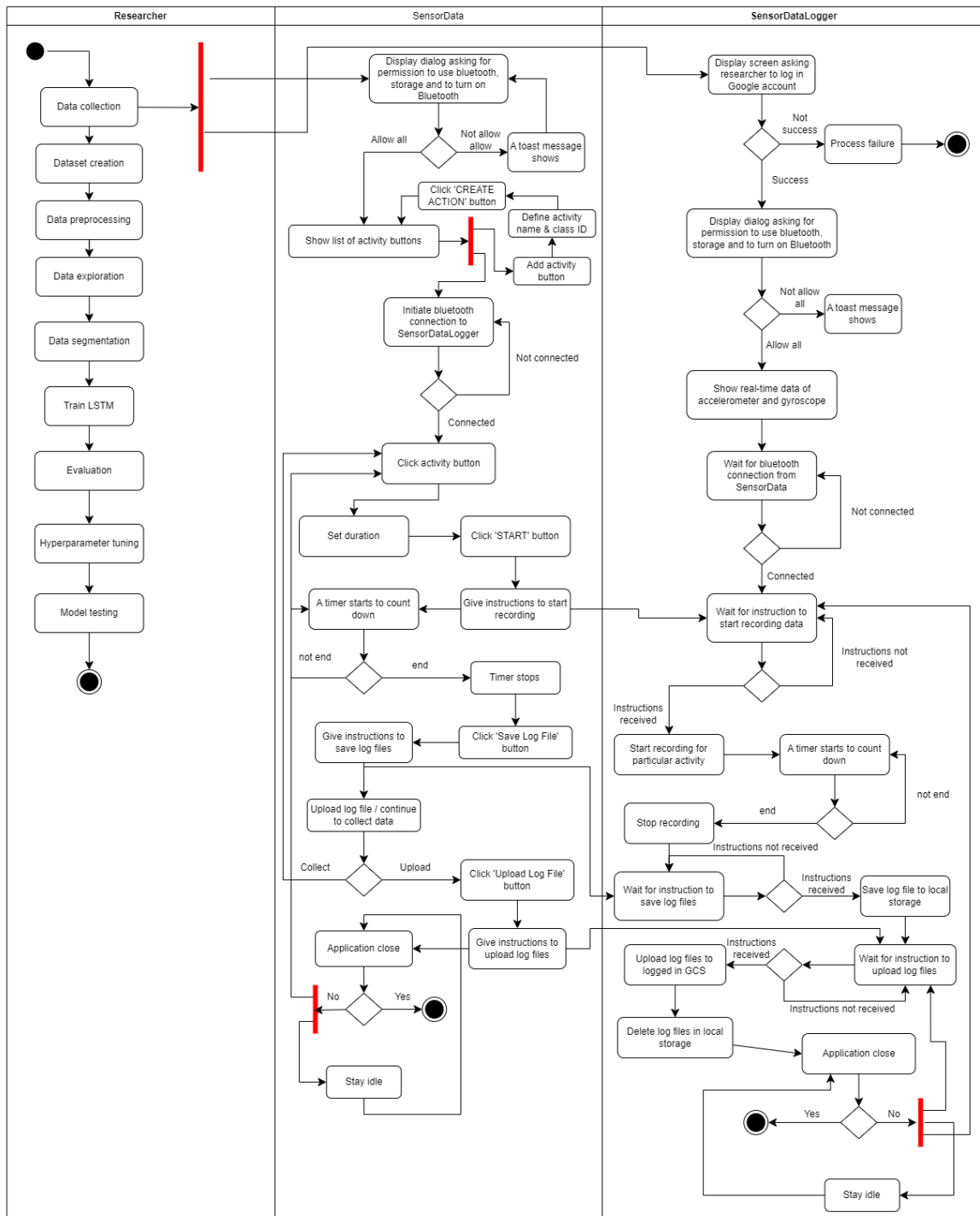


Figure 3.3: Activity Diagram

Each step in the proposed HAR system is depicted in Figure 3.3. Firstly, the researcher needs to collect sensor signals. The data collection involves the usage of SensorDataLogger and SensorData applications. The researcher has to open the applications. The applications will

Chapter 3

ask for permissions to use Bluetooth, storage and ask to turn on Bluetooth if it hasn't been turned on. After that, SensorDataLogger application will ask the researcher to log in the Google account. SensorData will then show the user interface consisting of list of activity buttons while SensorDataLogger will show real-time accelerometer and gyroscope data. Next, the researcher can either choose to add activity button that he or she wants to collect data for via SensorData, or the researcher can connect both applications via Bluetooth. If the researcher wants to add activity button, he or she needs to define activity name, class ID, and click 'CREATE ACTION' button. After both applications are connected via Bluetooth, the researcher can start the data collection process. The device installed with SensorDataLogger can be put inside pocket while the device installed with SensorData will be held on hand. To collect data, the researcher needs to click activity button that he or she wants to collect data for. A dialog will pop out, asking to set duration. After setting duration, click the 'START' button. The SensorData application will send instructions to start recording to SensorData application and start a timer. Similarly, when SensorData application receives the instructions, a timer will start. After timer stops, the SensorDataLogger and SensorData applications will stop data recording process. The log files created will be automatically annotated via SensorDataLogger application. The researcher now needs to click 'Save Log File' button and SensorData application will send instructions to SensorDataLogger to save the log file. Next, the researcher can choose to either continue collect sensor data for other activities or upload the existing log files to cloud server. If the researcher chooses to continue collecting data, then he or she just needs to click another activity button and repeats the same process. If the researcher chooses to upload the existing log files, then he or she needs to click 'Upload Log File' button. SensorData application will then send instructions to SensorDataLogger application to upload all the log files inside local storage to Google Cloud Storage. After uploading, the SensorDataLogger application will delete all the uploaded log files. This process continues until the researcher has collected data for all the activities.

After data collection, the researcher needs to create a dataset by combining all the log files into one single dataset. The researcher first convert all the CSV log files into Parquet log files. Then, the parquet log files are combined into single dataset.

Subsequently, data preprocessing is performed on the dataset created to make them more suitable for subsequent classification. Preprocessing steps such as dropping rows with null values, dropping rows when timestamp is zero and sorting the dataset in ascending order of timestamp are implemented.

Chapter 3

Then, data exploration is performed on the preprocessed dataset created in previous step. The steps such as identifying missing values or outliers, visualizing data, obtaining summary information regarding dataset, etc. are implemented to understand better the characteristics and quality of dataset. This step is crucial to identify subsequent data preparation steps.

Then, the researcher will perform data segmentation by breaking large datasets into fixed size of windows. After that, the windows produced are split into 80% of training dataset and 20% of testing data. The training dataset can be used to train LSTM model. After model training, the researcher will perform evaluation to assess the performance of the models by computing confusion matrices and accuracies.

Next, hyperparameter tuning is performed to identify the combination of hyperparameters that can produce the best model performance. Finally, the model will go through testing phase using the testing dataset.

Chapter 4

System Design

4.1 System Block Diagram

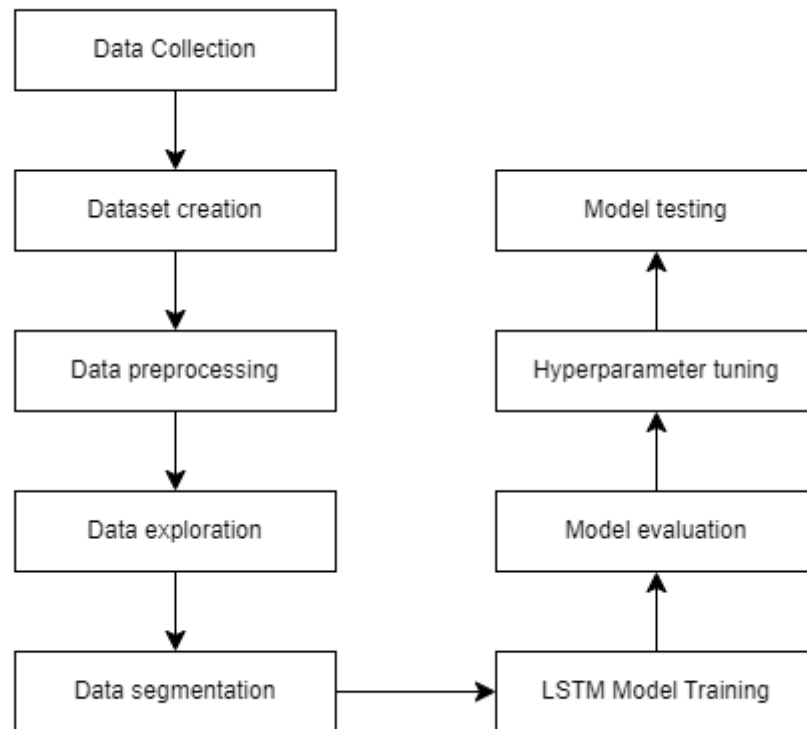


Figure 4.1: Block diagram for smartphone based HAR system.

The first step in the proposed smartphone based HAR system is data collection. The data collection involves sensor signals collection from accelerometer and gyroscope embedded in smartphone by using SensorData and SensorDataLogger applications being developed in this research. Two Android devices are used. After data collection, the researcher proceeds to dataset creation. Google Colab software is used starting from this step. This is because many individuals CSV log files are created during data collection. The researcher needs to convert the CSV log files into Parquet log files and combine all the log files to become single dataset. After that, the dataset produced is preprocessed. The preprocessing steps include dropping rows with null values, dropping rows with timestamp equals to zero and sorting the dataset in

ascending order of timestamp. Then, data exploration is performed on the preprocessed dataset. Through data exploration, the researcher can understand the data type of the features in the dataset, visualize the class label distribution, identify the missing values or outliers, and visualize the relationship between the variables in the dataset. This helps to decide the subsequent data preparation steps that need to be implemented. Next, the researcher will implement data segmentation. This is because raw time-series data cannot be fed directly to LSTM model. The researcher will break the large datasets into fixed size of windows, whereby each window consists of 50 rows of data. After that, the windows produced are split into 80% of training dataset and 20% of testing data. The training dataset can be used to train LSTM model. After model training, the researcher will perform evaluation to assess the performance of the models by computing confusion matrix and accuracy. Subsequently, hyperparameter tuning is performed to identify the combination of hyperparameters to further enhance the performance of LSTM model. Finally, the model will go through testing phase using the testing dataset.

4.2 System Components Specifications

4.2.1 Data Collection

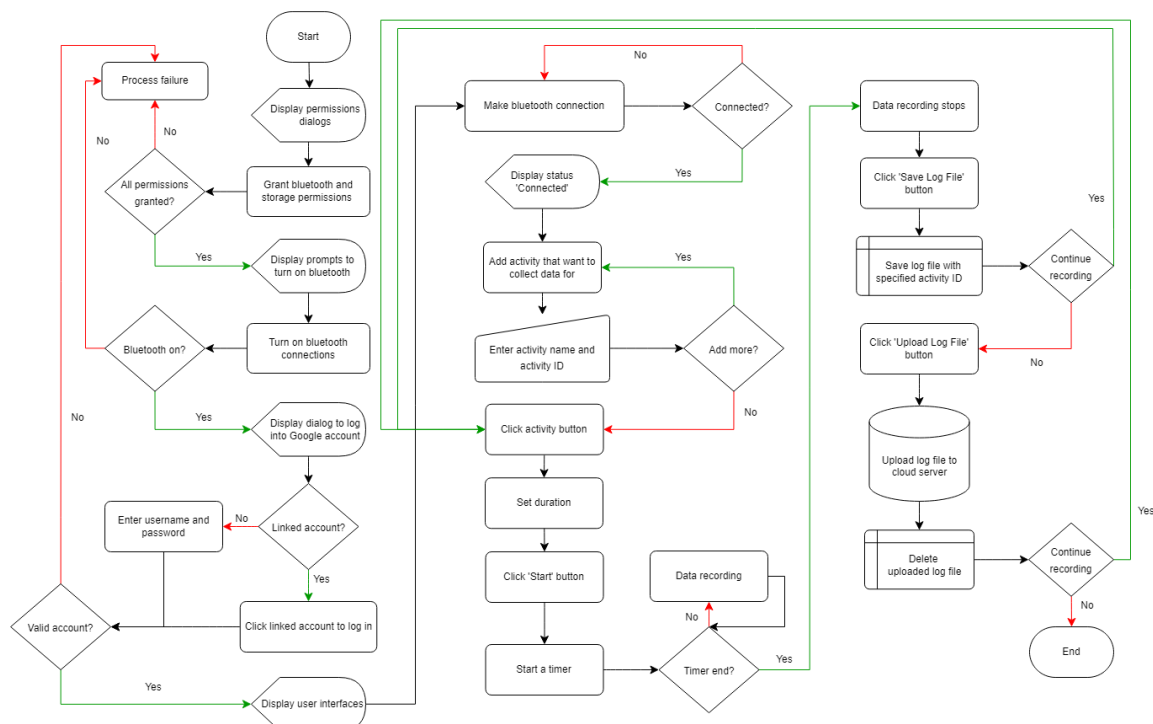


Figure 4.2: Flowchart for data collection in smartphone based HAR system.

Chapter 4

The first step in the smartphone based HAR system is data collection. The data collection involves the usage of SensorDataLogger and SensorData applications. The applications are connected by Bluetooth to send and receive instructions. When researcher opens the applications, the applications will ask for permissions to use Bluetooth, storage and ask to turn on Bluetooth if it hasn't been turned on. After that, SensorDataLogger application will ask the researcher to log in the Google account. SensorData will then show the user interface consisting of list of activity buttons while SensorDataLogger will show real-time accelerometer and gyroscope data as well as the sampling frequency. If the researcher uses these applications for the first time, then he or she needs to add activity button that they want to collect data for via SensorData. The researcher simply needs to click the 'Add Activity' button, define the activity name, activity ID, and click 'CREATE ACTION' button. The activity ID and activity name must be unique among the activity buttons created. The researcher will not be able to click the 'CREATE ACTION' button if there is duplication of names and IDs. Then, the buttons created will be shown in the user interface of SensorData application. To delete the button created, the research simply needs to swipe the particular button to the left.

After that, the researcher needs to connect both applications via Bluetooth. The status bar will change to 'Connected' when the Bluetooth connections are successfully established. After both applications are connected via Bluetooth, the researcher can start the data collection process. The device installed with SensorDataLogger can be put inside pocket while the device installed with SensorData will be held in hand. To collect data, the researcher needs to click activity button that he or she wants to collect data for. A dialog will pop out, asking to set duration. After setting duration, click the 'START' button. The SensorData application will send instructions to start recording to SensorDataLogger application and start a timer. Similarly, when SensorDataLogger application receives the instructions, a timer will start.

After timer stops, the SensorDataLogger and SensorData applications will stop data recording process. The researcher now needs to click 'Save Log File' button and SensorData application will send instructions to SensorDataLogger to save the log file. The log file name will be the activity ID and the datetime when the log files are created. The log files created are annotated with the activity ID automatically. Next, the researcher can choose to either continue collect sensor data for other activities or upload the existing log files to cloud server. If the researcher chooses to continue collecting data, then he or she just needs to click another activity button and repeats the same process. If the researcher chooses to upload the existing log files,

then he or she needs to click ‘Upload Log File’ button. SensorData application will then send instructions to SensorDataLogger application to upload all the log files inside local storage to Google Cloud Storage. After uploading, the SensorDataLogger application will delete all the uploaded log files. This process continues until the researcher has collected data for all the activities.

4.2.2 Dataset Creation

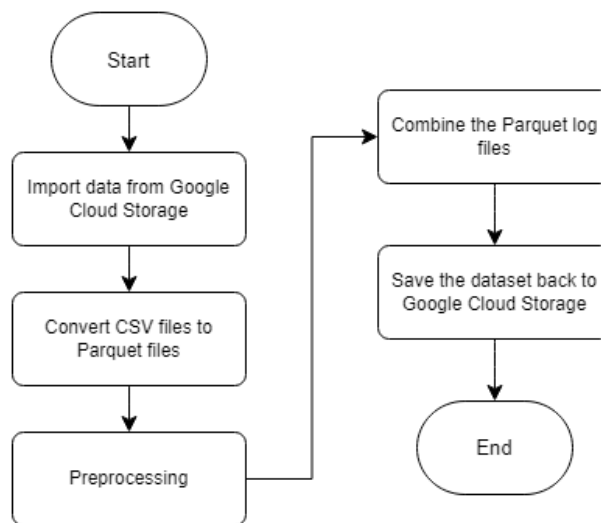


Figure 4.3: Flowchart for dataset creation in smartphone based HAR system.

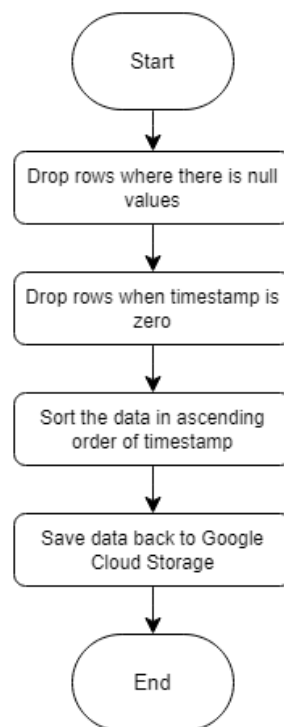


Figure 4.4: Flowchart for data preprocessing in smartphone based HAR system.

Starting from second step, Google Colab software is used. The second step in the smartphone based HAR system is dataset creation. When researchers are collecting sensor data for each activity, individual log files will be created. Therefore, these log files need to be combined to become single dataset for the purpose of subsequent data exploration, preprocessing, model training, etc.

The researcher first extracts the individual log files from Google Cloud Storage. This is performed by first performing authentication with Google Cloud Storage by uploading the .json file of private key and get authentication via `from google.colab import auth` and `auth.authenticate_user()`. Then, the researcher imports the necessary library, `from google.cloud import storage`, and retrieve the list of CSV files from the relevant Google Cloud Storage bucket by specifying the project id, bucket name and folder name.

Then, the researcher converts the CSV files into Parquet files. Parquet files are used instead of CSV files since Parquet is a columnar storage file format designed for efficient compression and storage. It can significantly reduce the amount of space required to store the dataset compared to CSV files. Besides, Parquet files store data in a columnar format, making it easier to skip over irrelevant columns during data retrieval. This can lead to faster query and analysis

performance, which is important in HAR research as we need to process and analyze data frequently. The researcher implements the conversion by first read the CSV files using *pandas.read_csv()* and convert to Parquet files using *pandas.to_parquet()* function provided by pandas library.

After that, the Parquet log files are preprocessed. The researcher will remove the rows of data with null values by using *pandas.dropna()* and the rows of data with timestamp equals to zero. Besides, the researcher will sort the dataset in ascending order of timestamp by using *pandas.sort_values()*.

Last but not least, all the log files are combined into single dataset. The dataset is then saved back to Google Cloud Storage by specifying the file path.

4.2.3 Data Exploration

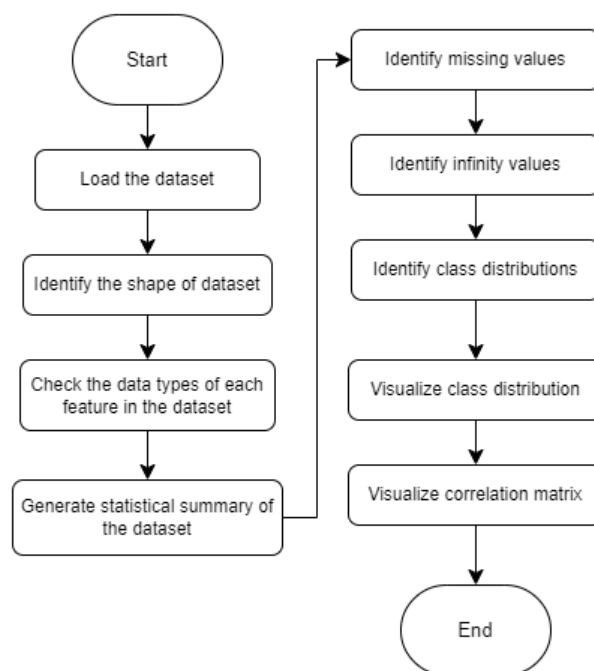


Figure 4.5: Flowchart for data exploration in smartphone based HAR system.

The next step in the smartphone based HAR system proposed is data exploration. This step is performed to allow researchers to better understand the structure, quality, and characteristics of the created dataset. This step is crucial to identify subsequent data preparation steps.

Chapter 4

The researcher first loads the dataset from Google Cloud Storage using `pandas.load_parquet()`. Then, the researcher identifies the shape of the dataset via `pandas.shape` attribute. The researcher also recognizes the data type of each attributes in the dataset, including the number of non-null values and memory usage by using `pandas.info()`. The researcher also generates statistical summary of the dataset by using `pandas.describe()`. This helps to gain insights into the central tendency and dispersion of the dataset, identifying potential outliers, and understanding the distribution of numerical variables in the dataset. Next, the researcher identifies missing values and infinity values in the dataset by using `pandas.isnull()`, `numpy.isinf()` and `numpy.isneginf()`. Subsequently, the researcher computes and visualize the class distributions in the dataset by using `pandas.crosstab()` and `seaborn.countplot()`. Lastly, the researcher visualizes the relationship between the numerical variables in the dataset by using correlation matrix via `pandas.corr()` and `seaborn.heatmap`. The squares with darker color indicate the stronger relationship between the variables, while squares with lighter color suggest weak relationships or no relationship.

4.2.4 Data Segmentation

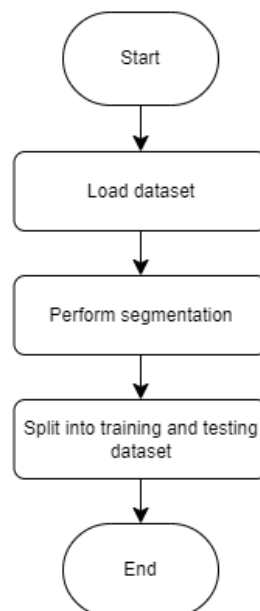


Figure 4.6: Flowchart for data segmentation in smartphone based HAR system.

Since raw time-series sensor data cannot be fed directly to LSTM model, therefore the subsequent step would be data segmentation by using window sliding techniques. The researcher first loads the preprocessed dataset. Then, the dataset is divided into smaller segments, called windows. Each window consists of 0.5 seconds of accelerometer and gyroscope signals, with 50% overlap between consecutive windows to minimize information loss at the edge of windows. After segmentation, the windows of data are divided into 80% of training dataset and 20% of testing dataset.

4.2.5 Model Training & Evaluation

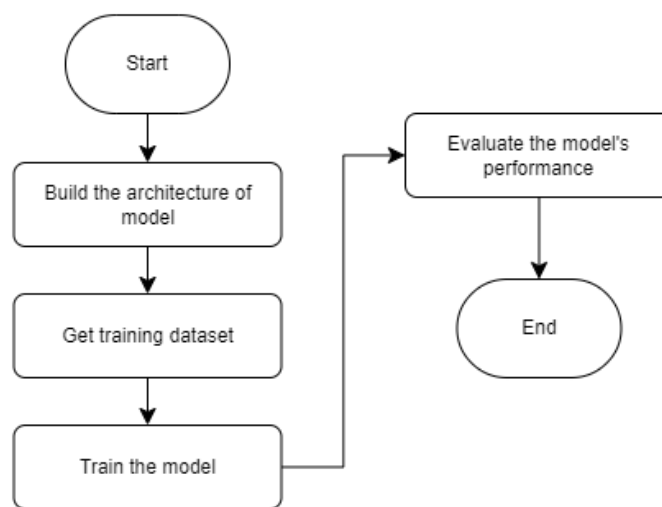


Figure 4.7: Flowchart for model training and evaluation in smartphone based HAR system.

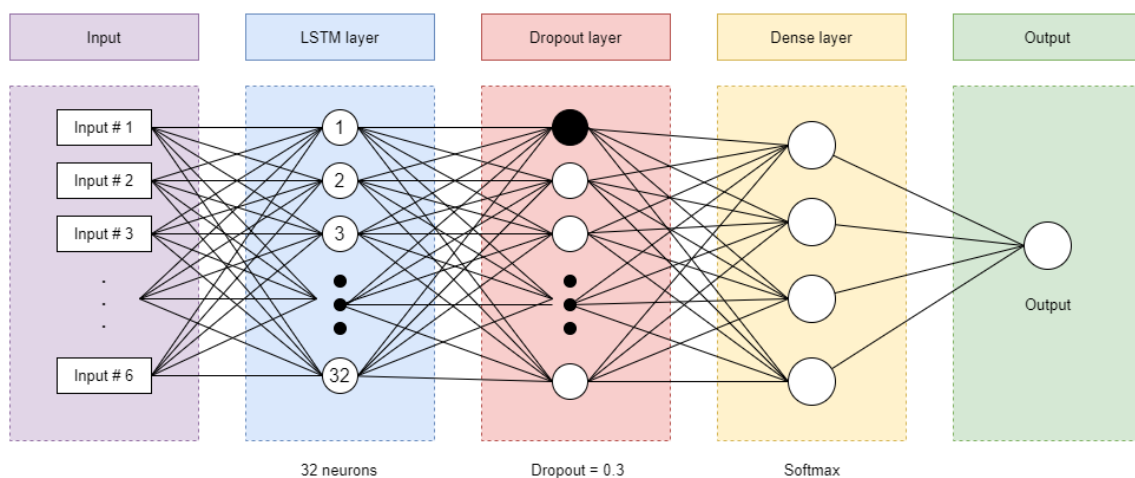


Figure 4.8: Architecture of LSTM model proposed.

For LSTM model training, the researcher first builds the architecture of the model. A unidirectional and simple LSTM model is proposed in this project by using TensorFlow and Keras. The model consists of one LSTM layer, one dropout layer and one dense layer. The first layer is a LSTM layer with 32 nodes to capture sequential patterns in the data. The number of node is chosen since our training dataset only consists of 3846 data points after data segmentation. LSTM model which is too complex will introduce overfitting problem during training when our dataset size is small. Next, the hidden layer incorporates a dropout layer with a 30% dropout rate to prevent overfitting problem. Overfitting happens when a model becomes too specialized in fitting certain data patterns. The dropout technique is a regularization techniques in neural network to reduce the interdependencies among neurons by randomly deactivating a subset of neurons in the layer during training. This introduces noise and variability during training, ensuring that each batch of data is trained using different subsets of neuron and thereby regularize the model. The dense layer is a fully connected dense layer implemented using Softmax activation function for multiclass classification. The Softmax activation function takes an input vector and computes the probability of the input belongs to each class. The probability scores computed are normalized to ensure that they sum up to 1. Then, the final output is the class with the highest probability score.

Additionally, the LSTM model is compiled with categorical cross-entropy loss. Categorical cross-entropy is a loss function which measures the difference between the predicted distribution of classes (output of neural network) and the true distribution of classes (ground truth labels). Besides, Adam optimizer is used, which is a stochastic gradient descent optimizer used to dynamically adjust the learning rates of the model to minimize loss function. The evaluation metric used is accuracy.

During training, a constant batch size of 4 is used. Batch size refers to the number of data samples that are processed together in a single forward and backward pass through a neural network during training. In this context, a batch size of 4 means that, during each iteration of model training, the neural network processes four data samples simultaneously. A batch size of 4 is chosen as smaller batch size are more suitable to work with small datasets and when we have limited computational resources.

The LSTM model training is conducted for 50 epochs with early stopping based on validation loss with a patience of 5 epochs. The code used to implement the early stopping is *EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)*. This means that the model training will run for a maximum of 50 iterations during training, but it will stop early

if the validation loss does not improve for 5 consecutive epochs. In other words, if the validation loss remains stagnant or increases for five consecutive epochs, the training process will terminate before reaching the maximum of 50 epochs to prevent overfitting and save time. Before stopping, the code will automatically restore the model's weight to the best observed point that achieved the best validation loss during training.

After building the architecture of LSTM model, we can proceed with the training of LSTM model using all the parameters mentioned. After training, the LSTM model's performance is evaluated by computing confusion matrix, accuracy, loss and RMSE. A graph on the change of train loss, train accuracy, validation loss and validation accuracy are also plotted using matplotlib library.

4.2.6 Hyperparameter Tuning

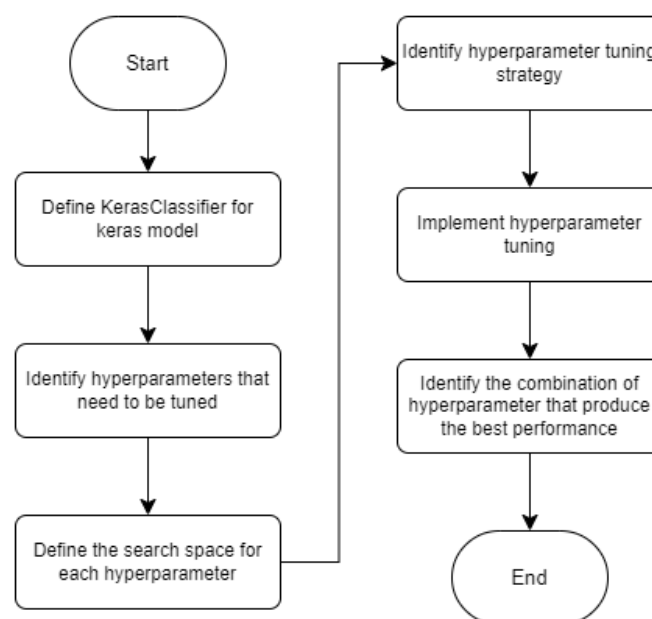


Figure 4.9: Flowchart for hyperparameter tuning in smartphone based HAR system.

After model training and evaluation, hyperparameter tuning is performed. Firstly, we define a KerasClassifier as a wrapper for the keras model. This is because we are going to use GridSearchCV by scikit-learn library. A KerasClassifier wrapper is necessary so that the keras model can be integrated with the hyperparameter tuning routines by scikit-learn library. Subsequently, we identify the hyperparameters that need to be tuned. In this project, we

perform tuning for batch size, epochs, learning rate and dropout rate. Then, we define the search space for each hyperparameter, which is the range of values the model is going to explore. For example, the search space defined for the batch size is 4, 16 and 32. Next, we need to identify the hyperparameter tuning strategy. In this context, we use GridSearchCV provided by scikit-learn library to perform the hyperparameter tuning. GridSearchCV trains and evaluates the model over all possible combinations of hyperparameter values within the defined search space to find the combinations that achieve the best accuracy. We also implement the early stopping that will monitor the validation loss during training. It will stop the training when the validation loss does not improve for 5 consecutive epochs to prevent overfitting problem. Finally, we implement the hyperparameter tuning and identify the combination of hyperparameter that achieve the best accuracy.

4.2.7 Model Testing

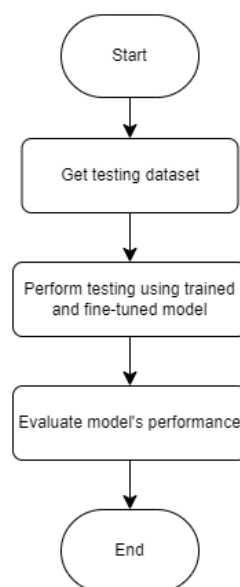


Figure 4.10: Flowchart for model testing in smartphone based HAR system.

The last step in the proposed smartphone based HAR system is model testing by using the dedicated testing dataset. By using the combination of hyperparameters identified in previous step, the trained and fine-tuned model is tested to measure the models' real-world performance by allowing the model to be presented with new, previously unseen data that simulates real-world scenarios.

Chapter 4

The researcher first gets the testing dataset. Then, he or she performs testing on the trained and fine-tuned model. After that, the researcher evaluates the performance of the model by computing accuracy, loss, RMSE and confusion matrix. The evaluation results obtained from testing process provide valuable insights into the effectiveness, practical utility, and reliability of the proposed HAR system.

Chapter 5

System Implementation

5.1 Hardware Setup

The hardware involved in this project is a laptop and two Android mobile devices. The two Android devices used involve a smartphone and a tablet. The smartphone will be installed with SensorDataLogger to show real-time sensor data from accelerometer and gyroscope, to collect sensor data, to annotate the log file automatically, to save the log file with specified filename, and to upload the log file to Google Cloud Storage server for further processing. The tablet will be installed with SensorData to give instructions to the smartphone.

The laptop is used to develop the mobile applications, SensorData and SensorDataLogger. The laptop is also used to perform HAR relevant operations such as dataset creation, data preprocessing, data exploration, data segmentation, model training and evaluation, hyperparameter tuning, and model testing.

Description	Specifications
Model	Lenovo IdeaPad S540
Processor	2nd Gen AMD Ryzen™7
Operating System	Windows 11
Graphic	AMD Integrated Graphics
Memory	12 GB DDR4
Storage	512 GB PCIe SSD

Table 5.1: Specifications of laptop

Description	Specifications
Model	Redmi Note 11 Pro 5G
Processor	Snapdragon 695, Octa-core Max 2.2 GHz
Operating System	Android 12, MIUI 13
Graphic	Mali-G57 MC2
Memory	8GB RAM

Storage	128GB
---------	-------

Table 5.2: Specifications of smartphone

Description	Specifications
Model	Samsung Galaxy Tab A8 10.5
Processor	Octa-core
Operating System	Android 13
Graphic	Mali-G52 MP2
Memory	4GB RAM
Storage	64GB

Table 5.3: Specifications of tablet

5.2 Software Setup

This project focuses on both the data engineering system and human activity recognition system. Several software is used in Android devices and laptop.

Firstly, software used in laptop include Android Studio and Google Colab. Android Studio is used for the purpose of developing mobile applications to implement the data engineering system. The Android emulators in Android Studio were also used to simulate the mobile applications being developed, i.e., the user interface and the functionalities, before being installed to the smartphones. Besides, Google Colab is used for the purpose of dataset creation, data preprocessing, data exploration, data segmentation, model training and evaluation, hyperparameter tuning, and model testing.

Next, the software that needs to be installed in Android devices include SensorData and SensorDataLogger. These applications are used to implement the data engineering system to collect sensor data from accelerometer and gyroscope embedded in smartphone. Besides, the applications can be used to annotate the log files automatically, to save the log files inside local storage with specified filename and upload all the log files to Google Cloud Storage server for further processing.

Chapter 5

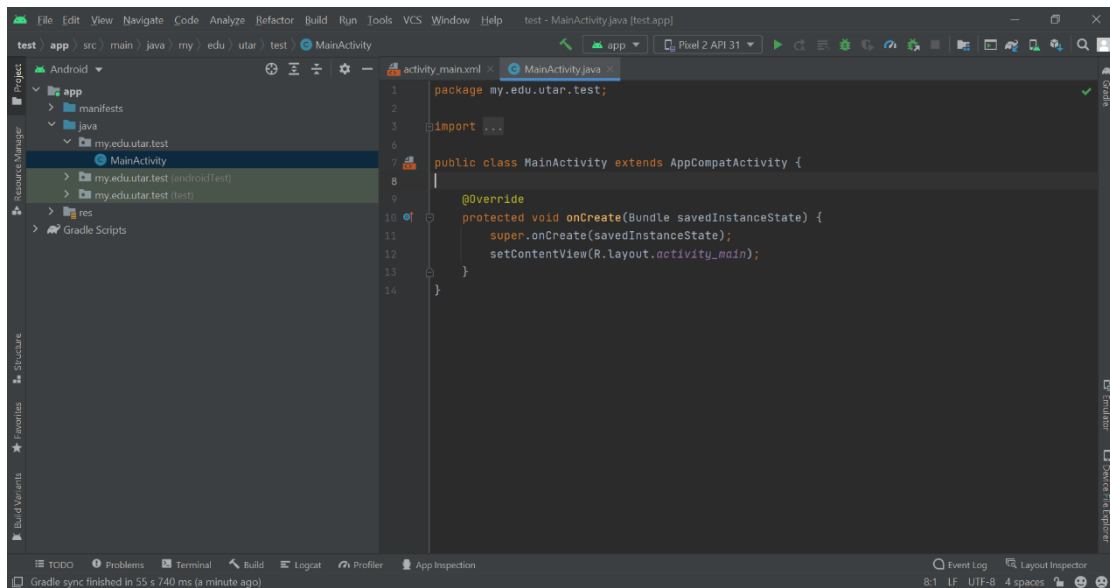


Figure 5.1: Android Studio Arctic Fox, 2020.3.1, Patch 4

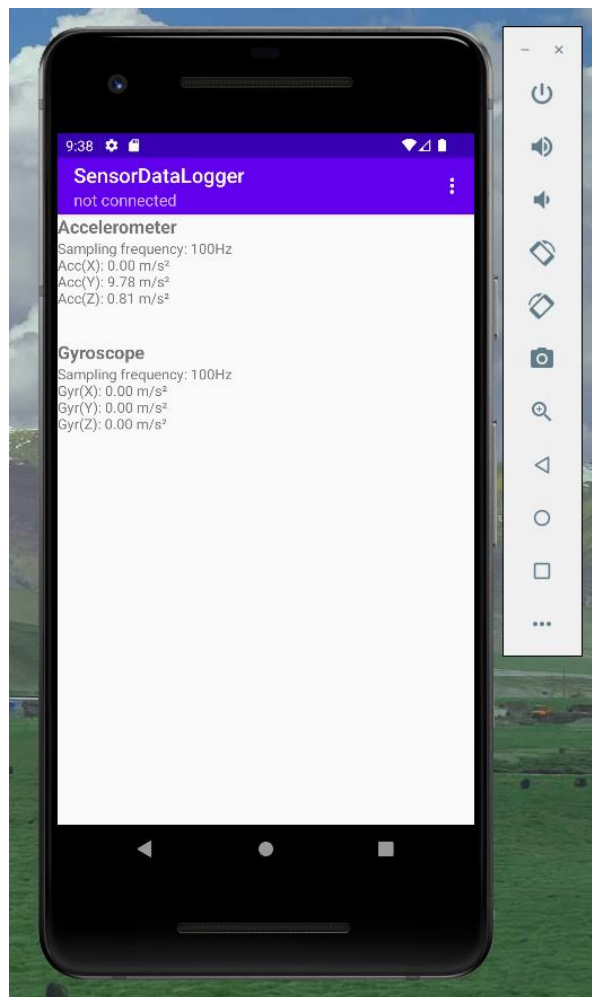
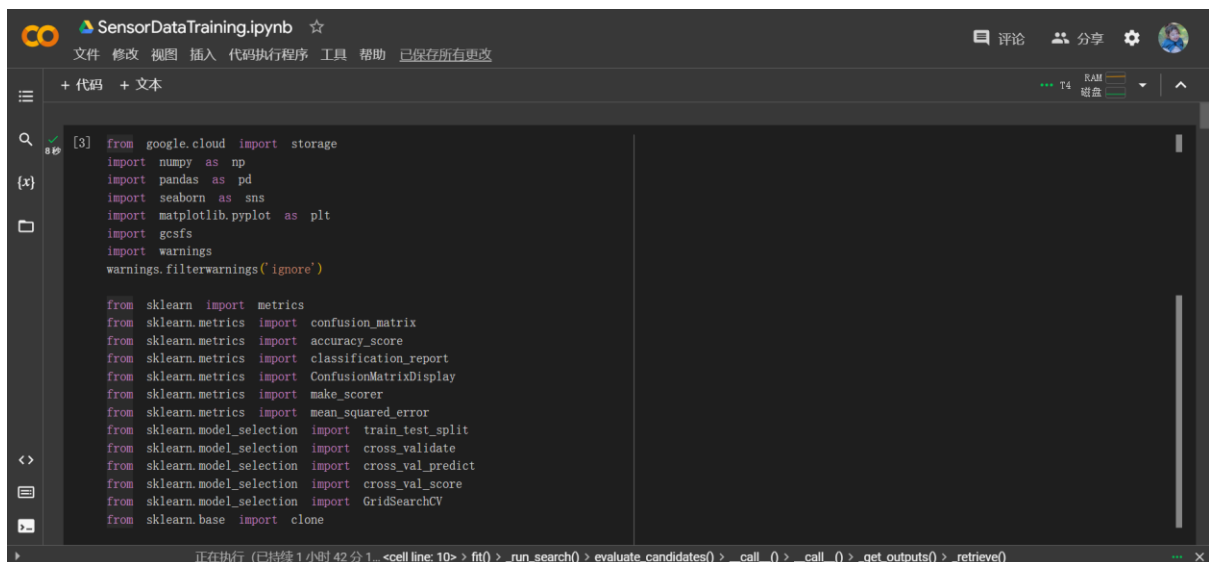


Figure 5.2: Android emulators used.



```

[3] from google.cloud import storage
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import gcfs
import warnings
warnings.filterwarnings('ignore')

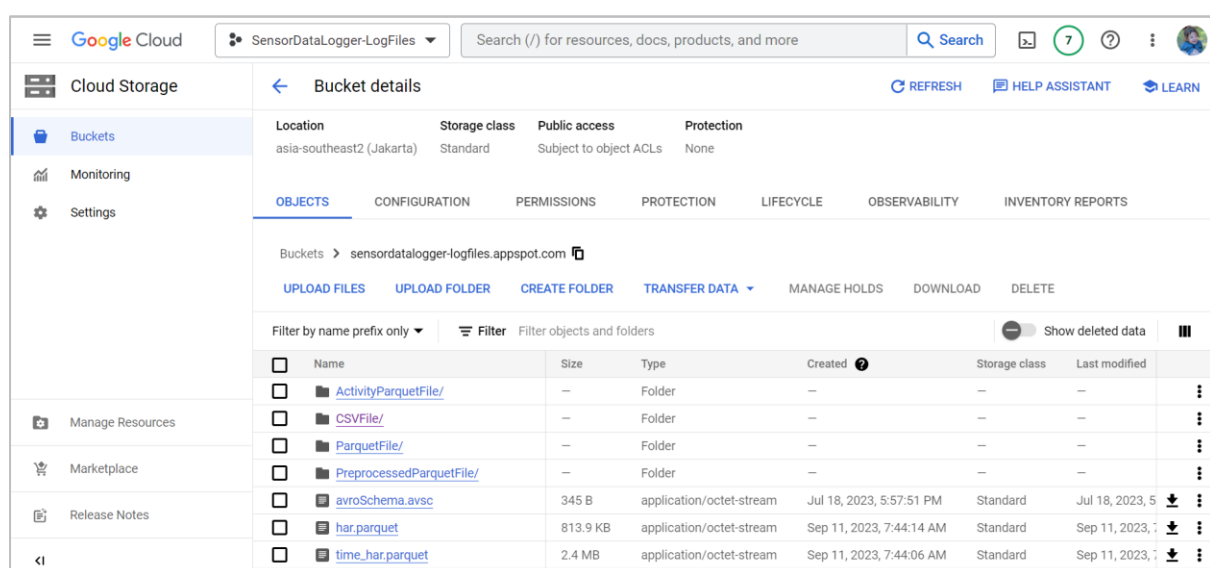
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.base import clone

```

Figure 5.3: Google Colab interface.

5.3 Cloud Setup

This project involves the usage of cloud platform, which is Google Cloud Storage to store the uploaded sensor data log files for further processing. Before the researcher starts to collect data, the SensorDataLogger application will request the researcher to log into Google account. After the researcher collects the data, the log files will be stored locally inside smartphone storage. After done collecting data for all the activities, the researcher can click the button ‘Upload Log Files’ via SensorData application to upload all the log files to the logged in Google Cloud Storage server.



Name	Size	Type	Created	Storage class	Last modified
ActivityParquetFile/	–	Folder	–	–	–
CSVFile/	–	Folder	–	–	–
ParquetFile/	–	Folder	–	–	–
PreprocessedParquetFile/	–	Folder	–	–	–
avroSchema.avsc	345 B	application/octet-stream	Jul 18, 2023, 5:57:51 PM	Standard	Jul 18, 2023, 5:57:51 PM
har.parquet	813.9 KB	application/octet-stream	Sep 11, 2023, 7:44:14 AM	Standard	Sep 11, 2023, 7:44:14 AM
time_har.parquet	2.4 MB	application/octet-stream	Sep 11, 2023, 7:44:06 AM	Standard	Sep 11, 2023, 7:44:06 AM

Figure 5.4: Google Cloud Storage bucket used to store the log files.

5.4 Setting and Configuration

For data collection, two Android devices installed with SensorData and SensorDataLogger are used. These two devices must be configured first before data collection. For instance, both applications need to be given permissions to access Bluetooth and storage of the devices. Bluetooth permission is needed because both applications need to send and receive instructions via Bluetooth. Storage permission is needed because SensorData will store the activity buttons' details created, which is the activity name and ID inside the local storage while SensorDataLogger will store the log files created inside the local storage temporarily before uploading to cloud. Besides, SensorDataLogger application will ask the user to log into Google account. This is because the log files collected later will be uploaded to Google Cloud Storage for further storage and processing. Furthermore, dialogs will pop out on both applications, asking to turn on Bluetooth if they are currently inactive. On the other hand, activity buttons need to be created before data collection by defining unique activity names and IDs. The configurations are also needed for GCS. Before we can upload the log files to cloud server, the researcher needs to create a project through the console and create a service account for authentication purpose. Then, we need to include the code to connect the SensorDataLogger application with the GCS during the development of mobile applications.

Next, the data collection process requires some settings. The device installed with SensorDataLogger should stay inside the front pocket of the data subject along the way when data collection process is performed. Whereas the device installed with SensorData will be held in hand by researcher or data subject to start the data collection process. This project is collecting sensor signals for 4 activities, including walking, standing, climbing upstairs, and climbing downstairs. There are in total 5 data subjects involved, whereby each data subject will perform each activity for 1 minute.

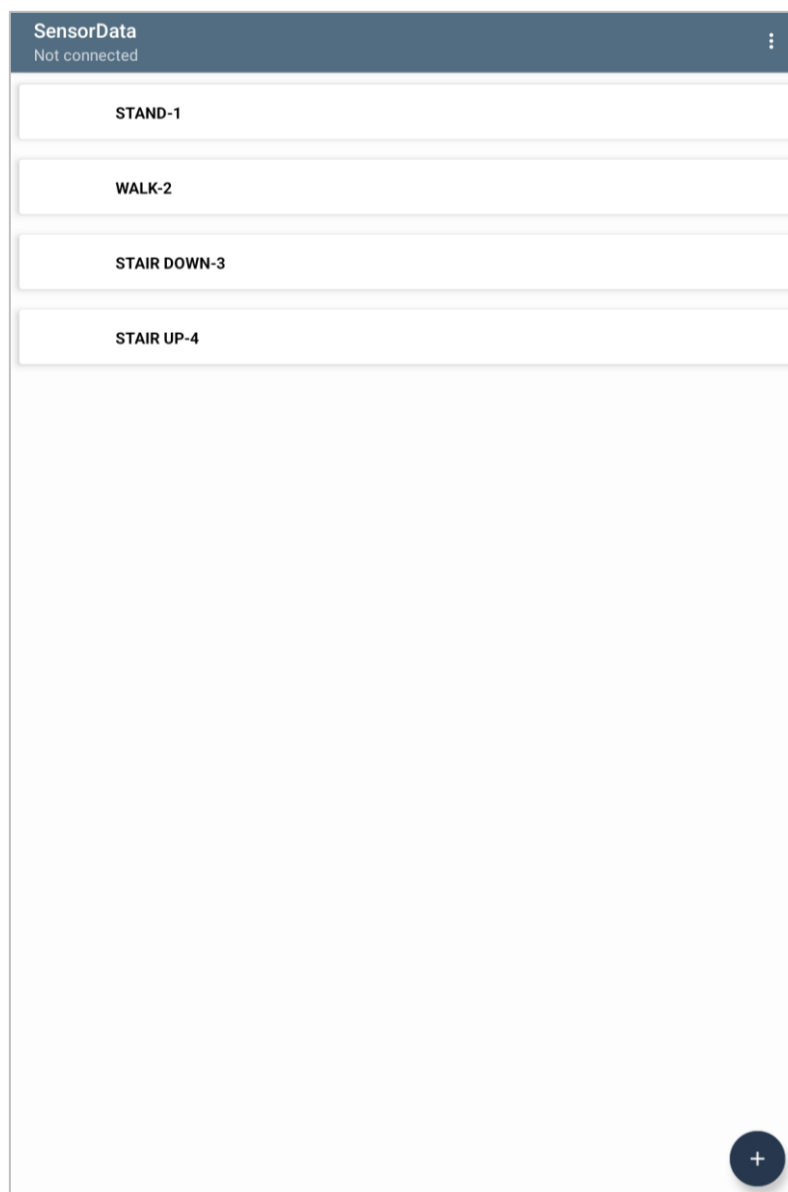
The HAR related operations such as dataset creation, data exploration, data segmentation, model training and evaluation, hyperparameter tuning and model testing are implemented in Google Colab. Before we can perform the operations, we need to first sign into the Google Colab using Google account and create a new .ipynb notebook. Besides, to extract the data stored in Google Cloud Storage to Google Colab, we need to create a service account via Google Cloud Console and upload the .json file of the service account to Google Colab for authentication purpose. Then, we need to run the code from *google.colab import auth* and *auth.authenticate_user()* to connect the Google Colab to the relevant GCS account.

5.5 System Operation

5.4.1 Data Engineering

The data engineering system in this project refers to sensor signals collection, annotation, and storage. For the implementation of data engineering system, two Android devices installed with SensorData and SensorDataLogger applications are used.

5.4.1.1 User Interface of SensorData and SensorDataLogger



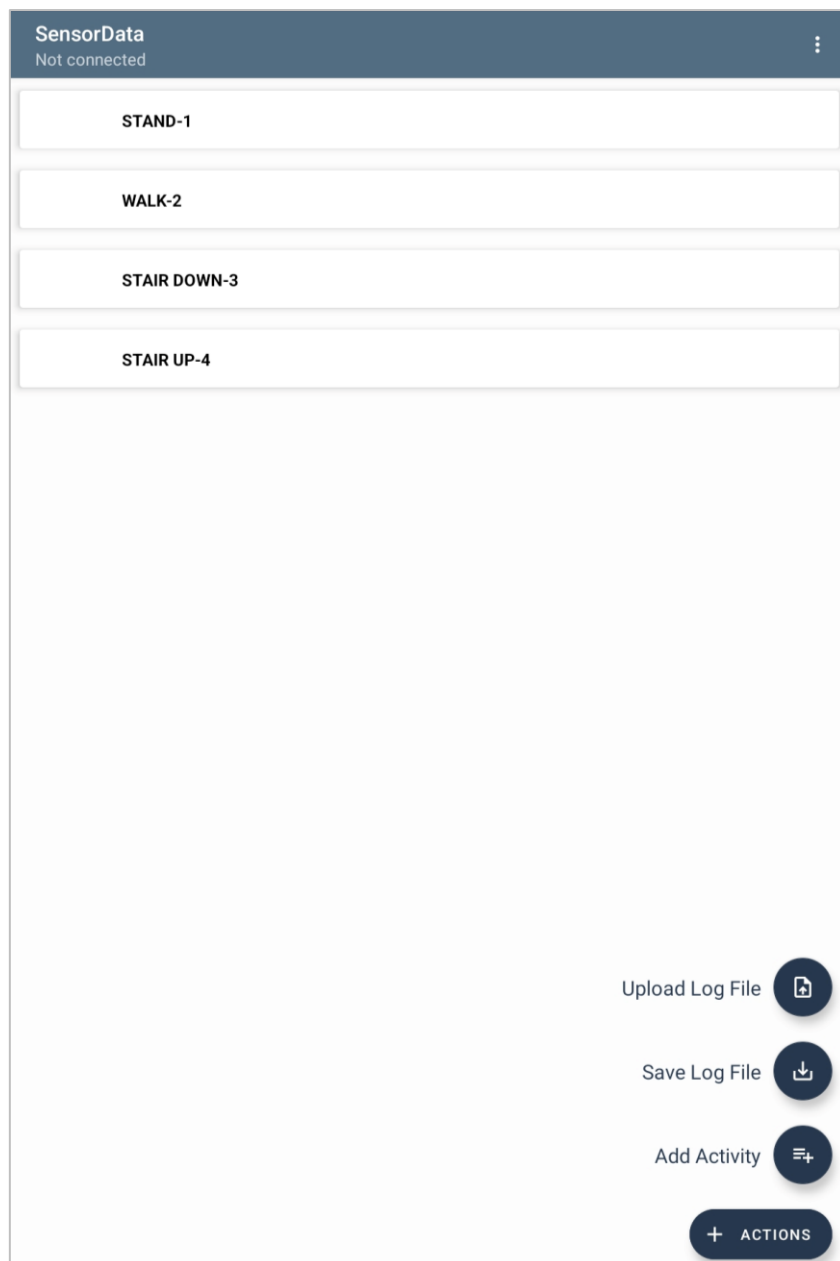


Figure 5.5: User Interface of SensorData

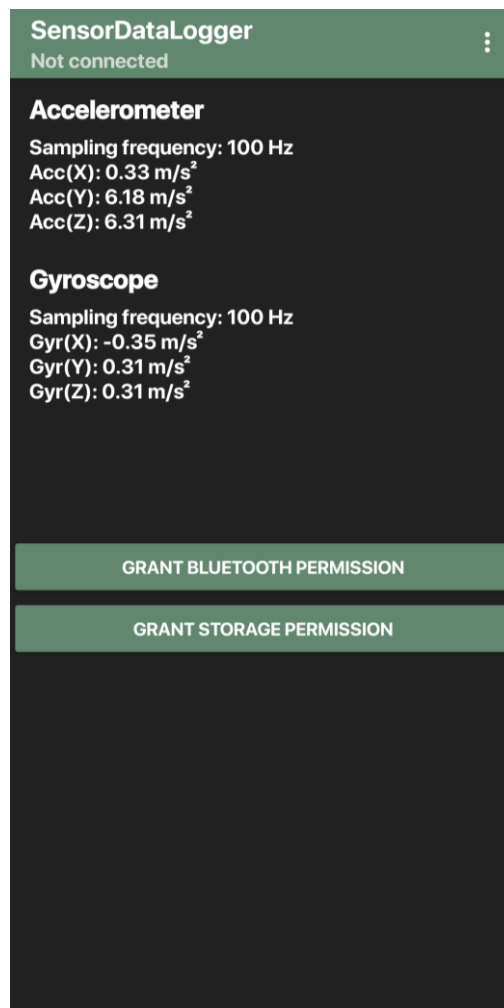


Figure 5.6: User Interface of SensorDataLogger

5.4.1.2 Authorization to use Bluetooth and storage

Before start collecting data, the SensorData and SensorDataLogger applications are given permissions to use Bluetooth and storage. The developed applications are connected using Bluetooth to send and receive instructions. Besides, devices' local storage will be needed to store the sensor data log files temporarily. Users need to grant permissions to allow the applications to access the Bluetooth service and local storage of devices. If any of the permissions are not granted by users, it will bring effects to the following processes of the applications. For instance, the applications cannot communicate via Bluetooth, or the application cannot store log files into local storage of device.

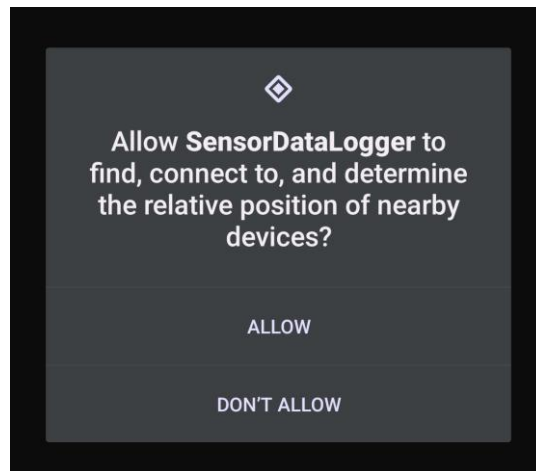


Figure 5.7: Dialog box asking for Bluetooth permission.

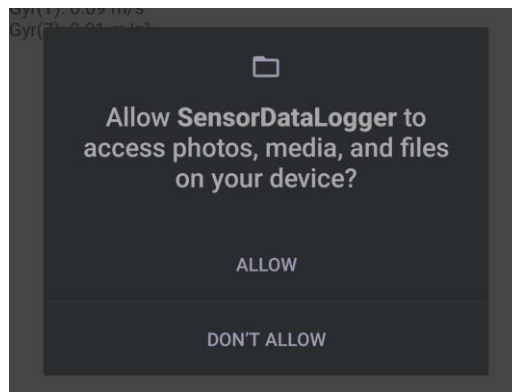


Figure 5.8: Dialog box asking for storage permission.

5.4.1.3 Turn on Bluetooth

The devices' Bluetooth need to be turned on before data collection. If the Bluetooth is inactive initially, a dialog box will pop out, asking to turn on the Bluetooth when the researcher starts the applications.

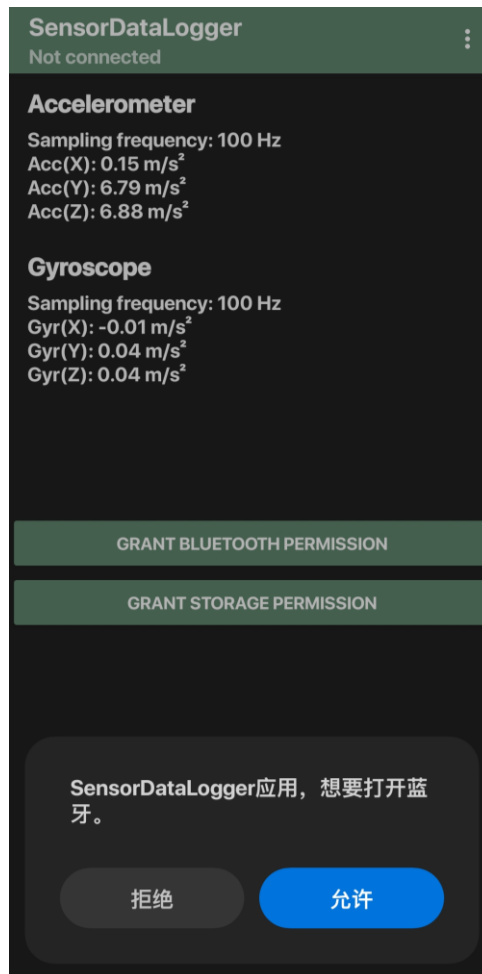


Figure 5.9: Dialog box asking to turn on Bluetooth in SensorDataLogger.



Figure 5.10: Dialog box asking to turn on Bluetooth in SensorData.

5.4.1.4 Google Cloud Storage server

One of the functions of SensorDataLogger is to upload all the sensor data log files in the specified path to Google Cloud Storage bucket, upon instructions from SensorData. To achieve this function, the application involves the usage of Google Cloud Platform and Google Cloud Storage. First, I will have to create a new project called SensorDataLogger in Google Cloud Platform through Google Cloud console. Next, I will create a bucket in Google Cloud Storage with the name *sensordatalogger-logfiles.appspot.com*. Last but not least, I will create a service account to authenticate SensorDataLogger application and authorize it to access user's account. Once these steps are completed, the SensorDataLogger application will be able to access user's Google account and upload log files to the created bucket for further processing.

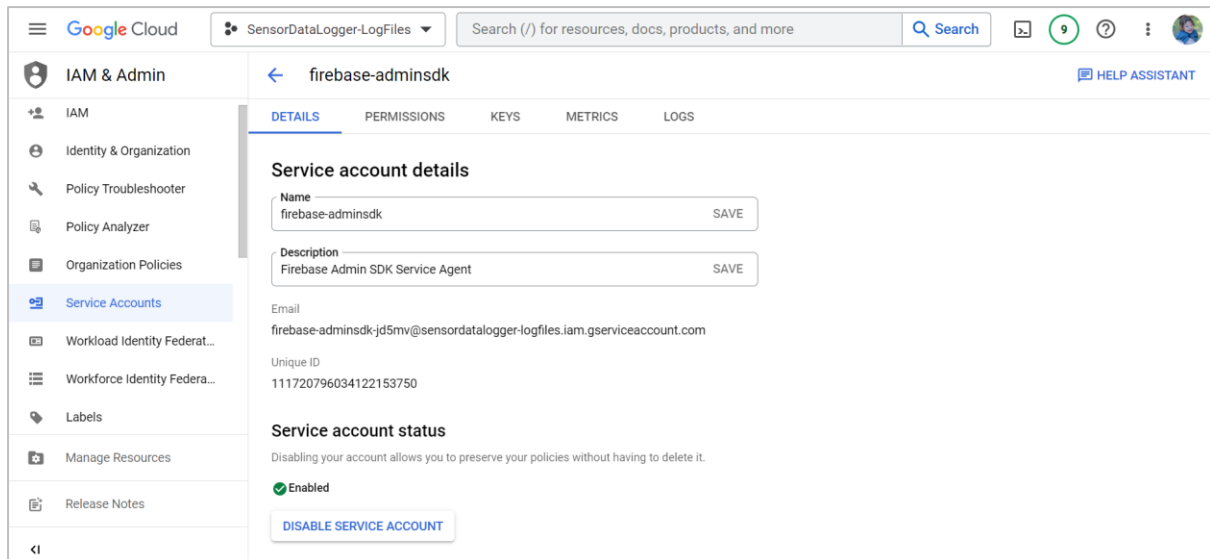


Figure 5.11: Service account created.

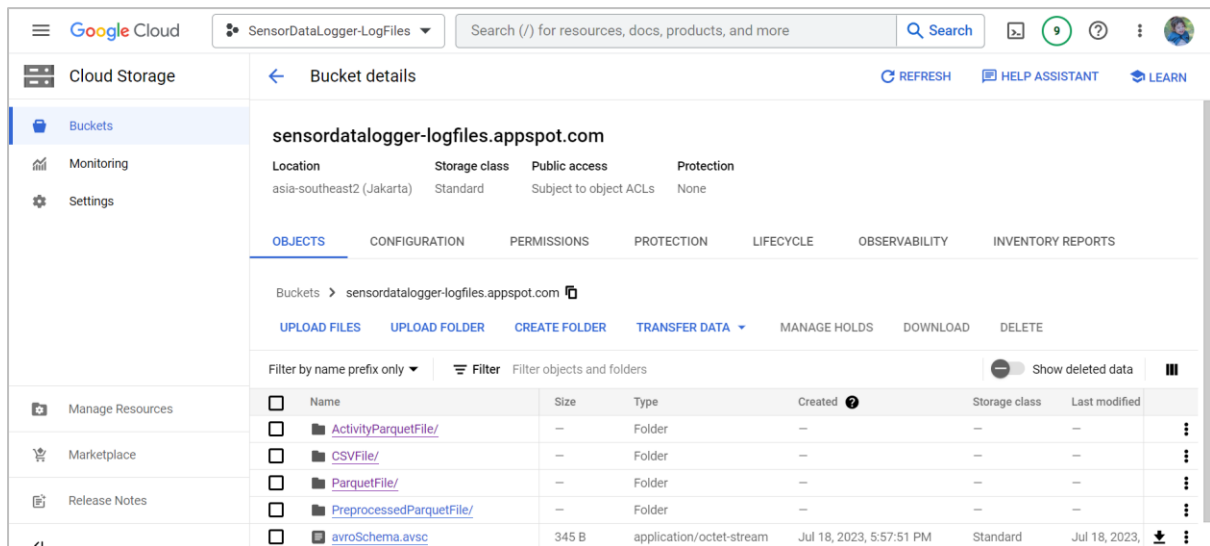


Figure 5.12: Bucket *sensordatalogger-logfiles.appspot.com* created.

5.4.1.5 Google account sign in

After the user grants the permissions to use Bluetooth and storage, the SensorDataLogger application will redirect user to Google account sign-in page. This step is necessary to ensure that the sensor data log files can be uploaded successfully to the relevant Google Cloud Storage bucket.

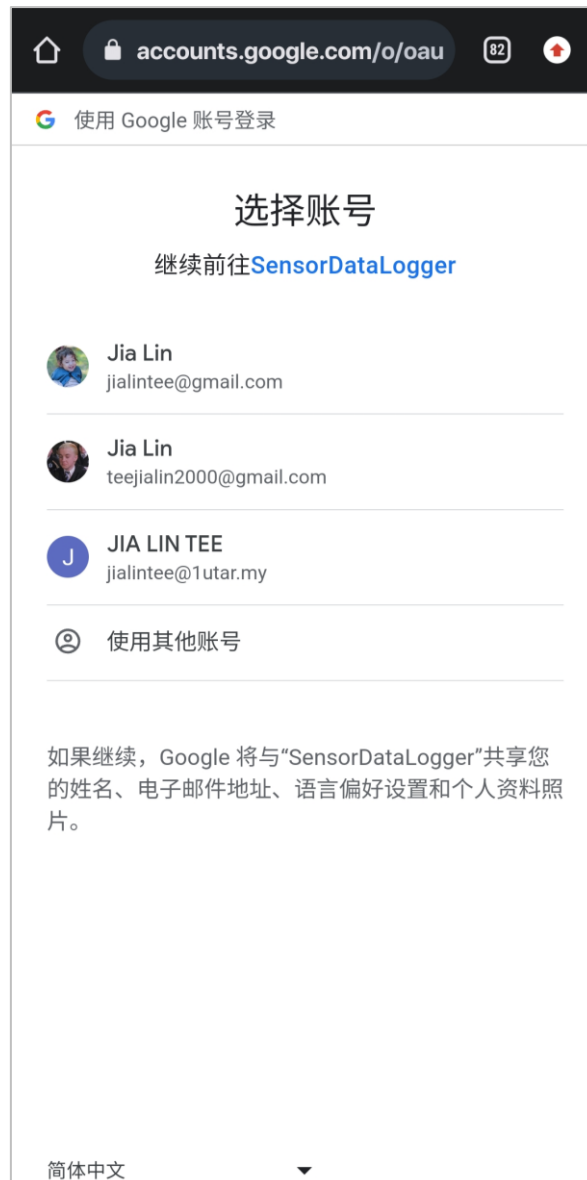


Figure 5.13: Google account sign-in page.

5.4.1.4 Add Activity Buttons

The researcher needs to add activity buttons that he or she wants to collect data for via SensorData application. The researcher needs to define unique activity name and class ID and click 'CREATE ACTION' button. The buttons created will be shown in the list in user interface. In this project, we create 4 activity buttons. with the name and ID as

- STAND - 1
- WALK - 2
- STAIR DOWN - 3
- STAIR UP - 4

Chapter 5

The activity ID specified here will be used as the filename of the log files. If users want to delete the activity buttons created all they need to do is swipe the created button in the list to the left.

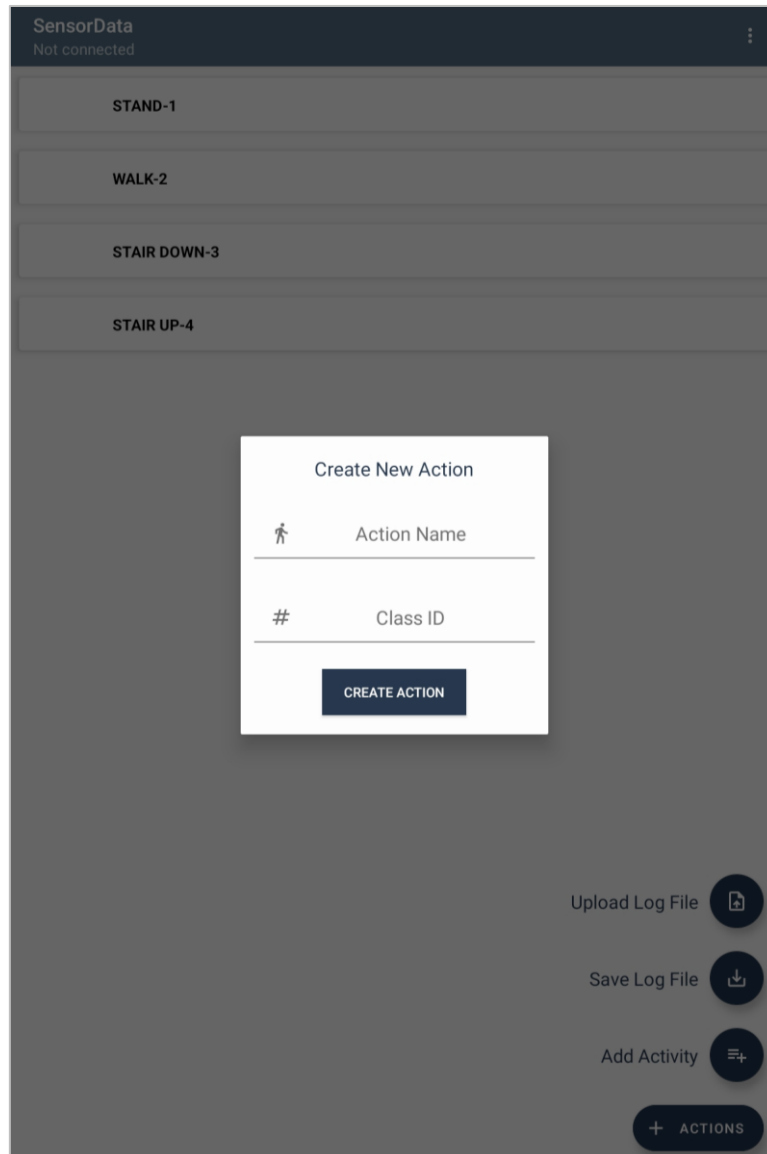


Figure 5.14: Dialog box to add activity buttons in SensorData.

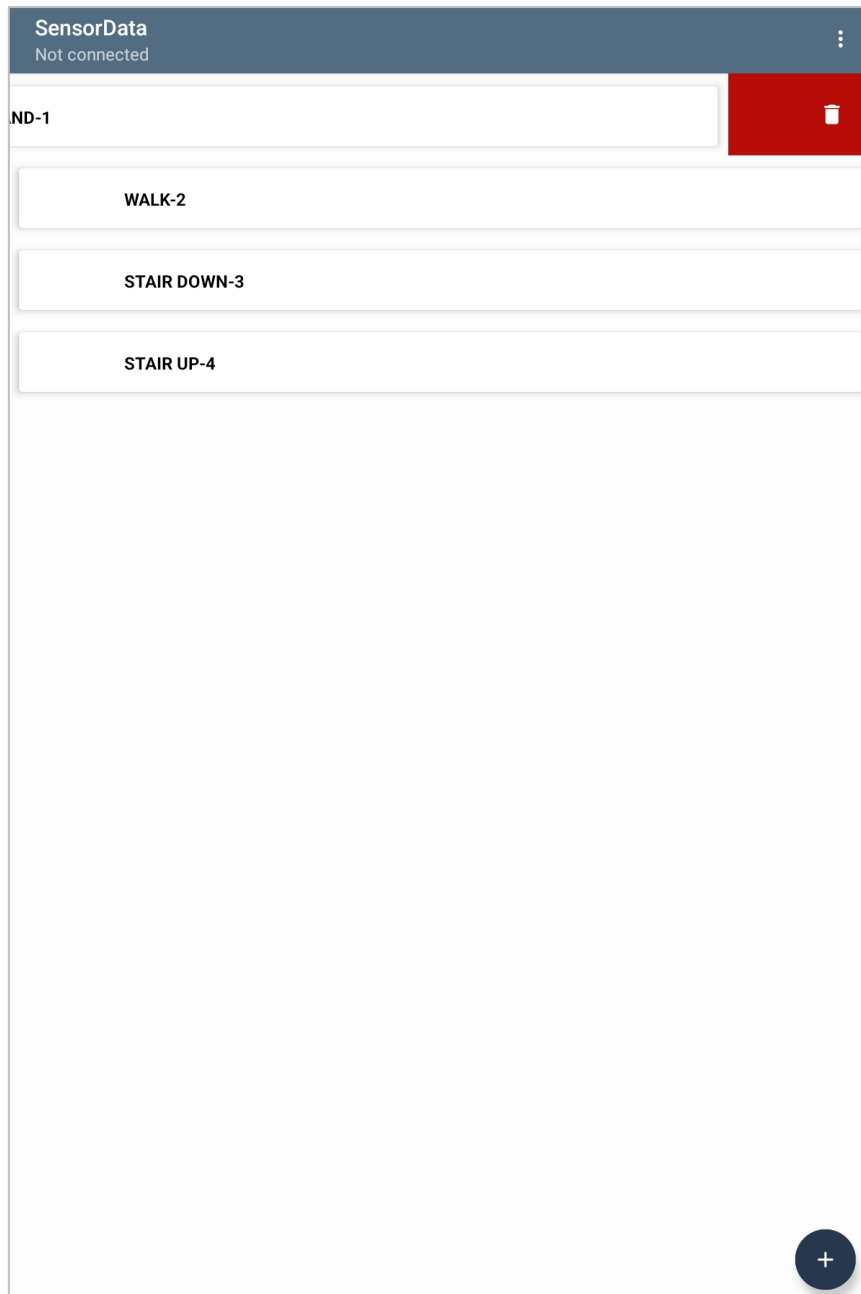


Figure 5.15: Swipe to the left to delete activity button.

5.4.1.5 Real-time accelerometer and gyroscope data

Once users grant the permission to use Bluetooth and storage as well as signing in Google account, the SensorDataLogger application will show real-time accelerometer and gyroscope data. This helps to monitor whether the application has access to accelerometer and gyroscope sensor or not. This function is achieved by using Sensor, SensorManager, SensorEvent and SensorEventListener classes imported from Android.hardware package.

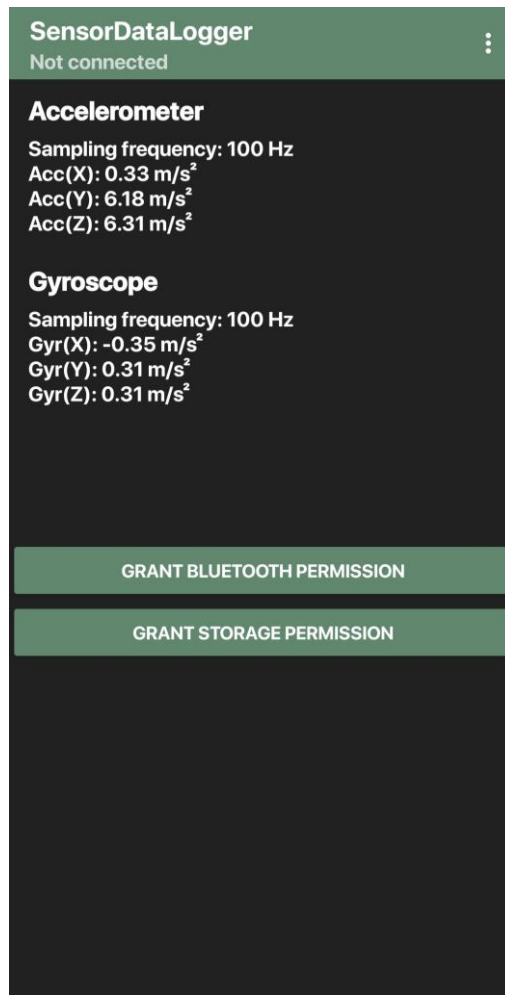


Figure 5.16: Real-time accelerometer and gyroscope data in SensorDataLogger.

5.4.1.6 List of activity buttons

Once users grant the permission to use Bluetooth and storage and users already went through the process of creating activity buttons, the SensorData application will show the list of activity buttons created.



Figure 5.17: List of activity buttons in SensorData.

5.4.1.8 Bluetooth connection

The researcher will connect both SensorData and SensorDataLogger via Bluetooth to start the data collection process. The applications contain features of showing list of paired devices and showing the Bluetooth connection status, whether it is disconnected or connected to certain device.

In this project, SensorData is installed on Samsung Galaxy A8 10.5 while SensorDataLogger is installed on Redmi Note 11 Pro 5G. To initiate Bluetooth connection from SensorData to SensorDataLogger, the user needs to click the option menu at the top right corner of the application and chooses the option 'Connect device'. Then, a list of pair devices will be shown. The user will then choose to connect with Redmi Note 11 Pro 5G. The status will then change from 'not connected' to 'connecting'. If the applications are successfully connected via Bluetooth, the status shown under the title bar of SensorData will change to 'Connected to Redmi Note 11 Pro 5G' and the status shown in SensorDataLogger will be 'Connected to Samsung Galaxy A8'.

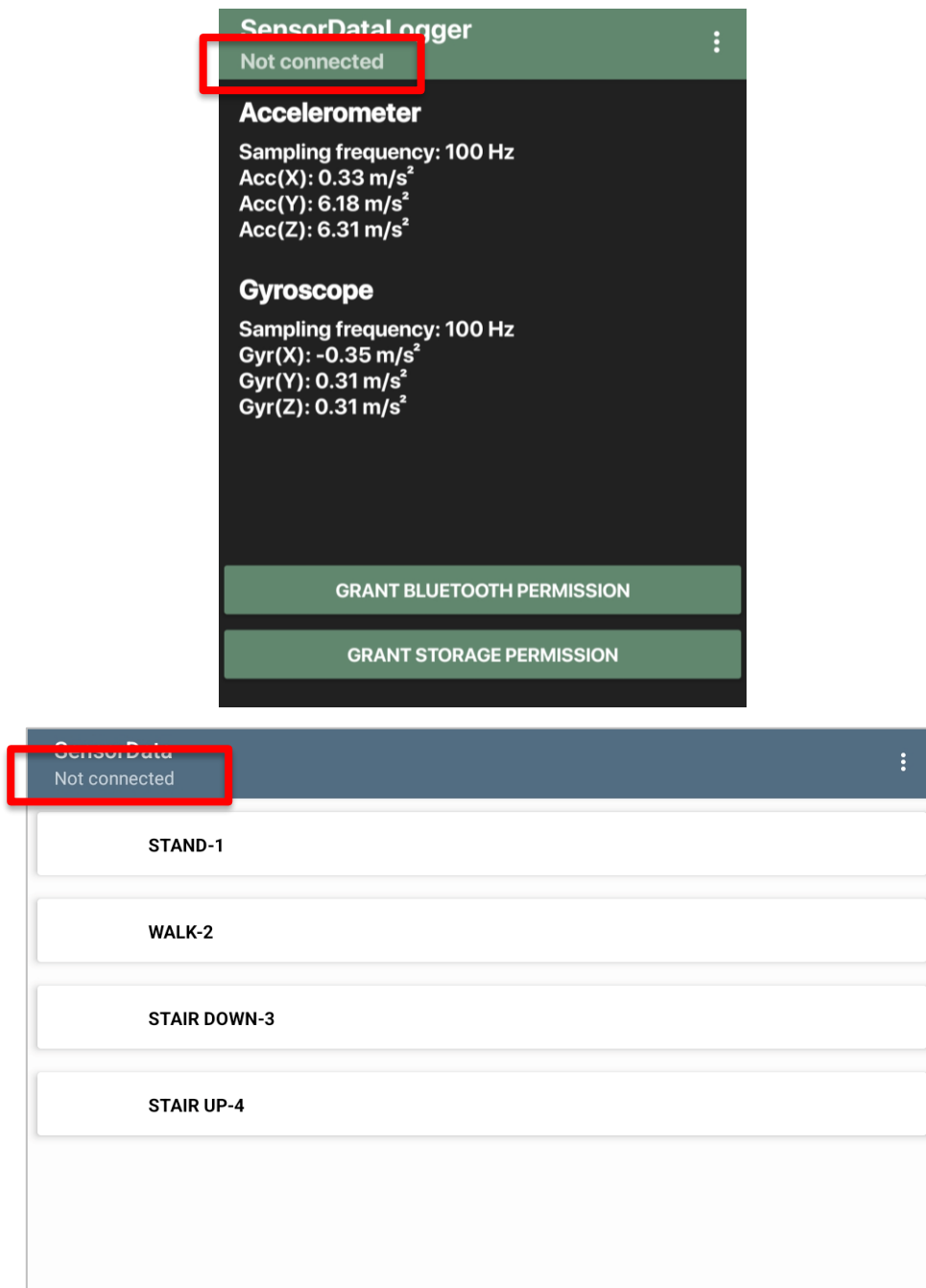


Figure 5.18: The devices show status 'Not connected'.

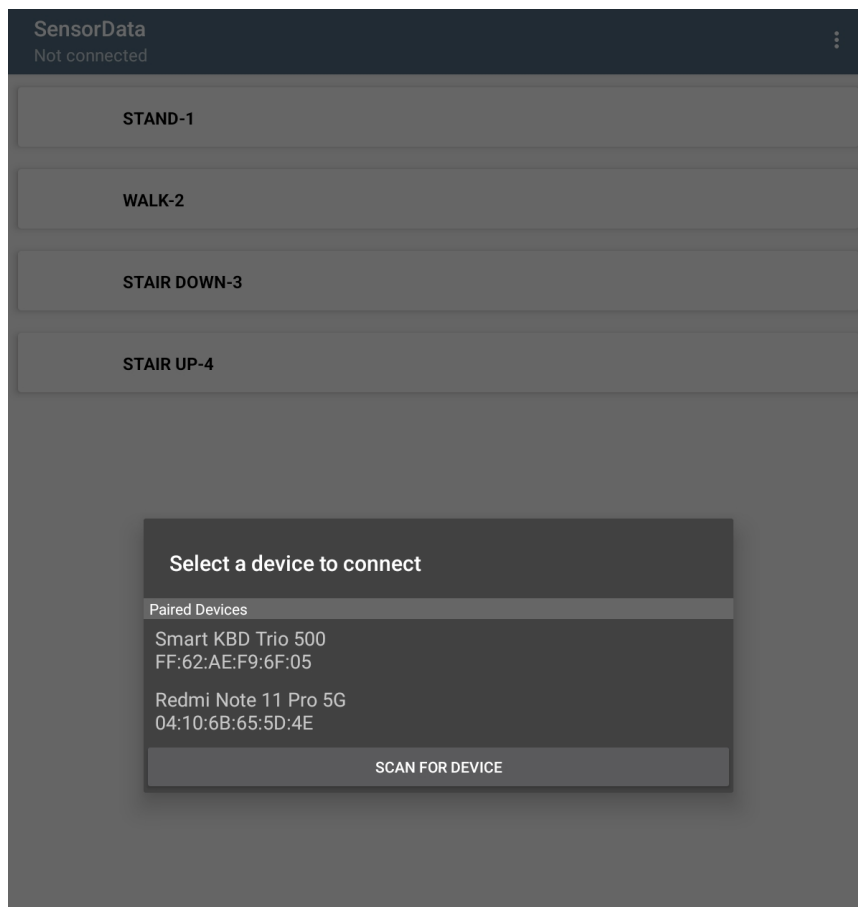
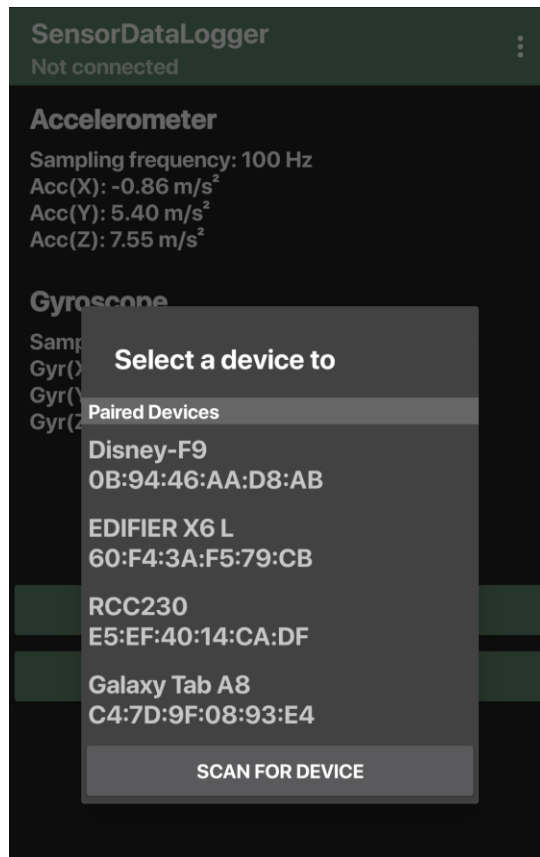


Figure 5.19: The device displays list of paired devices.



Figure 5.20: SensorData initiate Bluetooth connection to SensorDataLogger, status showing ‘Connecting...’

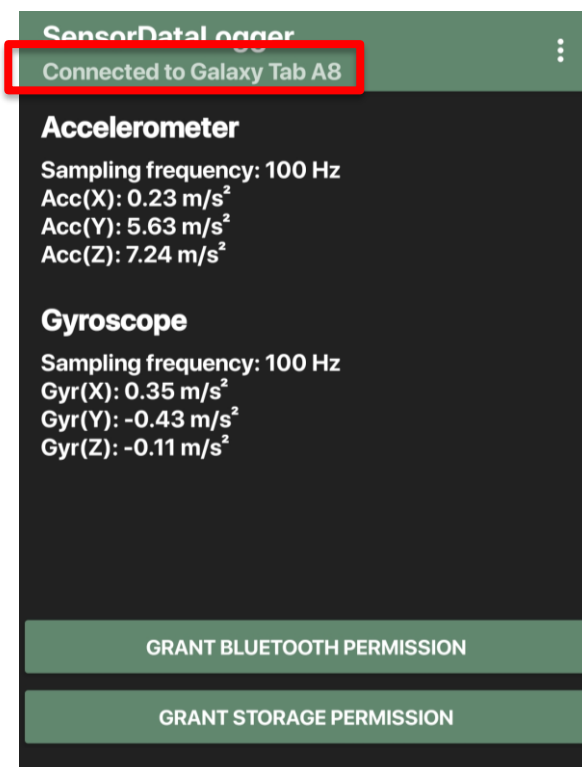




Figure 5.21: The devices show status ‘Connected to Redmi Note 11 Pro 5G’ and ‘Connected to Samsung Galaxy A8’.

5.4.1.9 Start recording

After the applications are connected via Bluetooth, they are now ready to send and receive instructions. Users can now start the data collection process. To collect data, the user needs to click the activity button that he or she wants to collect data for. After that, a dialog will pop out asking to set the duration for the relevant data collection process. Then, click the ‘START’ button. The SensorData application will send instructions to start recording to SensorDataLogger application and start a timer. Similarly, when SensorDataLogger application receives the instructions, a timer will start. Now the data subject starts to perform the activities, such as standing, walking, climbing upstairs and climbing downstairs. Note that the device installed with SensorDataLogger stays in the pocket of data subject along the way he or she is performing the activities to collect the accelerometer and gyroscope signals.



Figure 5.22: Click activity button 'WALK' via SensorDataLogger.

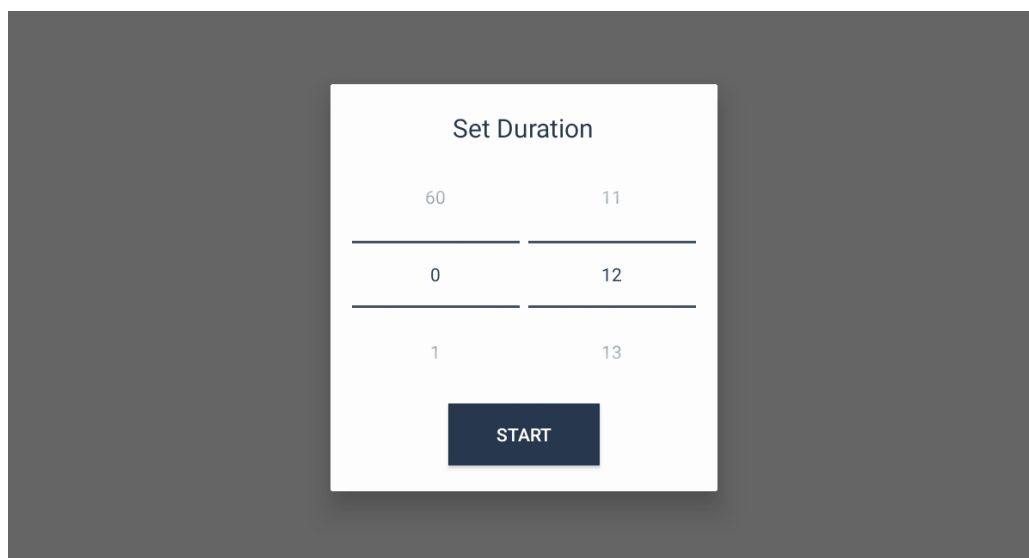


Figure 5.23: A dialog asking to set duration. After that, click 'START'.

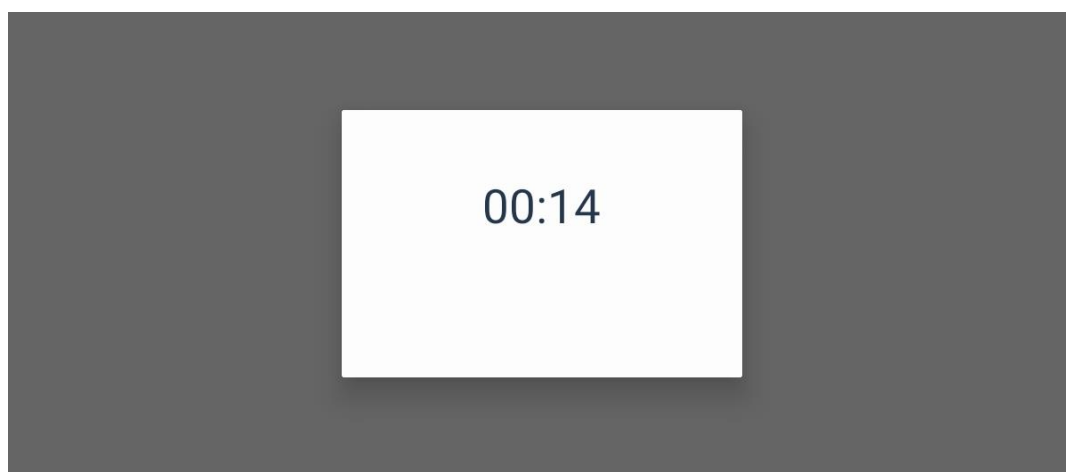


Figure 5.24: A timer starts at SensorData.

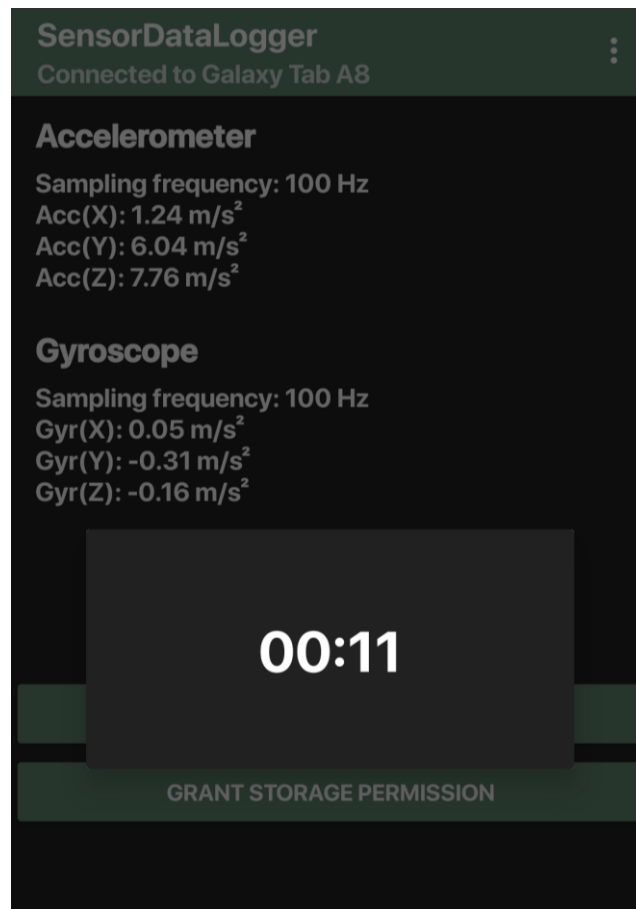


Figure 5.25: A timer starts at SensorDataLogger.

5.4.1.10 Stop recording

After timer stops, the SensorDataLogger and SensorData applications will stop data recording process. The log files created will be automatically annotated via SensorDataLogger application.

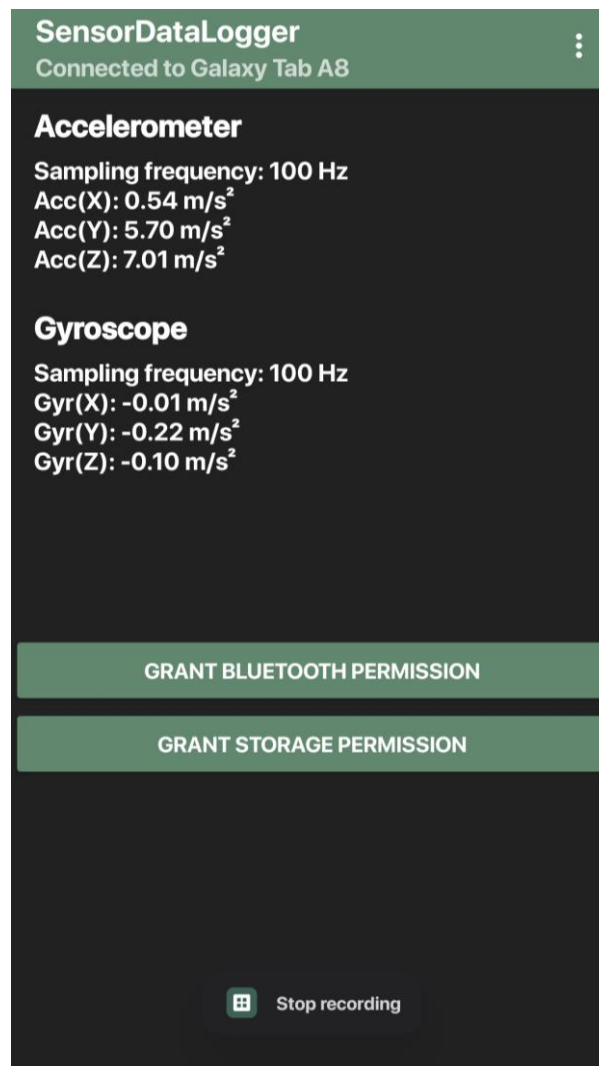


Figure 5.26: Timer stops. SensorDataLogger stops to collect sensor signals for ‘WALK’ activity.

5.4.1.11 Save log files

To save the log file, the researcher needs to click ‘Save Log File’ button and SensorData application will send instructions to SensorDataLogger to save the log file. The CSV log files will be saved in local storage of devices installed with SensorDataLogger, which is at `/storage/emulated/0/Android/data/my.edu.utar.sensordatalogger/files` with a fixed filename format, which is a combination of filename and datetime. Note that the user does not need to specify the filename manually. The filename for the relevant log file will be the activity ID specified when the activity button is created. For example, when the user creates the activity button for ‘WALK’ activity, the activity name defined is ‘WALK’ and the activity ID specified is ‘1’. Hence, after user collect data for walking activity and click ‘Save Log File’ button, the log files will be saved with the filename ‘Log_1_20230814202506.csv’.

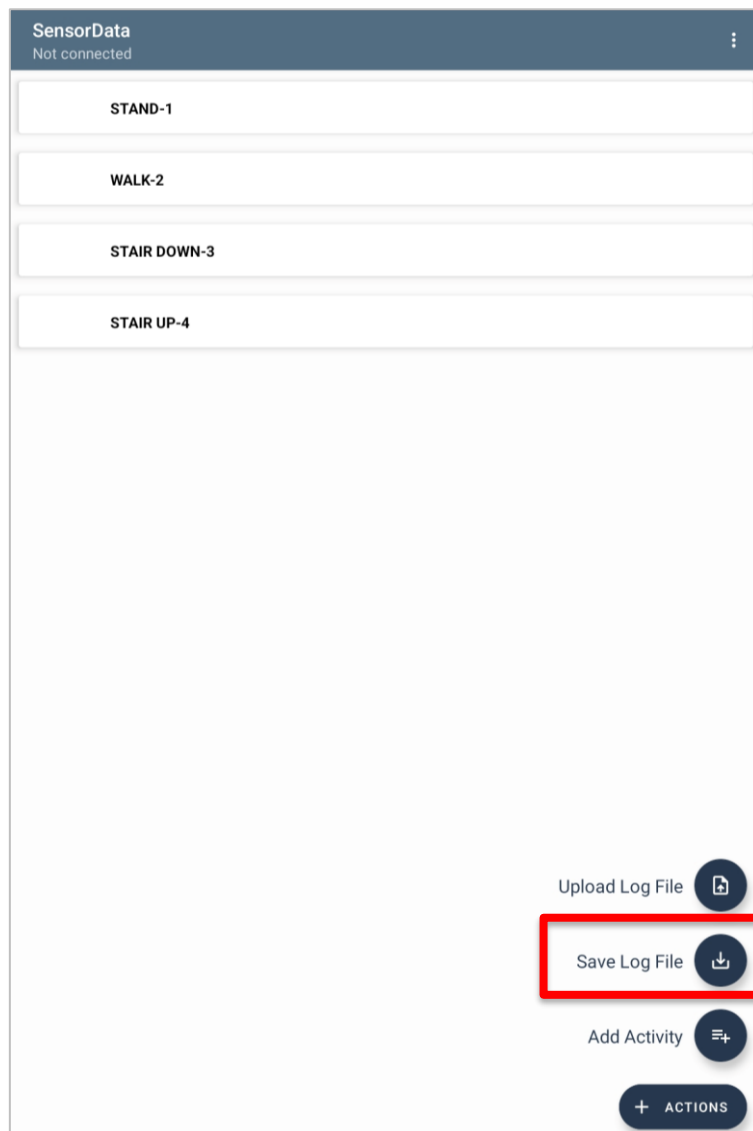


Figure 5.27: Click 'Save Log File' button.



Figure 5.28: Log files saved with specified filename.

	A	B	C	D	E	F
1	ACCE	0.16	0.41	9.76	192805386	2
2	ACCE	0.16	0.41	9.76	192805396	2
3	ACCE	0.18	0.41	9.76	192805406	2
4	ACCE	0.18	0.42	9.75	192805416	2
5	ACCE	0.17	0.41	9.75	192805426	2
6	ACCE	0.18	0.41	9.75	192805436	2
7	ACCE	0.21	0.43	9.75	192805447	2
8	ACCE	0.2	0.43	9.75	192805457	2
9	ACCE	0.17	0.42	9.77	192805467	2
10	ACCE	0.16	0.41	9.75	192805477	2
11	ACCE	0.16	0.41	9.77	192805487	2
12	ACCE	0.16	0.39	9.74	192805497	2
13	ACCE	0.17	0.4	9.76	192805507	2
14	ACCE	0.17	0.42	9.75	192805517	2
15	ACCE	0.18	0.41	9.74	192805527	2
16	ACCE	0.2	0.42	9.75	192805537	2
17	ACCE	0.2	0.42	9.77	192805547	2
18	ACCE	0.18	0.41	9.75	192805557	2
19	ACCE	0.19	0.42	9.74	192805567	2
20	ACCE	0.19	0.42	9.74	192805577	2
21	ACCE	0.18	0.42	9.76	192805587	2
22	ACCE	0.16	0.4	9.77	192805597	2
23	ACCE	0.17	0.43	9.75	192805607	2
24	ACCE	0.19	0.43	9.73	192805617	2
25	ACCE	0.21	0.45	9.74	192805627	2
26	ACCE	0.17	0.44	9.77	192805637	2
27	ACCE	0.18	0.48	9.77	192805648	2
28	ACCE	0.13	0.4	9.77	192805658	2
29	ACCE	0.16	0.4	9.77	192805668	2
30	ACCE	0.17	0.4	9.74	192805678	2
31	ACCE	0.15	0.42	9.77	192805688	2
32	ACCE	0.17	0.42	9.76	192805698	2
33	ACCE	0.14	0.38	9.78	192805708	2
34	ACCE	0.14	0.39	9.76	192805718	2
35	ACCE	0.14	0.41	9.77	192805728	2
36	ACCE	0.15	0.39	9.76	192805738	2
37	ACCE	0.18	0.41	9.75	192805748	2
38	ACCE	0.19	0.41	9.75	192805758	2
39	ACCE	0.21	0.45	9.73	192805768	2
40	ACCE	0.16	0.39	9.75	192805778	2

Figure 5.29: Log files content with automated annotation.

5.4.1.12 Upload log file to Google Cloud Storage bucket

Users can choose to either upload log file each time after collecting data and saving log file for one activity, or the user can choose to upload all the log files at once. The user would need to click the 'Upload Log File' button at SensorData. The instructions will be sent to SensorDataLogger, which will start to read and upload all the files available in the specified path to the specified bucket in the logged in Google Cloud Storage account. In this project, we are going to upload log files to the bucket *sensordatalogger-logfiles.appspot.com*. After uploaded successfully all the files, the log files in the local storage will be deleted.

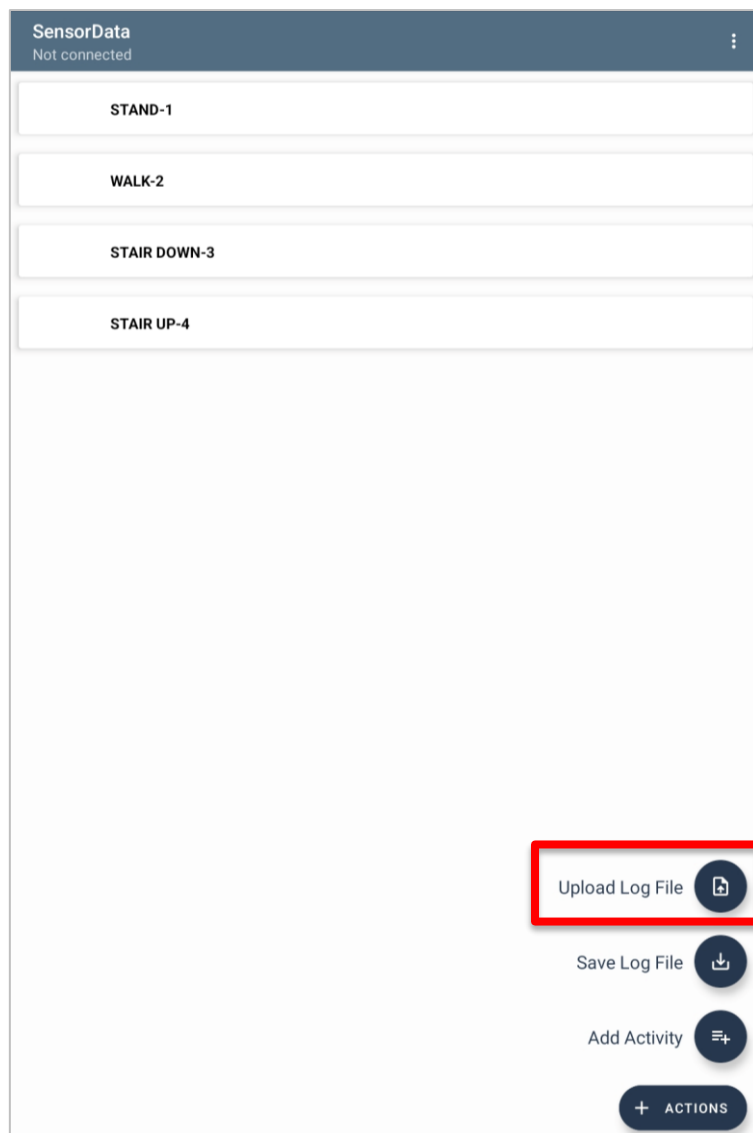


Figure 5.30: Click 'Upload Log File' button.

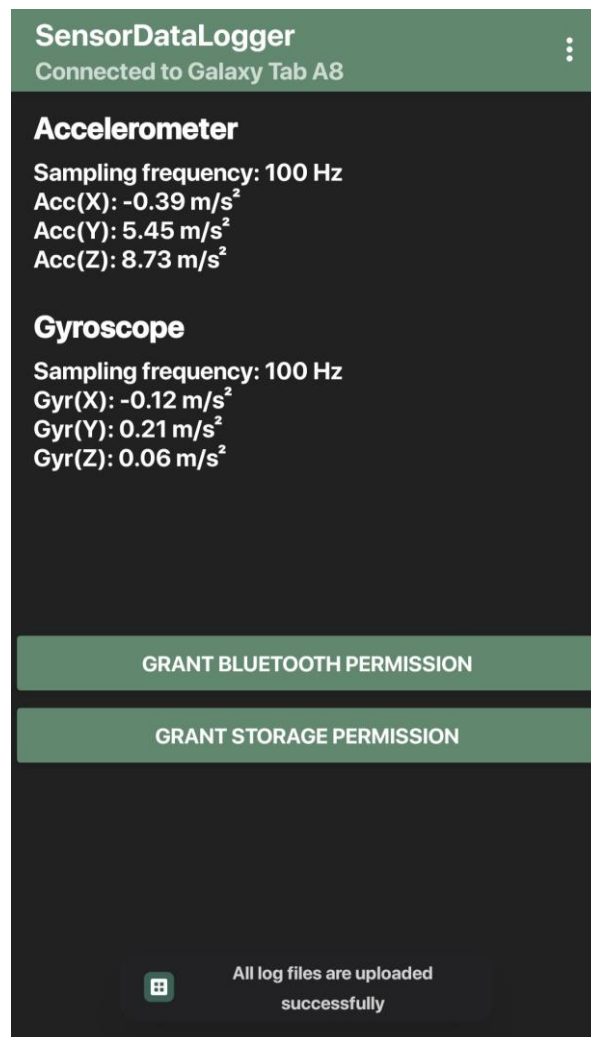


Figure 5.31: Upload log files to the bucket.

5.4.2 Dataset Creation

After data engineering process, the log files collected for all activities are now stored in the bucket *sensordatalogger-logfiles.appspot.com* in the folder *CSVFile/* in Google Cloud Storage server. This project involves 5 data subjects and collects accelerometer and gyroscope data for 4 activities. Each data subject is required to perform each activity for 1 minute. Hence, there are many individual log files being created and stored. Large amounts of individual log files make it difficult for subsequent HAR related operations such as data exploration, data segmentation, etc. Therefore, this dataset creation process is implemented to combine all the log files into single dataset and perform some basic preprocessing steps.

Before implementing the code for dataset creation, we need to import the necessary libraries, including *pandas*, *gcsfs* and *google.cloud*. We use the code from *google.cloud import storage*, *import pandas as pd* and *import gcsfs*.

This dataset creation process is implemented in Google Colab. First, we need to import the CSV log files. This is done by first retrieving the list of CSV file paths stored in GCS bucket. We specify the project ID, bucket name and folder name as shown in Figure 5.35. One of the example CSV file paths retrieved is “*CSVFile/Log_1_20230814202506.csv*”. To read the CSV file into Google Colab, gsutil URI, which is a file path specifically used by GCS, needs to be used. Hence, we need to format the filename of the CSV log files using the code in Figure 5.36. The example formatted file path looks like “*gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_1_20230814202506.csv*”. Then, we can read the CSV files by using *pd.read_csv()* and the formatted file path.

After that, we will convert the CSV log files to Parquet log files. Parquet files are used instead of CSV files since Parquet is a columnar storage file format designed for efficient compression and storage. It can significantly reduce the amount of space required to store the dataset compared to CSV files. This can be shown in Figure 5.37. Besides, Parquet files store data in a columnar format, making it easier to skip over irrelevant columns during data retrieval. This can lead to faster query and analysis performance, which is important in HAR research as we need to process and analyze data frequently. We convert the CSV files into Parquet files and later store the Parquet files back to GCS bucket inside folder *ParquetFile/* using the *to_parquet()*, as shown in Figure 5.38. To read the Parquet files, we need to format the filename to gsutil URI, as shown in Figure 5.39. Then, we can read the Parquet files using *read_parquet()* and the formatted file path.

Subsequently, we can combine the Parquet log files into single dataset. Before that, for each Parquet log files we will perform some preprocessing steps. For instance, drop rows with null values using *pandas.dropna()*, drop rows of data when timestamp equals to zero and sort the data in ascending order of timestamp using *pandas.sort_values()*. Then, we will combine the Parquet log files. The final dataset with the shape (120245, 7) is stored back into GCS with the name *har.parquet*. The final dataset consists of seven columns, whereby the first six columns are the signals data from accelerometer and gyroscope, and the last column is the activity label.

The complete code for dataset creation is shown in Appendix.

Chapter 5

The screenshot shows the Google Cloud Storage interface for a bucket named 'SensorDataLogger-LogFiles'. The interface includes a search bar, navigation tabs (Cloud Storage, Buckets, Monitoring, Settings, etc.), and a table of objects. The table has columns for Name, Size, Type, Created, Storage class, and Last modified. The objects listed are CSV files with names like 'Log_1_20230814202506.csv' and sizes ranging from 205.6 KB to 411.6 KB.

Name	Size	Type	Created	Storage class	Last modified
Log_1_20230814202506.csv	401 KB	application/octet-stream	Aug 14, 2023, 8:30:38 PM	Standard	Aug 14, 2023, 8:30:38 PM
Log_1_20230814202822.csv	402 KB	application/octet-stream	Aug 14, 2023, 8:30:38 PM	Standard	Aug 14, 2023, 8:30:38 PM
Log_1_20230814203019.csv	401.1 KB	application/octet-stream	Aug 14, 2023, 8:30:39 PM	Standard	Aug 14, 2023, 8:30:39 PM
Log_1_20230910133848.csv	407.3 KB	application/octet-stream	Sep 10, 2023, 2:13:02 PM	Standard	Sep 10, 2023, 2:13:02 PM
Log_1_20230910134100.csv	406.2 KB	application/octet-stream	Sep 10, 2023, 2:13:02 PM	Standard	Sep 10, 2023, 2:13:02 PM
Log_2_20230815131944.csv	400.7 KB	application/octet-stream	Aug 15, 2023, 3:35:40 PM	Standard	Aug 15, 2023, 3:35:40 PM
Log_2_20230815132233.csv	401 KB	application/octet-stream	Aug 15, 2023, 3:35:40 PM	Standard	Aug 15, 2023, 3:35:40 PM
Log_2_20230815132348.csv	400.5 KB	application/octet-stream	Aug 15, 2023, 3:35:41 PM	Standard	Aug 15, 2023, 3:35:41 PM
Log_2_20230910134549.csv	411.6 KB	application/octet-stream	Sep 10, 2023, 2:13:02 PM	Standard	Sep 10, 2023, 2:13:02 PM
Log_2_20230910134701.csv	410.9 KB	application/octet-stream	Sep 10, 2023, 2:13:03 PM	Standard	Sep 10, 2023, 2:13:03 PM
Log_3_20230910135254.csv	207 KB	application/octet-stream	Sep 10, 2023, 2:13:03 PM	Standard	Sep 10, 2023, 2:13:03 PM
Log_3_20230910135506.csv	206.5 KB	application/octet-stream	Sep 10, 2023, 2:13:04 PM	Standard	Sep 10, 2023, 2:13:04 PM
Log_3_20230910135659.csv	206.6 KB	application/octet-stream	Sep 10, 2023, 2:13:05 PM	Standard	Sep 10, 2023, 2:13:05 PM
Log_3_20230910135917.csv	205.6 KB	application/octet-stream	Sep 10, 2023, 2:13:05 PM	Standard	Sep 10, 2023, 2:13:05 PM
Log_3_20230910140101.csv	206.8 KB	application/octet-stream	Sep 10, 2023, 2:13:06 PM	Standard	Sep 10, 2023, 2:13:06 PM

Figure 5.32: Individual log files created and stored in Google Cloud Storage.

```
[ ] # get list of filepaths from google cloud bucket
csv_file_paths = []

def get_csv_files(project_id="sensordatalogger-logfiles"):
    storage_client = storage.Client(project=project_id)
    bucket = storage.Bucket(storage_client, 'sensordatalogger-logfiles.appspot.com')
    str_folder_name_on_gcs = 'CSVFile/'
    blobs = bucket.list_blobs(prefix=str_folder_name_on_gcs)
    print("Blob name:")
    for blob in blobs:
        csv_file_paths.append(blob.name)

get_csv_files()
print("List of csv files:")
print(csv_file_paths)
```

```
Blob name:
List of csv files:
['CSVFile/Log_1_20230814202506.csv', 'CSVFile/Log_1_20230814202822.csv', 'CSVFile/Log_1_20230814203019.csv', 'CSVFile/Log_1_20230910133848.csv', 'CSVFile/Log_1_20230910134100.csv', 'CSVFile/Log_2_20230815131944.csv', 'CSVFile/Log_2_20230815132233.csv', 'CSVFile/Log_2_20230815132348.csv', 'CSVFile/Log_2_20230910134549.csv', 'CSVFile/Log_2_20230910134701.csv', 'CSVFile/Log_3_20230910135254.csv', 'CSVFile/Log_3_20230910135506.csv', 'CSVFile/Log_3_20230910135659.csv', 'CSVFile/Log_3_20230910135917.csv', 'CSVFile/Log_3_20230910140101.csv']
```

Figure 5.33: Retrieve the list of CSV file paths stored in GCS.

```
[ ] # format file name
for i in range(len(csv_file_paths)):
    csv_file_paths[i] = "gs://sensordatalogger-logfiles.appspot.com/" + csv_file_paths[i]
print(csv_file_paths[i])
```

```
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_1_20230814202506.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_1_20230814202822.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_1_20230814203019.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_1_20230910133848.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_1_20230910134100.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_2_20230815131944.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_2_20230815132233.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_2_20230815132348.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_2_20230910134549.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_2_20230910134701.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910135254.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910135506.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910135659.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910135917.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910140101.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910140320.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910140552.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910140728.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910140921.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_3_20230910141143.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_4_20230910135127.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_4_20230910135350.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_4_20230910135604.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_4_20230910135755.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_4_20230910140005.csv
gs://sensordatalogger-logfiles.appspot.com/CSVFile/Log_4_20230910140203.csv
```

Figure 5.34: Format the filename of CSV file.

<input type="checkbox"/>	Name	Size	<input type="checkbox"/>	Name	Size
<input type="checkbox"/>	Log_1_20230814202506.csv	401 KB	<input type="checkbox"/>	Log_1_20230814202506.parquet	67.7 KB
<input type="checkbox"/>	Log_1_20230814202822.csv	402 KB	<input type="checkbox"/>	Log_1_20230814202822.parquet	66.9 KB
<input type="checkbox"/>	Log_1_20230814203019.csv	401.1 KB	<input type="checkbox"/>	Log_1_20230814203019.parquet	67.8 KB
<input type="checkbox"/>	Log_1_20230910133848.csv	407.3 KB	<input type="checkbox"/>	Log_1_20230910133848.parquet	68.3 KB
<input type="checkbox"/>	Log_1_20230910134100.csv	406.2 KB	<input type="checkbox"/>	Log_1_20230910134100.parquet	67 KB
<input type="checkbox"/>	Log_2_20230815131944.csv	400.7 KB	<input type="checkbox"/>	Log_2_20230815131944.parquet	108.2 KB
<input type="checkbox"/>	Log_2_20230815132233.csv	401 KB	<input type="checkbox"/>	Log_2_20230815132233.parquet	107.3 KB
<input type="checkbox"/>	Log_2_20230815132348.csv	400.5 KB	<input type="checkbox"/>	Log_2_20230815132348.parquet	107.8 KB
<input type="checkbox"/>	Log_2_20230910134549.csv	411.6 KB	<input type="checkbox"/>	Log_2_20230910134549.parquet	107.8 KB
<input type="checkbox"/>	Log_2_20230910134701.csv	410.9 KB	<input type="checkbox"/>	Log_2_20230910134701.parquet	108.1 KB
<input type="checkbox"/>	Log_3_20230910135254.csv	207 KB	<input type="checkbox"/>	Log_3_20230910135254.parquet	58.6 KB
<input type="checkbox"/>	Log_3_20230910135506.csv	206.5 KB	<input type="checkbox"/>	Log_3_20230910135506.parquet	60.1 KB
<input type="checkbox"/>	Log_3_20230910135659.csv	206.6 KB	<input type="checkbox"/>	Log_3_20230910135659.parquet	60 KB

Figure 5.35: Storage spaces needed for CSV log files (LHS) compared to Parquet log files (RHS).

```

▶ parquet_dir = "gs://sensordatalogger-logs.appspot.com/ParquetFile/"

def convert_csv_to_parquet(input_file_path, output_file_path):
    csv_df = pd.read_csv(input_file_path, names=['sensor', 'x', 'y', 'z', 'timestamp', 'label'])
    csv_df.to_parquet(output_file_path)

for i in range(len(csv_file_paths)):
    filename = csv_file_paths[i].split('/')[-1]
    filename = filename.replace('csv', 'parquet')
    parquet_file_name = parquet_dir + filename
    print(parquet_file_name)
    convert_csv_to_parquet(csv_file_paths[i], parquet_file_name)

```

Figure 5.36: Convert CSV files to Parquet files. Save back the Parquet files to GCS bucket inside folder ParquetFile/.

```

▶ # get list of filepaths from google cloud bucket
parquet_file_paths = []

def get_parquet_files(project_id="sensordatalogger-logs"):
    storage_client = storage.Client(project=project_id)
    bucket = storage.Bucket(storage_client, 'sensordatalogger-logs.appspot.com')
    str_folder_name_on_gcs = 'ParquetFile/'
    blobs = bucket.list_blobs(prefix=str_folder_name_on_gcs)
    print("Blob name:")
    for blob in blobs:
        parquet_file_paths.append(blob.name)

get_parquet_files()
print("List of parquet files:")
print(parquet_file_paths)

```

☞ Blob name:
 List of parquet files:
 ['ParquetFile/Log_1_20230814202506.parquet', 'ParquetFile/Log_1_20230814202822.parquet', 'ParquetFile/Log_1_20230814203019.parquet', 'ParquetFile/L

Figure 5.37: Retrieve the list of Parquet file paths stored in GCS.

```

[ ] # format file name
for i in range(len(parquet_file_paths)):
    parquet_file_paths[i] = "gs://sensordatalogger-logs.appspot.com/" + parquet_file_paths[i]
    print(parquet_file_paths[i])

```

```
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_1_20230814202506.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_1_20230814202822.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_1_20230814203019.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_1_20230910133848.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_1_20230910134100.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_2_20230815131944.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_2_20230815132233.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_2_20230815132348.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_2_20230910134549.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_2_20230910134701.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910135254.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910135506.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910135659.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910135917.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910140101.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910140320.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910140552.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910140728.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910140921.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_3_20230910141143.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_4_20230910135127.parquet
gs://sensordatalogger-logfiles.appspot.com/ParquetFile/Log_4_20230910135350.parquet
```

Figure 5.38: Format the filename of Parquet file.

```
# drop null values
preprocessed_df.dropna(inplace=True)

# drop the rows where timestamp is 0
preprocessed_df = preprocessed_df[preprocessed_df['timestamp'] != 0]

# sort in ascending order
acce_raw_df = acce_raw_df.sort_values(by = ['ts'], ignore_index=True)
gyro_raw_df = gyro_raw_df.sort_values(by = ['ts2'], ignore_index=True)
```

Figure 5.39: Preprocessing.

Chapter 5

Buckets > sensordatalogger-logfiles.appspot.com

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DATA MANAGE HOLDS DOWNLOAD DELETE

Filter by name prefix only Filter Filter objects and folders

<input type="checkbox"/>	Name	Size	Type	Created	Storage class
<input type="checkbox"/>	ActivityParquetFile/	–	Folder	–	–
<input type="checkbox"/>	CSVFile/	–	Folder	–	–
<input type="checkbox"/>	ParquetFile/	–	Folder	–	–
<input type="checkbox"/>	PreprocessedParquetFile/	–	Folder	–	–
<input type="checkbox"/>	avroSchema.avsc	345 B	application/octet-stream	Jul 18, 2023, 5:57:51 PM	Standard
<input type="checkbox"/>	har.parquet	813.9 KB	application/octet-stream	Sep 12, 2023, 11:03:05 AM	Standard
<input type="checkbox"/>	time_har.parquet	2.4 MB	application/octet-stream	Sep 12, 2023, 11:01:52 AM	Standard

```
➔      x      y      z      x2     y2     z2     label
0      -0.58  9.71 -0.22 -0.03 -0.05  0.02     1
1      -0.58  9.74 -0.23 -0.04 -0.04  0.02     1
2      -0.59  9.76 -0.23 -0.05 -0.03  0.02     1
3      -0.58  9.76 -0.22 -0.05 -0.01  0.03     1
4      -0.58  9.77 -0.22 -0.06  0.01  0.03     1
...    ...    ...    ...    ...    ...    ...    ...
120240 -0.36  9.89  0.32 -0.01 -0.05 -0.00     4
120241 -0.35  9.87  0.35 -0.01 -0.05 -0.00     4
120242 -0.34  9.84  0.38 -0.00 -0.05 -0.00     4
120243 -0.33  9.82  0.39  0.01 -0.05 -0.00     4
120244 -0.31  9.82  0.40  0.02 -0.05  0.00     4

[120245 rows x 7 columns]
(120245, 7)
```

Figure 5.40: Some part of the final dataset, *har.parquet*.

5.4.3 Data Exploration

Firstly, we need to import the necessary libraries such as pandas, numpy, seaborn, and matplotlib. We use the code *import numpy as np*, *import pandas as pd*, *import seaborn as sns*, and *import matplotlib.pyplot as plt*.

Then, we load the final dataset, *har.parquet* from GCS bucket by using *read_parquet()* and gsutil URI of the file into a pandas dataframe called *explore_df*. Then, we can print the shape of the dataset by using *explore_df.shape*. The shape of the dataset is (120245, 7).

Next, we use *explore_df.info()* to recognize the data type of each attributes in the dataset, including the number of non-null values and memory usage. The result shows that there are in total 120245 non-null rows of data for 7 attributes. The first six columns which consist of signals data from accelerometer and gyroscope have *float* datatype, whereas the last

column which consists of activity labels has *int* datatype. The total memory usage for this dataset is 6.4MB.

After that, we use *explore_df.describe()* to generate statistical summary of the dataset by calculating the mean, standard deviation, maximum, minimum, and quartile values of the data for each attribute in the dataset. This can help to gain insights into the central tendency and dispersion of the dataset, identifying potential outliers, and understanding the distribution of numerical variables in the dataset. The result shows that *x* attribute has the mean value of -0.151056, standard deviation of 1.559903, min value of -11.06 and max value of 10.51. -0.92, -0.34 and 1.05 are the values for 25%, 50% and 75% quartile respectively.

Next, the researcher identifies missing values and infinity values in the dataset by using *explore_df.isnull()*, *numpy.isinf()* and *numpy.isneginf()*. The result reveals that the dataset does not have any null values, negative infinity values and positive infinity values.

Subsequently, the researcher computes and visualize the class distributions in the dataset by using *pandas.crosstab()* and *seaborn.countplot()*. The result shows that the dataset has an approximately equal distribution of samples among all the activities. The activity labelled '1' has 30049 rows while the activity labelled '2' consists of 30040 rows. The activity labelled '3' has 30088 rows while the activity labelled '4' consists of 30068 rows.

Lastly, the researcher visualizes the relationship between the numerical variables in the dataset by using correlation matrix via *explore_df.corr()* and *seaborn.heatmap*. The squares with darker color indicate the stronger relationship between the variables, while squares with lighter color suggest weak relationships or no relationship. The result shows that there is a strong relationship between *x* and *z*, *y* and *y2*, etc.

The complete code for data exploration is shown in Appendix.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120245 entries, 0 to 120244
Data columns (total 7 columns):
#   Column   Non-Null Count  Dtype
---  ---      -
0   x         120245 non-null float64
1   y         120245 non-null float64
2   z         120245 non-null float64
3   x2        120245 non-null float64
4   y2        120245 non-null float64
5   z2        120245 non-null float64
6   label    120245 non-null int64
dtypes: float64(6), int64(1)
memory usage: 6.4 MB
```

Figure 5.41: Data type of each attribute in the dataset.

	x	y	z	x2	y2	z2	label
count	120245.000000	120245.000000	120245.000000	120245.000000	120245.000000	120245.000000	120245.000000
mean	-0.151056	9.628805	-0.740232	0.004703	-0.006436	-0.003936	2.500437
std	1.559903	2.348864	2.022521	0.710679	0.593695	0.309292	1.117998
min	-11.060000	1.270000	-10.190000	-1.800000	-4.290000	-1.550000	1.000000
25%	-0.920000	8.400000	-1.850000	-0.460000	-0.240000	-0.180000	2.000000
50%	-0.340000	9.740000	-0.390000	0.000000	0.000000	0.000000	3.000000
75%	1.050000	10.120000	0.030000	0.170000	0.240000	0.170000	4.000000
max	10.510000	28.430000	11.010000	3.230000	4.230000	1.440000	4.000000

Figure 5.42: Statistical summary of dataset.

```
x      0.0
y      0.0
z      0.0
x2     0.0
y2     0.0
z2     0.0
label  0.0
dtype: float64
```

Figure 5.43: Null values of dataset for each attributes.

```
[ ] # check whether the dataset contains infinity value

has_positive_infinity = np.isinf(explore_df).any().any()
has_negative_infinity = np.isneginf(explore_df).any().any()

if has_positive_infinity:
    print("The DataFrame contains positive infinity values.")
if has_negative_infinity:
    print("The DataFrame contains negative infinity values.")
if not has_positive_infinity and not has_negative_infinity:
    print("The DataFrame does not contain infinity values.")

The DataFrame does not contain infinity values.
```

Figure 5.44: Infinity values of dataset.

col_0	count
label	
1	30049
2	30040
3	30088
4	30068

Figure 5.45: Number of samples for each activity.

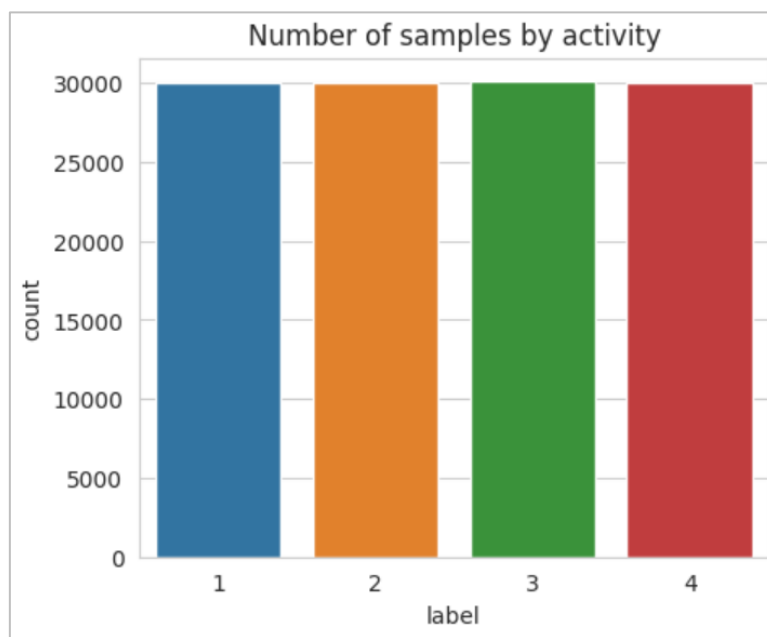


Figure 5.46: Bar graph showing number of samples for each activity.

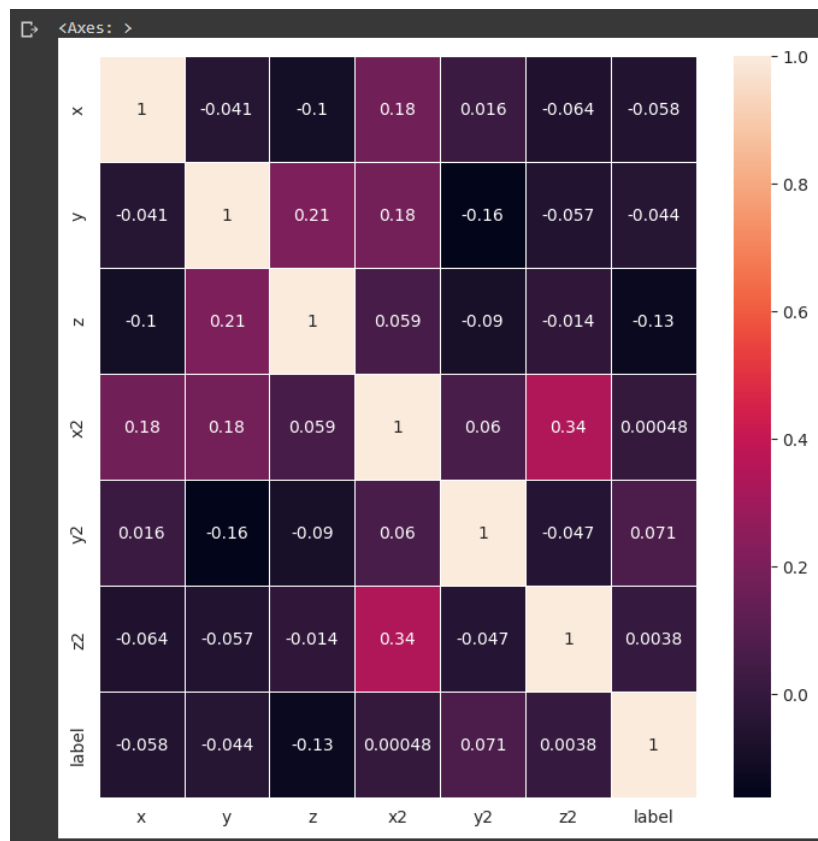


Figure 5.47: Correlation matrix.

5.4.4 Data Segmentation

Before data segmentation happens, we need to import necessary libraries such as *pandas*, *scipy* and *sklearn*. The codes being used are *from sklearn.model_selection import train_test_split*, *from scipy import stats*, *from scipy.stats import mode* and *import pandas as pd*.

Firstly, we load the final dataset, *har.parquet* from GCS bucket by using *read_parquet()* and gsutil URI of the file into a pandas dataframe called *lstm_df*.

Then, we split the dataset into small segments called windows. Each window consists of 0.5 seconds of data with 50% overlap to minimize information loss at the edge of windows. In other words, each segment consists of 50 consecutive data points from each of the six features. Sometimes, there may be a situation where there are two or more different labels in one segment of data. In this situation, the labels being assigned to each segment are decided by using the mode of labels. The mode of labels is calculated using *stats.mode*. Next, we split the signals and labels during segmentation and save the final results into different variables, *reshaped_segments* and *labels*. The result shows that the shape of the *reshaped_segments* is (4808, 50, 6), whereby the first number indicates the number of segments, the second number

indicates the number of time steps, and the third number indicates the number of features. The shape of the *labels* is (4808, 4), whereby it consists of 4808 segments and 4 classes.

Lastly, we split the dataset into 80% of training dataset and 20% of testing dataset by using *train_test_split()* and the input vector will be *reshaped_segments* and *labels* obtained from previous steps. The output vectors are *X_train*, *y_train*, *X_test* and *y_test*. The shape of the *X_train* is (3846, 50, 6) and *y_train* is (3846, 4). Whereas the shape of the *X_test* is (962, 50, 6) and *y_test* is (962, 4).

The complete code for data segmentation is shown in Appendix.

5.4.5 Model Training and Evaluation

In this project, we choose to use LSTM model. Beforehand, we need to import necessary libraries. The libraries we are going to use include *pandas*, *sklearn*, *matplotlib* and *tensorflow*.

Firstly, we need to build the architecture of the LSTM model. We define a *create_lstm_model()* which takes two parameters, dropout rate and learning rate. Inside the *create_lstm_model()* function, a sequential neural network is created by using *tf.keras.models.Sequential()*. After that, we add a LSTM layer with 32 units to the model by using *model.add(LSTM(units = 32, input_shape = (X_train.shape[1], X_train.shape[2])))*. Then, we add a dropout layer as the hidden layer by using *model.add(Dropout(dr))* where the default dropout rate, *dr* is 0.3. We also add a dense layer as the output layer with 4 units and softmax activation function by using *model.add(Dense(y_train.shape[1], activation = 'softmax'))*. Subsequently, we initialize an Adam optimizer with the learning rate, *lr* of 0.0025 by using *optimizer = Adam(learning_rate=lr)*. The LSTM model is then compiled with categorical cross-entropy loss function, the Adam optimizer initialized before and accuracy as the evaluation metrics. The code being involved here is *model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])*. We also run *model.summary()* to view the summary of LSTM architecture as shown in Figure 5.51.

Next, we can start training the LSTM model that we built by defining the *batch size* as 4 and *n_epochs* as 50. Early stopping is also defined here to monitor the validation loss during model training, by using *EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)*. Then, we start training by using *model.fit(X_train, y_train, epochs = n_epochs, validation_split = 0.20, callbacks=[early_stopping], batch_size =*

batch_size, verbose = 1). Figure 5.52 shows the result of loss, accuracy, validation loss and validation accuracy for each iteration of model training. The results show that the model training stops at 26th iteration with loss (0.0956), accuracy (0.9711), validation loss (0.1278) and validation accuracy (0.9688) before reaching the maximum 50th iteration. This is because the early stopping is monitoring the validation loss during model training. If the validation loss remains stagnant or increases for five consecutive epochs, then the training process will terminate to prevent overfitting. Early stopping will also automatically restore the model's weight to the best observed point that achieved the best validation loss during training.

Subsequently, a graph is plot as shown in Figure 5.53. The graph is plotted on the changes of train loss, train accuracy, validation loss and validation accuracy over the time during model training. The graph shows that the training loss and validation loss decrease gradually from 1st epoch until around 10th epoch. Then starting from 11th epoch until 26th epoch, the training loss and validation loss remains stagnant and slightly fluctuate. Whereas for training accuracy and validation accuracy, the result shows that they increase gradually from 1st to around 10th epoch and remain stagnant from 11th epoch to 26th epoch.

Lastly, we compute the accuracy, loss and RMSE of model training. The result is shown at Figure 5.54, with accuracy at 0.9828392863273621, loss at 0.06129278615117073 and RMSE at 0.08665741. We also compute the confusion matrix, as shown in Figure 5.55.

The complete code for model training and evaluation is shown in Appendix.

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow import keras
from tensorflow import stack
from keras.models import Sequential
from keras.layers import Dense, GlobalAveragePooling1D, BatchNormalization,
MaxPool1D, Reshape, Activation, Dropout
from keras.layers import LSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.optimizers import Adam
import matplotlib.pyplot as plt
```

Figure 5.48: Libraries or packages used.

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 32)	4992
dropout (Dropout)	(None, 32)	0
dense (Dense)	(None, 4)	132

```

Total params: 5124 (20.02 KB)
Trainable params: 5124 (20.02 KB)
Non-trainable params: 0 (0.00 Byte)

```

Figure 5.49: LSTM model architecture.

```

Epoch 1/50
769/769 [=====] - 19s 9ms/step - loss: 0.7869 - accuracy: 0.6704 - val_loss: 0.5030 - val_accuracy: 0.7987
Epoch 2/50
769/769 [=====] - 6s 8ms/step - loss: 0.5149 - accuracy: 0.8059 - val_loss: 0.3864 - val_accuracy: 0.8481
Epoch 3/50
769/769 [=====] - 5s 6ms/step - loss: 0.5878 - accuracy: 0.7493 - val_loss: 0.3904 - val_accuracy: 0.8727
Epoch 4/50
769/769 [=====] - 4s 6ms/step - loss: 0.3392 - accuracy: 0.8784 - val_loss: 0.2496 - val_accuracy: 0.9156
Epoch 5/50
769/769 [=====] - 4s 5ms/step - loss: 0.3054 - accuracy: 0.8904 - val_loss: 0.2871 - val_accuracy: 0.8909
Epoch 6/50
769/769 [=====] - 5s 6ms/step - loss: 0.2426 - accuracy: 0.9158 - val_loss: 0.1671 - val_accuracy: 0.9481
Epoch 7/50
769/769 [=====] - 4s 6ms/step - loss: 0.2479 - accuracy: 0.9129 - val_loss: 0.1818 - val_accuracy: 0.9519
Epoch 8/50
769/769 [=====] - 5s 7ms/step - loss: 0.2004 - accuracy: 0.9340 - val_loss: 0.1627 - val_accuracy: 0.9442
Epoch 9/50
769/769 [=====] - 4s 6ms/step - loss: 0.1768 - accuracy: 0.9451 - val_loss: 0.1689 - val_accuracy: 0.9429
Epoch 10/50
769/769 [=====] - 4s 6ms/step - loss: 0.1713 - accuracy: 0.9473 - val_loss: 0.1289 - val_accuracy: 0.9584
Epoch 11/50
769/769 [=====] - 5s 6ms/step - loss: 0.1513 - accuracy: 0.9522 - val_loss: 0.1285 - val_accuracy: 0.9649
Epoch 12/50
769/769 [=====] - 4s 6ms/step - loss: 0.1605 - accuracy: 0.9496 - val_loss: 0.1991 - val_accuracy: 0.9377
Epoch 13/50
769/769 [=====] - 4s 6ms/step - loss: 0.1743 - accuracy: 0.9405 - val_loss: 0.1793 - val_accuracy: 0.9377
Epoch 14/50
769/769 [=====] - 5s 6ms/step - loss: 0.1696 - accuracy: 0.9473 - val_loss: 0.2074 - val_accuracy: 0.9247
Epoch 15/50
769/769 [=====] - 4s 6ms/step - loss: 0.1360 - accuracy: 0.9568 - val_loss: 0.1164 - val_accuracy: 0.9662
Epoch 16/50
769/769 [=====] - 5s 6ms/step - loss: 0.1220 - accuracy: 0.9597 - val_loss: 0.1185 - val_accuracy: 0.9636
Epoch 17/50
769/769 [=====] - 5s 6ms/step - loss: 0.1240 - accuracy: 0.9587 - val_loss: 0.1826 - val_accuracy: 0.9429

Epoch 18/50
769/769 [=====] - 4s 6ms/step - loss: 0.1270 - accuracy: 0.9587 - val_loss: 0.1165 - val_accuracy: 0.9675
Epoch 19/50
769/769 [=====] - 5s 7ms/step - loss: 0.1101 - accuracy: 0.9636 - val_loss: 0.1135 - val_accuracy: 0.9727
Epoch 20/50
769/769 [=====] - 4s 6ms/step - loss: 0.1635 - accuracy: 0.9467 - val_loss: 0.1370 - val_accuracy: 0.9623
Epoch 21/50
769/769 [=====] - 4s 6ms/step - loss: 0.1115 - accuracy: 0.9652 - val_loss: 0.0920 - val_accuracy: 0.9727
Epoch 22/50
769/769 [=====] - 5s 7ms/step - loss: 0.1094 - accuracy: 0.9629 - val_loss: 0.1480 - val_accuracy: 0.9519
Epoch 23/50
769/769 [=====] - 5s 6ms/step - loss: 0.1057 - accuracy: 0.9629 - val_loss: 0.1178 - val_accuracy: 0.9636
Epoch 24/50
769/769 [=====] - 5s 6ms/step - loss: 0.0809 - accuracy: 0.9704 - val_loss: 0.1689 - val_accuracy: 0.9584
Epoch 25/50
769/769 [=====] - 5s 6ms/step - loss: 0.0779 - accuracy: 0.9743 - val_loss: 0.2457 - val_accuracy: 0.9260
Epoch 26/50
769/769 [=====] - 4s 6ms/step - loss: 0.0956 - accuracy: 0.9711 - val_loss: 0.1278 - val_accuracy: 0.9688

```

Figure 5.50: Results of model training.



Figure 5.51: A graph plotted on the changes of train loss, train accuracy, validation loss and validation accuracy.

```

962/962 [=====] - 4s 4ms/step - loss: 0.0613 - accuracy: 0.9828
962/962 [=====] - 3s 2ms/step
Train Accuracy: 0.9828392863273621
Train Loss: 0.06129278615117073
Train RMSE: 0.08665741
    
```

Figure 5.52: Accuracy, loss and RMSE of training.

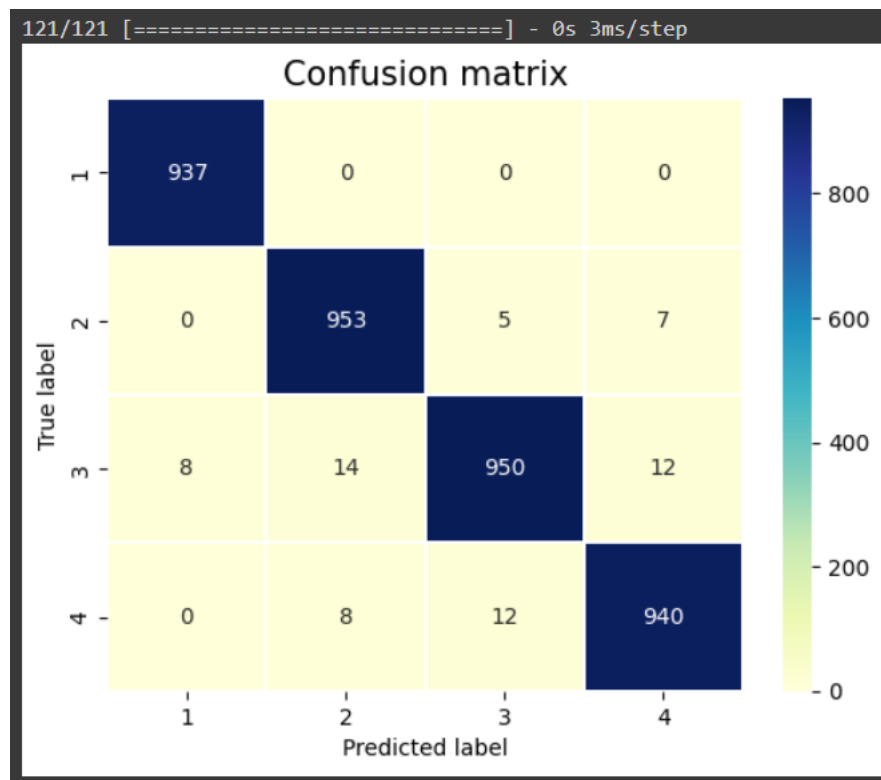


Figure 5.53: Confusion matrix.

5.4.6 Hyperparameter Tuning

Before implementing hyperparameter tuning, we need to import necessary libraries. The libraries used include *scikeras*, *sklearn* and *tensorflow*.

Firstly, we define a *KerasClassifier* as a wrapper for our Keras model by using *KerasClassifier(build_fn=create_lstm_model, verbose=0, batch_size=4, dr=0.2, epochs=30, lr=0.001)*. Next, we define the *param_grid*, which is the search space for the hyperparameters that the model is going to explore. Here, we perform hyperparameter tuning for batch size, dropout rate, learning rate and epochs. After that, we initialize the early stopping just like what we did during model training to monitor validation loss and prevent overfitting. The code involved is *early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)*. Then, we start to implement hyperparameter tuning using *GridSearchCV*, whereby we use 3-fold cross validation and we split the training dataset into 80% of training data and 20% of validation data. The code being used is *GridSearchCV(estimator=keras_model, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1)* and *grid_result = grid.fit(X_train, y_train, validation_split=0.2, callbacks=[early_stopping])*.

Finally, we access the best parameters and best accuracy using `grid_result.best_params_` and `grid_result.best_score_`. The result shows that the best combination of hyperparameters is batch size of 16, dropout rate of 0.2, epochs of 50 and learning rate 0.0025. The accuracy achieved is 0.96.

The complete code for hyperparameter tuning is shown in Appendix.

```
from sklearn.model_selection import GridSearchCV
import scikeras
from scikeras.wrappers import KerasClassifier
from tensorflow import keras
from keras.callbacks import EarlyStopping
```

Figure 5.54: Libraries or packages used.

```
param_grid = {
    'batch_size': [4, 16, 32],
    'dr': [0.2, 0.3, 0.4],
    'epochs': [30, 40, 50],
    'lr': [0.001, 0.0025]
}
```

Figure 5.55: param_grid.

```
Best Parameters: {'batch_size': 16, 'dr': 0.2, 'epochs': 50, 'lr': 0.0025}
Best Accuracy: 0.9648985959438378
```

Figure 5.56: Best combination of hyperparameters and accuracy.

5.4.7 Model Testing

Firstly, we re-train the LSTM model by using the combination of hyperparameters identified in previous step and on the training dataset. Then, we performed testing by using the dedicated testing. The codes involved are `tuned_model.evaluate(X_test, y_test, batch_size = batch_size, verbose = 1)` and `tuned_model.predict(X_test)`. Then, we compute the accuracy, loss, and RMSE, as shown in Figure 5.60. The result shows that after hyperparameter tuning, the model can achieve an accuracy of 0.9656964540481567, loss of 0.11889327317476273 and RMSE of 0.11827322. We also compute confusion matrix, as shown in Figure 5.61.

The complete code for model testing is shown in Appendix.

```

61/61 [=====] - 0s 4ms/step - loss: 0.1189 - accuracy: 0.9657
31/31 [=====] - 0s 2ms/step
Test Accuracy: 0.9656964540481567
Test Loss: 0.11889327317476273
Test RMSE: 0.11827322

```

Figure 5.57: Accuracy, loss and RMSE of testing.

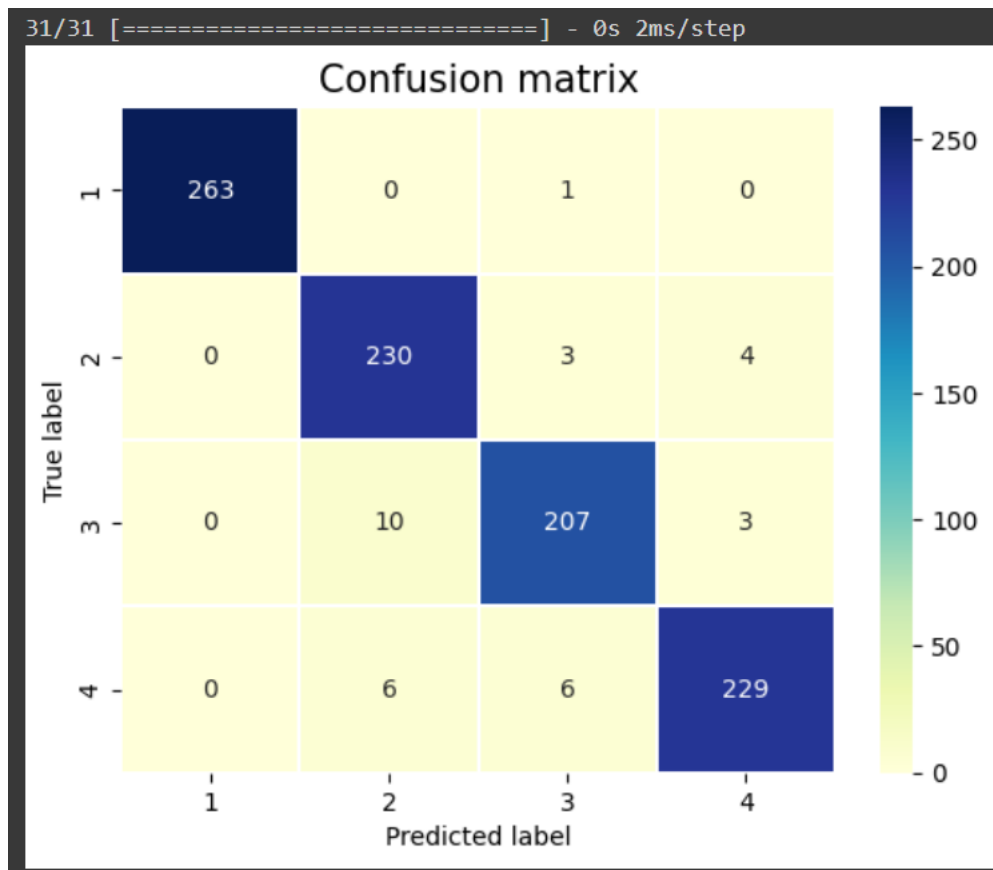


Figure 5.58: Confusion matrix.

5.6 Implementation Issues and Challenges

The challenging part during development of the project is sensor data synchronization. This project involves collecting sensor signals from accelerometer and gyroscope. Hence, I need to ensure that the sensor data recorded are synchronized correctly.

Next, another challenging part is ensuring the reliable Bluetooth connectivity between SensorData and SensorDataLogger. Bluetooth connectivity between SensorData and SensorDataLogger is important because the applications send and receive instructions via Bluetooth. I need to determine the furthest distance that the Bluetooth connectivity can hold between the devices, to ensure the reliability of the dataset being created later.

Besides, another challenging part is technical issues with the Android Studio while developing mobile applications. This is due to unfamiliarity with the software interface and functionalities and can be overcome by searching solutions from Internet.

Additionally, another challenge while performing human activity recognition research is the technical issues with Google Colab. Sometimes there would be problems like incompatible versions of the libraries, missing packages, missing modules, etc. This challenge can be overcome by searching solutions and opinions from Internet.

5.7 Concluding Remark

In the nutshell, we have shown here what are the hardware, software and cloud component being used in this project. The hardware used are laptop and two Android devices. The software used are Google Colab, Android Studio SensorData and SensorDataLogger. The cloud used is GCS. We have also stated their specifications and functions in the proposed system and how to properly configure them before usage. Laptop is used for the development of mobile applications using Android Studio and HAR related operations using Google Colab. Two Android devices are installed with SensorData and SensorDataLogger to implement the data engineering system proposed. It is important to make configurations and settings before we start to perform data engineering and HAR related operations. This helps to prevent common errors that might happen during system implementation and reveal some errors that we might overlook. For instance, the applications need to be given permissions to use Bluetooth and storage and turn on Bluetooth before data collection.

The codes and partial results for the HAR related operations, including data exploration, data segmentation, model training and evaluation, hyperparameter tuning and model testing are described in detail here and attached in Appendix. This helps other researchers who are referencing this project to have better understanding and are able to replicate and further enhance this project. Finally, the implementation issues and challenges serve as a reminder for other researchers who are referring to this project to avoid making the same mistake.

Chapter 6

System Evaluation and Discussion

6.1 Model Testing and Performance Metrics

In this project, model testing is implemented using the dedicated testing datasets and the hyperparameter combination obtained from hyperparameter tuning. Later, the performance of model testing is evaluated using accuracy, loss, RMSE, and confusion matrix.

Firstly, we re-train the LSTM model by using the combinations of hyperparameters that achieve the best performance during hyperparameter tuning. The hyperparameters used involve a batch size of 16, epochs of 50, dropout rate of 0.2 and learning rate of 0.0025. We also implement early stopping and dropout techniques to prevent overfitting. Other hyperparameters remain the same.

```
tuned_model = tf.keras.models.Sequential()
tuned_model.add(LSTM(units = 32, input_shape = (X_train.shape[1], X_train.shape[2])))
tuned_model.add(Dropout(0.2))
tuned_model.add(Dense(y_train.shape[1], activation = 'softmax'))
optimizer = Adam(learning_rate=0.0025)
tuned_model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
batch_size = 16
n_epochs = 50
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
history = tuned_model.fit(X_train, y_train, epochs = n_epochs, validation_split = 0.20,
callbacks=[early_stopping], batch_size = batch_size, verbose = 1)
```

After that, we start to perform testing. The accuracy and loss are obtained by first using the tuned models to make evaluations on testing datasets. In this project, our testing dataset consists of shape (962, 50, 6). The accuracy represents the proportion of correctly classified samples out of the total samples in the dataset. The loss represents how well the predictions match the true labels. Additionally, RMSE is computed by first using the tuned model to make predictions on testing dataset and store the predictions. Then, we use *mean_squared_error()* to calculate MSE and RMSE is obtained by applying square root on MSE. RMSE represents the magnitude of predictions errors. Lower value of RMSE indicates better performance.

Confusion matrix is also computed.

```
# loss and accuracy
loss_tuned, accuracy_tuned = tuned_model.evaluate(X_test, y_test, batch_size =
batch_size, verbose = 1)

# RMSE
y_pred_tuned = tuned_model.predict(X_test)
mse_tuned = mean_squared_error(y_pred_tuned, y_test)
rmse_tuned = np.sqrt(mse_tuned)

#confusion matrix
predictions = tuned_model.predict(X_test)
class_labels = ['1', '2', '3', '4']
max_test = np.argmax(y_test, axis=1)
max_predictions = np.argmax(predictions, axis=1)
confusion_matrix = metrics.confusion_matrix(max_test, max_predictions)
sns.heatmap(confusion_matrix, xticklabels = class_labels, yticklabels =
class_labels, annot = True, linewidths = 0.1, fmt='d', cmap = 'YlGnBu')
plt.title("Confusion matrix", fontsize = 15)
plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.show()
```

6.2 Testing Setup and Result

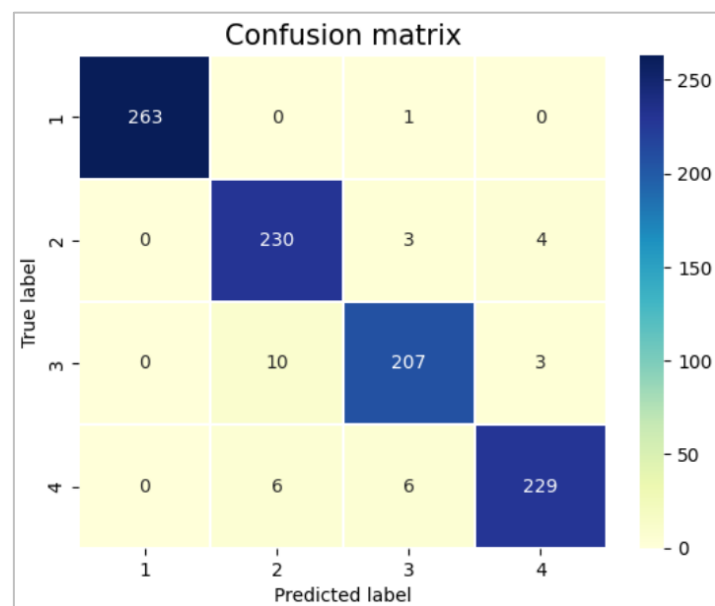


Figure 6.1: Confusion matrix of model testing.

```

61/61 [=====] - 0s 4ms/step - loss: 0.1189 - accuracy: 0.9657
31/31 [=====] - 0s 2ms/step
Test Accuracy: 0.9656964540481567
Test Loss: 0.11889327317476273
Test RMSE: 0.11827322

```

Figure 6.2: Accuracy, loss and RMSE of model testing.

Firstly, we re-train the LSTM model by using the combinations of hyperparameters that achieve the best performance during hyperparameter tuning. Then, we performed the testing using the tuned model and testing dataset.

After testing, we compute accuracy, loss and RMSE and the result is shown in Figure 6.2. The result shows that the tuned model can achieve an accuracy of 96.57% when being used on new, unseen dataset. This proves the generalization capabilities of LSTM model trained using dataset created from the proposed data engineering system. In essence, this validates the efficiency and capability of the proposed data engineering system in producing high-quality datasets and conducive to robust model generalization. Overfitting problem is not expected to occur since we are implementing early stopping and dropout technique.

Furthermore, a confusion matrix is also computed, as shown in Figure 6.1. According to the result, the LSTM model predicted quite well for each activity, including standing (1), walking (2), climbing downstairs (3) and climbing upstairs (4). However, the model sometimes confuses between walking (2), climbing downstairs (3) and climbing downstairs (4). Overall, the model achieves good accuracy and generalization capability.

6.3 Project Challenges

One of the challenges encountered while implementing this project is deficiencies in knowledge and experience. This projects involve data engineering and human activity recognition which requires knowledge about development of mobile applications, machine learning pipeline and techniques, deep learning models architecture, etc. Prior starting this project, only limited opportunities are available to get involved in developing mobile applications, machine learning pipeline and deep learning process, and most of them are basic practices for beginners. Consequently, the successful execution of this project necessitated a substantial investment of time and effort into intensive research and investigative efforts to have a profound understanding of the underlying concepts and technologies essential for its realization.

Besides, another project challenge includes limited number of data subjects involved in this project. Deep learning model training typically necessitates access to large datasets for effective generalization. However, since only five data subjects are involved, the resulting dataset size is quite small. This raises concerns about the potential susceptibility to overfitting of model. To mitigate this issue, dropout techniques and early stopping techniques have been incorporated during model training, hyperparameter tuning and model testing to prevent overfitting.

Furthermore, another project challenge is constraint imposed by limited computational resources. In this project, all the HAR related operations are implemented in Google Colab. One of the tasks in HAR requires substantial number of computational resources, which is hyperparameter tuning. A substantial investment of time, which is about 2 hours are still required for hyperparameter tuning although the hardware specifications had been changed to GPU T4. Extra payment is needed if more resources are needed.

6.4 Objectives Evaluation

Index	Objectives	States
1	To address the issue of inefficient data engineering infrastructure being used in current Human Activity Recognition (HAR) research.	Completed
2	To develop a comprehensive, efficient, and user-friendly data engineering system for data collection, annotation, and storage.	Completed
3	To design and develop mobile applications, SensorData and SensorDataLogger.	Completed
4	To evaluate the efficiency of proposed data engineering system on HAR algorithm.	Completed

Table 6.1: Objectives evaluation.

The Table 6.1 has shown the overall objectives evaluation of this project.

We have successfully addressed the issue of inefficient data engineering infrastructure being used in current Human Activity Recognition research by developing a comprehensive, efficient, and user-friendly data engineering system for data collection, annotation, and storage. To implement this data engineering system, we have designed and developed two mobile

applications, SensorData and SensorDataLogger. The applications are connected using Bluetooth to send and receive instructions. Users can give instructions via SensorData while SensorDataLogger is responsible to receive and execute instructions. In SensorData, users can start the data recording process by clicking at the activity buttons created and set the durations they want to collect data for. After users click the ‘START’ button, SensorData sends the instructions to SensorDataLogger to start data recording process and starts a timer. Similarly, when SensorDataLogger receives the instructions, a timer is also started. When the timer stops, SensorData will automatically send instructions to SensorDataLogger to stop the recording process. The log files created in SensorDataLogger are automatically annotated with the ID of the activity. When users click the ‘Save Log File’ button, SensorData will send instructions to SensorDataLogger to save the log files with the specific filename, which is the ID of the activity. When users click ‘Upload Log File’, SensorData will send instructions to SensorDataLogger to upload log files to Google Cloud Storage bucket.

On the other hand, the applications offer user-friendly interface and intuitive functionalities so that non-technical researchers can easily utilize the system proposed in their research. The proposed data engineering system is said to be comprehensive since the system proposed covers all the components required to create a dataset needed for Human Activity Recognition, including data collection, annotation, and storage. Additionally, the proposed system is said to be efficient as it requires least human effort to create a dataset of high-quality and is able to achieve good accuracy and generalization. For instance, it includes the features of automated annotation, eliminating the need for manual annotations, which requires a lot of effort. Besides, the log files can be directly uploaded to cloud server, which is more easier to be extracted for further processing.

Next, we have also successfully evaluated the efficiency of the proposed data engineering system on HAR algorithm by looking at the performance of the HAR algorithms trained using the dataset collected via data engineering system proposed. This assessment encompasses performance metrics such as accuracy and the generalization capabilities of HAR algorithms. In other words, if the dataset created using the proposed data engineering system can achieve good accuracy and generalization during training and testing, it means that the proposed data engineering system is effective. After data collection, we have performed HAR related operations such as dataset creation, data exploration and data segmentation. Then, we build and train a simple and unidirectional LSTM model, perform hyperparameter tuning and model testing. During model training, we have successfully achieved an accuracy of 0.9828,

loss of 0.0613 and RMSE of 0.0867. Besides, during model testing using dedicated testing dataset and tuned model, we obtained an accuracy of 0.9657, loss of 0.1189 and RMSE of 0.1183. This statistics validates the efficiency of the proposed data engineering system. It also shows the practicality and feasibility of integrating the proposed data engineering system into real-world HAR applications as it can improve the scalability, reproducibility, and comparability of project without compromising recognition accuracy or efficiency.

6.5 Concluding Remark

In a nutshell, this chapter has discussed the details regarding the model testing and performance metrics used. We are using tuned model and dedicated testing dataset for model testing. The performance metrics used include accuracy, loss, RMSE and confusion matrix. Besides, we also show and analyze the results of model testing based on accuracy, loss, RMSE, and confusion matrix. Our project validates the efficiency of the proposed data engineering system in producing high-quality dataset. Furthermore, we have talked about some of the challenges or constraints faced while implementing this project and proposed our solutions to tackle the challenges. This helps to give clear understanding to other researchers who are referencing this project. Last but not least, an objective evaluation process is performed to show the current progress of this project and how far it achieves.

Chapter 7

Conclusion and Recommendations

7.1 Conclusion

In recent years, HAR research has primarily focused on refining algorithms and techniques, often overshadowing the critical need for efficient data engineering systems in terms of sensor data collection, annotation, and storage. This results in the use of incomplete and inefficient data engineering systems, which brings negative effects to the accuracy, efficiency, scalability, and reproducibility of HAR research.

This project addresses this vital issue of inefficient data engineering infrastructure by introducing an efficient, comprehensive, and user-friendly data engineering system for data collection, annotation, and storage. To implement the data engineering system proposed, two mobile applications, SensorData and SensorDataLogger with user-friendly interfaces and intuitive functionalities are developed. The applications are utilized to collect raw sensor signals. The raw signals went through some preprocessing and a dataset is created. The dataset is later used to train a unidirectional LSTM model with 32 neurons. To prevent overfitting, early stopping monitoring validation loss and dropout technique using dropout rate of 0.3 are applied during model training. The training accuracy obtained is 0.9711.

Hyperparameter tuning is then implemented and identify the best combinations, which is [batch size: 16, dropout rate: 0.2, learning rate: 0.0025, epochs: 50]. These hyperparameters are used to re-train a LSTM model and this model is used to perform model testing.

In evaluating the efficiency of our proposed data engineering system, we compute accuracy, loss, RMSE, and the confusion matrix. This project proves that the proposed data engineering system is efficient, which is able to achieve an accuracy rate of 96.57% during model testing.

In conclusion, this project represents a significant stride towards the advancement of HAR research. By addressing the critical need for efficient data engineering in data collection, annotation, and storage, we have contributed to improving accuracy, reliability, scalability, and reproducibility in HAR research. We offer a comprehensive and effective toolkit for data engineering, aimed to further enhance the performance of the field.

7.2 Recommendation

Bachelor of Computer Science (Honours)
Faculty of Information and Communication Technology (Kampar Campus), UTAR

This section serves as a recommendation to other researchers who are referencing and citing this research. These recommendations aim to continue the current work and further enhance the performance of smartphone based HAR research.

One of the recommendations is to increase the size of the dataset used in future studies, by adding number of samples and variations of activities being collected. Currently, this project only collects data from five data subjects and for four activities, resulting in a small dataset. By incorporating larger and more diverse datasets, researchers can capture a broader range of activities and user behaviors, which can lead to more robust and generalizable HAR models.

Next, another recommendation is to diversify the sources of data by collecting signals from different types of sensor. This project only collects data from smartphone embedded accelerometer and gyroscope, which is sufficient for human daily activities like walking, standing, climbing upstairs, etc. The inclusion of additional sensor such as magnetometer, GPS sensor, proximity sensor, ambient light sensor, etc. can provide a more comprehensive understanding of human behavior. Researcher should explore the integration of data from different sensors to capture a more detailed and holistic view of users' activities and environments. This helps to improve the performance and versatility of activity recognition system, making them applicable to a broader range of scenarios or use cases.

Furthermore, we can explore other deep learning architecture for activity recognition purpose. While this project is employing a simple and unidirectional LSTM model with 32 neurons, other deep learning models are worth explored as well. Researchers can consider bidirectional LSTM model, CNN, or more complex architectures that combine multiple types of neural networks such as LSTM-CNN or 4-layered LSTM-CNN. This helps to enhance recognition accuracy and broaden the scope of smartphone based HAR.

REFERENCES

- [1] Y. K. Han and Y. B. Choi, "Human Action Recognition based on LSTM Model using Smartphone Sensor," in *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, Jul. 2019, pp. 748-750, doi: 10.1109/ICUFN.2019.8806065.
- [2] M. Ermes, J. PÄrkkÄ, J. MÄntyjÄrvi and I. Korhonen, "Detection of Daily Activities and Sports With Wearable Sensors in Controlled and Uncontrolled Conditions," in *IEEE Transactions on Information Technology in Biomedicine*, vol. 12, no. 1, pp. 20-26, Jan. 2008, doi: 10.1109/TITB.2007.899496.
- [3] E. Bulbul, A. Cetin and I. A. Dogru, "Human Activity Recognition Using Smartphones," in *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Oct. 2018, pp. 1-6, doi: 10.1109/ISMSIT.2018.8567275.
- [4] A. Bayat, M. Pomplun and D. A. Tran, "A Study on Human Activity Recognition Using Accelerometer Data from Smartphones," in *Procedia Computer Science*, vol. 34, pp. 450-457, 2014, doi: <https://doi.org/10.1016/j.procs.2014.07.009>.
- [5] F. Demrozi, G. Pravadelli, A. Bihorac and P. Rashidi, "Human Activity Recognition Using Inertial, Physiological and Environmental Sensors: A Comprehensive Survey," in *IEEE Access*, vol. 8, pp. 210816-210836, Nov. 2020, doi: 10.1109/ACCESS.2020.3037715.
- [6] A. K. Muhammad Masum, A. Barua, E. H. Bahadur, M. R. Alam, M. Akib Uz Zaman Chowdhury and M. S. Alam, "Human Activity Recognition Using Multiple Smartphone Sensors," in *2018 International Conference on Innovations in Science, Engineering and Technology (ICISSET)*, Oct. 2018, pp. 468-473, doi: 10.1109/ICISSET.2018.8745628.
- [7] M. Alema Khatun and M. Abu Yousuf, "Human Activity Recognition Using Smartphone Sensor Based on Selective Classifiers," in *2020 2nd International Conference on Sustainable Technologies for Industry 4.0 (STI)*, Dec. 2020, pp. 1-6, doi: 10.1109/STI50764.2020.9350486.

- [8] J. Manjarrés, V. Russo, J. Peñaranda and M. Pardo, “Human Activity and Heart Rate Monitoring System in a Mobile Platform,” in *2018 Congreso Internacional de Innovación y Tendencias en Ingeniería (CONIITI)*, Oct. 2018, pp. 1-6, doi: 10.1109/CONIITI.2018.8587094.
- [9] F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, and Y. Amirat, “Physical Human Activity Recognition Using Wearable Sensors,” in *Sensors*, vol. 15, no. 12, pp. 31314–31338, Dec. 2015, doi: 10.3390/s151229858.
- [10] P. William, G. R. Lanke, D. Bordoloi, A. Shrivastava, A. P. Srivastavaa and S. V. Deshmukh, “Assessment of Human Activity Recognition based on Impact of Feature Extraction Prediction Accuracy,” presented at the *2023 4th International Conference on Intelligent Engineering and Management (ICIEM)*, London, United Kingdom, May 2023.
- [11] Y. Liu, L. Nie, L. Liu and D.S. Rosenblum, “From action to activity: Sensor-based activity recognition,” in *Neurocomputing*, Mar. 2016, pp. 108-115, doi: <https://doi.org/10.1016/j.neucom.2015.08.096>.
- [12] J. G. Ponsam, S. V. J. B. Gracia, G. Geetha, K. Nimala, S. Chepuri and R. S. Rajline, “Human Activity Recognition Using LSTM Network with Dropout Technique,” presented at the *2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*, Chennai, India, December 2022.
- [13] P. P. Ariza-Colpas et al., “Human Activity Recognition Data Analysis: History, Evolutions, and New Trends,” in *Sensors 2022*, vol. 22, no. 9, pp. 3401, April 2022, doi: <https://doi.org/10.3390/s22093401>.
- [14] S. Mekruksavanich and A. Jitpattanakul, “LSTM Networks Using Smartphone Data for Sensor-Based Human Activity Recognition in Smart Homes,” in *Sensors 2021*, vol. 21, no. 5, p. 1636, Feb. 2021, doi: <https://doi.org/10.3390/s21051636>.
- [15] J. Parkka, M. Ermes, P. Korpiä, J. Mantyjarvi, J. Peltola and I. Korhonen, “Activity classification using realistic data from wearable sensors,” in *IEEE Transactions on Information*

Technology in Biomedicine, vol. 10, no. 1, pp. 119-128, Jan. 2006, doi: 10.1109/TITB.2005.856863.

[16] F. Cruciani, I. Cleland, C. Nugent, P. McCullagh, K. Synnes and J. Hallberg, “Automatic Annotation for Human Activity Recognition in Free Living Using a Smartphone,” in *Sensors*, vol. 18, no. 7, pp. 2203, Jul. 2018, doi: 10.3390/s18072203.

[17] M. Shoaib, S. Bosch, O. Incel, H. Scholten, and P. Havinga, “A Survey of Online Activity Recognition Using Mobile Phones,” in *Sensors*, vol. 15, no. 1, pp. 2059-2085, Jan. 2015, doi: 10.3390/s150102059.

[18] M. Kheirhahan, S. Nair, A. Davoudi, P. Rashidi, A. A. Wanigatunga, D. B. Corbett, T. Mendoza, T. M. Manini and S. Ranka, “A smartwatch-based framework for real-time and online assessment and mobility monitoring,” in *Journal of Biomedical Informatics*, vol. 89, pp. 19-40, Jan. 2019, doi: <https://doi.org/10.1016/j.jbi.2018.11.003>.

[19] F. Shahmohammadi, A. Hosseini, C. E. King and M. Sarrafzadeh, “Smartwatch Based Activity Recognition Using Active Learning,” in *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE), Philadelphia*, pp. 321-329, Jul. 2017, doi: 10.1109/CHASE.2017.115.

[20] R. San-Segundo, H. Blunck, J. Moreno-Pimentel, A. Stisen and M. Gil-Martín, “Robust Human Activity Recognition using smartwatches and smartphones,” in *Engineering Applications of Artificial Intelligence*, vol. 72, pp. 190-202, Jun. 2018, doi: <https://doi.org/10.1016/j.engappai.2018.04.002>.

[21] I. D. Luptáková, M. Kubovčák and J. Pospíchal, “Wearable Sensor-Based Human Activity Recognition with Transformer Model,” in *Sensors 2022*, vol. 22, no. 5, pp. 1911, March 2022, doi: <https://doi.org/10.3390/s22051911>.

[22] I. Cleland, M. Han, C. Nugent, H. Lee, S. McClean, S. Zhang and S. Lee, “Evaluation of Prompted Annotation of Activity Data Recorded from a Smart Phone,” in *Sensors 2014*, vol. 14, no. 9, pp. 15861-15879, Aug. 2014, doi: 10.3390/s140915861.

[23] S. Mekruksavanich and A. Jitpattanakul, "A Comparative Study of Deep Learning Robustness for Sensor-based Human Activity Recognition," presented at the *2023 46th International Conference on Telecommunications and Signal Processing (TSP)*, Prague, Czech Republic, July 2023.

[24] S. Harada, K. Patel, J. A. Landay and T. S. Saponas, "VoiceLabel: using speech to label mobile sensor data," presented at the *Proceedings of the 10th International Conference on Multimodal Interfaces, ICMI 2008, Chania, Crete, Greece, October 20-22, 2008*, Oct. 2008.

[25] S. Das, L. Green, B. Perez and M. J. Murphy, "Detecting User Activities using the Accelerometer on Android Smartphones," July 2010.

[26] B. Moradi, M. Aghapour and A. Shirbandi, "Compare of Machine Learning and Deep Learning Approaches for Human Activity Recognition," presented at the *2022 30th International Conference on Electrical Engineering (ICEE)*, Tehran, Iran, Islamic Republic of, May 2022.

APPENDIX

Configurations & Libraries

```
# import libraries
from google.cloud import storage
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import gcsfs
import warnings
warnings.filterwarnings('ignore')
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.base import clone
import tensorflow as tf
from tensorflow import keras
from tensorflow import stack
import scikeras
from scikeras.wrappers import KerasClassifier
from keras.models import Sequential
```

```

from keras.layers import Dense, GlobalAveragePooling1D, BatchNormalization, MaxPool1D,
Reshape, Activation, Dropout
from keras.layers import LSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.optimizers import Adam
from scipy import stats
from scipy.stats import mode
from google.colab import auth

# GCS authentication
auth.authenticate_user()

```

Dataset Creation Coding

```

# get list of filepaths from google cloud bucket
csv_file_paths = []

def get_csv_files(project_id="sensordatalogger-logfiles"):
    storage_client = storage.Client(project=project_id)
    bucket = storage.Bucket(storage_client, 'sensordatalogger-logfiles.appspot.com')
    str_folder_name_on_gcs = 'CSVFile/'
    blobs = bucket.list_blobs(prefix=str_folder_name_on_gcs)
    print("Blob name:")
    for blob in blobs:
        csv_file_paths.append(blob.name)

get_csv_files()
print("List of csv files:")
print(csv_file_paths)

# format file name
for i in range(len(csv_file_paths)):
    csv_file_paths[i] = "gs://sensordatalogger-logfiles.appspot.com/" + csv_file_paths[i]
    print(csv_file_paths[i])

```

```

# Convert CSV file to Parquet file
parquet_dir = "gs://sensordatalogger-logfiles.appspot.com/ParquetFile/"

def convert_csv_to_parquet(input_file_path, output_file_path):
    csv_df = pd.read_csv(input_file_path, names=['sensor', 'x', 'y', 'z', 'timestamp', 'label'])
    csv_df.to_parquet(output_file_path)

for i in range(len(csv_file_paths)):
    filename = csv_file_paths[i].split('/')[-1]
    filename = filename.replace('csv', 'parquet')
    parquet_file_name = parquet_dir + filename
    convert_csv_to_parquet(csv_file_paths[i], parquet_file_name)

# Import Parquet file
# Get list of file paths from google cloud bucket
parquet_file_paths = []

def get_parquet_files(project_id="sensordatalogger-logfiles"):
    storage_client = storage.Client(project=project_id)
    bucket = storage.Bucket(storage_client, 'sensordatalogger-logfiles.appspot.com')
    str_folder_name_on_gcs = 'ParquetFile/'
    blobs = bucket.list_blobs(prefix=str_folder_name_on_gcs)
    print("Blob name:")
    for blob in blobs:
        parquet_file_paths.append(blob.name)

get_parquet_files()
print("List of parquet files:")
print(parquet_file_paths)

# format file name
for i in range(len(parquet_file_paths)):

```

```

    parquet_file_paths[i] = "gs://sensordatalogger-logfiles.appspot.com/" +
parquet_file_paths[i]
    print(parquet_file_paths[i])

# Combine log files
preprocessed_parquet_dir =
"gs://sensordataloggerlogfiles.appspot.com/PreprocessedParquetFile/"

for i in range(len(parquet_file_paths)):
    preprocessed_df = pd.read_parquet(parquet_file_paths[i], engine='auto')
    parts = parquet_file_paths[i].split('/')
    preprocessed_parquet_file_path = preprocessed_parquet_dir + parts[-1]

    # drop null values
    preprocessed_df.dropna(inplace=True)

    # drop the rows where timestamp is 0
    preprocessed_df = preprocessed_df[preprocessed_df['timestamp'] != 0]

    index = 0

    # split accelerometer and gyroscope data
    for r in range(preprocessed_df.shape[0]):
        if preprocessed_df['sensor'].iloc[r] == 'GYRO':
            index = r
            break;

    signals_acce = preprocessed_df.values[:index, 1:6]
    signals_gyro = preprocessed_df.values[index:, 1:6]

    acce_row = signals_acce.shape[0]
    gyro_row = signals_gyro.shape[0]
    keep_row = min(acce_row, gyro_row)

```

```

signals_acce = signals_acce[:keep_row]
signals_gyro = signals_gyro[:keep_row]

acce_raw_df = pd.DataFrame(signals_acce, columns=['x', 'y', 'z', 'ts', 'label'])
gyro_raw_df = pd.DataFrame(signals_gyro, columns=['x2', 'y2', 'z2', 'ts2', 'label2'])

# sort in ascending order
acce_raw_df = acce_raw_df.sort_values(by = ['ts'], ignore_index=True)
gyro_raw_df = gyro_raw_df.sort_values(by = ['ts2'], ignore_index=True)

# combine and arrange side by side
final_df = pd.concat([acce_raw_df, gyro_raw_df], axis=1, join='outer')
final_df.to_parquet(preprocessed_parquet_file_path)

# Get preprocessed file path
preprocessed_parquet_file_paths = []

def get_preprocessed_parquet_files(project_id="sensordatalogger-logfiles"):
    storage_client = storage.Client(project=project_id)
    bucket = storage.Bucket(storage_client, 'sensordatalogger-logfiles.appspot.com')
    str_folder_name_on_gcs = 'PreprocessedParquetFile/'
    blobs = bucket.list_blobs(prefix=str_folder_name_on_gcs)
    print("Blob name:")
    for blob in blobs:
        preprocessed_parquet_file_paths.append(blob.name)

get_preprocessed_parquet_files()

# Format file name
for i in range(len(preprocessed_parquet_file_paths)):
    preprocessed_parquet_file_paths[i] = "gs://sensordatalogger-logfiles.appspot.com/" +
preprocessed_parquet_file_paths[i]

```

```

print(preprocessed_parquet_file_paths[i]

# Combine file paths
same_activity_file_paths = {}

for i in range(len(preprocessed_parquet_file_paths)):
    p1 = preprocessed_parquet_file_paths[i].split('/')[-1]
    p2 = p1.split('_')[1]

    if p2 in same_activity_file_paths:
        same_activity_file_paths[p2].append(preprocessed_parquet_file_paths[i])
    else:
        same_activity_file_paths[p2] = [preprocessed_parquet_file_paths[i]]

print(same_activity_file_paths)

# Combine files for each activity
activity_file_dir = 'gs://sensordatalogger-logfiles.appspot.com/ActivityParquetFile/'
column_names = ['x', 'y', 'z', 'ts', 'label', 'x2', 'y2', 'z2', 'ts2', 'label2']

for key in same_activity_file_paths:
    activity_list = []
    same_activity_df = pd.DataFrame()

    print(key)

    # change data to numpy and append to activity_list
    for l in range(len(same_activity_file_paths[key])):
        activity_df = pd.read_parquet(same_activity_file_paths[key][l], engine='auto')
        activity_np = activity_df.to_numpy()
        activity_list.append(activity_np)

    if l == 0:

```

```

parts = same_activity_file_paths[key][1].split('/')
parts2 = parts[-1].split('_')[0] + '_' + parts[-1].split('_')[1] + '.parquet'
activity_parquet_file_path = activity_file_dir + parts2
print(activity_parquet_file_path)

# Check if all elements in num_columns_list are the same
num_columns_list = [array.shape[1] for array in activity_list]
print(num_columns_list)

if all(num == num_columns_list[0] for num in num_columns_list):
    print("The files have similar number of columns.")
else:
    print("The files does not have similar number of columns.")

# append data to file
for n in range(len(activity_list)):
    df = pd.DataFrame(activity_list[n])
    same_activity_df = pd.concat([same_activity_df, df], ignore_index=True)

same_activity_df.columns = column_names
same_activity_df.to_parquet(activity_parquet_file_path)

#print(same_activity_df)
print(same_activity_df.shape)
print('\n')

# Get files
activity_parquet_file_paths = []

def get_activity_parquet_files(project_id="sensordatalogger-logfiles"):
    storage_client = storage.Client(project=project_id)
    bucket = storage.Bucket(storage_client, 'sensordatalogger-logfiles.appspot.com')
    str_folder_name_on_gcs = 'ActivityParquetFile/'

```

```

blobs = bucket.list_blobs(prefix=str_folder_name_on_gcs)
print("Blob name:")
for blob in blobs:
    activity_parquet_file_paths.append(blob.name)

get_activity_parquet_files()
print("List of parquet files:")
print(activity_parquet_file_paths)

# Format file name
for i in range(len(activity_parquet_file_paths)):
    activity_parquet_file_paths[i] = "gs://sensordatalogger-logfiles.appspot.com/" +
activity_parquet_file_paths[i]
    print(activity_parquet_file_paths[i])

# Combine vertically
har_df = pd.DataFrame()

for k in range(len(activity_parquet_file_paths)):
    df = pd.read_parquet(activity_parquet_file_paths[k], engine='auto')
    df.drop(['ts', 'ts2', 'label'], axis=1, inplace=True)
    df.rename(columns = {'label2':'label'}, inplace = True)
    df['label'] = df['label'].astype(int)
    har_df = pd.concat([har_df, df], axis=0, ignore_index=True)

har_df.to_parquet('gs://sensordatalogger-logfiles.appspot.com/har.parquet', engine='auto',
index=False)
print(har_df)
print(har_df.shape)

```

Data Exploration Coding

```

explore_df = pd.read_parquet("gs://sensordatalogger-logfiles.appspot.com/har.parquet",
engine='auto')

```



```

explore_df.shape
explore_df.info()
explore_df.describe()

# Check null value
explore_df.isnull().sum()*100/len(explore_df)

# Check infinity value
has_positive_infinity = np.isinf(explore_df).any().any()
has_negative_infinity = np.isneginf(explore_df).any().any()
if has_positive_infinity:
    print("The DataFrame contains positive infinity values.")
if has_negative_infinity:
    print("The DataFrame contains negative infinity values.")
if not has_positive_infinity and not has_negative_infinity:
    print("The DataFrame does not contain infinity values.")

# Class distribution
pd.crosstab(index =explore_df["label"],columns="count")

sns.set_style("whitegrid")
plt.figure(figsize = (5, 4))
sns.countplot(x = 'label', data = explore_df)
plt.title('Number of samples by activity')
plt.show()

corr_matrix=explore_df.corr()
corr_matrix
fig, ax = plt.subplots(figsize=(8,8))
sns.heatmap(data=corr_matrix, annot=True, linewidths=.5, ax=ax)

```

Data Segmentation Coding

```

# Read dataset
Bachelor of Computer Science (Honours)
Faculty of Information and Communication Technology (Kampar Campus), UTAR

```

```

lstm_df = pd.read_parquet("gs://sensordatalogger-logfiles.appspot.com/har.parquet",
engine='auto')
print(lstm_df.shape)

# windowing
n_time_steps = 50
n_features = 6
step = 25
n_classes = 4
segments = []
labels = []

for i in range(0, lstm_df.shape[0]- n_time_steps, step):

    x_acce = lstm_df['x'].values[i: i + 50]
    y_acce = lstm_df['y'].values[i: i + 50]
    z_acce = lstm_df['z'].values[i: i + 50]
    x_gyro = lstm_df['x2'].values[i: i + 50]
    y_gyro = lstm_df['y2'].values[i: i + 50]
    z_gyro = lstm_df['z2'].values[i: i + 50]
    label_mode_result = stats.mode(lstm_df['label'][i: i + 50])
    if np.isscalar(label_mode_result[0]):
        label = label_mode_result[0]
    else:
        label = label_mode_result.mode[0]

    segments.append([x_acce, y_acce, z_acce, x_gyro, y_gyro, z_gyro])
    labels.append(label)

reshaped_segments = np.asarray(segments, dtype= np.float32).reshape(-1, n_time_steps,
n_features)
labels = np.asarray(pd.get_dummies(labels), dtype = np.float32)
print(reshaped_segments.shape)

```

```

print(labels.shape)

# Split into training & testing
X_train, X_test, y_train, y_test = train_test_split(reshaped_segments, labels, test_size = 0.2,
random_state = 42)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

```

Model Training & Evaluation Coding

```

def create_lstm_model(dr=0.3, lr=0.0025):
    model = tf.keras.models.Sequential()
    model.add(LSTM(units = 32, input_shape = (X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(dr))
    model.add(Dense(y_train.shape[1], activation = 'softmax'))
    optimizer = Adam(learning_rate=lr)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

model = create_lstm_model()
model.summary()

batch_size = 4
n_epochs = 50
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
history = model.fit(X_train, y_train, epochs = n_epochs, validation_split = 0.20,
callbacks=[early_stopping], batch_size = batch_size, verbose = 1)

plt.plot(np.array(history.history['loss']), "r--", label = "Train loss")
plt.plot(np.array(history.history['accuracy']), "g--", label = "Train accuracy")
plt.plot(np.array(history.history['val_loss']), "r-", label = "Validation loss")
plt.plot(np.array(history.history['val_accuracy']), "g-", label = "Validation accuracy")

```

```

plt.title("Training session's progress over iterations")
plt.legend(loc='lower left')
plt.ylabel('Training Progress (Loss/Accuracy)')
plt.xlabel('Training Epoch')
plt.ylim(0)
plt.show()

loss, accuracy = model.evaluate(X_train, y_train, batch_size = batch_size, verbose = 1)
y_pred_train = model.predict(X_train, batch_size = batch_size, verbose=1)
mse_train = mean_squared_error(y_pred_train, y_train)
rmse_train = np.sqrt(mse_train)

print("Train Accuracy: ", accuracy)
print("Train Loss: ", loss)
print ("Train RMSE: ", rmse_train)

predictions = model.predict(X_train)
class_labels = ['1', '2', '3', '4']
max_train = np.argmax(y_train, axis=1)
max_predictions = np.argmax(predictions, axis=1)
confusion_matrix = metrics.confusion_matrix(max_train, max_predictions)
sns.heatmap(confusion_matrix, xticklabels = class_labels, yticklabels = class_labels, annot =
True, linewidths = 0.1, fmt='d', cmap = 'YlGnBu')
plt.title("Confusion matrix", fontsize = 15)
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

```

Hyperparameter Tuning Coding

```

keras_model = KerasClassifier(build_fn=create_lstm_model, verbose=0, batch_size=4,
dr=0.2, epochs=30, lr=0.001)
param_grid = {
    'batch_size': [4, 16, 32],

```

```

'dr': [0.2, 0.3, 0.4],
'epochs': [30, 40, 50],
'lr': [0.001, 0.0025]
}
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
grid = GridSearchCV(estimator=keras_model, param_grid=param_grid, cv=3,
scoring='accuracy', n_jobs=-1)
grid_result = grid.fit(X_train, y_train, validation_split=0.2, callbacks=[early_stopping])
best_params = grid_result.best_params_
best_model = grid_result.best_estimator_.model

print("Best Parameters: ", grid_result.best_params_)
print("Best Accuracy: ", grid_result.best_score_)

```

Model Testing Coding

```

# re-train model
tuned_model = tf.keras.models.Sequential()
tuned_model.add(LSTM(units = 32, input_shape = (X_train.shape[1], X_train.shape[2])))
tuned_model.add(Dropout(0.2))
tuned_model.add(Dense(y_train.shape[1], activation = 'softmax'))
optimizer = Adam(learning_rate=0.0025)
tuned_model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
tuned_model.summary()

batch_size = 16
n_epochs = 50
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
history = tuned_model.fit(X_train, y_train, epochs = n_epochs, validation_split = 0.20,
callbacks=[early_stopping], batch_size = batch_size, verbose = 1)

# Testing using best hyperparameter

```

```

loss_tuned, accuracy_tuned = tuned_model.evaluate(X_test, y_test, batch_size = batch_size,
verbose = 1)
y_pred_tuned = tuned_model.predict(X_test)
mse_tuned = mean_squared_error(y_pred_tuned, y_test)
rmse_tuned = np.sqrt(mse_tuned)

print("Test Accuracy: ", accuracy_tuned)
print("Test Loss: ", loss_tuned)
print("Test RMSE: ", rmse_tuned)

# confusion matrix
predictions = tuned_model.predict(X_test)
class_labels = ['1', '2', '3', '4']
max_test = np.argmax(y_test, axis=1)
max_predictions = np.argmax(predictions, axis=1)
confusion_matrix = metrics.confusion_matrix(max_test, max_predictions)
sns.heatmap(confusion_matrix, xticklabels = class_labels, yticklabels = class_labels, annot =
True, linewidths = 0.1, fmt='d', cmap = 'YlGnBu')
plt.title("Confusion matrix", fontsize = 15)
plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.show()

```

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	Study week no.: 3
Student Name & ID: Tee Jia Lin 2005546	
Supervisor: Ts Dr Ooi Boon Yaik	
Project Title: Human Activity Recognition Via Accelerometer and Gyro Sensors	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Review objective, background, problem statement of the research.
- Made research on machine learning / deep learning pipeline.

2. WORK TO BE DONE

- Collect data and create dataset.

3. PROBLEMS ENCOUNTERED


- Unable to decide what machine learning / deep learning model to be used.

4. SELF EVALUATION OF THE PROGRESS

- Should start earlier.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	Study week no.: 4
Student Name & ID: Tee Jia Lin 2005546	
Supervisor: Ts Dr Ooi Boon Yaik	
Project Title: Human Activity Recognition Via Accelerometer and Gyro Sensors	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Decide to use LSTM model.
- Raw dataset collected.

2. WORK TO BE DONE

- Study on LSTM model architecture and implementation.
- Preprocess dataset.

3. PROBLEMS ENCOUNTERED

- No.

4. SELF EVALUATION OF THE PROGRESS

- Progress is on track.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	Study week no.: 6
Student Name & ID: Tee Jia Lin 2005546	
Supervisor: Ts Dr Ooi Boon Yaik	
Project Title: Human Activity Recognition Via Accelerometer and Gyro Sensors	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Data preprocessing done. Dataset was created.
- Made study on LSTM architecture. Decide to use the simple unidirectional LSTM.

2. WORK TO BE DONE

- Try to implement the model training by referencing other researcher's work.

3. PROBLEMS ENCOUNTERED

- Not familiar with the machine learning process.

4. SELF EVALUATION OF THE PROGRESS

- Progress is on track.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	Study week no.: 8
Student Name & ID: Tee Jia Lin 2005546	
Supervisor: Ts Dr Ooi Boon Yaik	
Project Title: Human Activity Recognition Via Accelerometer and Gyro Sensors	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Perform data exploration on preprocessed dataset.
- Perform data segmentation on the dataset.

2. WORK TO BE DONE

- Build architecture of LSTM model.
- LSTM model training.

3. PROBLEMS ENCOUNTERED


- No.

4. SELF EVALUATION OF THE PROGRESS

- Progress is on track.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	Study week no.: 9
Student Name & ID: Tee Jia Lin 2005546	
Supervisor: Ts Dr Ooi Boon Yaik	
Project Title: Human Activity Recognition Via Accelerometer and Gyro Sensors	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- LSTM model architecture built.
- LSTM model training.

2. WORK TO BE DONE

- Solve the overfitting problem.

3. PROBLEMS ENCOUNTERED

- LSTM model training results in overfitting problem.

4. SELF EVALUATION OF THE PROGRESS

- Progress is a bit delayed.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	Study week no.: 10
Student Name & ID: Tee Jia Lin 2005546	
Supervisor: Ts Dr Ooi Boon Yaik	
Project Title: Human Activity Recognition Via Accelerometer and Gyro Sensors	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Use early stopping and dropout technique to solve the overfitting problem.

2. WORK TO BE DONE

- Hyperparameter tuning.
- Model testing.
- Start writing report.

3. PROBLEMS ENCOUNTERED

- No.

4. SELF EVALUATION OF THE PROGRESS

- Progress is a bit delayed.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Trimester 3, Year 3	Study week no.: 11
Student Name & ID: Tee Jia Lin 2005546	
Supervisor: Ts Dr Ooi Boon Yaik	
Project Title: Human Activity Recognition Via Accelerometer and Gyro Sensors	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Hyperparameter tuning done.
- LSTM model testing done.

2. WORK TO BE DONE


- Complete FYP2 report.

3. PROBLEMS ENCOUNTERED


- No.

4. SELF EVALUATION OF THE PROGRESS

- Progress is on track.



Supervisor's signature



Student's signature

POSTER



HUMAN ACTIVITY RECOGNITION VIA ACCELEROMETER AND GYRO SENSORS

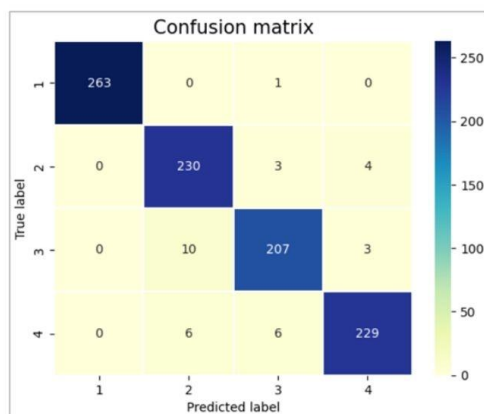
Supervisor: Ts Dr Ooi Boon Yaik

Student: Tee Jia Lin

INTRODUCTION

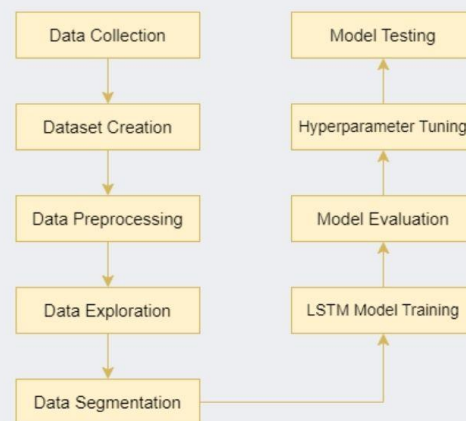
Much HAR research has been conducted, focusing on the development of algorithms and techniques. Less emphasis has been given to the data engineering system used. This project proposes an efficient, comprehensive, and user-friendly data engineering system used to collect, annotate and store sensor data. Two mobile applications, SensorData and SensorDataLogger are developed. The dataset created are then used to train LSTM model. The LSTM model achieved an accuracy of 96.57%, which validates the efficiency and practicality of our proposed system.

RESULTS



Test Accuracy: 0.9656964540481567
Test Loss: 0.11889327317476273
Test RMSE: 0.11827322

METHODS



DISCUSSIONS

- The tuned LSTM model can achieve an accuracy of 96.57%
- Overfitting problem is not expected to occur since we implement early stopping and dropout technique.
- The model sometimes confuses between walking (2), climbing downstairs (3) and climbing downstairs (4)

CONCLUSION

Data engineering system developed can improve the efficiency and accuracy of the sensor data collection, annotation and storage process.

PLAGIARISM CHECK RESULT

Human activity recognition fyp2

ORIGINALITY REPORT

9%

SIMILARITY INDEX

7%

INTERNET SOURCES

5%

PUBLICATIONS

3%

STUDENT PAPERS

PRIMARY SOURCES

1

www.cs.umb.edu

Internet Source

<1%

2

www.mdpi.com

Internet Source

<1%

3

www.ncbi.nlm.nih.gov

Internet Source

<1%

4

J. Godwin Ponsam, S.V. Juno Bella Gracia, G. Geetha, K. Nimala, Swetha Chepuri, Rithik S. Rajline. "Human Activity Recognition Using LSTM Network with Dropout Technique", 2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), 2022

Publication

<1%

5

www.arxiv-vanity.com

Internet Source

<1%

6

link.springer.com

Internet Source

<1%

7

dokumen.pub

Internet Source

<1%

8	www.hindawi.com Internet Source	<1 %
9	www.analyticsvidhya.com Internet Source	<1 %
10	www.researchgate.net Internet Source	<1 %
11	Submitted to Napier University Student Paper	<1 %
12	Lucas Borges Ferreira, Fernando França da Cunha. "Multi-step ahead forecasting of daily reference evapotranspiration using deep learning", Computers and Electronics in Agriculture, 2020 Publication	<1 %
13	Submitted to Universiti Tunku Abdul Rahman Student Paper	<1 %
14	eprints.utar.edu.my Internet Source	<1 %
15	YONGCAI GUO, WEIHUA HE, CHAO GAO. "HUMAN ACTIVITY RECOGNITION BY FUSING MULTIPLE SENSOR NODES IN THE WEARABLE SENSOR SYSTEMS", Journal of Mechanics in Medicine and Biology, 2012 Publication	<1 %
16	github.com Internet Source	<1 %

17	scholar.sun.ac.za Internet Source	<1 %
18	www.biorxiv.org Internet Source	<1 %
19	d197for5662m48.cloudfront.net Internet Source	<1 %
20	huggingface.co Internet Source	<1 %
21	ipfs.io Internet Source	<1 %
22	Submitted to University of Economics & Law Student Paper	<1 %
23	Submitted to University of Wales central institutions Student Paper	<1 %
24	bbrc.in Internet Source	<1 %
25	coek.info Internet Source	<1 %
26	dspace.cvut.cz Internet Source	<1 %
27	Submitted to University of Ulster Student Paper	<1 %

28	Bayat, Akram, Marc Pomplun, and Duc A. Tran. "A Study on Human Activity Recognition Using Accelerometer Data from Smartphones", <i>Procedia Computer Science</i> , 2014. Publication	<1 %
29	Submitted to University of Strathclyde Student Paper	<1 %
30	repository.essex.ac.uk Internet Source	<1 %
31	"Human Activity Recognition using Deep and Machine Learning Algorithms", <i>International Journal of Innovative Technology and Exploring Engineering</i> , 2020 Publication	<1 %
32	Sheik Murad Hassan Anik, Xinghua Gao, Na Meng. "Towards automated occupant profile creation in smart buildings: A machine learning-enabled approach for user persona generation", <i>Energy and Buildings</i> , 2023 Publication	<1 %
33	www.medrxiv.org Internet Source	<1 %
34	Poornachandra Sarang. "Artificial Neural Networks with TensorFlow 2", Springer Science and Business Media LLC, 2021 Publication	<1 %

35	researchrepository.wvu.edu Internet Source	<1 %
36	spectrum.library.concordia.ca Internet Source	<1 %
37	Christian Ebere Enyoh, Qingyue Wang. "Automated Classification of Undegraded and Aged Polyethylene Terephthalate Microplastics from ATR-FTIR Spectroscopy using Machine Learning Algorithms", Research Square Platform LLC, 2023 Publication	<1 %
38	Submitted to University of Glasgow Student Paper	<1 %
39	edoc.pub Internet Source	<1 %
40	Kathrin Kalischewski, Daniel Wagner, Jorg Velten, Anton Kummert. "Activity recognition for indoor movement and estimation of travelled path", 2017 10th International Workshop on Multidimensional (nD) Systems (nDS), 2017 Publication	<1 %
41	Marek Natkaniec, Marcin Bednarz. "Wireless Local Area Networks Threat Detection Using 1D-CNN", Sensors, 2023 Publication	<1 %

42	Submitted to Victoria University Student Paper	<1 %
43	Submitted to Baker College Student Paper	<1 %
44	Submitted to Coventry University Student Paper	<1 %
45	Submitted to University of Birmingham Student Paper	<1 %
46	Carlos Parga, Xiaoou Li, Wen Yu. "Tele-manipulation of robot arm with smartphone", 2013 6th International Symposium on Resilient Control Systems (ISRCS), 2013 Publication	<1 %
47	Submitted to Rochester Institute of Technology Student Paper	<1 %
48	hdl.handle.net Internet Source	<1 %
49	Submitted to Auston Institute of Management and Technology Student Paper	<1 %
50	www.coursehero.com Internet Source	<1 %
51	Submitted to University of Wales Institute, Cardiff Student Paper	<1 %

52	Submitted to Westcliff University Student Paper	<1 %
53	jurnal.atmaluhur.ac.id Internet Source	<1 %
54	machinelearningmastery.com Internet Source	<1 %
55	medium.com Internet Source	<1 %
56	wrap.warwick.ac.uk Internet Source	<1 %
57	www.dell.com Internet Source	<1 %
58	Submitted to Canterbury Christ Church University Student Paper	<1 %
59	Iris Iddaly Méndez-Gurrola, Ramón Iván Barraza-Castillo, Abdiel Ramírez Reyes, Alejandro Israel Barranco-Gutiérrez. "Chapter 40 An Approach to Mobile App Design and Development Combining Design Thinking, User Experience, and Iterative-Incremental Development", Springer Science and Business Media LLC, 2023 Publication	<1 %

60	Submitted to King Mongkut's University of Technology North Bangkok (Graduate College) Student Paper	<1%
61	Submitted to University of Sydney Student Paper	<1%
62	Submitted to University of Warwick Student Paper	<1%
63	saucis.sakarya.edu.tr Internet Source	<1%
64	Paul A. Demers, Kay Teschke, Hugh W. Davies, Susan M. Kennedy, Victor Leung. "Exposure to Dust, Resin Acids, and Monoterpenes in Softwood Lumber Mills", AIHAJ - American Industrial Hygiene Association, 2000 Publication	<1%
65	Submitted to Politecnico di Milano Student Paper	<1%
66	Submitted to University of Western Sydney Student Paper	<1%
67	cdas.cancer.gov Internet Source	<1%
68	jestec.taylors.edu.my Internet Source	<1%

69	echenshe.com Internet Source	<1 %
70	thesai.org Internet Source	<1 %
71	Dinh Le, Tuan, and Chung Van Nguyen. "Human activity recognition by smartphone", 2015 2nd National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS), 2015. Publication	<1 %
72	Submitted to Liverpool John Moores University Student Paper	<1 %
73	Submitted to Sikkim Manipal University Student Paper	<1 %
74	Submitted to University of East London Student Paper	<1 %
75	Submitted to Xiamen University Student Paper	<1 %
76	assets.researchsquare.com Internet Source	<1 %
77	deepai.org Internet Source	<1 %
78	intellipaat.com Internet Source	<1 %

		<1 %
79	mdpi.com Internet Source	<1 %
80	neptune.ai Internet Source	<1 %
81	udspace.udel.edu Internet Source	<1 %
82	Dimitrios Kolosov, Vasilios Kelefouras, Pandelis Kourtessis, Iosif Mporas. "Contactless Camera-Based Heart Rate and Respiratory Rate Monitoring Using AI on Hardware", Sensors, 2023 Publication	<1 %
83	db.teddyboom.com Internet Source	<1 %
84	ebin.pub Internet Source	<1 %
85	pdfcoffee.com Internet Source	<1 %
86	pgjonline.com Internet Source	<1 %
87	www.frontiersin.org Internet Source	<1 %

88	"Cyberspace Safety and Security", Springer Science and Business Media LLC, 2019 Publication	<1 %
89	"Machine Learning for Brain Disorders", Springer Science and Business Media LLC, 2023 Publication	<1 %
90	Ali Chelli, Matthias Patzold. "A Machine Learning Approach for Fall Detection and Daily Living Activity Recognition", IEEE Access, 2019 Publication	<1 %
91	Kwon, Yongjin, Kyuchang Kang, and Changseok Bae. "Unsupervised Learning for Human Activity Recognition Using Smartphone Sensors", Expert Systems with Applications, 2014. Publication	<1 %
92	Kybernetes, Volume 43, Issue 5 (2014-09-16) Publication	<1 %
93	Ramnath Kumar, G Geethakumari. "Temporal Dynamics and Spatial Content in IoT Malware detection", TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), 2019 Publication	<1 %
94	Zhuo Chen, Danqing Song. "Modeling landslide susceptibility based on convolutional	<1 %

neural network coupling with metaheuristic optimization algorithms", International Journal of Digital Earth, 2023

Publication

95	archive.org Internet Source	<1 %
96	arxiv.org Internet Source	<1 %
97	ia601805.us.archive.org Internet Source	<1 %
98	idoc.pub Internet Source	<1 %
99	mdpi-res.com Internet Source	<1 %
100	www.researchsquare.com Internet Source	<1 %
101	"Food Processing", Wiley, 2004 Publication	<1 %
102	Sepp Hochreiter, Jürgen Schmidhuber. "Long Short-Term Memory", Neural Computation, 1997 Publication	<1 %

	Student Paper	<1%
10	digitalcommons.lsu.edu Internet Source	<1%
11	www.essaysauce.com Internet Source	<1%
12	eprints.utm.edu.my Internet Source	<1%
13	Submitted to Laureate Higher Education Group Student Paper	<1%
14	Submitted to Letterkenny Institute of Technology Student Paper	<1%
15	Submitted to Kingston University Student Paper	<1%
16	sites.google.com Internet Source	<1%
17	www.3dreshaper.com Internet Source	<1%
18	core.ac.uk Internet Source	<1%
19	www.glendale.k12.or.us Internet Source	<1%
20	Foundations of Joomla, 2015. Publication	<1%

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1

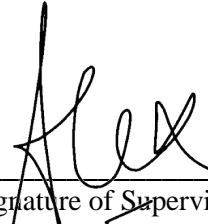
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Tee Jia Lin
ID Number(s)	20ACB05546
Programme / Course	Bachelor of Computer Science (Honours)
Title of Final Year Project	Human Activity Recognition Via Accelerometer and Gyro Sensors

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)
Overall similarity index: <u>9</u> % Similarity by source Internet Sources: <u>7</u> % Publications: <u>5</u> % Student Papers: <u>3</u> %	
Number of individual sources listed of more than 3% similarity: <u>0</u>	
Parameters of originality required and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.



Signature of Supervisor

Name: Ts Dr Ooi Boon Yaik

Date: 15 September 2023

Signature of Co-Supervisor

Name: _____

Date: _____



UNIVERSITI TUNKU ABDUL RAHMAN

**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)**

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB05546
Student Name	Tee Jia Lin
Supervisor Name	Ts Dr Ooi Boon Yaik

TICK (✓)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	Front Plastic Cover (for hardcopy)
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
✓	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 15 September 2023