青　山　学　院　大　学

理　工　学　研　究　科

理工学専攻　知能情報　コース

修　士　論　文

学 生 番 号　35623252

氏　名　LIU ZEYU

研究指導教員　Guillaume　LOPEZ

SmartWT: Fitness activity monitoring using wearable devices

Liu Zeyu

January 2025

理工学専攻修士論文要旨

提出年度　　：2024 年度
提出日　　：2025 年 1 月 31 日
専修コース　：知能情報　コース
学生番号　　：35623252
学生指名　　：LIU ZEYU
研究指導教員：ロペズ・ギヨーム 教授

（論文題目）
SmartWT: Fitness activity monitoring using wearable devices

（内容の要旨）
近年，ウェアラブルデバイスは急速に進化し，心拍数測定，歩数計測，睡眠分析など多様な健康管理機能を提供するようになった．特にスマートウォッチは，加速度センサーやジャイロスコープを搭載し，運動データのリアルタイム収集が可能となっている．しかし，現在のフィットネス活動追跡アプリケーションソフトウェア（以降アプリ）は，有酸素運動の記録には優れているものの，ウェイトトレーニングなどの無酸素運動においては，ユーザーが回数や負荷を手動で入力する必要があり，トレーニングの集中力を妨げる要因となっている．

これを解決するため，本研究では，スマートウォッチとスマートフォンを活用したリアルタイムフィットネスモニタリングシステム「SmartWT」を開発し，既存のフィットネスアプリが無酸素運動の自動記録に対応できていない課題を解決することを目的としている．

本研究では，スマートウォッチで加速度データを収集し，スマートフォンに転送したのち，スマートフォン上でリアルタイム処理を行う TLSW（三層スライディングウィンドウ）アルゴリズムを考案し，運動状態の自動識別，運動種別の分類，および回数の自動カウントを実現することで，手動入力の必要性を排除する．TLSW アルゴリズムは，状態認識，ピーク検出，運動分類の 3 つの主要ステップで構成される．まず，状態分類モデルを用いてユーザーが運動状態にあるかを判定し，運動中であると判断された場合，合成加速度を計算してピークを検出する．ピークがウィンドウの中心に位置する場合，それを 1 回の運動としてカウントし，機械学習モデルで運動種別を分類する．精度向上のため，最大値，最小値，スペクトルセントロイド，スペクトルフラットネス，ゼロ交差率などの時間領域・周波数領域の特徴量を抽出し，ランダムフォレストを用いた分類モデルを訓練した．

パソコンを用いたオフライン環境において，分類精度 96.7%，カウント精度 96.6%を達成したが，高性能なプログラミング環境と処理能力のないモバイル端末において，データ転送や計算環境の違いにより，それぞれ 95%，88%に若干低下した．さらに，9 名の被験者を対象とした実証実験では，8 種類の運動において分類精度 86.1%，カウント精度 81.4%を記録した．特定の動作において精度が低下した主な原因は，個人の運動幅の違いや疲労による動作速度の低下である．モバイル端末の開発では，システムのリアルタイム性と安定性を確保するために，Data Layer API を使用してスマートウォッチとスマートフォン間のデータ転送を実装し，スライディングウィンドウのデータ管理を最適化するためにリングバッファを採用し，データコピーによる負荷を軽減した．また，パソコンを用いたオフライン環境と特性計算の精度を統一するため，モバイル端末に Chaquopy を統合し，フーリエ変換や周波数領域の特徴抽出をオフライン処理環境と同じスクリプトで実行できるようにした．

実験結果から，本システムは高精度な運動種別識別および回数カウントを実現し，フィットネスデータの記録を効率化できることが確認された．加えて，Android 端末の最適化により，データ処理の効率と計算の安定性が確保された．

青山学院大学大学院理工学研究科

The Academic Year of 2024, Submitted on January 31st, 2025

Graduate School of Science and Engineering, Aoyama Gakuin University

Title: SmartWT: Fitness activity monitoring using wearable devices

Student Name: LIU ZEYU
ID Number: 35623252
Degree: Master of Engineering
Course: Intelligence and Information
Thesis Advisor: Professor Guillaume Lopez

Abstract

In recent years, wearable devices have rapidly evolved, offering various health management functions such as heart rate monitoring, step counting, and sleep analysis. Smartwatches, in particular, are equipped with accelerometers and gyroscopes, enabling real-time movement data collection. However, while current fitness tracking applications effectively record aerobic exercises, they still require users to manually input repetitions and weights for anaerobic exercises like weight training, which disrupts workout focus.

This study developed SmartWT, a real-time fitness monitoring system utilizing smartwatches and smartphones, to address the limitations of existing fitness applications in automatically tracking anaerobic exercises. It proposes the Three-Layer Sliding Window (TLSW) algorithm, which collects acceleration data via a smartwatch and processes it in real-time on a smartphone to automatically recognize exercise states, classify exercise types, and count repetitions, eliminating the need for manual input. The algorithm consists of three key steps: state recognition, peak detection, and exercise classification & counting. First, the system uses a state classification model to determine whether the user is actively exercising. Once an exercise state is detected, composite acceleration is calculated, and peaks are identified. If a peak is located at the center of the window, it is counted as a valid repetition, and a machine learning model classifies the exercise type. To improve accuracy, this study extracted various time-domain and frequency-domain features, including maximum and minimum values, spectral centroid, spectral flatness, and zero-crossing rate, and trained a classification model using a Random Forest algorithm.

In offline experiments conducted in a PC environment using Python programming language, the algorithm achieved 96.7% classification accuracy and 96.6% counting accuracy. However, in offline tests on Android devices, accuracy slightly decreased to 95% and 88%, respectively, due to differences in data transmission and computational environments. Furthermore, a user study with nine participants performing eight different exercises recorded a classification accuracy of 86.1% and a counting accuracy of 81.4%. Lower accuracy in some exercises was mainly attributed to variations in individual movement ranges and reduced motion speed caused by fatigue.

For the mobile device implementation, the system's real-time performance and stability were ensured by utilizing the Data Layer API to transmit data between the smartwatch and smartphone. A ring buffer was employed to optimize sliding window data management, reducing computational overhead caused by continuous data copying. Additionally, to maintain consistency with the Python environment in feature calculations, Chaquopy library was integrated into the mobile device system, allowing Fourier Transform and frequency-domain feature extraction to be executed using Python scripts directly on the mobile device.

The experimental results confirm that SmartWT can accurately recognize exercise types and count repetitions, significantly improving the efficiency and accuracy of fitness data recording. Furthermore, optimizations in the mobile device implementation ensure efficient data processing and computational stability, making the system a promising solution for automated fitness tracking.

Contents

Chapter 1

Introduction

1.1 Background

In recent years, wearable devices have gained widespread popularity due to their powerful functionalities in monitoring health and fitness activities. These devices not only help individuals track their physical condition in real time but also provide personalized recommendations for fitness enthusiasts, promoting more scientific and efficient workout methods. A 2023 survey revealed that wearable devices consistently attract significant attention among fitness enthusiasts, especially smartwatches equipped with sensors such as accelerometers and gyroscopes, which have become essential tools for tracking physical activity.

Scientific research has demonstrated that regular exercise has significant positive effects on physical health. Exercise not only strengthens muscles and improves cardiovascular function but also effectively reduces the risk of chronic diseases such as cardiovascular diseases, diabetes, and obesity. Additionally, fitness activities can enhance mood, relieve stress, and improve overall quality of life. However, unstructured or unscientific exercise may lead to suboptimal outcomes or even increase the risk of injury. Therefore, planned fitness routines are particularly important. A well-structured fitness plan helps individuals select appropriate exercise intensity and volume based on their physical condition and goals, thereby improving workout efficiency and minimizing the likelihood of injuries.

With the rapid advancement of technology, wearable devices continue to improve in both hardware and software functionalities. Early wearable devices had relatively simple features, such as step counting and heart rate measurement. In contrast, today ' s smartwatches and fitness bands offer much more comprehensive monitoring capabilities, including sleep quality

Top 20 Worldwide Fitness Trends for 2023

Rank	Trend
1	Wearable technology
2	Strength training with free weights
3	Body weight training
4	Fitness programs for older adults
5	Functional fitness training
6	Outdoor activities
7	High-intensity interval training (HIIT)
8	Exercise for weight loss
9	Employing certified fitness professionals
10	Personal training
11	Core training

Wearable devices receive attention [1].

tracking, real-time three-axis acceleration data recording, blood oxygen monitoring, and personalized fitness plan recommendations.

These devices not only raise users' awareness of their health but also generate detailed health reports based on collected data, helping users adjust their fitness strategies more scientifically. For example, real-time heart rate monitoring enables users to control exercise intensity to remain in the optimal range for fat burning or cardiovascular improvement. Additionally, motion recognition features help users correct improper movements and prevent injuries. For fitness enthusiasts, these functions significantly enhance the safety and scientific accuracy of their workouts. At the same time, fitness tracking applications are flourishing, serving as essential companions to wearable devices. These applications seamlessly connect with smart devices, allowing users to synchronize workout data in real time and providing detailed fitness analyses and plan customization. Many fitness apps now include social elements, such as online rankings, challenges, and badge rewards. These features not only enable users to track their fitness progress and long-term achievements but also increase engagement and motivation through social interactions and competitive incentives. In the future, with the further development of artificial intelligence and big data technologies, fitness tracking apps will be able to identify users' workout states more accurately, provide highly personalized fitness advice, and predict potential health risks, further advancing data-driven fitness applications.

In summary, the rapid development of wearable devices and the growing prevalence of fit-

Figure 1.2: Monitoring heart rate with a smartwatch

ness tracking apps are profoundly transforming people's approach to fitness, offering new possibilities for healthier lifestyles. These tools not only help fitness enthusiasts achieve their goals more efficiently but also promote health management and the establishment of fitness habits through technological innovations. In the future, as technology continues to evolve, this field will further develop, bringing greater convenience and health benefits to a broader audience.

1.2 Research problem

Currently, many fitness tracking apps are highly effective in monitoring aerobic exercises. For example, Apple Fitness and Google Fit provide comprehensive tracking for long-duration aerobic activities like running and cycling. However, their support for anaerobic exercises remains inadequate. To address this issue, fitness apps such as RepCount and Strong Workout Tracker Gym Log, which focus on anaerobic exercises, have gained popularity. These apps assist users in creating detailed training plans and tracking long-term fitness achievements. However, users are still required to manually input data such as repetitions and weights after completing each set of exercises. This frequent input not only disrupts user focus but also diminishes the overall workout experience. If user actions could be recognized in real time and recorded automatically, it would significantly enhance workout efficiency and user experience.

Current research on human action recognition can be broadly divided into two approaches: image-based recognition and sensor-based recognition. Image recognition uses cameras to capture images of users during exercise and applies deep learning techniques to identify movements, enabling precise capture of complex motion details. However, this approach requires

high device performance and network reliability and may raise privacy concerns. Sensor-based research, on the other hand, leverages built-in sensors like accelerometers and gyroscopes to collect user movement data and applies methods such as template matching or machine learning to recognize actions. This approach offers better real-time performance and portability. Nonetheless, it faces challenges such as the potential inconvenience caused by wearing multiple sensors or the reliance on stable network connections for data transmission.

Although wearable devices have made significant progress in the field of human action recognition, existing research still has limitations. On the one hand, many developed algorithms remain confined to theoretical models or simulated environments and lack deployment and testing on actual devices, leaving their performance in real-world scenarios insufficiently validated. On the other hand, many models used in research are computationally intensive and require high hardware performance, making it impossible for them to operate independently on wearable devices. Instead, they rely on server-based computation. Furthermore, this dependence on server computation necessitates a stable network connection between the device and server. When the network is unstable or offline, system performance and user experience are significantly impacted.

In this study, we developed an algorithm capable of long-term tracking of user fitness activities, automatically recognizing and counting exercise actions. The algorithm replaces manual operations at the start and end of each action with automatic segmentation and improves upon the traditional fixed-step sliding window algorithm, reducing computational overhead while enhancing recognition accuracy. Testing results show that the algorithm achieves an accuracy of 94.31% in long-term tracking of fitness data, correctly identifying exercise types and counting sets.

1.2.1 Questionnaire for fitness enthusiasts on automated monitoring of fitness data

To understand the use of wearable devices by fitness enthusiasts and their perspectives on fitness tracking software, we designed a questionnaire survey. The following are the results of the survey.

Respondents Fitness Situation Statistics

We conducted an anonymous questionnaire survey with 71 participants. The distribution of their years of fitness experience and regular workout frequency is shown in the chart Figure 1.3 below.

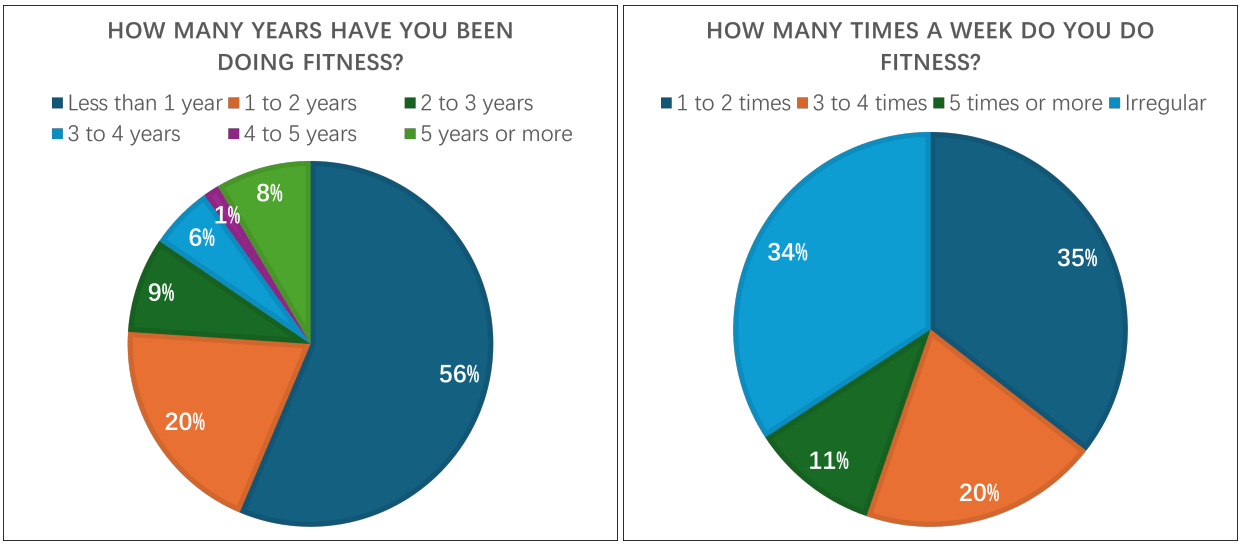


Figure 1.3: Participants' number of years and frequency of fitness

The Current State of Planned Fitness

We analyzed respondents' views on fitness plans, their habits of creating fitness plans, and the implementation of these plans. The results are presented in Figure 1.4.

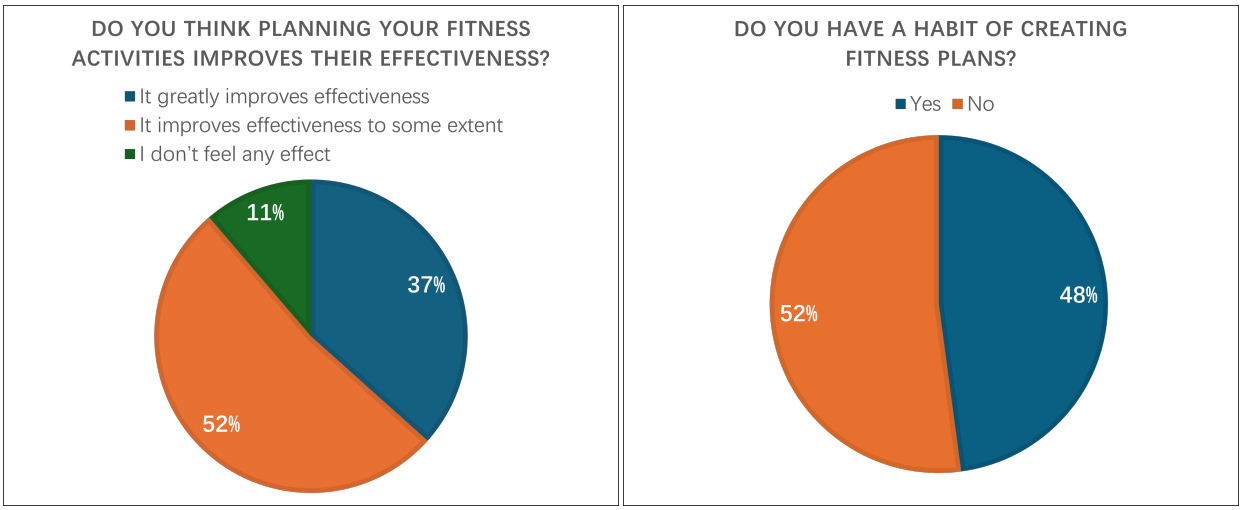


Figure 1.4: Participants' number of years and frequency of fitness

Most respondents believe that fitness plans can enhance workout effectiveness. Among them, 52% stated that plans "somewhat improve effectiveness," 37% believed that plans "greatly enhance effectiveness," and only 11% indicated that plans have no effect. Of the respondents, 48% reported having the habit of creating fitness plans, while 52% stated they did not have such a habit. Based on whether respondents have the habit of creating fitness plans, we designed corresponding questions and analyzed the responses. Below are the most representative results

from the analysis.

Fitness enthusiasts with a habit of creating plans

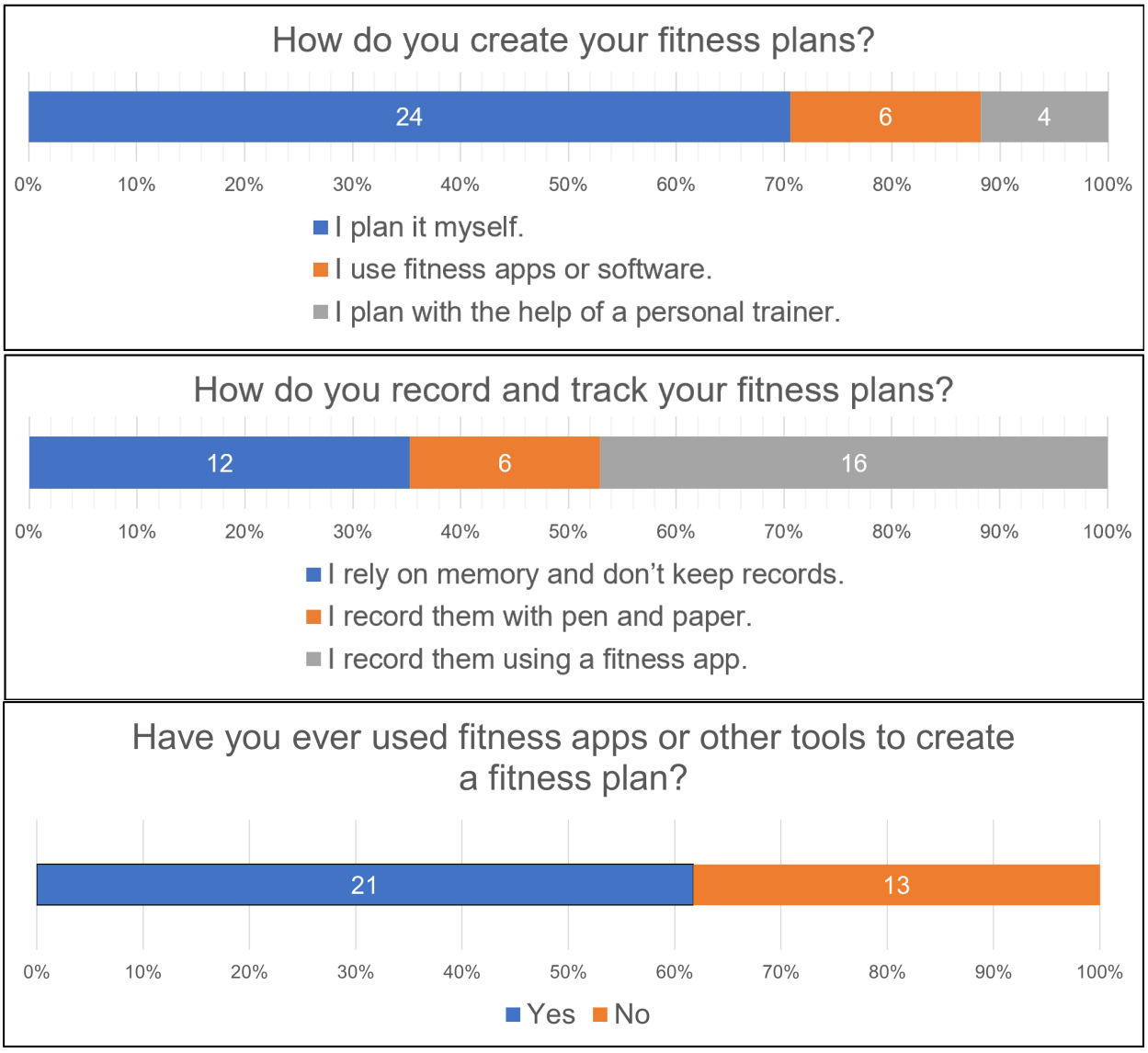


Figure 1.5: Views on fitness plans from people who have plans to work out

From the Figure 1.5 above, it can be observed that most respondents who create fitness plans tend to design their plans independently, while the proportion using fitness apps or software is relatively low. When recording fitness plans, they prefer simpler methods or even rely on memory. Additionally, more than half of the respondents have tried using fitness apps or tools to assist in creating their plans.

Fitness enthusiasts without a habit of creating plans

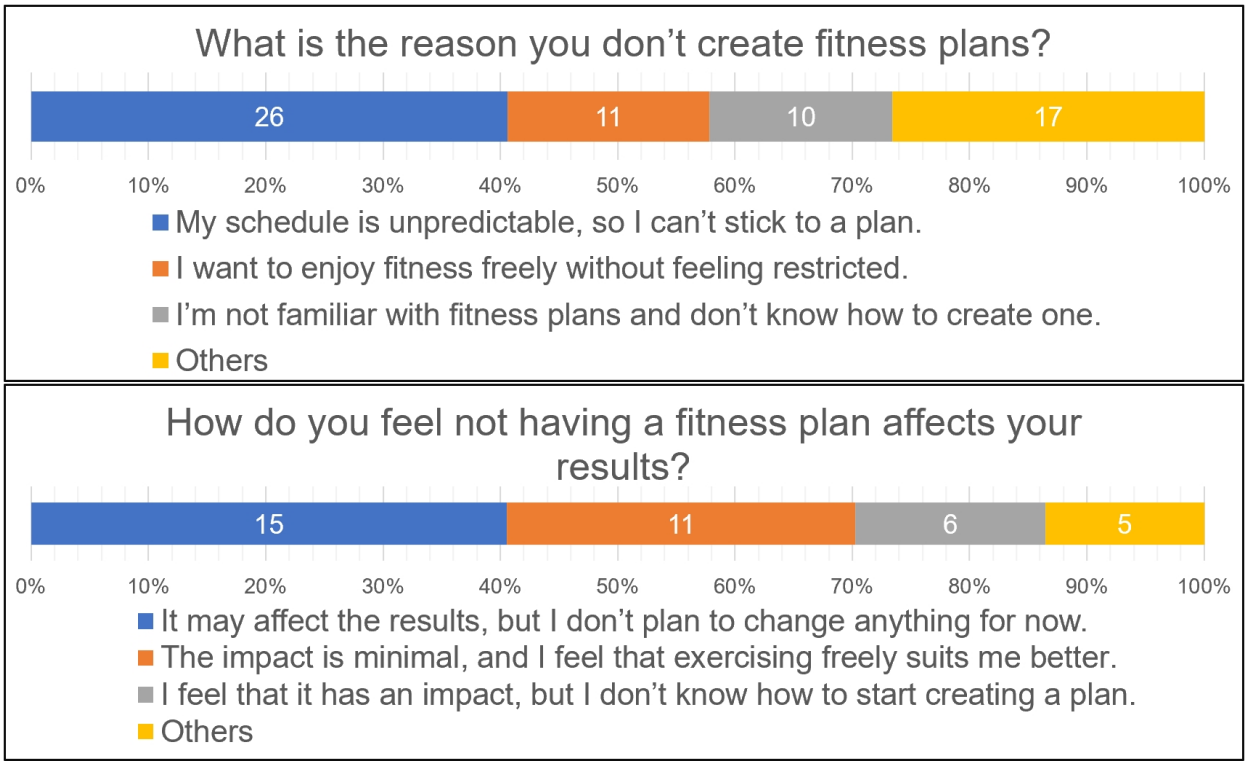


Figure 1.6: Views on fitness plans from people who have no plan to work out

For users who do not create fitness plans, as shown in Figure 1.6, the main reasons include unpredictable schedules, a preference for unrestricted workouts, and unfamiliarity with plan creation. Opinions on the impact of not having a plan on workout effectiveness are divided. Some users believe the impact is minimal and prefer the freedom of unstructured exercise. Others, while aware of the potential drawbacks, have not taken action due to uncertainty about how to start or a lack of willingness to change.

Respondent perceptions of tracking apps

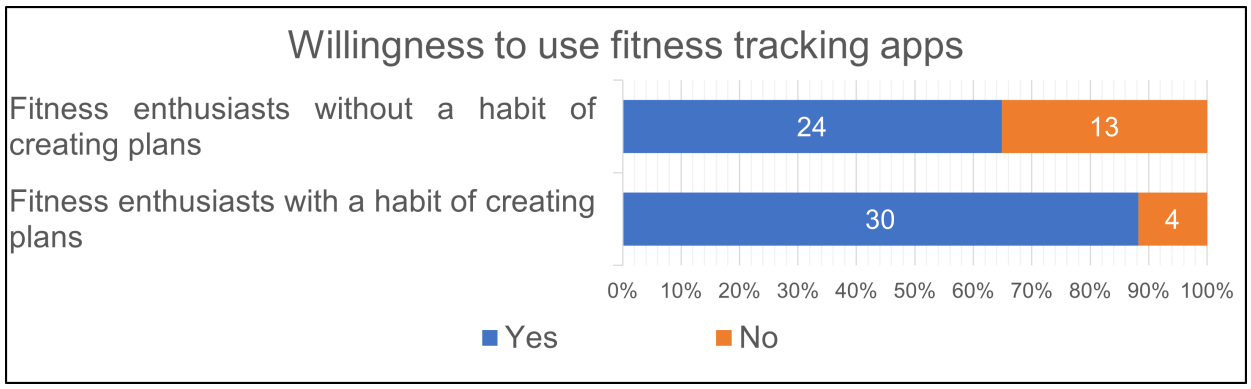


Figure 1.7: The willingness of different fitness enthusiasts to use fitness tracking apps

Before analyzing fitness enthusiasts' opinions on fitness tracking apps, we first examined their willingness to use such applications. As shown in Figure 1.7, 76.1% of respondents expressed a willingness to use fitness tracking apps. Among them, individuals with planned workout routines demonstrated a significantly higher inclination toward using these applications.

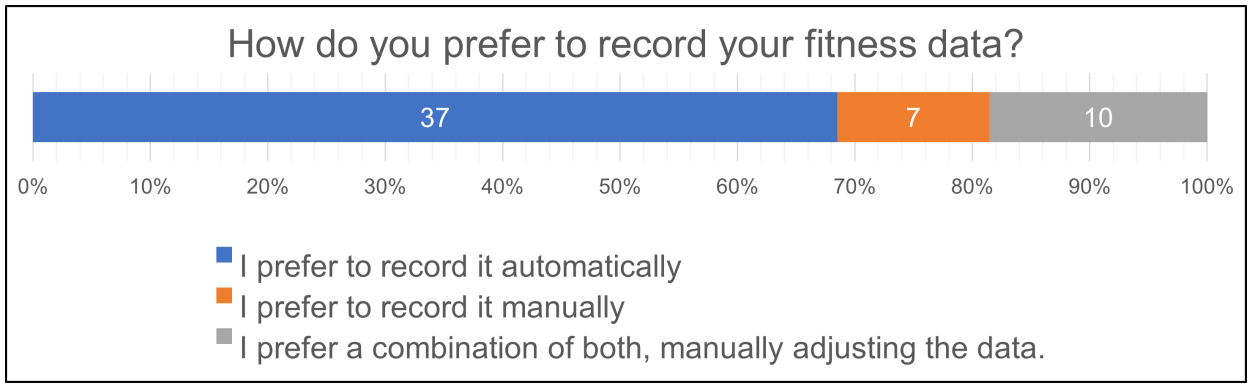


Figure 1.8: Views of fitness enthusiasts on recording data

Figure 1.8 shows that most respondents preferred automatic recording of fitness data, while only a small number chose manual recording or a combination of automatic and manual adjustments for data recording.

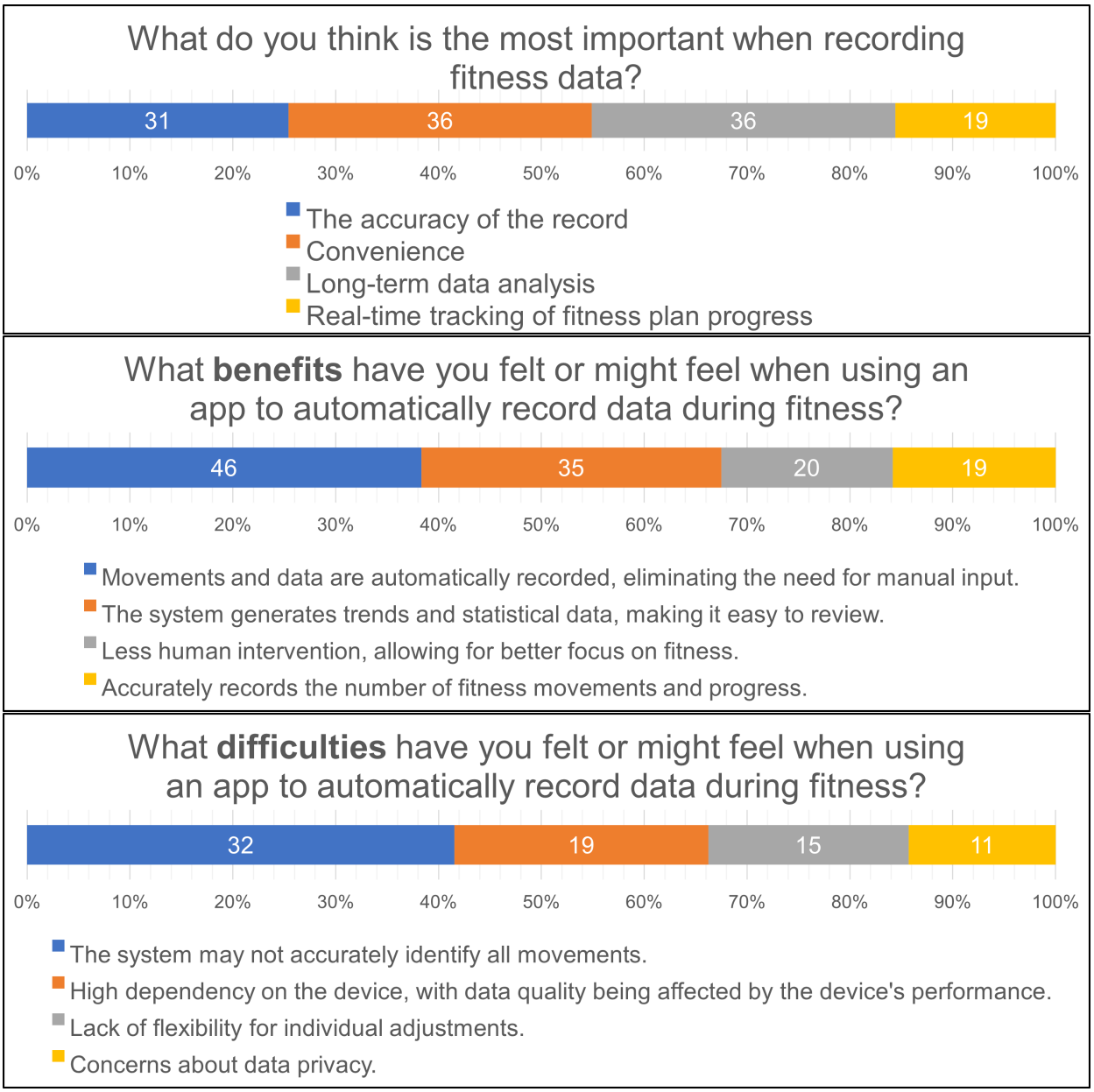


Figure 1.9: Views of fitness enthusiasts on automatically recording fitness data

As shown in Figure 1.9, respondents value accuracy and convenience the most when recording fitness data. The primary advantages of automatic recording applications include reducing manual input, improving management efficiency, and helping users focus more on their workouts. However, these applications also face challenges such as recognition inaccuracies, high dependency on devices, limited flexibility, and concerns about data privacy.

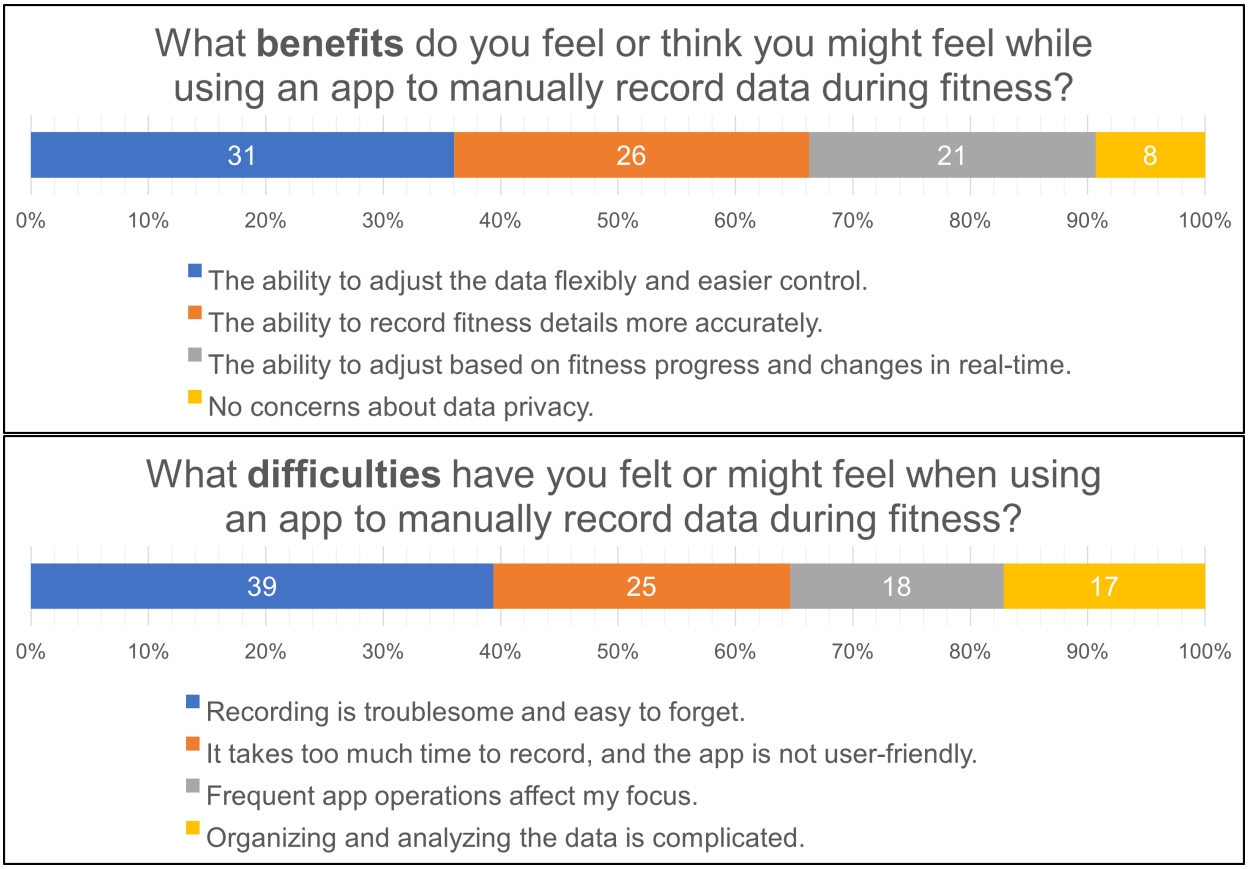


Figure 1.10: Views of fitness enthusiasts on automatically recording fitness data

As shown in the results of Figure 1.10, fitness apps that rely on manual data recording offer certain advantages, such as flexibility in data adjustments, more accurate detail recording, and the ability to make adjustments based on real-time progress. However, users also face numerous challenges, including the tedious and forgettable nature of the recording process, its time-consuming and user-unfriendly aspects, frequent operations that disrupt focus, and the complexity of data organization and analysis. This indicates that while manual recording excels in precise control, the user experience requires further optimization.

Expectations for automated fitness tracking app

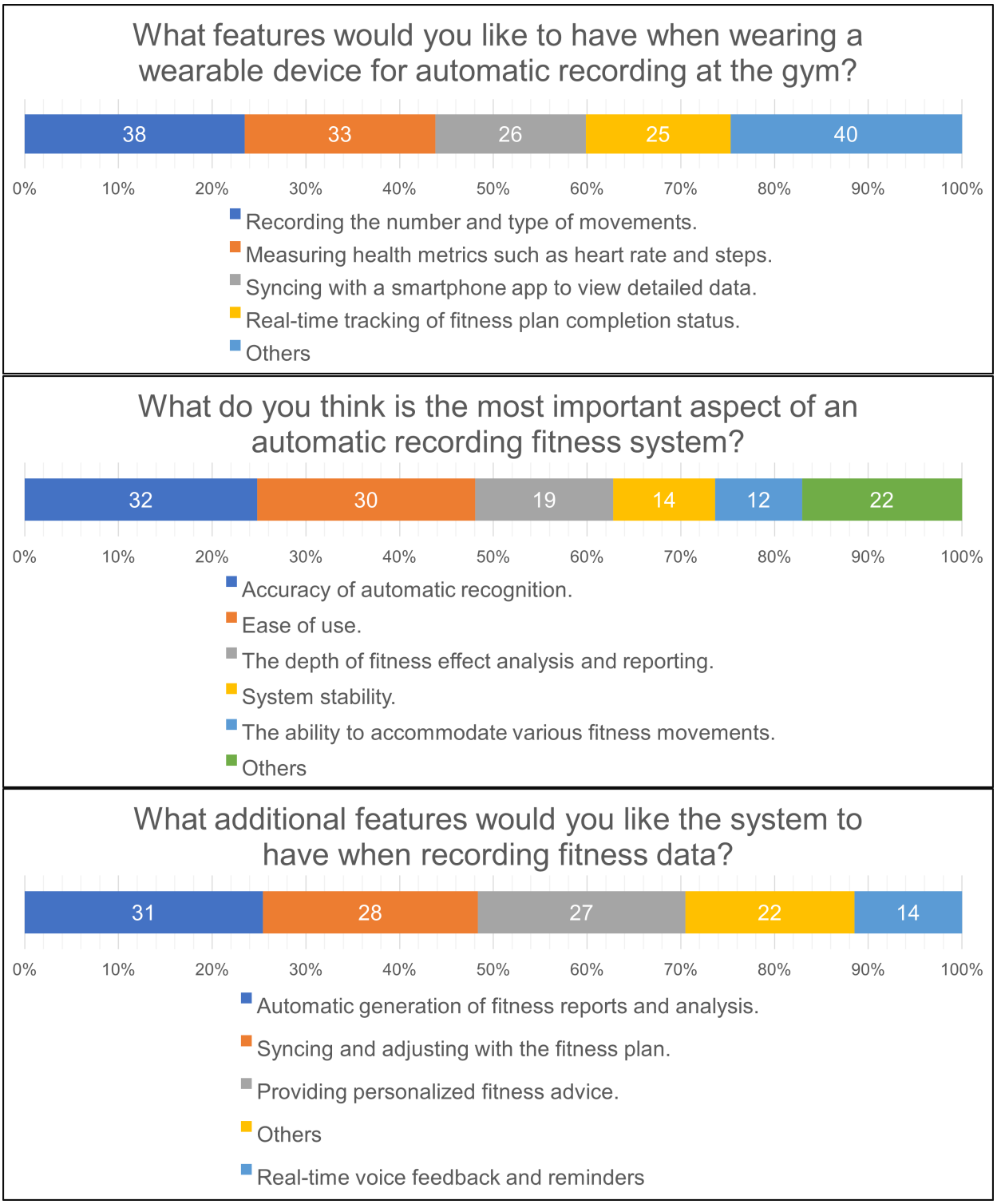


Figure 1.11: Expectations of fitness enthusiasts for automated fitness tracking app

Figure 1.11 illustrates that users ' demands for automated fitness data recording systems focus on accurately recognizing fitness actions, real-time tracking of plan completion, and monitoring health metrics such as heart rate. Additionally, they consider the accuracy of automatic recognition and system usability as the most critical aspects. Users also expressed a desire for features such as automatically generated fitness reports, synchronized plan adjustments, and personalized fitness recommendations. This indicates a preference for systems that are precise, efficient, and capable of providing in-depth analysis.

Summary of questionnaire results

The survey results indicate that most respondents believe fitness plans can enhance workout effectiveness and show a strong willingness to use automatic fitness tracking applications. Compared to manual recording, automatic tracking is favored for its efficiency and convenience. Users hope to reduce manual input through automatic tracking while achieving precise action recognition, real-time plan tracking, and comprehensive monitoring of health metrics. This highlights a clear demand for fitness tracking applications with intelligent and automated features.

1.3 Outline of the thesis

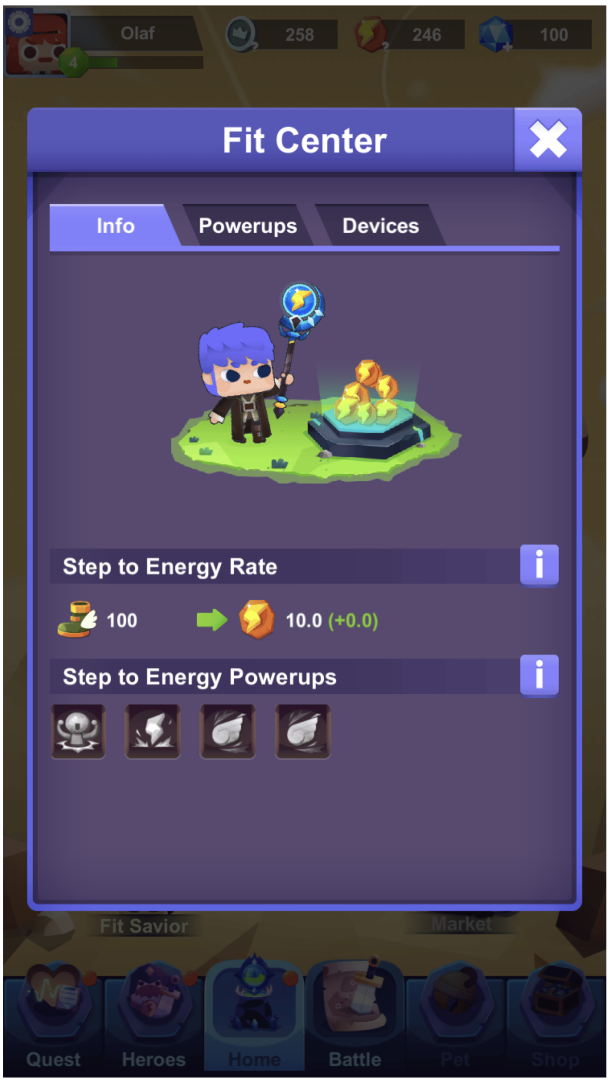
This thesis is divided into seven chapters, covering the research background, related work, methodology, experimental validation, system implementation, experimental results analysis, and future prospects. **Chapter 1** introduces the research background, highlighting the limitations of existing fitness tracking applications in recording anaerobic exercises and presenting the objectives of this study. **Chapter 2** reviews mobile fitness applications and motion recognition methods, analyzing their advantages and limitations. **Chapter 3** details the Triple-Layer Sliding Window (TLSW) algorithm, including exercise state recognition, movement classification, counting, and machine learning-based feature extraction. **Chapter 4** presents offline experiments conducted in a Python environment to evaluate the algorithm's performance. **Chapter 5** describes the implementation of the Android application, focusing on data interaction between the smartwatch and smartphone, data transmission, and feature computation. **Chapter 6** evaluates the system through offline and real-world experiments on Android devices, comparing performance across different environments. **Chapter 7** discusses potential improvements, including enhanced data processing and personalized fitness feedback.

Chapter 2

Related work

2.1 Impact of Mobile Fitness Applications

According to a 2023 survey, wearable devices have been a popular topic of discussion among fitness enthusiasts since 2016 [1]. Mobile health applications integrated into wearable devices have also become increasingly prevalent. These technological advancements not only enhance individuals' health awareness but also create opportunities for research in automatic activity recognition and data-driven fitness applications. Meanwhile, fitness tracking apps are highly popular among fitness enthusiasts. By monitoring fitness data, these apps not only help users record workout completion and long-term fitness achievements but also boost their motivation through features such as badge collection and online competitions [2][3][4].[5]introduces gamified fitness tracking applications that incorporate step count data into gameplay. By mapping existing currency-based taxonomies to step-based games and providing different types of examples, it demonstrates the value of treating steps as currency.



Screenshot from the Fitness RPG app showing canonical game elements found in traditional RPG games. [5].

The widespread adoption and advancement of this technology are closely related to the numerous health benefits of physical activity. Studies have shown that regular exercise can significantly reduce resting heart rate (RHR), particularly through endurance training and yoga. This not only lowers all-cause mortality but is also correlated with pre-intervention RHR and participant age [6]. Additionally, physical exercise enhances cognitive function and extends its positive effects by maintaining good cardiovascular health [7]. During special periods, such as the COVID-19 pandemic, digital fitness apps provided convenient solutions for home workouts, enabling individuals to stay active during lockdowns, although their benefits were primarily observed in specific social groups [8].

These fitness applications make it easier for users to track their workout progress, challenge themselves, and compete with others, further motivating them to maintain their exercise habits [7][9][10][11]. Studies on physical activity and wearable devices suggest that the reminder functions and continuous notification mechanisms provided by these devices effectively encourage adults to engage in physical exercise, with particularly significant results for sedentary individuals [8]. Additionally, participating in online competitions or sharing fitness achievements within communities can effectively boost users ' fitness motivation. This interactive approach allows users to experience social support and the joy of competition, inspiring them to engage more actively in exercise [7].

However, research shows that compared to sharing fitness achievements, more users prefer to focus on setting personal goals and tracking their progress. When users achieve their goals and receive rewards, they often feel "good about themselves" and "more motivated," which further promotes the sustainability of their workouts [12]. Moreover, fitness tracking applications enhance physical activity levels through data feedback and goal management, strengthening users' self-monitoring and fitness motivation [11].

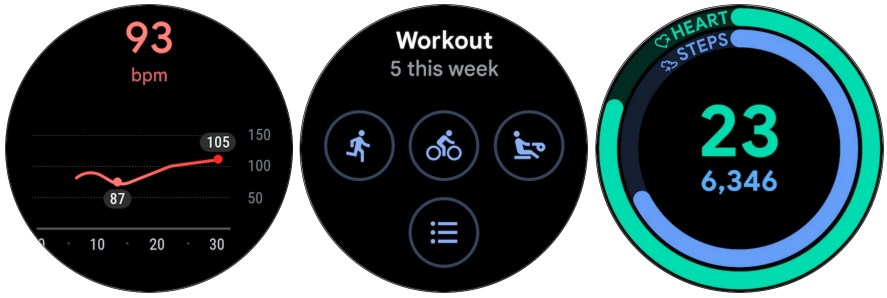


Figure 2.2: Capabilities in Google Fit

Currently, widely used fitness tracking software, such as Apple Fitness and Google Fit, can track daily walking, running, and cycling data. Apps like RepCount [13] and Strong Workout Tracker Gym Log [14], designed specifically for fitness enthusiasts, help users create workout plans and record completion progress. These apps support long-term fitness tracking, enabling users to monitor their progress over time. However, most of these apps require users to manually record repetitions after completing each set, which not only causes inconvenience but also disrupts workout focus.

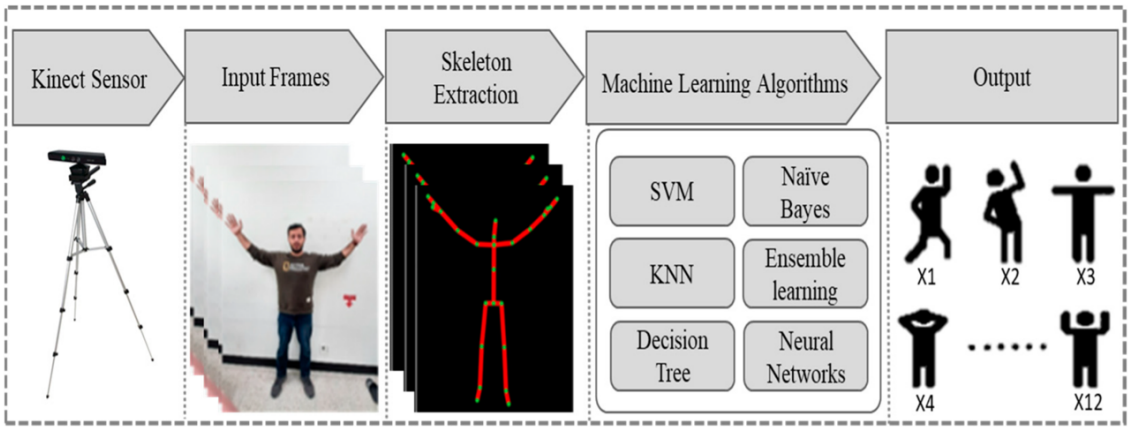
2.2 Motion Recognition Methods

In the rapid development of mobile health applications, the implementation of automatic fitness tracking relies on motion recognition technology. By converting users' movement patterns into quantifiable data, this technology not only helps users track their workout progress in real-time but also automatically identifies exercise types and calculates activity levels. To achieve long-term automatic tracking of various fitness activities, along with counting and recording them, three main objectives must typically be accomplished: (1) segmenting the user's active and inactive states, (2) recognizing the user's fitness activities, and (3) counting the fitness activities.

In research on user state and motion recognition, two primary approaches are visual-based and sensor-based methods. In visual-based research, Xie et al. [15] introduced an interactive core training system called CoreUI. This system utilizes monocular camera input and 3D human shape estimation technology to provide users with visual feedback for posture correction. However, due to a processing time of approximately 2 seconds per frame, the system has high latency, making it unsuitable for high-speed activities requiring real-time feedback, such as gymnastics or ball sports. M. Pasula et al. [16] proposed a hybrid model combining X3D and SlowFast networks for exercise classification and muscle group activation prediction in videos. The results demonstrated improved accuracy and efficiency compared to single models, but limitations remain in real-world generalization, real-time performance, and small muscle group prediction accuracy.

[17] developed a hybrid model combining a Convolutional Neural Network (CNN) and a Long Short-Term Memory (LSTM) network for activity recognition. The CNN extracts spatial features, while the LSTM captures temporal information. Additionally, a new dataset containing 12 types of human activities was created, collected from 20 participants using a Kinect V2 sensor. Through an ablation study on various traditional machine learning and deep learning models, the CNN-LSTM approach achieved an accuracy of 90.89%, demonstrating its strong performance in human activity recognition (HAR) applications.

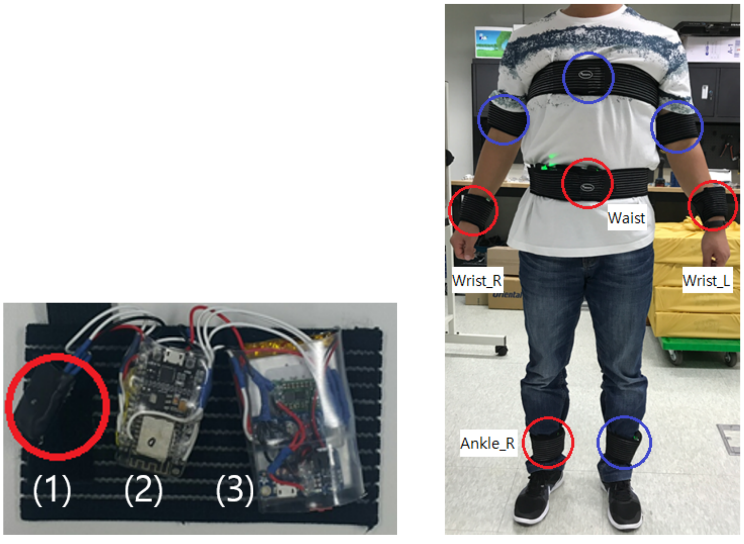
Other visual-based studies have also achieved promising recognition results. However, due to the need to set up devices like smartphones for real-time video capture, these methods are not suitable for public spaces such as gyms. Sensor-based methods, on the other hand, are more suitable for public settings because they offer higher privacy and do not require additional space for data collection.Chaurasia et al. [18] provided a comprehensive review covering various aspects of Activity Recognition and Classification (ARC) research using smartphones and wearable



Activity recognition through different Machine learning algorithms.[17]

sensors. The review included data collection, feature extraction, classification modeling, and key factors influencing performance. Additionally, it summarized the strengths and limitations of existing methods, offering clear directions for future research.

Chung et al. [19] built a testing platform consisting of eight wearable inertial measurement unit (IMU) sensors and an Android mobile device for activity data collection. They also developed a Long Short-Term Memory (LSTM) framework to train a deep learning model. The results showed that activity data from four sensors located on the wrist, right ankle, and both sides of the waist, sampled at a rate as low as 10 Hz, was sufficient to recognize daily activities (ADL), including eating and driving.



(a) (b)

Testbed configuration.[19]

Highly relevant to this study are the works of [20][21] [22][23]and [24], all of which have achieved long-term automatic recognition of various exercises and completed repetition counting. Dan et al. [20] introduced the RecoFit system, which uses inertial sensors in a smartphone worn on the arm to automatically track repetitive exercises such as weightlifting and aerobics. The system achieved segmentation, recognition, and counting of fitness data, with segmentation precision and recall exceeding 95%. Recognition rates for different numbers of repetitions were 99%, 98%, and 96%, respectively, with a counting accuracy of $\pm$ 1. However, RecoFit has limitations, such as requiring 5 seconds for segmentation and recognition, performing poorly with a small number of repetitions, and relying on specialized devices attached to the forearm, which increases user inconvenience and cost.

Data collection and evaluation hardware.[20]

Shen et al. [21] proposed the MiLift system, which combines a two-stage classification model with a lightweight weightlifting detection algorithm to autonomously and efficiently track aerobic and weightlifting exercises. The system achieved an average precision and recall of over 90% for aerobic activity classification, weightlifting detection, and weightlifting type classification. MiLift also calculated weightlifting repetitions with an average error of 1.12 repetitions (mean of 9.65 repetitions). However, it relied solely on gravity data for recognition, which reduced accuracy when movements lacked significant changes in the gravity direction.

Soro et al. [22] used deep learning methods for recognizing and counting complex full-body movements, achieving a recognition accuracy of 99.96% on constrained movement datasets. However, their approach required a CNN+RNN architecture to process multi-sensor data, posing challenges for real-time recognition on devices.

Skawinski et al. [23] proposed a machine learning approach using convolutional neural networks to count repetitions of recognized exercise types with a single 3D accelerometer worn on the chest. The method achieved an average recognition accuracy of 89.9%, with an average repetition counting accuracy of 97.9% across all exercise types. However, the method was limited

to four types of exercises with significant differences, potentially restricting its applicability to other exercise types.

Ishii et al. [24] developed ExerSense, which uses correlation-based methods and Dynamic Time Warping (DTW) for exercise segmentation, classification, and counting. The authors also tested different device positions to optimize performance. When data was limited, this approach outperformed machine learning methods in accuracy. However, template-based algorithms are highly dependent on template quality, posing challenges for generalization across different populations and exercise scenarios.

2.3 Contribution of this research

To enable real-time human motion recognition on Android devices, our study has made the following contributions compared to the aforementioned research:

1. Lower Sampling Rate: Unlike the algorithms requiring 50 Hz or 100 Hz sampling rates in prior studies, our algorithm achieves recognition and counting using only a 12.5 Hz sampling rate, significantly reducing battery consumption [25].

2. Reduced Feature Set: Our algorithm utilizes fewer features. After feature selection, the two Random Forest (RF) models employed in our study use only 24 and 26 features, respectively.

3. Double Layer Sliding Window Method: We implemented a Double Layer Sliding Window approach, which, compared to traditional sliding window methods [18][25], reduces the number of computations required by the model while maintaining accuracy, thereby lowering computational overhead.

Chapter 3

Introduction to Research Methods

3.1 Proposal of Research Methods

This study aims to develop an algorithm and design an Android application based on it to track fitness activities in real time and record data for eight categories of exercises. The system collects acceleration data in real time through a smartwatch and processes it on the smartphone. With this application, users do not need to manually select exercise types or record the number of repetitions during their workouts. The application automatically handles the recording and classification of fitness data.

Figure 3.1 illustrates the eight target fitness activities in our study, all of which are upper-body resistance exercises performed using gym machines. Resistance exercises provide numerous benefits for individuals across different populations and conditions, such as reducing arterial blood pressure [26], increasing muscle strength, and serving as a treatment for various musculoskeletal disorders [27].

Before detailing the components of our algorithm, we first present Figure 3.2, which illustrates how the entire algorithm operates in a real-time environment.

The algorithm employs a triple-layer sliding window structure to process acceleration data collected during user activities in real time. The detailed workflow is as follows: **1. State Detection:** The algorithm first uses three-axis acceleration data to determine whether the user is in an active state. **2. Composite Acceleration Calculation:** Once the user is identified as being in an active state, compcosite acceleration is computed. **3. Repetition Counting:** By detecting the peaks of the composite acceleration, the algorithm counts the user ' s repetitive actions. **4. Action Recognition:** Finally, based on the three-axis acceleration data, the algorithm identifies the

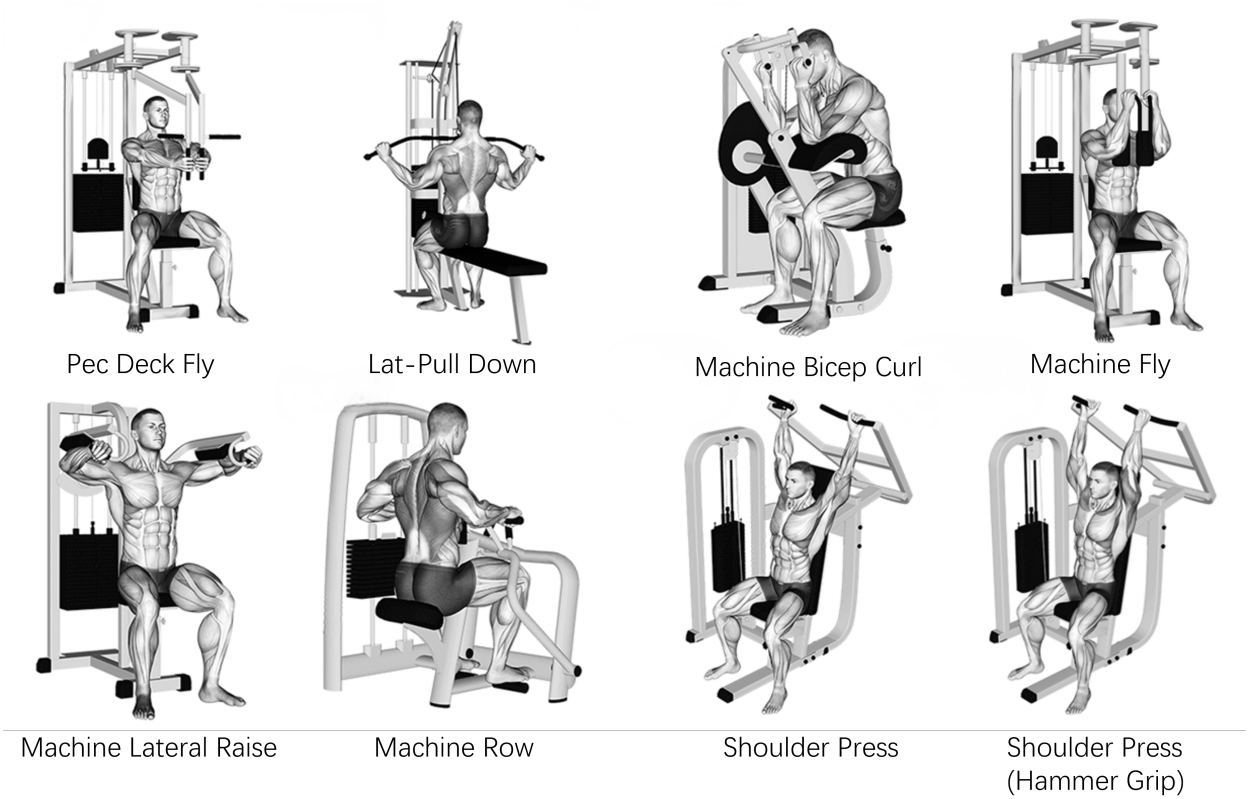


Figure 3.1: 8 Types of fitness activities

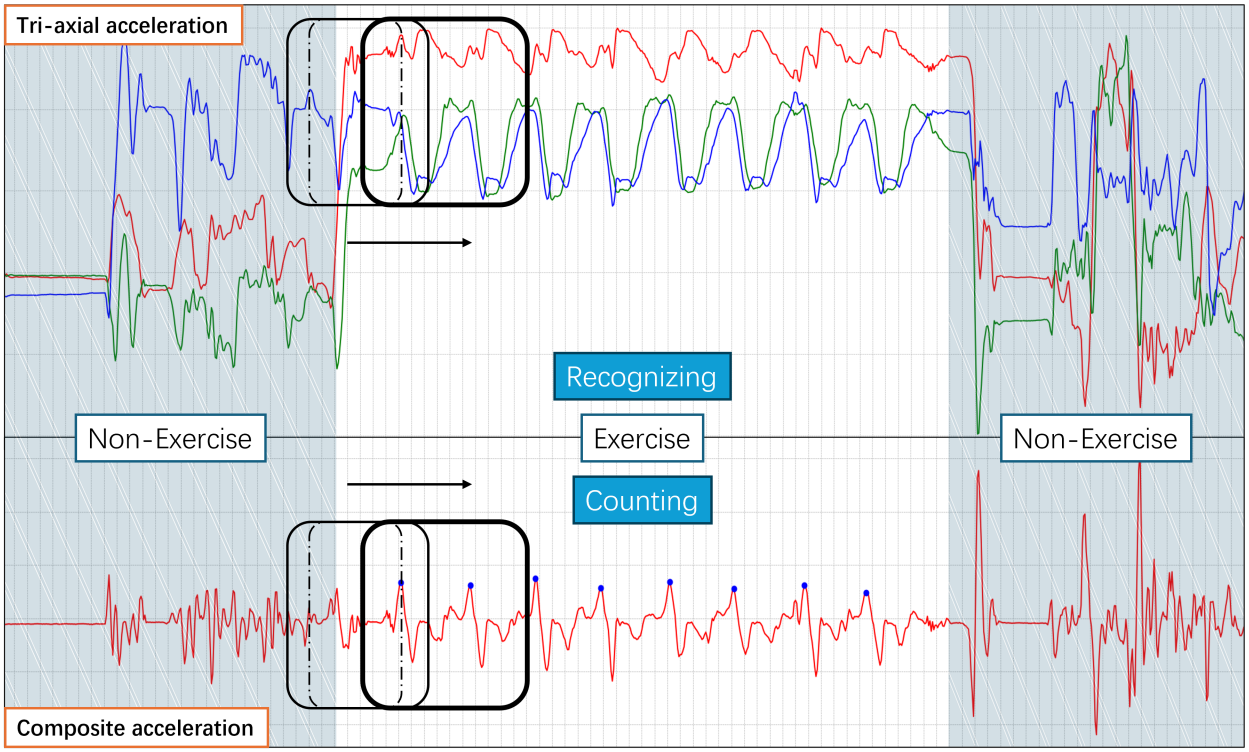


Figure 3.2: Algorithm working schematic

type of activity and outputs the results. Through these steps, the algorithm achieves real-time automatic tracking and recording of the user ' s fitness activities.

3.2 Overview of TLSW: A method for monitoring fitness exercise in real time

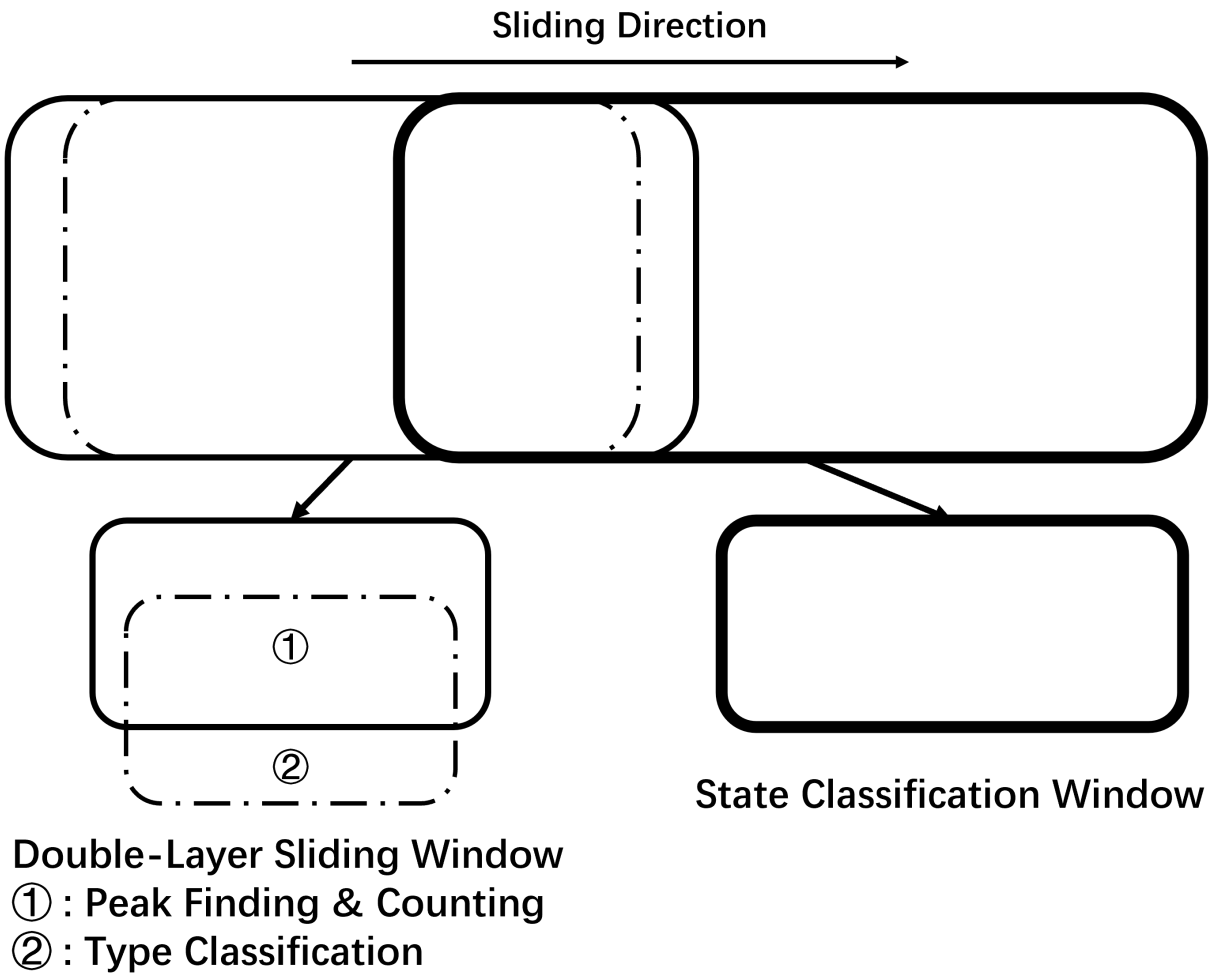


Figure 3.3: Structure of the TLSW algorithm

Figure 3.3 illustrates the algorithm's structure in this study, which utilizes a Triple-layer sliding window data structure with a total length of approximately 5.6 seconds. The structure slides one data point at a time, performing computations with each slide, and moves to the right (with new data entering the state classification window first). The state recognition window continuously classifies the activity state in real time, determining whether the user is engaged in a fitness activity. Once a fitness state is identified, the double-layer window is activated, where: **Layer[1]:** This layer identifies the peak positions within the window and determines whether the

peaks are centered. **Layer[2]:** This layer classifies the data within the window using a machine learning model for activity type classification.

This structure ensures efficient real-time state recognition, repetition counting, and activity classification.

Figure 3.4 presents the flowchart of the algorithm, with details of each component explained in the next section. The algorithm consists of two main parts:

1. **State Classification:**

   When the user presses the "Start Workout" button, the collected three-axis acceleration data enters the window for real-time state classification. The purpose of this step is to determine whether the user is exercising. A machine learning model trained to recognize states is used for classification:

   - If the classification result is "non-exercise," the accumulator - 1.
   - If the classification result is "exercise," the accumulator + 1.

2. **Type Classification and Counting:**

   When the state classification determines the user is exercising, this part is activated. First, peak detection is performed within the window to locate the positions of peaks. The algorithm then checks whether the peak position lies at the center of the window:

   - If the peak is not at the center, it is skipped.
   - If the peak is at the center, it is counted as a valid exercise repetition. Centered around the peak, the data within the window is classified using a machine learning model to identify the exercise type, which is recorded as one completed action of that type.

This dual-step process ensures accurate recognition of the user's activity state, repetition counting, and exercise type classification in real time.

3.3 Method for separating and extracting the fitness state

In this section, we will detail the working principle of the state classification window, as illustrated in the structural diagram. To eliminate the cumbersome process of manually starting and stopping between each set of exercises, we introduced a real-time state recognition feature. First, the system continuously detects the user's motion state in real time, automatically distinguishing between active (exercise) and inactive (non-exercise) states. Once the system identifies an active state, it proceeds to recognize the user's specific actions during exercise. Based

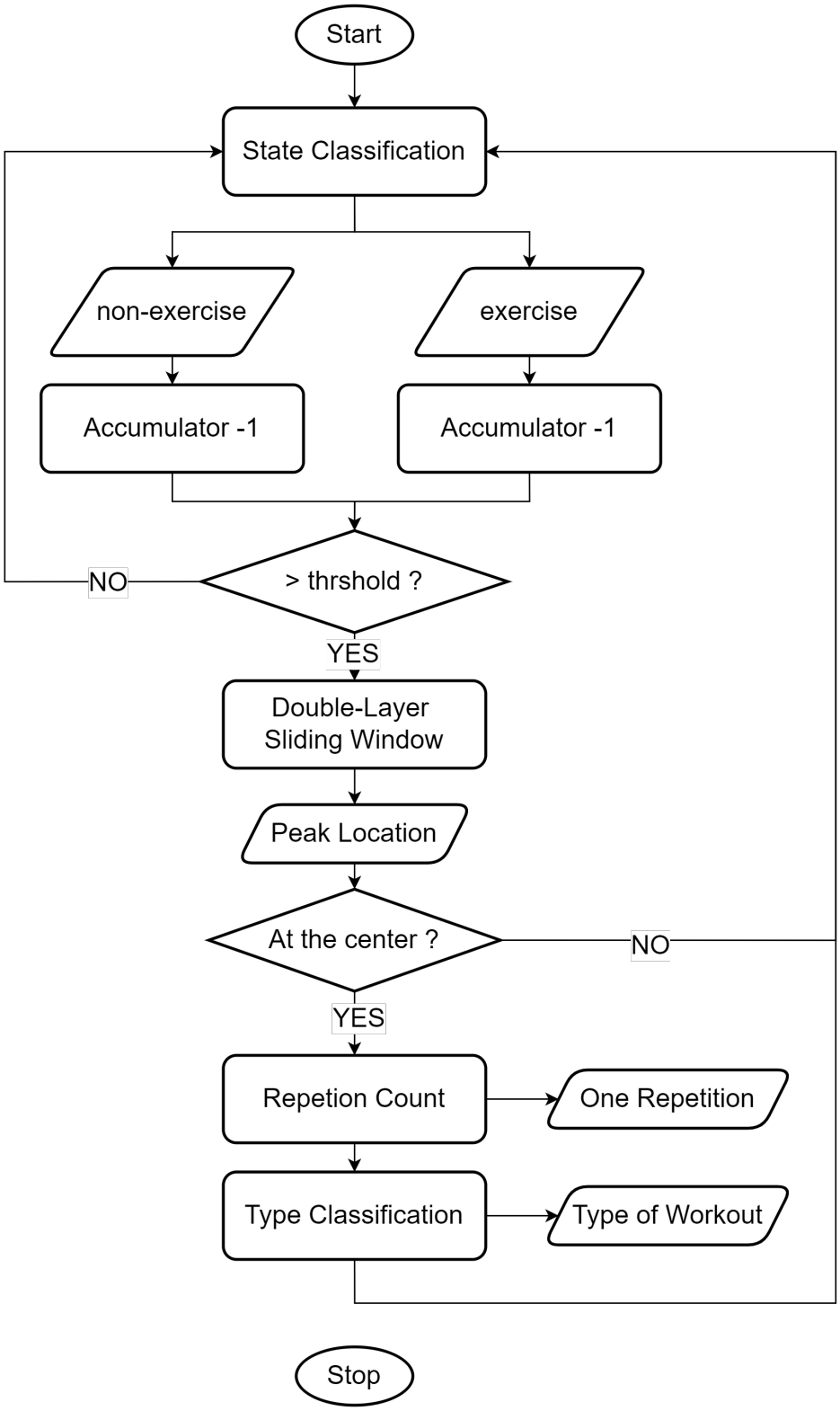


Figure 3.4: Flowchart of the TLSW algorithm

on data collection and analysis, we categorized the user's state during workouts into four distinct types. Figure 3.5 shows the corresponding three-axis acceleration data for these four types. This approach ensures efficient and seamless transitions between exercise detection and action recognition, significantly enhancing user convenience and workout tracking accuracy.

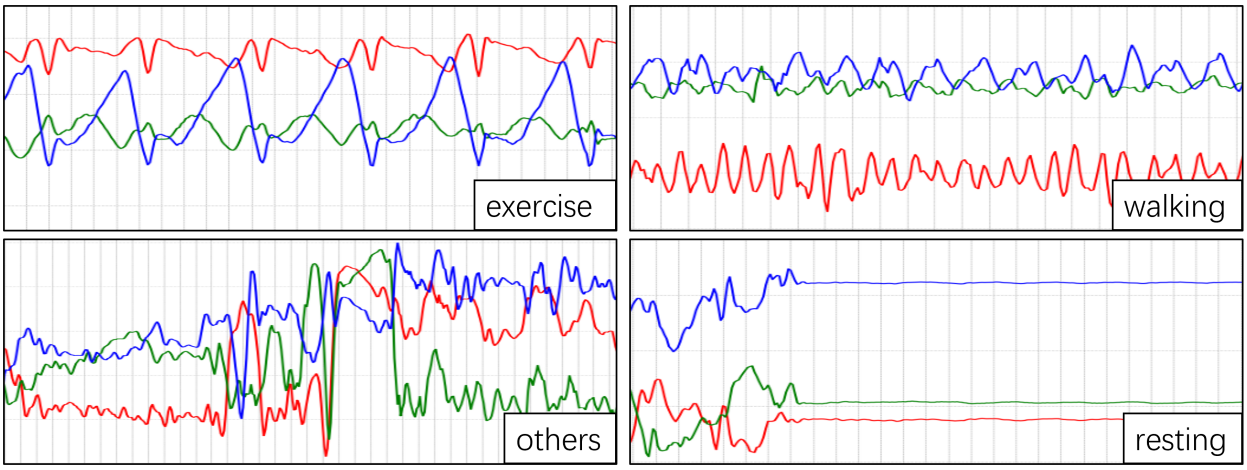


Figure 3.5: Triaxial acceleration diagrams for 4 states

The four states include exercise, walking, resting while seated, and other activities (e.g., drinking water, sanitizing gym equipment). Since this study focuses solely on processing the exercise state, all states other than exercise are defined as "non-exercise," while all target fitness activities are classified as "exercise."

To recognize motion states, the action recognition window is designed such that three-axis acceleration data enters the window, and every 5 data points (approximately 0.4 seconds), a machine learning model performs classification to determine the state. The window length is 70 data points, equivalent to approximately 5.6 seconds. Since activities like walking or repetitive non-exercise motions can often be misclassified as exercise, relying on a single classification result to determine the exercise state could lead to numerous errors. To address this, an accumulator system with a threshold is incorporated to optimize the decision process:

- When the classification result is "exercise," the accumulator increases by 1.

- When the result is "non-exercise," the accumulator decreases by 1.

The accumulator has a threshold value and a maximum limit. Once the accumulator value exceeds the threshold, the system transitions to the exercise state and activates the next processing stage. When the accumulator reaches its maximum limit, it will no longer increase.

Adjusting the threshold affects the ease of classifying the user as being in the exercise state.

For example, lowering the threshold allows quicker transitions to the exercise state but increases the risk of misclassifying other states as exercise. Similarly, modifying the difference between the maximum limit and the threshold adjusts the transition difficulty from exercise to non-exercise: A small difference results in a quick reversion to the non-exercise state after completing the last action in a set, potentially losing the count of the final action due to the delay. Conversely, a large difference may lead to non-exercise data being passed to the next stage, causing redundant counts.

In summary, adjusting the threshold and maximum limit influences the weights for transitions between **[non-exercise]** → **[exercise]** and **[exercise]** → **[non-exercise]** states. After multiple tests in this experiment, the delay is approximately 1.6 seconds. Under ideal conditions, the system determines the user has entered the exercise state 1.6 seconds after starting the workout.

3.4 Method of counting and recognizing fitness movements

This section details the workflow of the Double-Layer Sliding Window illustrated in the structural diagram. This component consists of an inner window and an outer window, both centered at the same position.

Outer Window (Peak Detection Window): The outer window is responsible for peak detection. It has a size of 60 data points, equivalent to approximately 4.8 seconds.

Inner Window (Action Recognition Window): The inner window handles action recognition. Its size is 50 data points, equivalent to approximately 4 seconds.

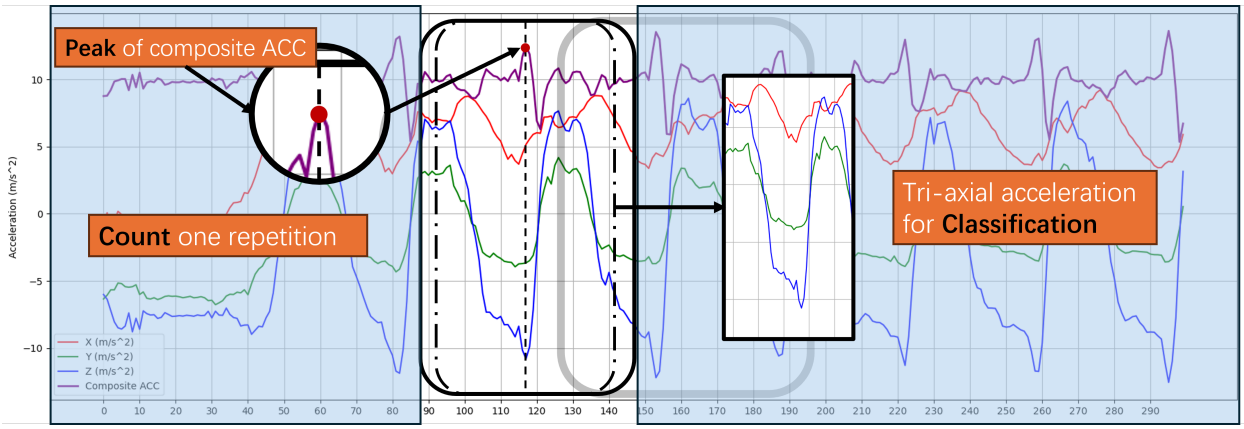


Figure 3.6: Schematic of the working of DLSW algorithm

Figure 3.6 shows the four steps of data processing in the Double-Layer Sliding Window algorithm:

1. Composite Acceleration Calculation Compute the Composite acceleration from the three-axis acceleration data entering the window.
2. Peak Detection Detect the peaks of the composite acceleration within the outer window. As the window slides, calculate the position of each peak.
3. Repetition Recording and Classification When a peak is located at the center of the outer window, record one repetition. Simultaneously, classify the fitness activity type using the three-axis acceleration data within the inner window.
4. Result Aggregation and Output Temporarily store each recorded repetition. When a transition out of the exercise state occurs, aggregate the classification results within the current set, select the majority result as the activity type for the set, and output the classification result along with the count.

In studies [20] and [22], repetition counting is performed by selecting a specific axis from the three-axis acceleration data for peak detection. However, in our research, we observed that for certain exercises, such as the Lat Pull Down(Figure 3.7), variations in the smartwatch's angle and position on the left wrist during data collection, along with differences in the motion angles of the exercises themselves, result in the most representative axis differing across actions.

To address this issue, we opted to compute the composite acceleration, allowing us to perform peak detection on a unified dimension for all exercises. This approach enhances the algorithm's adaptability and consistency across various fitness activities.

During the sliding of the window, the composite acceleration peaks for each action pass sequentially through the center of the window. The center point is used as a marker, and when a peak aligns with the center, it is recorded as one repetition. This method offers the following advantages:

- **Reduction in Duplicate Counting**

  By aligning peak positions with the center point, the method ensures that the same peak appearing in adjacent windows is not counted twice. This effectively reduces the likelihood of duplicate counts for the same action.

- **Improved Action Classification Accuracy**

  To enhance classification accuracy, special preprocessing was applied to the training data. Specifically, composite acceleration peaks were used as markers to segment each action into windows centered around the peaks. This approach minimizes errors caused by variations in user movement speed and ensures that the window data covers the primary or

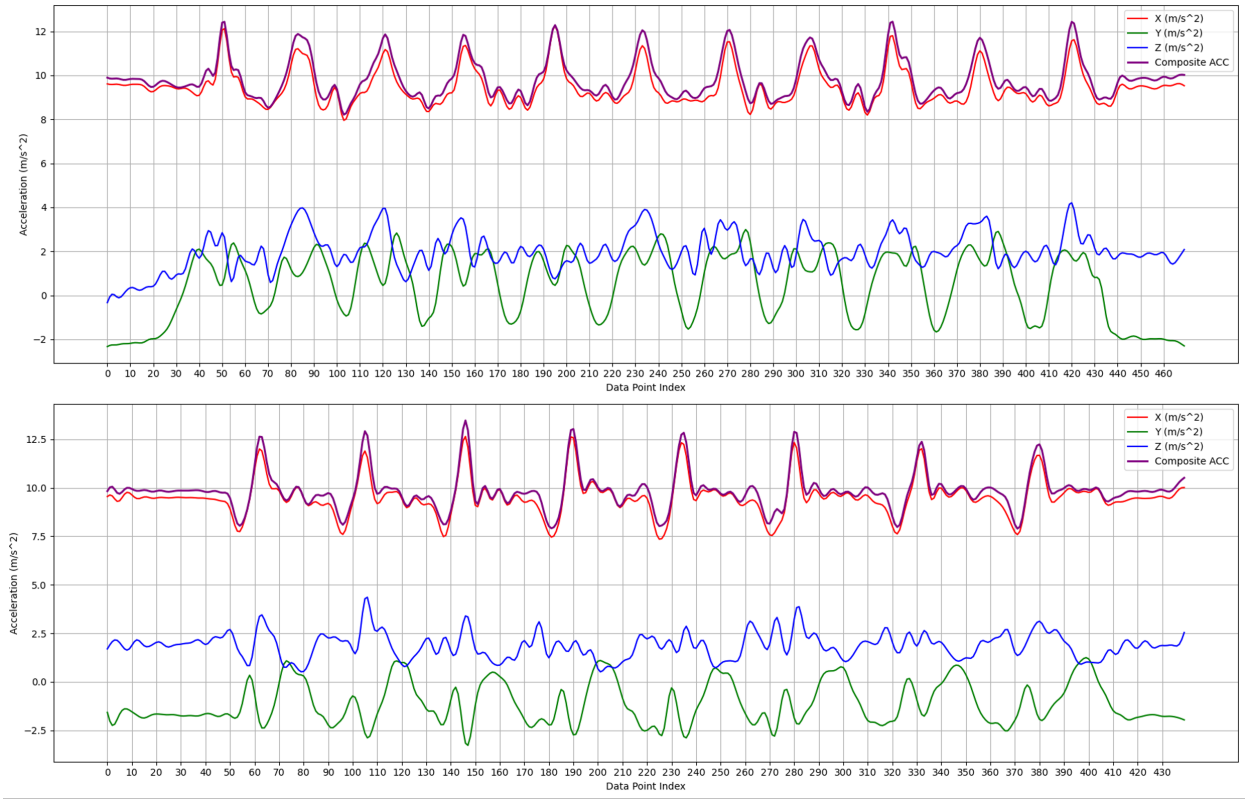


Figure 3.7: Composite acceleration and triaxial acceleration of Lat Put Down

complete segment of each fitness action. Even when the rhythm of the actions changes, data integrity is preserved, significantly improving model classification accuracy.

- **Reduced Computational Overhead**

  The algorithm mandates that composite acceleration peaks must pass through the window center. Ideally, the number of feature extractions and model classifications matches the actual number of fitness actions. Compared to fixed-step sliding window methods, this approach effectively reduces unnecessary computational costs.

Analysis revealed that the duration of a single user action typically ranges from 2.7 to 5 seconds. To maximize coverage of a single action ' s data while avoiding the inclusion of two complete actions in one window, we set the inner window size to 4 seconds. Additionally, we optimized the size of the outer window to reduce counting errors:

- **Issues with an Oversized Window**

  When users complete the last few repetitions in a set, their movements tend to slow down due to fatigue, resulting in lower acceleration peaks. An oversized window may include the peak from the previous action, which can interfere with detecting the next action ' s acceleration peak, ultimately reducing the count.

- **Issues with an Undersized Window**

  If the outer window is too small, secondary peaks between two consecutive actions may be mistaken for the synthetic acceleration peak of a full action. This can lead to oversegmentation of a single action and result in overcounting.

In some actions, a single motion may produce two closely spaced positive peaks of similar magnitude, along with a distinct negative peak in the opposite direction that can serve as a marker. Relying solely on peak-to-peak distance to filter out duplicate peaks may affect fast-paced actions or those with close peak spacing within a single motion.

To address this, we incorporated synthetic acceleration inversion into the peak detection process shows in Figure 3.8: Using gravity acceleration (9.8 m/s^2) as the reference value, if the difference between the minimum value and the reference exceeds the difference between the maximum value and the reference significantly, the synthetic acceleration data within the current window is inverted.The inverted data is then used for subsequent peak detection.

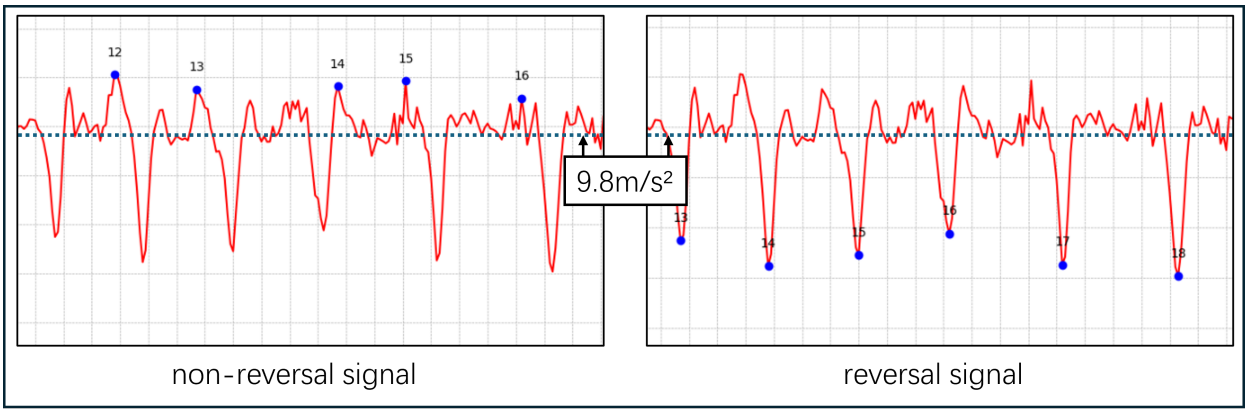


Figure 3.8: Reverse acceleration optimized for counting

Figure 3.9 illustrates the workflow for determining whether a peak position lies at the center:

Here are the detailed steps for calculating peak positions:

1. **Smooth Data**

   To eliminate noise interference in the acceleration data, a Simple Moving Average (SMA) is applied to the three-axis acceleration data (X, Y, Z). A window size of 3 is used to calculate the average of each data point and its neighboring points.

2. Calculate the Composite Acceleration

   Composite acceleration is computed from the smoothed three-axis acceleration data and serves as the foundation for subsequent processing.

3. **Calculate Whether to Reverse the Signal**

   To distinguish different phases of an action, the signal's direction is evaluated for potential inversion:

   Calculate the deviation of the maximum and minimum acceleration values from the standard gravitational acceleration (9.8 m/s^2).If the minimum value's deviation significantly exceeds the maximum value's deviation, the signal is inverted.The inversion ensures consistent peak orientation, facilitating more accurate peak detection.

4. **Calculate Relative Height**

   The relative height of peaks is calculated based on the maximum and minimum acceleration values within the window. A relative height threshold of 40% of the acceleration range is used to filter out minor fluctuations.The prominence of a peak is determined using the

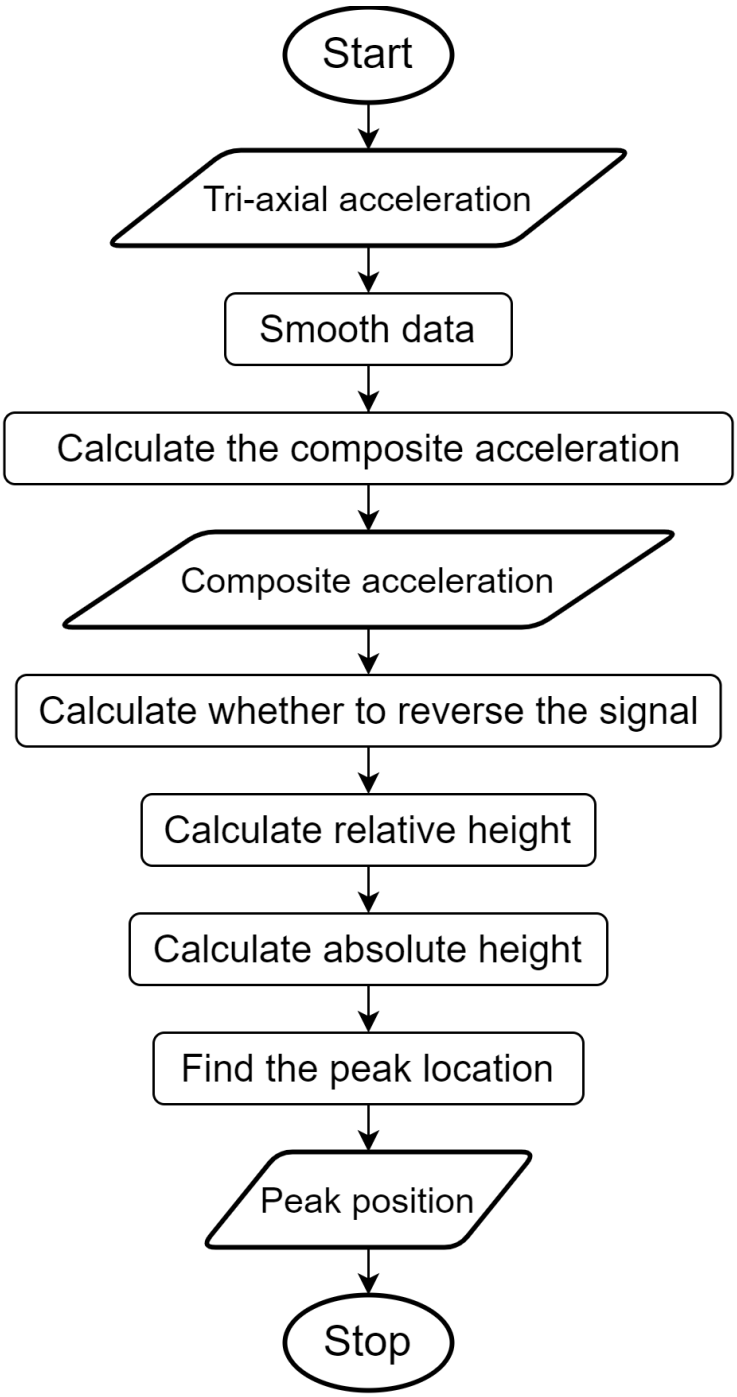


Figure 3.9: Process for finding peak locations

prominence parameter from Python's [scipy.signal.find_peaks] method.Prominence is defined as:

$$\text{Prominence} = \text{Peak Height} - \max(\text{Left Base}, \text{Right Base}) \tag{3.1}$$

The left and right bases represent the lowest points on either side of the peak.After testing, setting the prominence threshold to 40% of the acceleration range(Prominence=(max_acc − min_acc) * 0.4)yielded the best results.

5. **Calculate Absolute Height**

   Values below the gravitational acceleration (9.8 m/s^2) are adjusted to 9.8 to mitigate the impact of low values on calculations.The adjusted signal ' s mean is calculated as the baseline.A peak's absolute height threshold is set to 101.5% of the baseline (a 1.5% increment), filtering significant peaks related to actions while reducing false detections caused by general fluctuations.

6. **Find the Peak Location**

   Using the [scipy.signal.find_peaks] method in Python, peaks are filtered based on relative height, absolute height, and a minimum distance of 22 data points between peaks. This process identifies valid peaks in the window and calculates their positions.

3.5 State Recognition Model and Fitness Movement Recognition Model using Machine Learning

To support the above algorithm in classifying users' states and fitness activity types, we trained two machine learning models using common Python libraries such as *NumPy* and *scikit-learn*. Below are the details of the model training process.

3.5.1 Data collection

This study utilized the Google Pixel Watch 2 running Android OS 4.0 (API 34) to collect three-axis acceleration data, as shown in Figure 3.10. The device was worn on the left wrist of 10 volunteers. Data collection was conducted in the gymnasium of Aoyama Gakuin University's Sagamihara Campus, recorded at a sampling rate of 12.5 Hz.

In total, data for eight types of fitness activities were collected, including 1,701 fitness actions and approximately 120 minutes of non-exercise state data during the fitness sessions, as shown

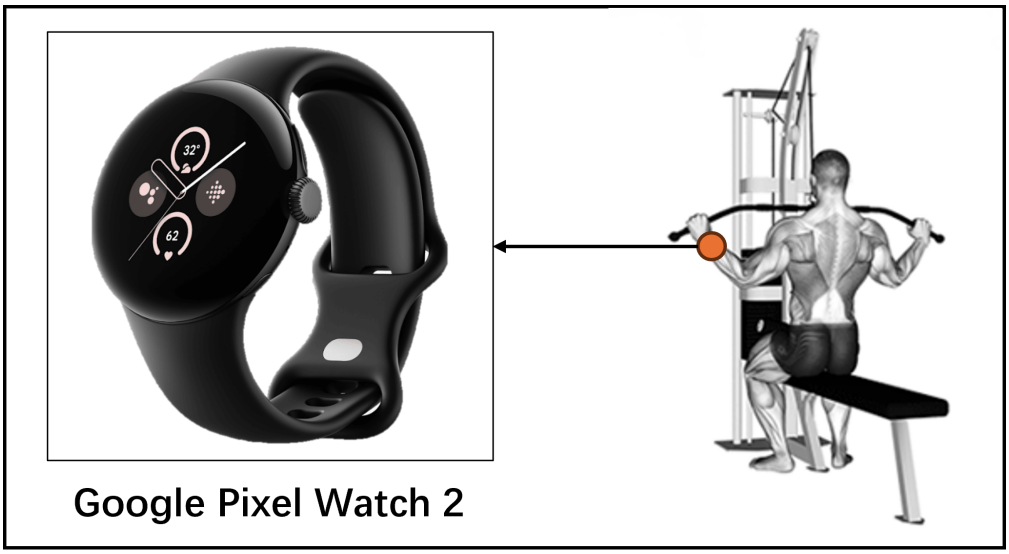


Figure 3.10: Devices and wear positions

Type	Number of movements	Label
Machine Row	220	1
Machine Bicep Curl	202	2
Machine Fly	205	3
Pec Deck Fly	201	4
Lat Pull Down	204	5
Machine Lateral Raise	233	6
Shoulder Press_1	234	7
Shoulder Press_2	202	8

Figure 3.11: Distribution in data sets

in Figure 3.11. Figure 3.12 illustrates the three-axis acceleration corresponding to each activity type.

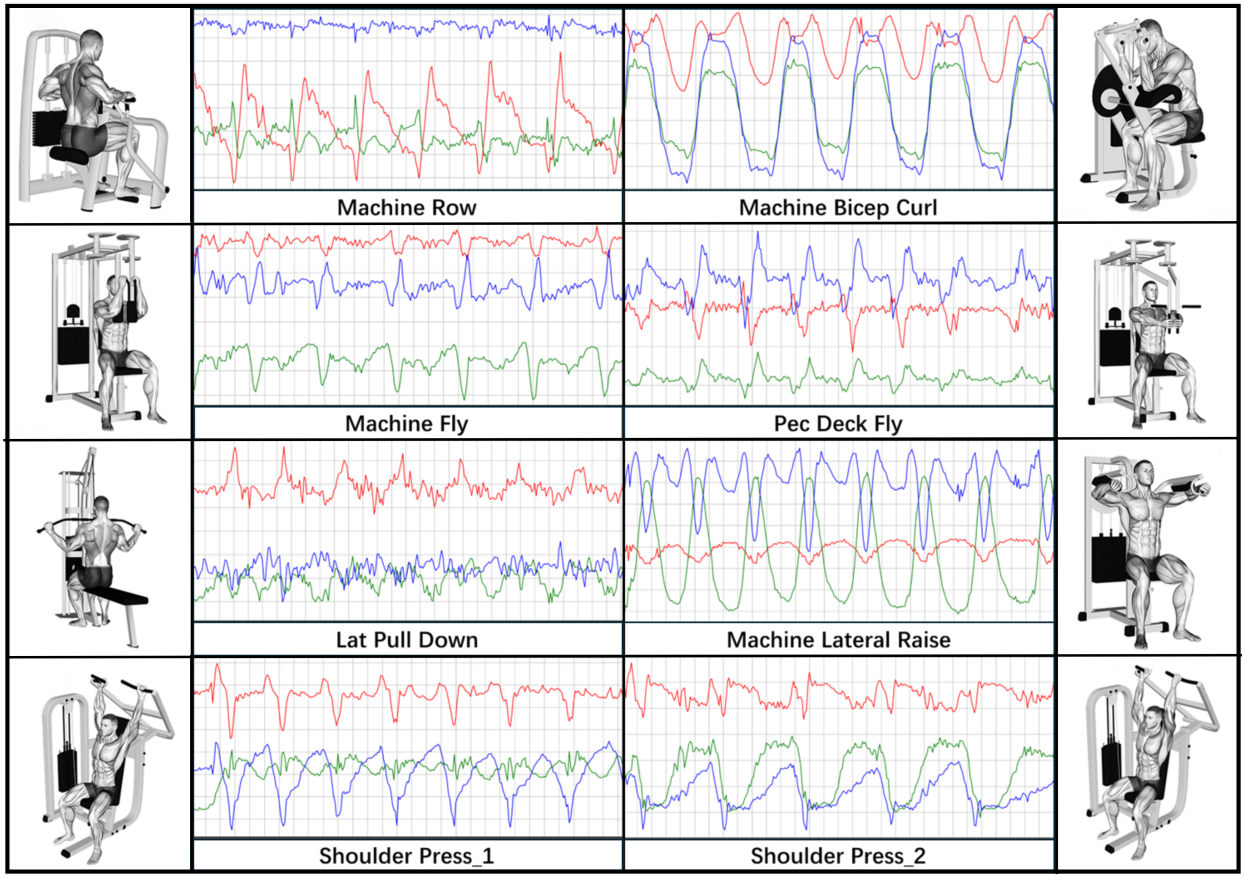


Figure 3.12: 3-axis acceleration for 8 types of movements

During data collection, volunteers performed various predefined fitness activities using designated gym machines. However, no strict requirements were imposed on speed, weight load, or repetitions per set. This ensured that the data closely resembled real-world fitness scenarios, providing valuable support for model training.

3.5.2 Data preprocessing

In this study, the collected acceleration data was smoothed to reduce the impact of potential outliers during data collection on subsequent analysis. To enhance computational efficiency, we applied a *sliding window averaging method* for smoothing. Specifically, the sliding window size was set to 3, and the value of each data point was calculated as the average of that point and its two adjacent points (one before and one after).

This method effectively smooths short-term fluctuations while preserving the primary trends and characteristics of the data, providing a more stable and reliable input for subsequent data segmentation and feature extraction.

3.5.3 Data set segmentation

After smoothing the data, it was segmented based on the requirements of different models:

1. **Segmentation for the State Classification Model**

   - Manual Separating:

     The data was manually cleaned to separate exercise states from non-exercise states within the same dataset. Exercise state data was saved as independent datasets for each set of actions, while non-exercise state data was saved in segments, ensuring continuity within each dataset and avoiding concatenation from different segments.

   - Window:

     The window size was set to match real-time recognition parameters, 70 data points with a stride of 5, to maintain consistency across data processing standards.

2. **Segmentation for the Type Classification Model**

   Following the approach of the Double-Layer Sliding Window (DLSW) algorithm, segmentation points were determined based on the positions of synthetic acceleration peaks:

   - Synthetic Acceleration Calculation:

     Composite acceleration was calculated from three-axis acceleration data.

   - Peak Detection:

     Peaks were detected using the *find_peaks* method in Python ' s *scipy.signal* library.

   - Windowing Around Peaks:

     For each peak position, a window of 50 data points centered around the peak was extracted, providing the input data for subsequent feature extraction.

3.5.4 Feature Selection

After splitting the dataset for each model, we first performed feature extraction in Python. Feature extraction is crucial, as the selected features directly affect classification accuracy. Choosing the right features means identifying key attributes that best represent the original data. In

other studies, researchers select different types of features based on the classification model they are building. These features are mainly divided into time-domain and frequency-domain features. Compared to time-domain features, frequency-domain features have a higher computational cost [18].

We referred to other studies and selected commonly used features such as maximum, minimum, median, kurtosis, skewness, zero-crossing rates, correlation coefficient, peak-to-peak value, and correlation between axes [28]. Below are the features used in this study for the state recognition model and the category recognition model. Some feature calculations were based on [28][29][30][31] and were implemented using the *NumPy* and *SciPy* libraries in Python.

Table 3.1: Feature Descriptions

Feature Name	**Explanation**
max	Maximum value in the window
min	Minimum value in the window
mdn	Median value
kts	Kurtosis
peak_value	Maximum absolute value in the window
peak_to_peak_value	Difference between maximum and minimum values
skewness	Measures the degree of skewness of the data distribution relative to a symmetric distribution
sma	Signal Magnitude Area [32]
num_prominent_peaks_3	Number of peaks with prominence > 0.3
num_prominent_peaks_5	Number of peaks with prominence > 0.5
num_prominent_peaks_7	Number of peaks with prominence > 0.7
max_peak_value	Peak value with prominence > 0.7
spectral_centroid	Weighted average of signal frequency distribution
spectral_flatness	Indicator of the uniformity of the signal's spectral distribution
cor_xy	Correlation coefficient of x-axis and y-axis
zcr	Zero-Crossing Rate
total_energy	The power spectral density of the signal is calculated and integrated to obtain the total energy of the signal.
bandpass_energy_0_2_0_3	Total energy in the frequency range 0.2 Hz to 0.3 Hz
bandpass_energy_0_4_0_6	Total energy in the frequency range 0.4 Hz to 0.6 Hz
bandpass_energy_0_7_0_8	Total energy in the frequency range 0.7 Hz to 0.8 Hz

The calculation methods for commonly selected features such as maximum and minimum are omitted here. Instead, we will focus on explaining the calculation methods for other features specifically defined in this study.

- **Signal Magnitude Area**

  The normalized Signal Magnitude Area (SMA) reflects the overall average absolute magnitude of a signal and is used to quantify its intensity or energy characteristics in a specific dimension. In time series analysis, it helps compare the average magnitude of different signals, such as the acceleration characteristics in various directions during fitness movements.

$$x_{\text{sma}} = \frac{\sum_{i=1}^{N} |x[i]|}{N}$$

  – $|x[i]|$: The absolute value of the i-th sample of the signal;

  – N: The total number of samples in the signal;

  – $\sum |x[i]|$: The sum of the absolute values of the signal amplitude.

- **num_prominent_peaks**

  We found that the number of prominent peaks exceeding a certain threshold differs between fitness and non-fitness states. Therefore, we included the number of peaks with prominence greater than 0.3, 0.5, and 0.7 as features. Using the *'find_peaks'* method with a specified prominence parameter, we detected significant peaks in the signal and recorded their count as feature values.

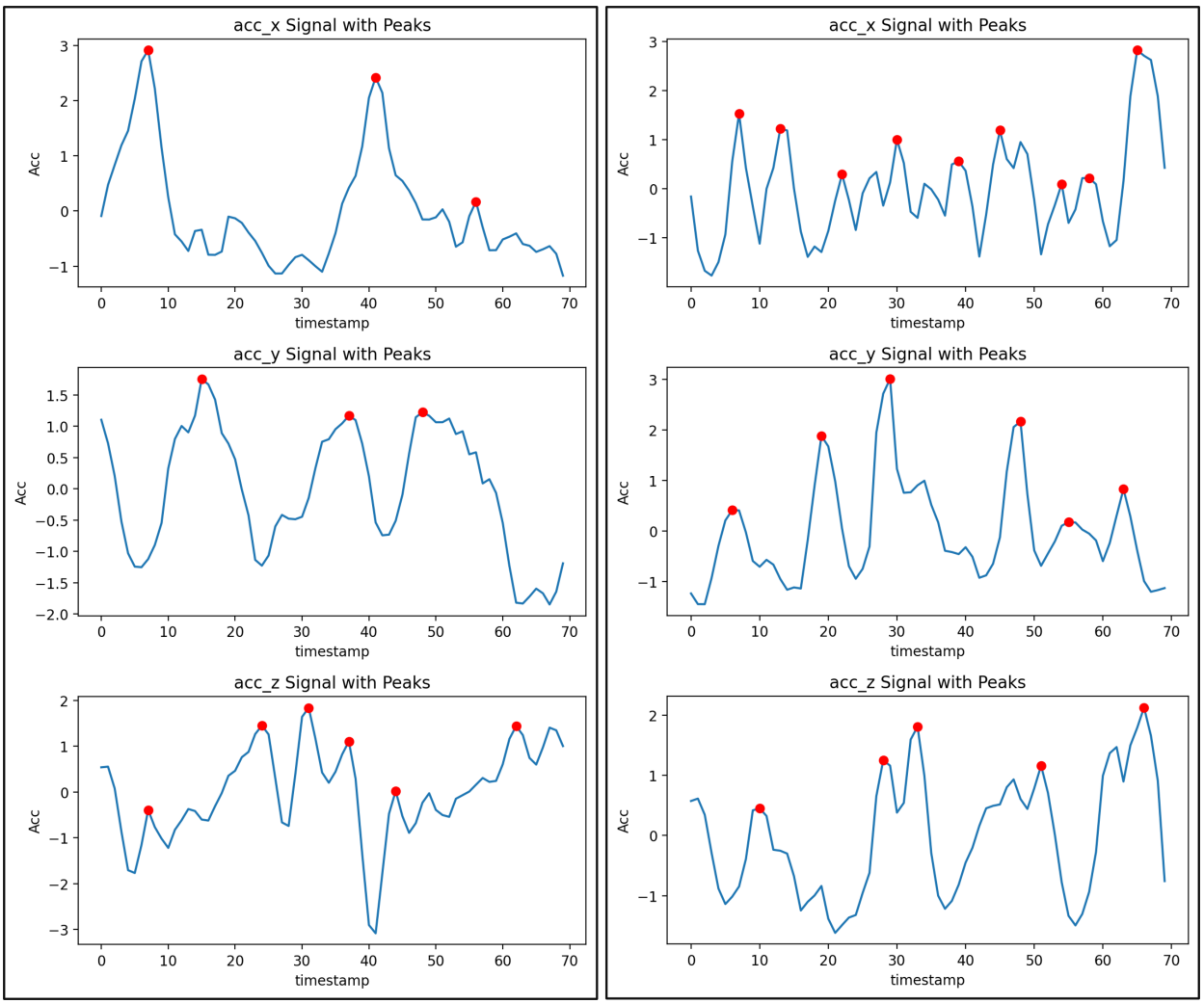


Figure 3.13: The number of peak points exceeding the threshold in three-axis acceleration: (left) fitness state, (right) non-fitness state.

- **max peak value**

  The algorithm detects peaks in the signal based on the specified prominence parameter (prominence > 0.7) and returns the maximum value among these peaks. If no peaks are found, it returns 0.

- **Power Spectral Density**

  Next, to compute frequency features, we analyzed the peak frequency distribution of three-axis acceleration within each segmented data window. First, we calculated the power spectral density (PSD) for each window using the ‘welch‘ method from the ‘scipy.signal‘ library. This method analyzes the energy distribution of the signal in the frequency domain by segmenting the signal, computing the spectrum for each segment, and averaging the results.

To accommodate the characteristics of the signal and the limitations of data length, we set the segment size for Welch ' s method to 50 data points. Additionally, adjacent segments had an overlap of 10 points (20% overlap) to further smooth the PSD curve and reduce estimation variance. With this configuration, each window of the signal was first processed with a Hanning window to minimize edge effects, followed by a Fast Fourier Transform (FFT) to compute its spectrum. The squared values of all windowed spectra were then averaged to generate the final PSD curve.

$$S_{xx}(f) = \frac{1}{L}\sum_{k=1}^{L}\frac{1}{n_{\text{perseg}}}\left|\sum_{n=0}^{n_{\text{perseg}}-1} x_k[n]\,w[n]e^{-i2\pi f n\Delta t}\right|^2$$

- **spectral centroid**

  In the analysis of acceleration signals during fitness activities, the spectral centroid represents the weighted average of the signal's frequency spectrum, reflecting the primary frequency characteristics of the movement. A lower spectral centroid value indicates that low-frequency components dominate the signal, which is typically associated with slower or more controlled movements. In contrast, a higher spectral centroid suggests that high-frequency components are dominant, often corresponding to faster and more dynamic movements [30]. We calculate the spectral centroid based on the previously computed PSD, using the following formula:

  For a given frequency array $f[i]$ and PSD values $S[i]$, the spectral centroid C is defined as:

$$C = \frac{\sum_i f[i]\cdot S[i]}{\sum_i S[i]}$$

  - $f[i]$: The i-th frequency point;
  - $S[i]$: The PSD value corresponding to the i-th frequency point;
  - The numerator is the weighted sum of frequencies by their corresponding power;
  - The denominator is the total power, used for normalization.

- **spectral flatness**

  Spectral Flatness is calculated as the ratio of the geometric mean to the arithmetic mean of the spectrum and is used to measure the uniformity of the spectral distribution.

  First, the magnitude spectrum is obtained by applying the Fast Fourier Transform (FFT) to the input signal. Then, the geometric mean and arithmetic mean of the spectrum are

computed. The geometric mean is derived by taking the logarithm of the magnitude values, averaging them, and then applying the exponential function. The arithmetic mean is simply the average of the spectral magnitudes.

Finally, spectral flatness is obtained by dividing the geometric mean by the arithmetic mean. This metric reflects the frequency distribution characteristics of the signal: higher spectral flatness indicates a more uniform energy distribution across frequencies (e.g., white noise), while lower spectral flatness suggests that energy is concentrated at a few specific frequencies.

The Spectral Flatness is defined as:

$$\text{Spectral Flatness} = \frac{\left(\prod_{i=1}^{N}\left(S_x(f_i)+\epsilon\right)\right)^{\frac{1}{N}}}{\frac{1}{N}\sum_{i=1}^{N}\left(S_x(f_i)+\epsilon\right)}$$

- $S_x(f_i)$: The power spectral density (PSD) corresponding to the i-th frequency point;
- N: The total number of frequency points;
- $\epsilon = 1 \times 10^{-10}$: A small constant added to avoid division by zero or taking logarithms of zero.

- **Total energy**

  By computing the total energy of the Power Spectral Density (PSD), the overall energy distribution across the entire frequency domain can be quantified. This reflects the overall intensity characteristics of the signal, providing a measure of movement intensity or activity level.

  The total energy is calculated by summing the PSD values across all frequency points and multiplying by the frequency resolution $\Delta f = \text{freqs}[1] - \text{freqs}[0]$. The formula is as follows:

$$E_{\text{total}} = \sum_{i} S_x(f_i) \cdot \Delta f$$

  where $S_x(f_i)$ represents the power spectral density value at the ith frequency point.

- **bandpass_energy**

  To analyze the energy distribution of the signal within specific frequency ranges, we computed the Bandpass Energy within each window. First, we determined the dominant frequency range by calculating the signal's frequency distribution using Power Spectral Density (PSD). Then, we integrated the PSD within this frequency band to obtain the total energy in that range.

After extracting the peak frequency from the PSD in each window, we used Kernel Density Estimation (KDE) to visualize the distribution of dominant frequencies. KDE is a non-parametric method that smooths discrete frequency distributions into a continuous probability density function, revealing the concentration and overall trend of frequency components.

For implementation, we used the kdeplot function from the Seaborn library to generate KDE visualizations. The results are shown in the figure.

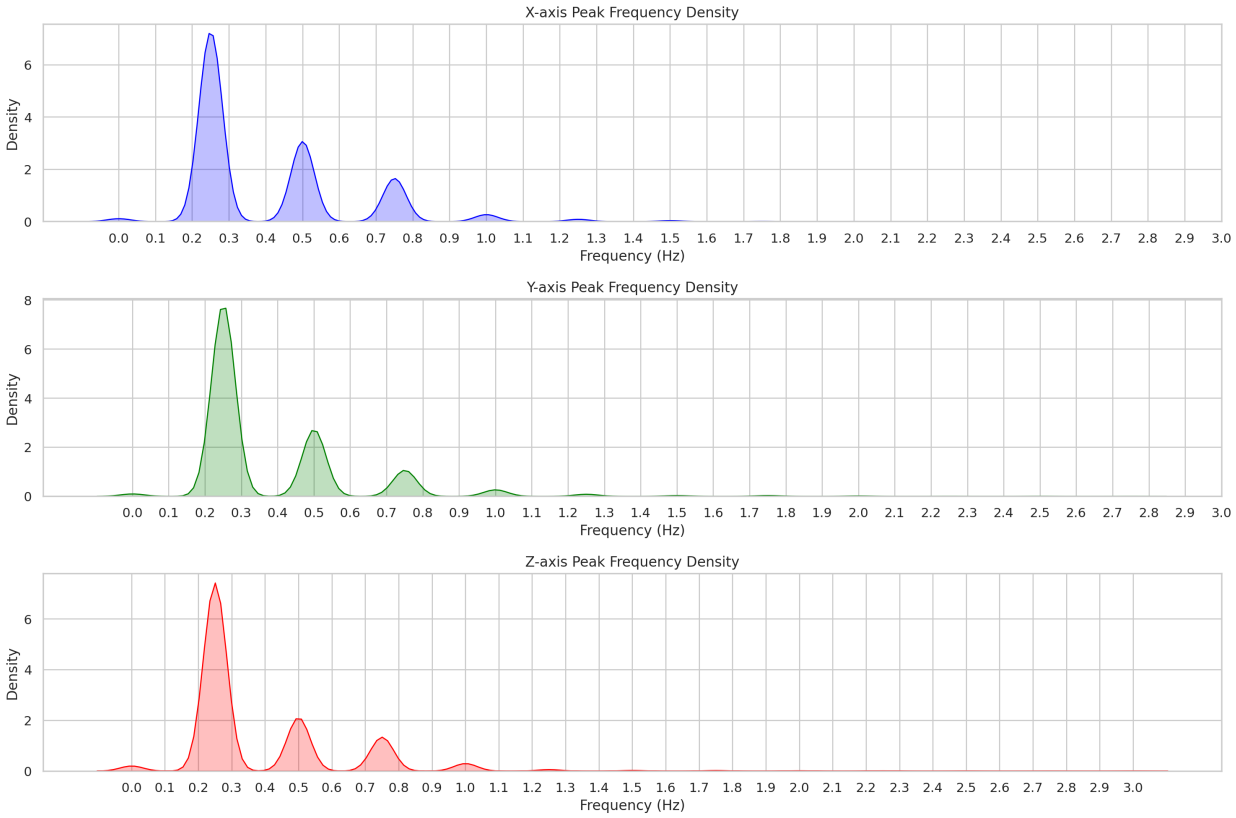


Figure 3.14: Main Frequency Distribution in Exercise State

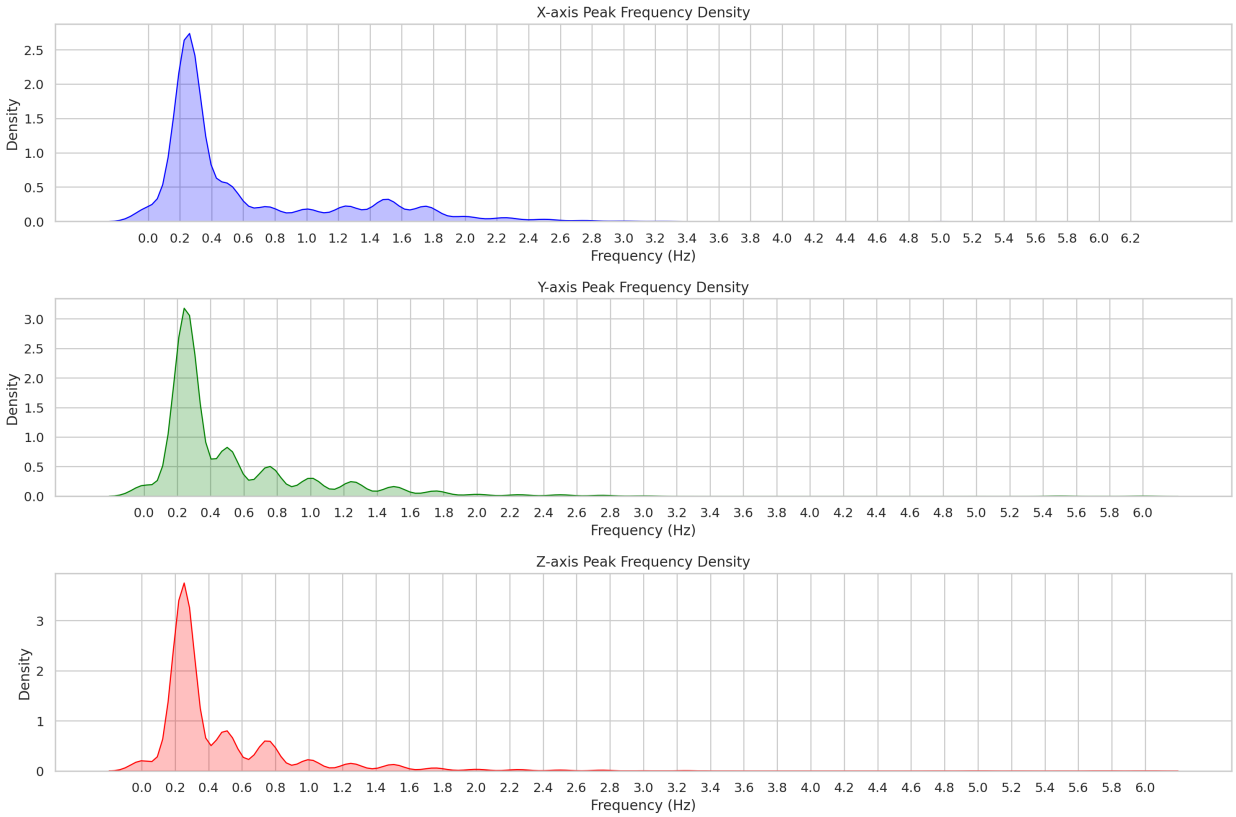


Figure 3.15: Main Frequency Distribution in Non-Exercise State

From the figure, it can be observed that in the exercise state, the dominant frequencies are concentrated in the ranges of 0.2–0.3 Hz, 0.4–0.6 Hz, and 0.7–0.8 Hz, whereas in the non-exercise state, the frequency distribution is more dispersed.

We then used the following formula to compute the total energy within each of these three frequency ranges.

$$E_{\text{bandpass}} = \sum_{i \in \text{band}} S_x(f_i) \Delta f$$

- $S_x(f_i)$: The PSD value at the frequency point f_i;
- $\Delta f = \text{freqs}[1] - \text{freqs}[0]$: The frequency resolution;
- band: The set of frequency points within the bandpass range.

The figure shows the differences in total energy across different frequency ranges between the exercise and non-exercise states. Therefore, we defined the total energy in each of the three frequency ranges as separate feature values.

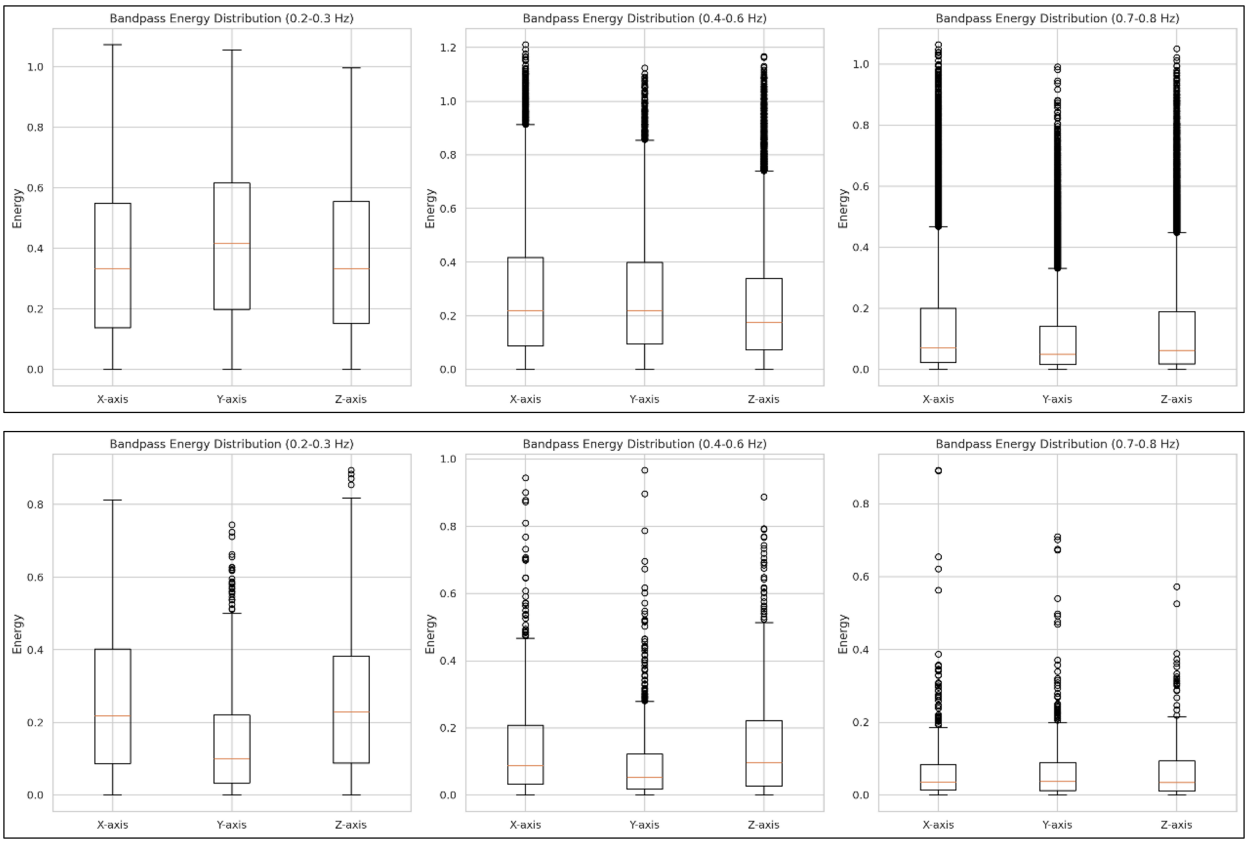


Figure 3.16: Comparison of Energy in Three Frequency Ranges for Different States: (Top) Exercise State (Bottom) Non-Exercise State

- **Zero Crossing Rate**

  Zero Crossing Rate (ZCR) is a frequency-domain feature that measures the rate of signal changes by calculating the proportion of zero crossings within a unit time. It is determined by detecting sign changes between adjacent sampling points, counting the total number of zero-crossings, and dividing this count by the signal length.

$$\mathrm{ZCR} = \frac{1}{2}\sum_{n=2}^{N}\left|\mathrm{sign}(x_n) - \mathrm{sign}(x_{n-1})\right|$$

- **Correlation Coefficient**

  By computing the correlation coefficients between the X, Y, and Z axes of the acceleration signal, the relationship between different directional movements is quantified. Specifically, the X, Y, and Z components of the signal are first extracted, and then the Pearson correlation coefficients for X-Y, X-Z, and Y-Z are calculated using *np.corrcoef*.

$$\text{cor}(X,Y) = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$$

3.5.5 State Classification Model Training and Feature Reduction

We referred to previous studies [30][33] that summarized commonly used machine learning methods for state recognition and selected Random Forest, Decision Tree, SVM, K-Means, and Naïve Bayes for comparison. The models were first trained in Python, and their performance was evaluated using accuracy and F1-score. The results are shown in the table below.

Table 3.2: Comparison of State Classification Model Performance

Model	**F1-Score**	**Accuracy**
Random Forest	0.972	0.974
Decision Tree	0.927	0.922
SVM	0.919	0.915
K-MEANS	0.968	0.967
Naive Bayes	0.762	0.776

Similar to findings in other studies, the Random Forest algorithm demonstrated strong performance, making it the chosen method for training the state recognition model.

Since the model is intended to run on an offline device, dimensionality reduction was necessary to lower computational complexity. To achieve this, we combined Gini-based feature selection [34][35][36] and SHAP (Shapley Additive Explanations) [37][38] to evaluate feature importance.

In Random Forest, the Gini Impurity is a key metric for measuring node purity and assessing each feature ' s contribution to classification decisions. By analyzing feature importance, we performed feature selection and dimensionality reduction, retaining only the most valuable features for classification. This approach helps reduce computational complexity and mitigates the risk of overfitting.

The Gini Impurity quantifies classification impurity and is calculated using the following formula:

$$G = 1 - \sum_{k=1}^{K} p_k^2$$

- K: The number of classes;

- p_k: The proportion of samples belonging to the k-th class in the current node.

The smaller the Gini Index, the purer the samples in the current node, and the better the classification effect. During the training process of a Random Forest, the Gini Index is used to measure the change in purity before and after the split.

SHAP (SHapley Additive exPlanations) is based on Shapley values from game theory, which provide a method for fairly distributing rewards. In machine learning, Shapley values quantify each feature's average marginal contribution to the overall prediction, ensuring fairness, consistency, and efficiency in explanations.

Within the SHAP framework, this method is used to assess each feature ' s contribution to model predictions. The core idea is to evaluate all possible subsets of features, compute the impact of adding a specific feature to each subset, and assign contribution values using a weighted average. This approach provides an interpretable explanation of the model's decision-making process.

Specifically, we set the number of trees (n_estimators) between 50 and 150, increasing by 5 each time, and evaluated model accuracy using 5-fold cross-validation. The results showed that model performance stabilized when the number of trees ranged between 75 and 125. Next, we refined the search within this range, increasing by 1 each time and performing another 5-fold cross-validation, ultimately determining the optimal number of trees to be 100.

Based on this, we further tuned the tree depth (max_depth) from 1 to 11, again using 5-fold cross-validation, to determine the best depth parameter. Using the optimized model, we then calculated the contribution of each feature to the prediction results.

Finally, by combining the results from Gini-based selection and SHAP analysis, we selected the top 25 most important features. The final selected features are shown in the table below.

Next, we extracted data for these selected features and split it into a training set and test set with an 80:20 ratio. We trained the model using 10-fold cross-validation with the GridSearchCV method to fine-tune hyperparameters. The model's performance was evaluated using accuracy and F1-score, and a confusion matrix was generated to visualize the classification results.

The final results are shown in the figure below.

Table 3.3: Feature Names in State Classification Model

Feature Names	
x_zcr	x_spectral_flatness
x_num_prominent_peaks_7	y_num_prominent_peaks_3
x_spectral_centroid	x_bandpass_0_4_0_6
y_spectral_centroid	z_zcr
x_bandpass_0_7_0_8	y_zcr
	y_peak_value
y_bandpass_0_2_0_3	x_total_energy
x_max_peak	y_kts
y_bandpass_0_4_0_6	y_total_energy
z_spectral_flatness	y_spectral_flatness
cor_yz	x_skewness
x_sma	x_peak_to_peak_value
y_peak_to_peak_value	y_sma

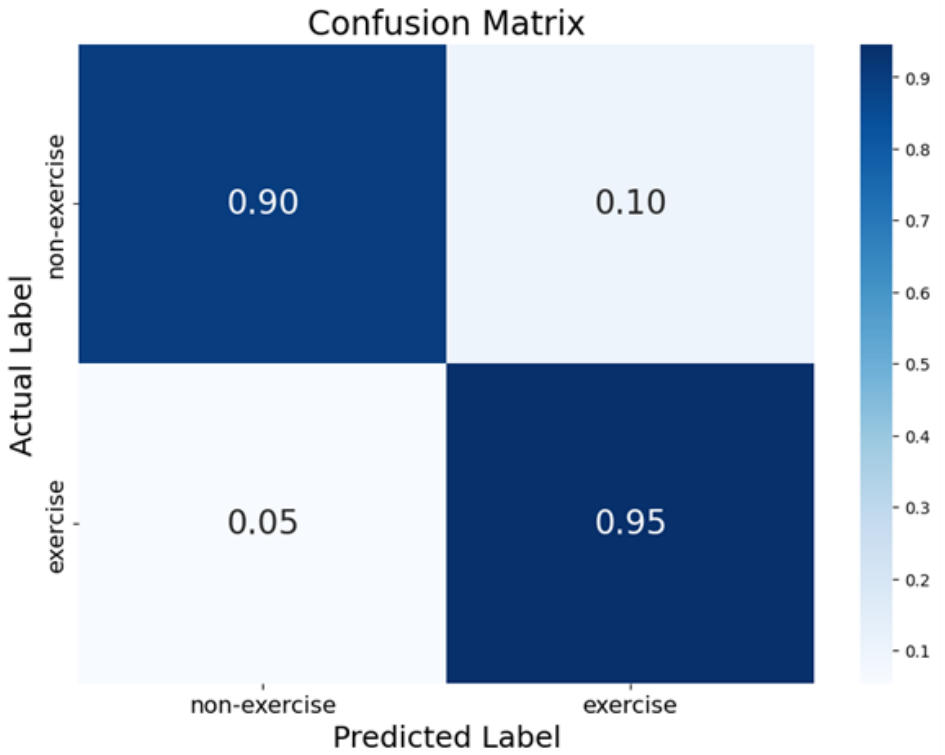


Figure 3.17: Confusion Matrix for State Classification Models

The results showed an accuracy of 92.1% and an F1-score of 0.917. The recognition accuracy for the fitness state was particularly high, likely because the selected features were more distinc-

tive in fitness-related movements. This aligns with our objective of assigning higher weight to the fitness state for improved classification performance.

3.5.6 Type Classification Model Training and Feature Reduction

Similar to the state classification model, we first created a dataset using all features. Then, we trained multiple machine learning classification models for comparison. The results are as follows:

Table 3.4: Comparison of Type Classification Performance

Model	**F1-Score**	**Accuracy**
Random Forest	0.948	0.943
Decision Tree	0.871	0.868
SVM	0.941	0.940
K-MEANS	0.940	0.936
Naive Bayes	0.894	0.887

Following the same approach as in the state classification model, we first trained a random forest model using all features and fine-tuned its key parameters to assess feature importance.

In the initial tuning phase, we performed cross-validation to evaluate the impact of different parameter combinations on model accuracy, identifying a stable parameter range. We then refined the tuning process within this range to determine the optimal combination of parameters, including the number and depth of trees.

Based on the optimized random forest model, we computed the contribution of each feature to classification performance. From the feature importance analysis, we selected the most significant features and split them into training and test sets. As shown in the table, most of the high-contribution features are related to acceleration along the X-axis, which aligns with the characteristics of our target movements—most actions exhibit significant variations along the smartwatch ' s X-axis direction.

For further model training, we used grid search and cross-validation to optimize performance, ensuring stability and generalization. Finally, we evaluated the model using accuracy and F1-score, and visualized the classification results through a confusion matrix, leading to the selection of the best-performing model.

Table 3.5: Feature Names in Type Classification Model

Feature Names	
cor_xy	cor_xz
cor_yz	x_bandpass_energy
x_kts	x_max
x_max_peak_value	x_min
x_peak_to_peak_value	x_peak_value
x_skewness	x_spectral_centroid
y_max	y_mdn
y_min	y_num_prominent_peaks
y_peak_to_peak_value	y_peak_value
y_skewness	y_spectral_centroid
z_bandpass_energy	z_skewness
z_spectral_centroid	z_zcr

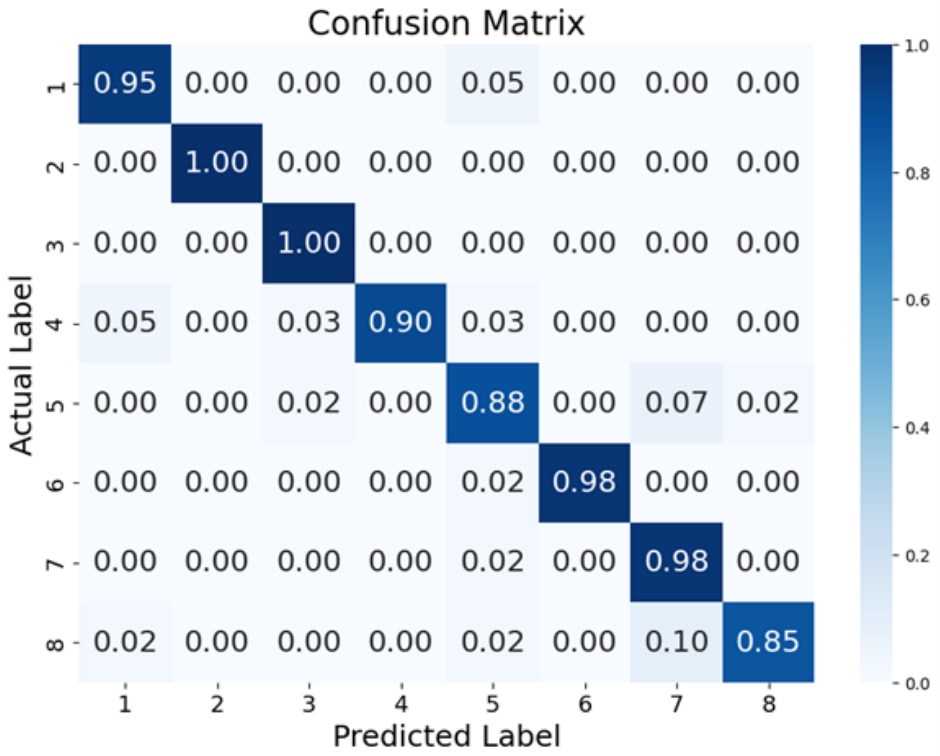


Figure 3.18: Confusion Matrix for Type Classification Models

The category classification model achieved an accuracy of 97.1% $\pm$ 2.1 and an F1-score of 0.97 $\pm$ 0.022. Among the classifications, Lat Pull Down (Class 5) and Shoulder Press_1 (Class 7) were more prone to misclassification. We speculate that this is due to the similarity in movement

patterns between the two exercises. After standardization, the range of variation in three-axis acceleration was reduced, making it harder to distinguish between them.

Chapter 4

Algorithm offline experiments and results

Before developing the Android application incorporating this algorithm, we first conducted offline experiments to verify its performance and applicability. The experiments were divided into two main parts: DLSW algorithm validation for category classification and TLSW algorithm validation for overall performance. All offline experiments were conducted in a Python environment on Google Colaboratory.

4.1 Experimental data collection

We first collected the necessary experimental data with the assistance of seven male volunteers aged 21 to 26, all with prior fitness experience. Before data collection, volunteers were informed of the available fitness activities and created personal workout plans based on their preferences. Each participant selected 2–3 types of exercises, performing at least two sets per exercise.

Before starting the workout, volunteers engaged in light physical activity as a warm-up, during which data collection began. Volunteers then followed their planned workout routines, and data collection ended when they either completed their workout or could no longer continue.

In total, we collected approximately 3.3 hours of data, including 892 recorded fitness movements and around 2.4 hours of non-exercise data.

4.2 DLSW Algorithm Experiment

In the Double-Layer Sliding Window (DLSW) algorithm test, we segmented and separately saved each set of fitness activity data. We then conducted comparative testing in a Python environment, using a traditional fixed-step sliding window approach with step sizes set to 25%, 50%, and 75% of the window size.

Following the same classification method as in the algorithm, we recorded the prediction results for each set of data. To evaluate the performance, we used two key metrics:

1. Probability of Correct Single Classification – Measures the classification accuracy for each individual sliding window.

2. Probability of Correct Classification in the Result – Evaluates the proportion of correctly classified target labels within a set of actions.

Since the algorithm determines the final classification based on the most frequently predicted label within a set of movements, minor misclassifications do not significantly affect the overall result. These two metrics effectively reflect the algorithm's classification performance.

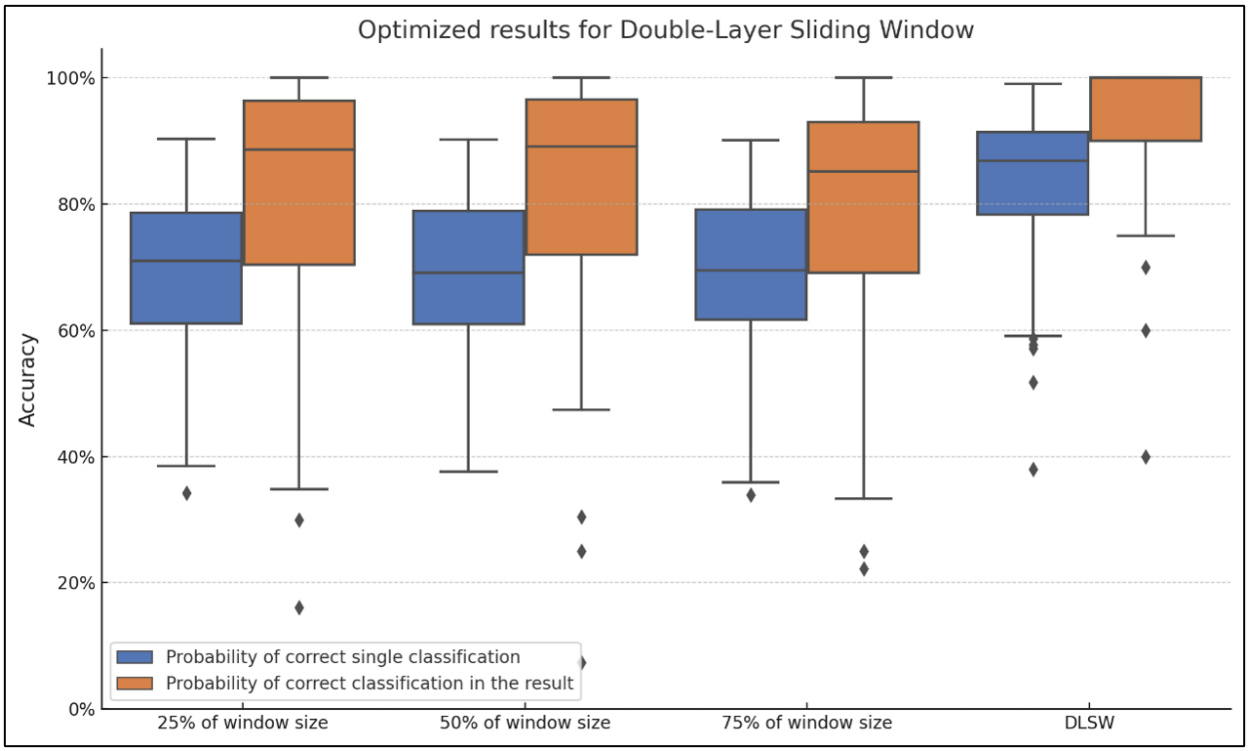


Figure 4.1: Optimized results for Double-Layer Sliding Window

The results showed that, compared to the fixed-step sliding window, this method reduces the number of computations, lowering computational overhead while maintaining better accuracy in both single classification and correct label proportion within a set of movements.

During testing, we observed that when movements were slower, the sliding window often included a higher proportion of transition data between two actions, leading to a decrease in classification accuracy. In such cases, using the peak point within the window as the center for classification proved to be a more effective approach.

4.3 TLSW Algorithm Experiment

For the TLSW algorithm experiment, we first simulated a real-time environment that closely resembles actual usage conditions. In this test, we used the collected experimental dataset and strictly followed the sliding window size and step length defined in the TLSW algorithm to ensure consistency between the experiment and real-world execution.

Specifically, we applied the sliding window technique to incrementally segment and process the data, simulating the characteristics of a real-time data stream. At each step, feature values were computed and fed into the classification model for evaluation.

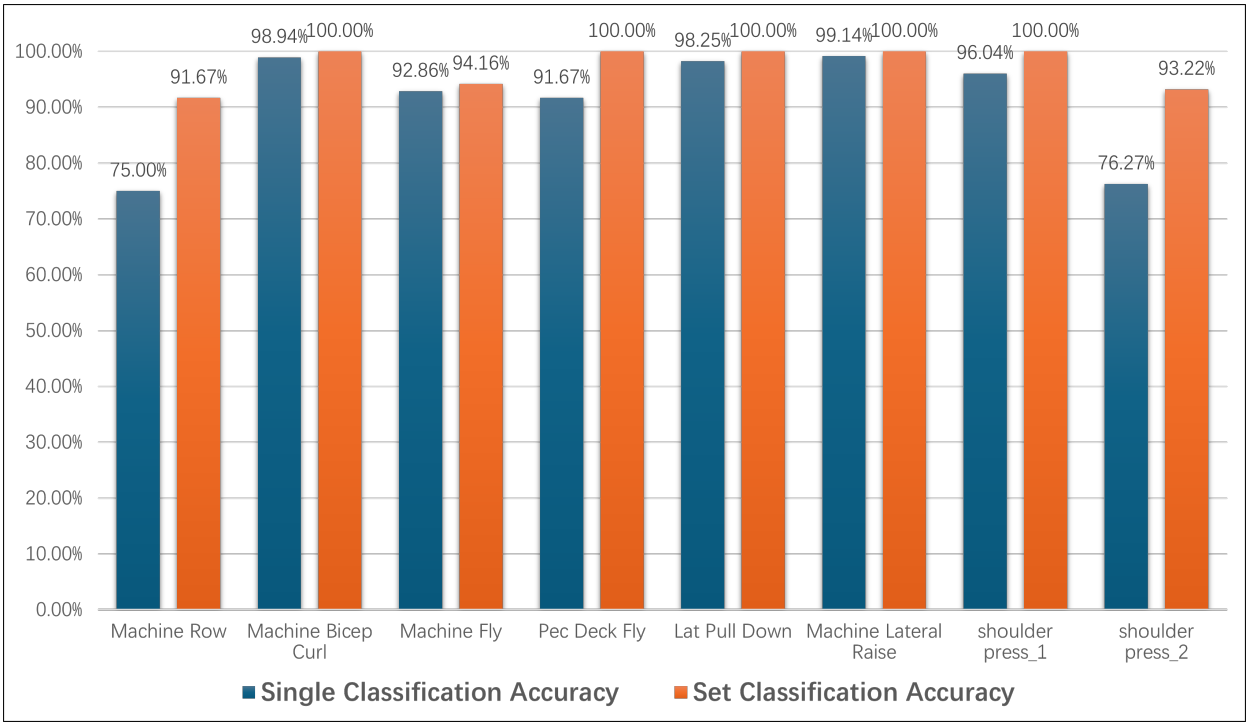


Figure 4.2: Optimized results for Double-Layer Sliding Window

We compared the single classification accuracy of different exercise categories with the corrected classification accuracy for a full set of movements. As shown in the figure, although Machine Row and Shoulder Press_2 had relatively low single classification accuracy, at around 75%, the majority of movements within each set were correctly classified. After applying the correction strategy, the overall classification accuracy for these exercises significantly improved, demonstrating the effectiveness of the correction method in enhancing overall classification performance.

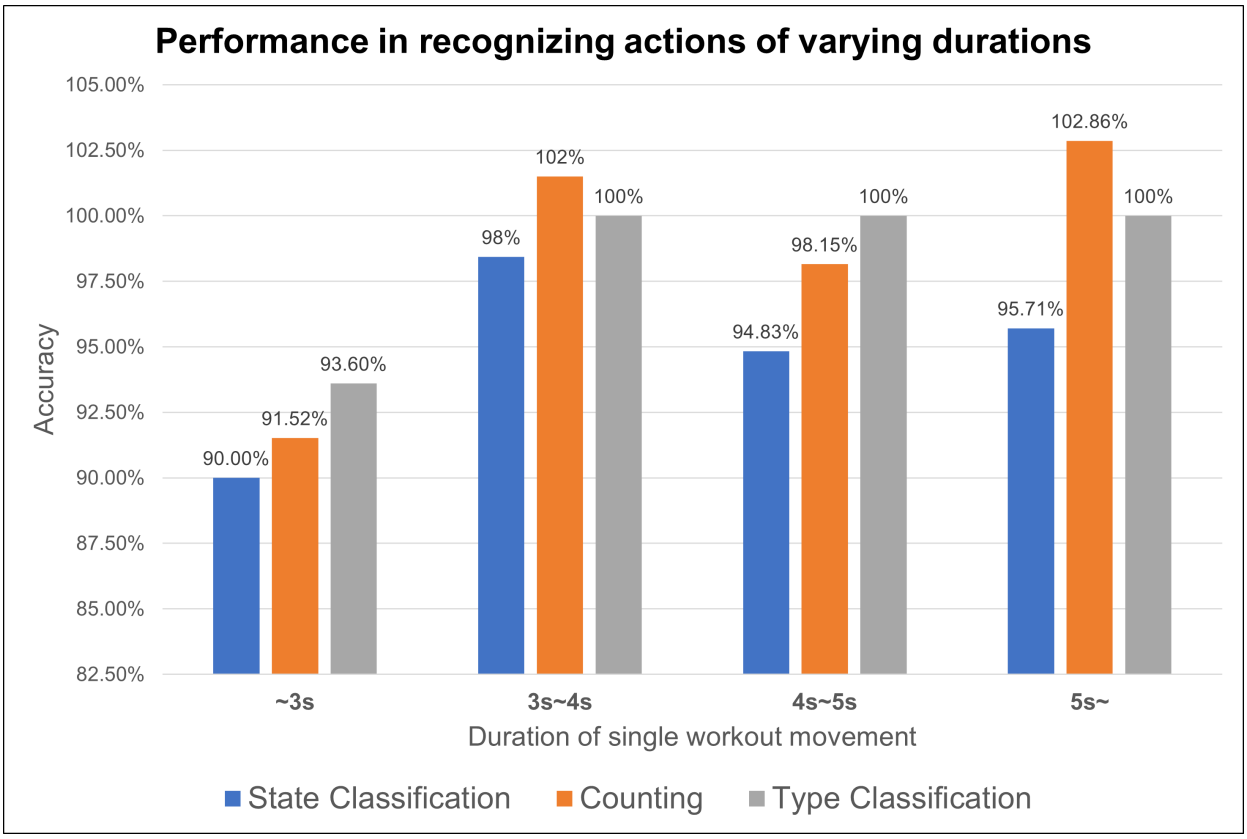


Figure 4.3: Performance in recognizing actions of varying durations

In addition to classifying different exercise categories, we also calculated the average duration per movement and divided it into four time ranges, from less than 3 seconds to more than 5 seconds. We then conducted a detailed analysis for each range.

The results showed that the algorithm performed well when movement durations were between 3 and 5 seconds. However, when movements lasted less than 3 seconds, the accuracy of the three evaluation metrics dropped to approximately 90

To investigate these issues, we conducted further analysis. For movements shorter than 3 seconds, since the window size for action classification was set to 4 seconds, each window may

have contained parts of the previous or next movement, leading to mixed data that affected feature calculations and classification performance. For movements longer than 5 seconds, since the peak detection window was 4.8 seconds, when the window moved between two movement peaks, some secondary peaks might have been mistakenly identified as primary peaks, causing overcounting. Although we introduced restrictions in the peak detection algorithm, they did not fully eliminate the impact of secondary peaks.

Beyond the effect of movement duration on algorithm performance, we also observed other potential sources of error. For instance, short pauses between repetitions and changes in grip position (e.g., incorrect posture) during exercises sometimes led to misclassification in state recognition or movement type recognition.

Despite these challenges, the overall results of the TLSW experiment demonstrated strong performance. The algorithm achieved: 92.84% accuracy for fitness state recognition,97.83% accuracy for fitness type classification,96.33% accuracy for movement counting.

Chapter 5

SmartWT : An Android fitness monitoring app that uses TLSW methods

After validating the TLSW algorithm through offline experiments, we confirmed that its performance meets the expected goals. It demonstrated strong results in fitness state recognition, exercise classification, and movement counting, providing a solid foundation for the next phase of our work.

Next, we plan to integrate the TLSW algorithm and the optimized machine learning model into an Android application. Based on feedback from the questionnaire survey, we aim to develop a smart fitness assistant capable of real-time fitness state and movement recognition.

This application will collect acceleration data via a smartwatch and transmit it to a smartphone for real-time processing. The smartphone will utilize the TLSW algorithm for sliding window processing, extract feature values, and classify movements using a pre-trained machine learning model.

5.1 Related work

In the current mobile device market, Android and iOS are the two dominant operating systems. As an open-source system, Android not only holds a significant share in the smartphone market but is also widely used in tablets, in-car systems, and IoT devices [39].

In human activity recognition research, researchers have not only developed innovative algorithms but also implemented them as Android applications, enabling functions such as fall detection [41][44], gait symmetry analysis [40], and activity recognition [42][43]. Some stud-

ies have leveraged machine learning or deep learning, deploying trained models on Android devices using frameworks like TensorFlow, Keras, and ONNX, achieving efficient real-time applications on mobile platforms [45][46][47].

5.2 Overview of System Architecture

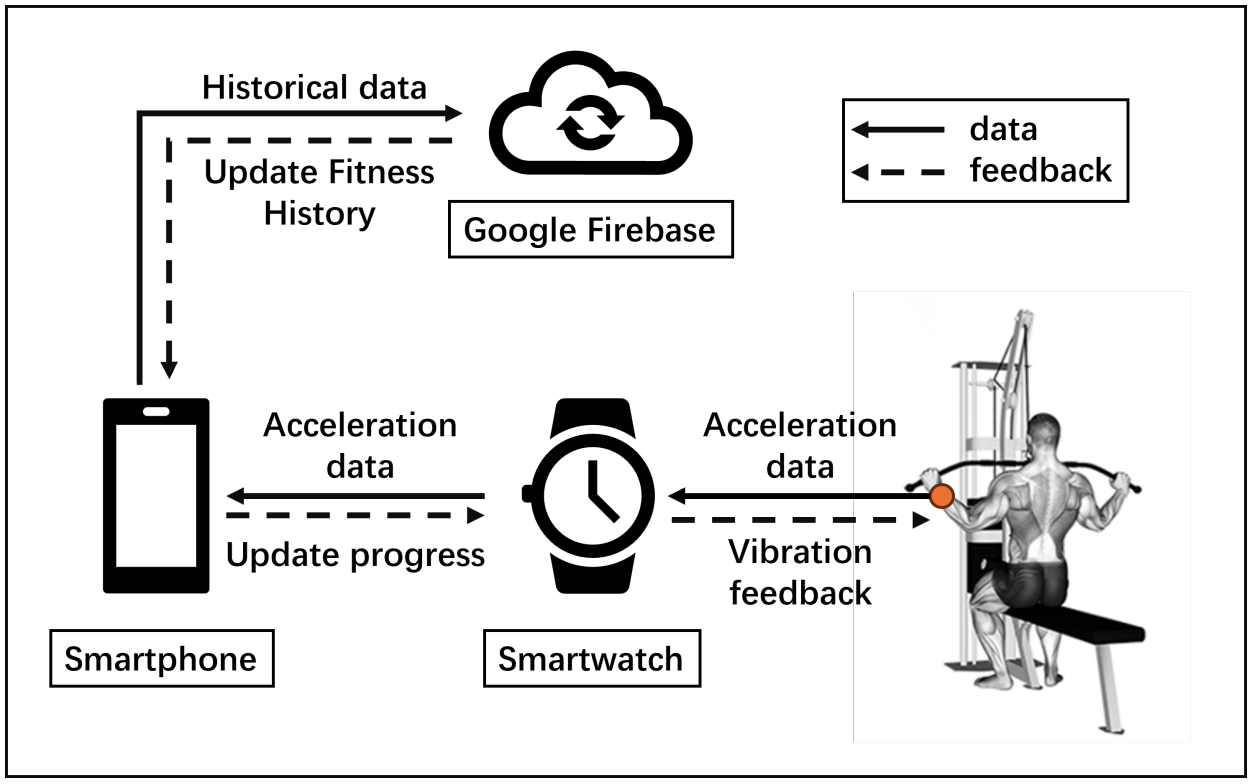


Figure 5.1: System Composition Diagram

Figure 5.1 illustrates the overall system architecture, which consists of a smartwatch, smartphone, and cloud database. The devices used in this study include a Motorola g52j 5G smartphone and a Google Pixel 2 smartwatch, with Google Firebase serving as the cloud database.

Both the smartwatch and smartphone run on the Android operating system, with the smartphone using Android 14 and the smartwatch running Android Wear OS 4.0. The software development was conducted on Windows 11 using Android Studio Giraffe | 2022.3.1 Patch 2.

Figure 5.2 illustrates the system workflow. The process begins with the user logging into their account, which is authenticated through Firebase. Upon successful authentication, the system automatically checks the connection status of the smartwatch.

Table 5.1: Functions of Each Component

Component	**Functionality**
Smartphone	Transmission of acceleration data, Show fitness plan, Vibration feedback
Smartwatch	User login, Operate the watch, View historical data & fitness movement descriptions, Real-time recognition
Firebase	Store user accounts, Stores historical user data

After logging in, users can view historical data and fitness tutorials on the smartphone, regardless of the smartwatch ' s connection status. However, the automatic tracking function is only available when the smartwatch app is open and online.

Before starting a workout, users input their fitness plan on the smartphone, which is then synchronized to the smartwatch and displayed on its screen. When the Start button is pressed, the SmartWT system begins operating. The smartwatch starts collecting three-axis acceleration data during the workout and transmits it in real time to the smartphone.

The smartphone processes the received data in real time, recognizing fitness movements and counting repetitions. It updates the progress according to the predefined workout plan and sends updates back to the smartwatch, dynamically displaying the remaining exercises. When a set is completed, the smartwatch vibrates to notify the user of their progress.

During the workout, users can check the remaining repetitions at any time via the smartwatch or smartphone. When the workout plan is completed, or if the user chooses to stop early, pressing the Stop button on the smartphone halts the SmartWT system. The system then uploads the workout plan and completion progress to the cloud database, allowing users to review their workout history on the smartphone.

5.3 Android application for smartphone

In this study, the Android application on the smartphone is responsible for data processing, device management, and model inference, enabling automated fitness tracking and user interaction.

Data transmission is handled using Data Layer API [48], an Android synchronization mechanism designed for efficient data transfer between smartphones and smartwatches. Additionally, fitness data is synchronized to the cloud via Firebase. Upon user login, the application automatically checks the smartwatch ' s connection status, and the automatic tracking function is

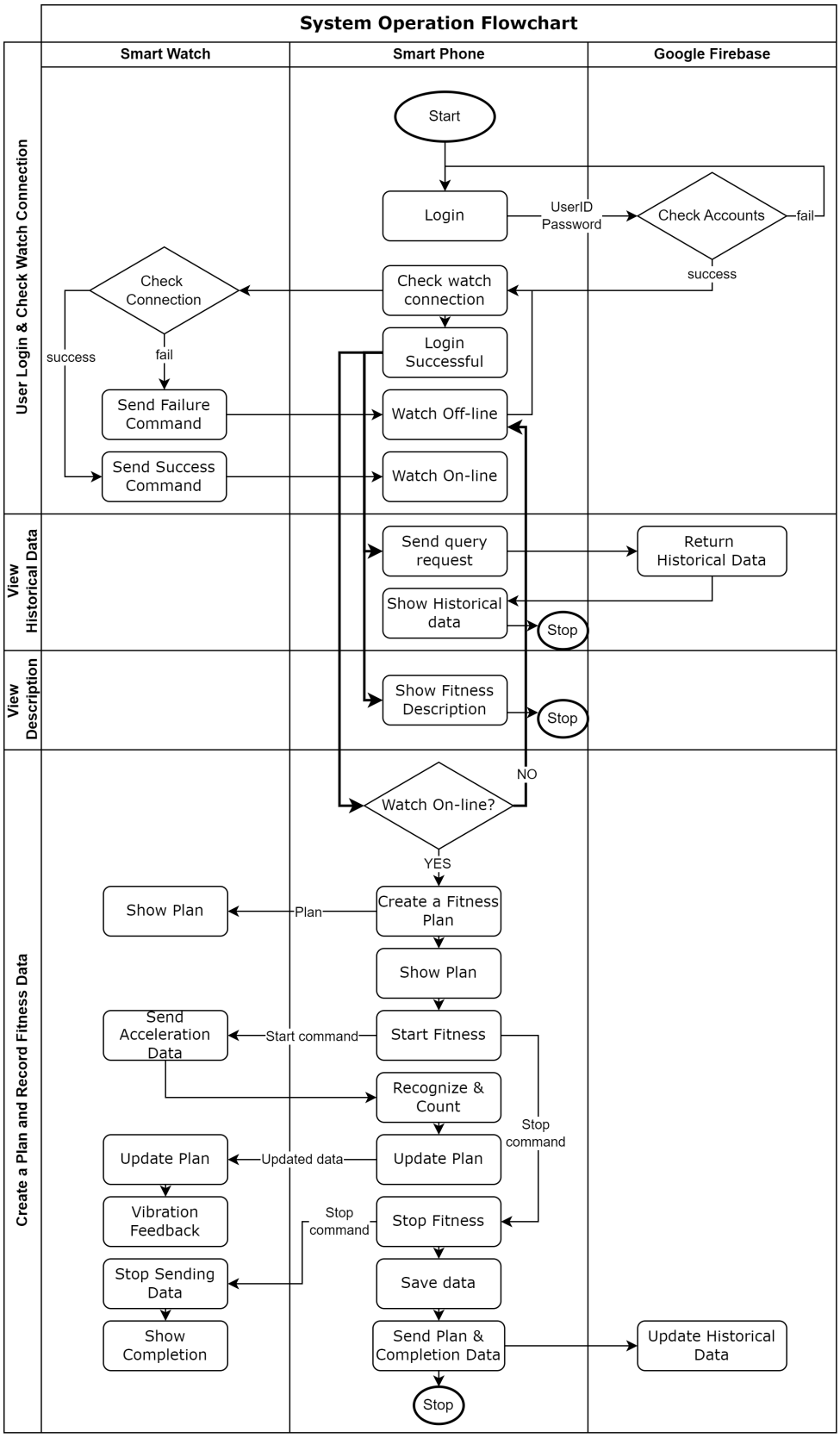


Figure 5.2: System Operation Flowchart

only enabled when the smartwatch is online and the app is running.

For model inference, the application utilizes ONNX Runtime, a cross-platform deep learning inference engine that efficiently runs pretrained machine learning models on Android devices [49]. This enables fitness activity recognition and repetition counting.

In this chapter, we focus on the development details of the automatic recognition system, which is directly relevant to this study. Explanations of device connection status, data upload, and database retrieval are omitted.

The figure illustrates the workflow of the automatic recognition function, which follows the same process as the TLSW algorithm described in Chapter 3. The system consists of four core components: data reception, fitness state recognition, movement classification, and corrected result output.

The entire process is governed by three key decision points, which determine when each computation step is triggered to ensure the algorithm operates correctly. However, due to differences in computing environments and the fact that offline Python code does not involve data transmission, adjustments were made on the Android side to accommodate data transmission, sliding window data structures, and feature computation.

5.3.1 Data transmission

We used Google's Data Layer API to enable real-time transmission of three-axis acceleration data between the smartphone and smartwatch. This method is highly convenient as it does not rely on an internet connection, allowing communication even when both devices are offline. Additionally, communication is restricted to paired devices, ensuring data security.

In our algorithm, one system computation is performed for each received batch of three-axis acceleration data. Ideally, data transmission should follow this sequence: smartwatch sends data $\rightarrow$ smartphone receives data $\rightarrow$ computation is completed $\rightarrow$ smartwatch sends the next batch.

Initially, we designed the system so that the smartwatch would send data immediately after collecting each batch. However, testing showed that frequent API calls at high transmission rates caused significant data loss due to system-level asynchronous operations, with a loss rate of approximately 10

To address this issue, we modified the data transmission logic so that the smartwatch now collects and transmits data in batches of five. On the smartphone side, data is received and

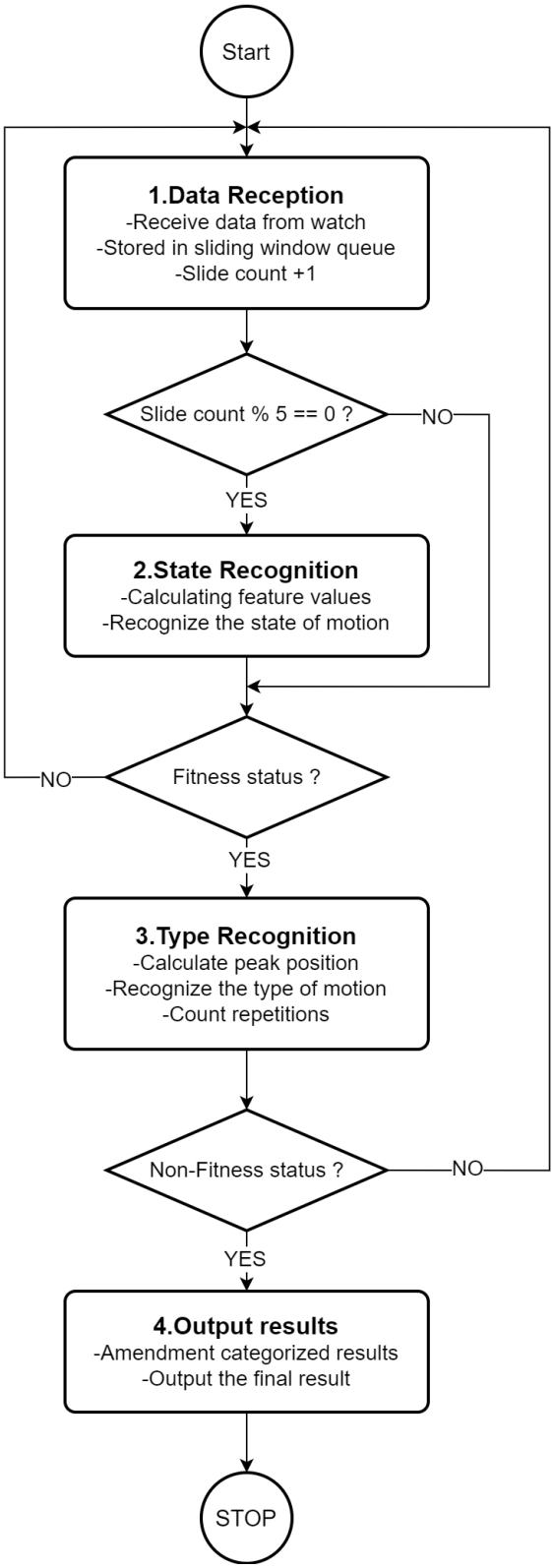


Figure 5.3: Flowchart of smartphone app auto-recognition function

inserted into the sliding window structure at 80ms intervals, followed by a computation cycle. Testing showed that this approach increased data transmission success to 98.3%, though a small degree of data loss still remained.

5.3.2 Data structure of the sliding window

In the offline Python experiments, while testing the TLSW algorithm, we simulated sliding windows by reading data points from the dataset at predefined positions. Specifically, we created a new data structure, populated it with data from the original dataset, and then applied the algorithm for computation.

However, in real-time execution, to improve efficiency and reduce computational overhead caused by continuous data copying, we implemented the sliding window using a circular queue. By maintaining queue indices (start_index and end_index), we dynamically tracked the head and tail positions of the data queue. This ensured that extracted data preserved its original time sequence, allowing the system to accurately retrieve data from different positions within the window for state recognition and movement classification.

5.3.3 Feature Value Calculation

During model training and testing in the Python environment, we used various libraries to compute feature values. However, in the Android environment, when computing Fourier Transform and other feature values using Java libraries, we observed significant discrepancies in the results.

We attempted to manually adjust all parameters to minimize errors, but the discrepancies remained substantial. To resolve this, we integrated Chaquopy (version 15.0.1) to execute Python scripts directly on Android. This allowed us to use the same Python code for feature computation as in the offline experiments.

For complex feature calculations or peak detection algorithms, the application calls a Python script, processes the computation, and returns the results to the Android environment. Testing confirmed that this method successfully reduced errors, although minor precision differences remained due to variations in computing environments. Nevertheless, the error margin was significantly lower compared to using Java-based computations.

5.3.4 Interface of Smartphone Application

The Figure 5.4 presents the user interface of the smartphone application developed in this study.

The upper section of the figure displays the login screen, menu screen, and history screen, while the lower section includes the automatic workout tracking interface, workout plan creation screen, and real-time progress tracking screen during workouts.

In the automatic workout tracking interface, users first press the "Create a Plan" button to set up a personalized workout plan, specifying the number of sets and repetitions per exercise. Once the plan is created, users can start the automatic tracking feature by pressing "Start Work Out".

During the workout, as users complete each set, the system updates the remaining repetitions in the plan. When an exercise is fully completed, the system displays "Fin", indicating that the movement is finished. Users can follow the plan sequentially or choose to end the workout early.

To prevent accidental interruptions, stopping the workout requires a double-tap on the "Stop Work Out" button.

5.4 Android application for smartwatch

In this study, the smartwatch serves three main functions:

Collecting acceleration data from the user and transmitting it to the smartphone.

Displaying the user ' s workout plan and updating progress in real time.

Providing vibration feedback when a set is completed.

Data transmission is handled via Data Layer API, as explained in the previous section. Therefore, this section focuses on interface design and user interaction workflow.

The Figure5.5 shows the operational flow of the smartwatch in the automatic workout tracking function. The process starts with the following steps:

1. Checking device connection.
2. Receiving the user ' s workout plan.

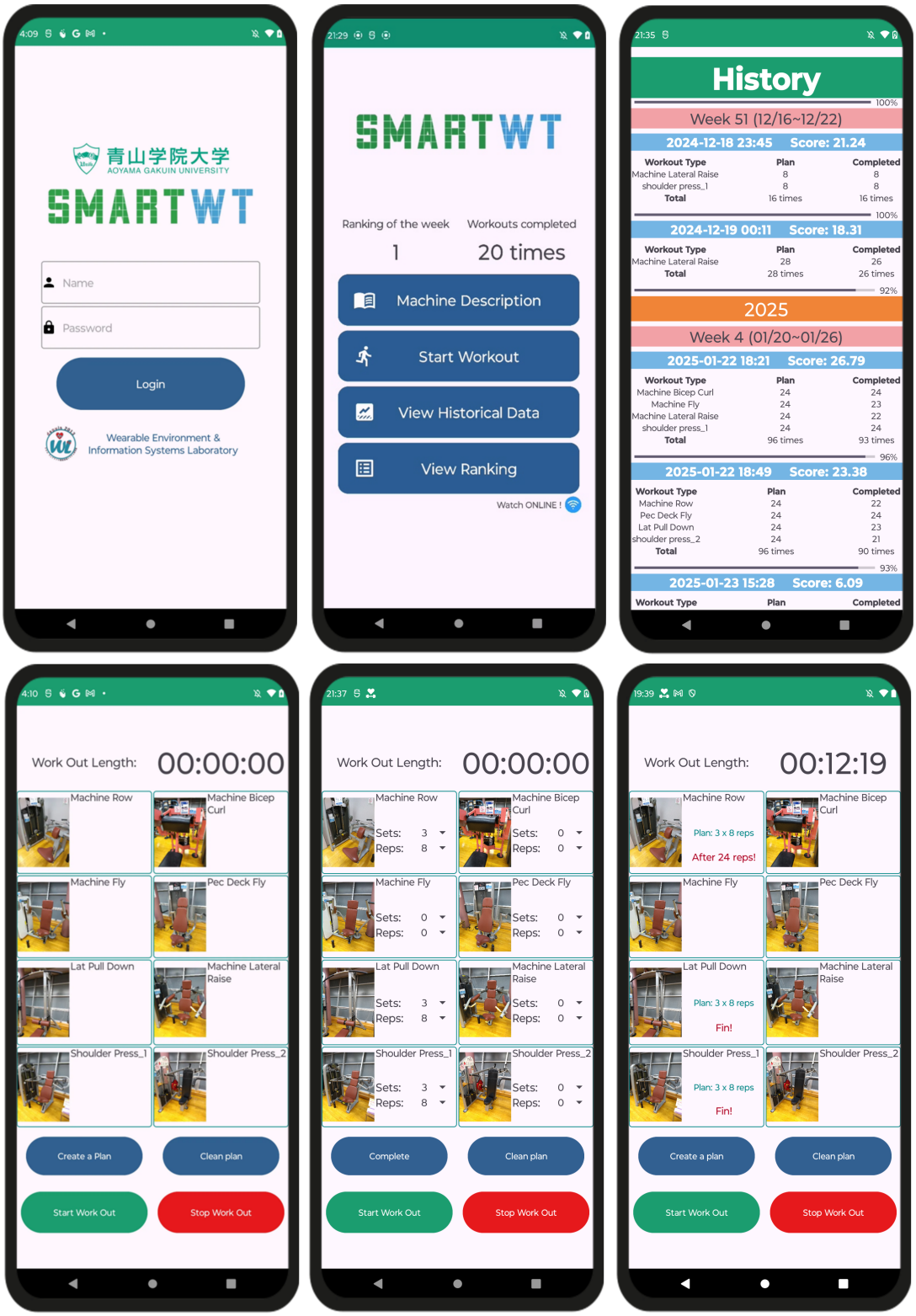


Figure 5.4: user interface of the smartphone application

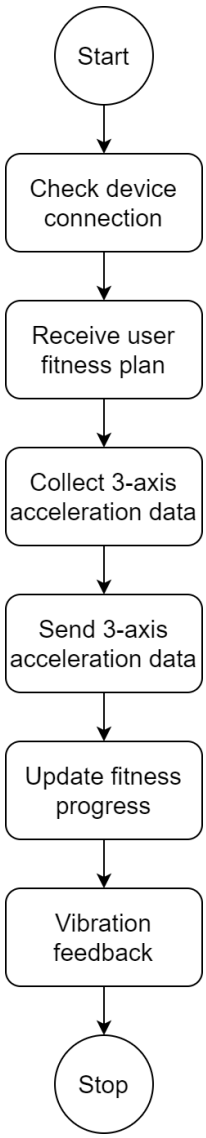


Figure 5.5: Flowchart of smartwatch app auto-recognition function

3. Collecting three-axis acceleration data.

4. Transmitting the data to the smartphone via Data Layer API.

After the smartphone processes the data, it updates the workout plan progress on the smartwatch. When the user completes a set, the smartwatch provides vibration feedback to notify the user. The process concludes when the user either completes all sets or manually stops the program.

5.4.1 Interface of Smartphone Application

As shown in the figure5.6, when the smartwatch app is opened, the initial screen displays the connection status with the smartphone (top left). After receiving the connection information from the smartphone, the screen transitions to the waiting for fitness data screen (top right).

Once the user inputs their workout plan on the smartphone and starts the workout, the smartwatch receives the user ' s plan and displays the remaining repetitions for each exercise. After completing a movement, the screen shows "Fin", and the smartwatch provides vibration feedback to notify the user.

Figure 5.6: Interface of Smartphone Application

Chapter 6

Experiments and Results

6.1 Android device offline experiments and results

6.1.1 Description of the experiment

After completing the system development for both the smartwatch and smartphone, we conducted offline experiments to verify the system's performance before real-world use and testing. We used the same dataset that was used in the Python environment for algorithm testing to conduct the SmartWT system tests.

In the offline experiments, we stored the pre-prepared dataset on the smartwatch and simulated the data transmission process as it would occur in an actual experiment. Specifically, the smartwatch sent test data to the smartphone in groups at the same time intervals as in real usage. Upon receiving the data, the smartphone processed and recognized it according to the actual operating logic.

This approach allowed us to test the smartphone system's recognition performance without relying on actual user interaction. We were able to assess the algorithm ' s performance within the SmartWT system, verify whether the data transmission and recognition processes were functioning properly, and evaluate whether the system demonstrated good stability and accuracy.

6.1.2 Resurts

Figure 6.1 shows the real-time recognition performance of the SmartWT system on Android devices. Compared to the results from the Python environment shown in Figure 3.1, it can be ob-

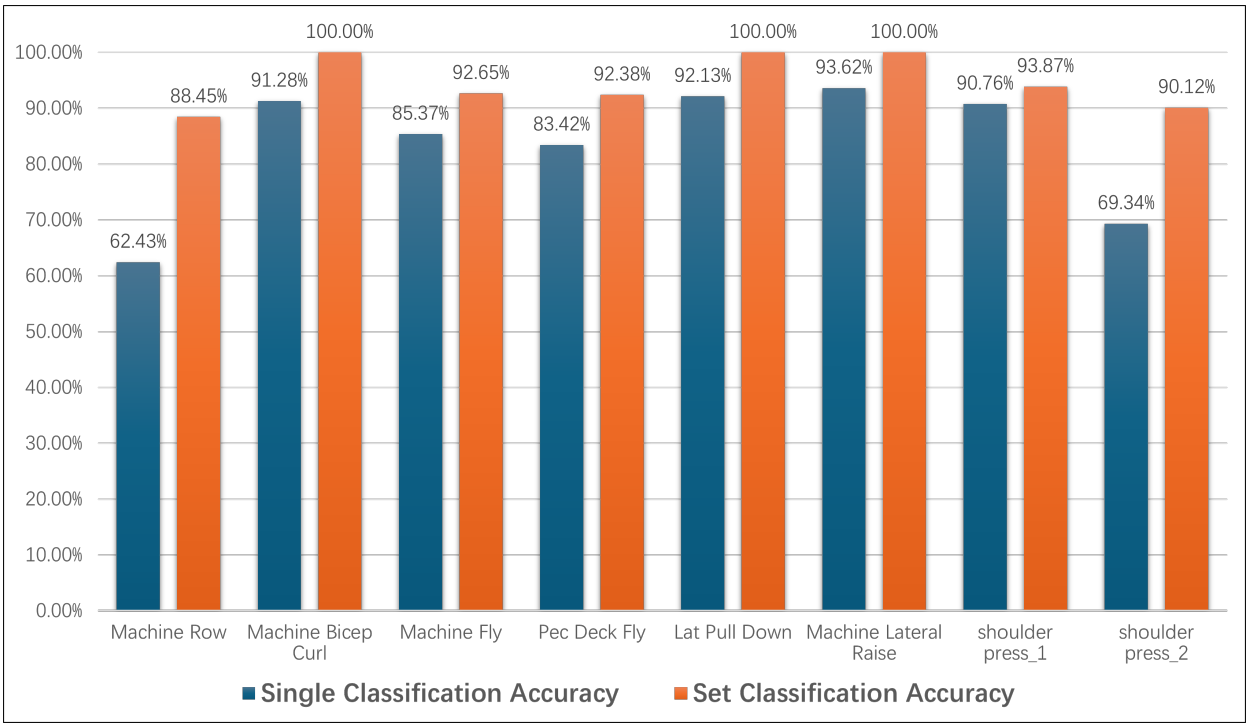


Figure 6.1: Real-time recognition results of the SmartWT system on Android devices

served that, although the same algorithm and dataset were used, the recognition performance on the Android side has slightly decreased. This is especially evident in the recognition of Machine Row and Shoulder Press_2 movements. The accuracy of single recognition on Android dropped by approximately 10% compared to the Python environment.

Similarly, we analyzed the classification accuracy corresponding to different movement durations. Compared to the results in Figure 3.2, there was a noticeable decline in accuracy for fast movements with an average period of less than 3 seconds. For slower movements, there was also a slight decrease in both state recognition and repetition count accuracy. This indicates that movement duration does have a certain impact on the system's recognition performance.

6.2 Android devices actual experiments and results

6.2.1 Description of the experiment

We then designed and conducted real-world experiments, using the same smartphone and smartwatch as in the offline experiments. The experimental process was as follows:

Volunteers were asked to complete all exercises, with each exercise consisting of 3 sets, each

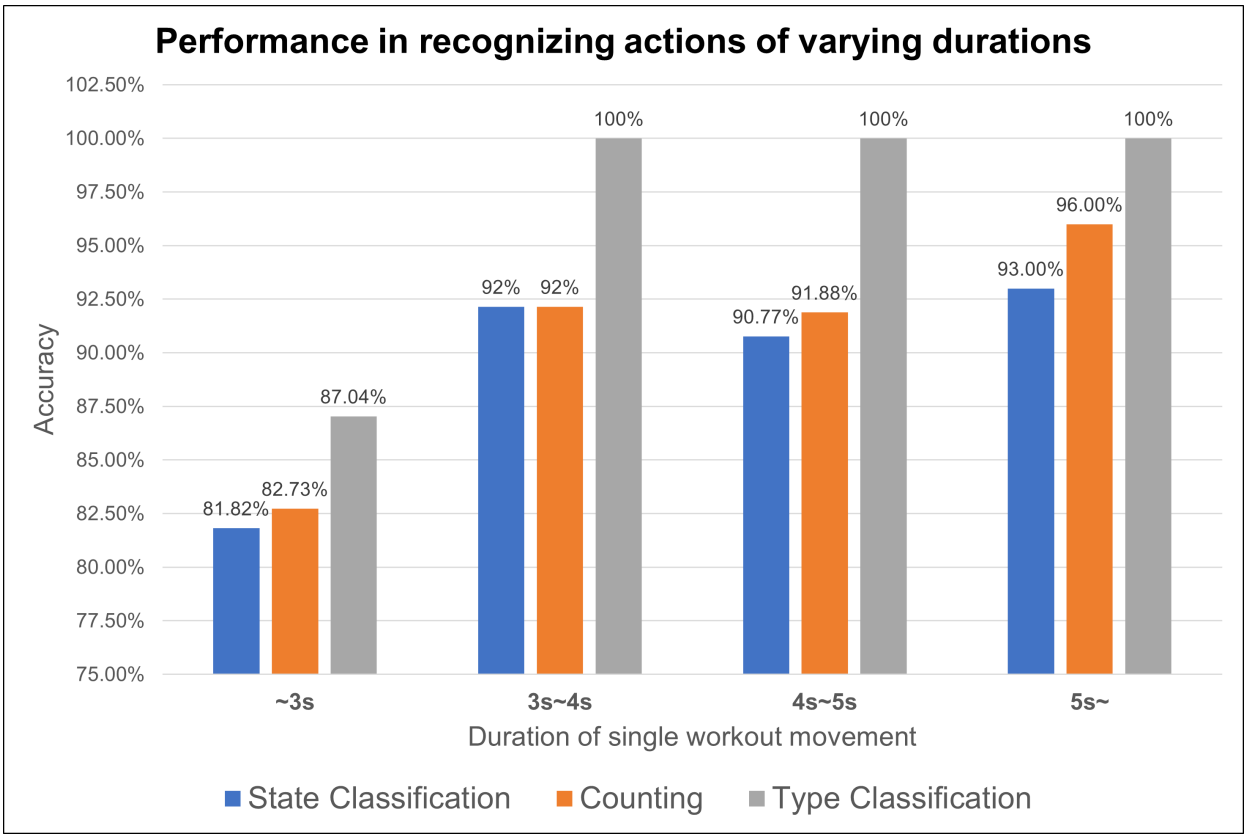


Figure 6.2: Recognition of different simultaneous movements by SmartWT system in an experiment on Android devices

set containing 8 repetitions, for a total of 192 movements. To avoid excessive strain and reduce the risk of injury, the experiment was divided into two sessions for each volunteer. In the first session, volunteers could freely choose 4 exercises from the 8 available exercises, and in the second session, they completed the remaining 4 exercises.

During the experiment, volunteers were required to complete the specified number of repetitions regardless of the remaining repetitions displayed. However, if the volunteers felt fatigued or unable to complete a set, they could reduce the number of repetitions or discontinue the experiment as they saw fit. The experiment was conducted under the supervision and guidance of the researcher, who explained the equipment usage and exercise form to the volunteers beforehand, recorded the number of repetitions during the experiment, and provided necessary assistance.

A total of 9 participants took part in the experiment, including 8 males and 1 female, aged between 21 and 25 years. The participants had 1 to 3 years of fitness experience and had no recent injuries or illnesses. The experiment was conducted in the fitness room at Aoyama Gakuin University. To protect the privacy of the participants and other gym users, no images of the participants were taken; only images of the researcher were used to illustrate the experimental process.

6.2.2 Resurts

After analyzing the experimental results of the 9 volunteers, we statistically divided the classification accuracy and counting accuracy based on different volunteers and exercise categories. The final results are shown in the figure below.

Figure 6.3: Experimental schematic

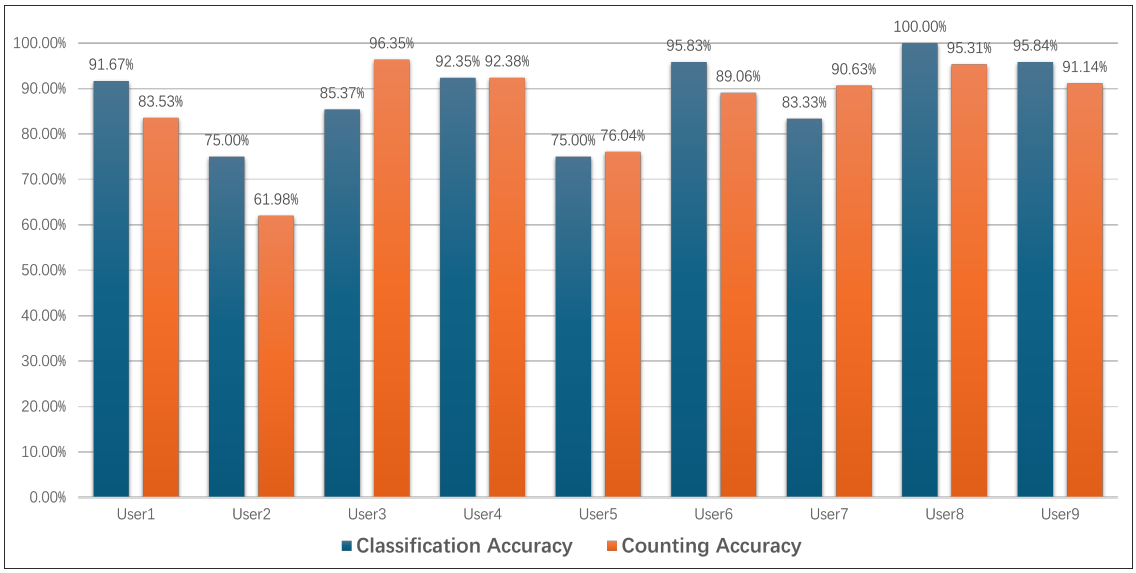


Figure 6.4: Results of the experiment for each volunteer

From Figure 6.4, it can be seen that there are certain variations in classification accuracy and counting accuracy among the different volunteers. Overall, the classification accuracy is generally high, with most volunteers achieving over 90% accuracy. Notably, User7 and User8 reached 100% classification accuracy.

In contrast, the counting accuracy shows more fluctuation. Some volunteers, such as User2 and User6, had relatively low counting accuracy, with values of 63.55% and 63.33%, respectively. On the other hand, User8 achieved the highest counting accuracy at 98.00%.

This indicates that, while the system performs relatively consistently in movement classification, there is still room for improvement in movement counting, especially in optimizing performance for certain individuals.

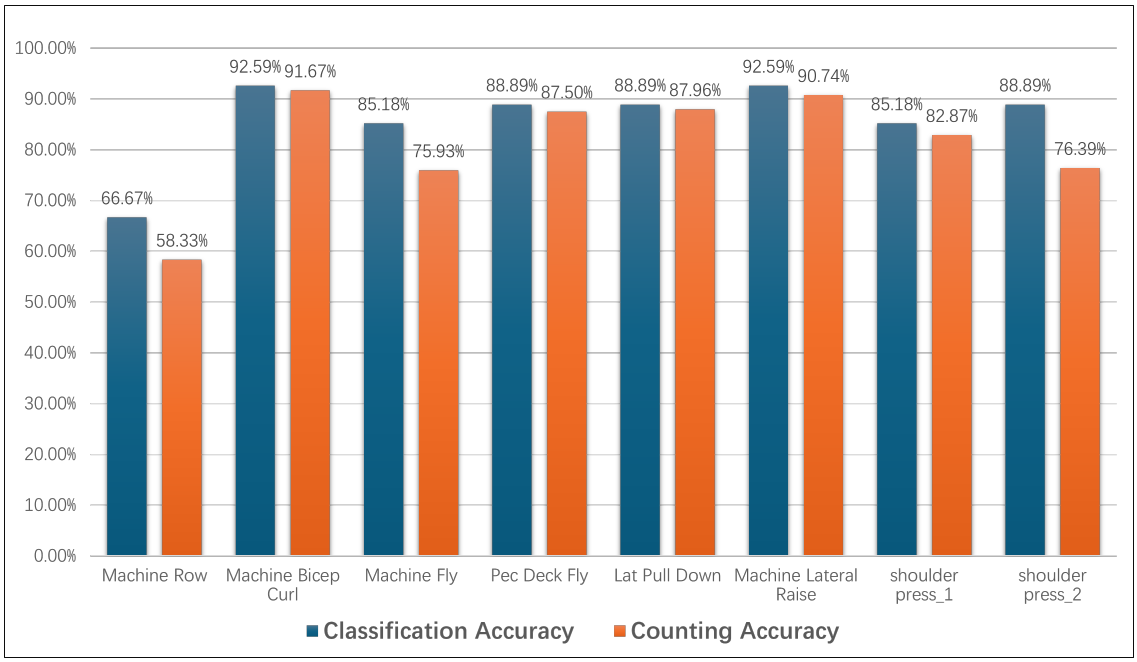


Figure 6.5: Results of the experiment for each volunteer

From Figure 6.5, it is evident that there are significant differences in both classification accuracy and counting accuracy across different exercises. The classification accuracy is generally high, with most exercises achieving over 88% accuracy. For instance, Machine Bicep Curl and Machine Lateral Raise had classification accuracies of 92.59% and 92.50%, respectively. However, some exercises, such as Machine Row, had lower classification accuracy, with a performance of only 66.67%, the lowest among all exercises.

The counting accuracy shows even more variation. Some exercises performed well, such as Pec Deck Fly and Lat Pull Down, with counting accuracies of 97.50% and 97.96%, respectively. However, other exercises, such as Machine Row and Shoulder Press_2, had relatively low counting accuracies of 58.33% and 75.00%, which were noticeably lower compared to other exercises.

Overall, the system performs well in classification and counting for most exercises, but further optimization is needed for movements like Machine Row and Shoulder Press_2, especially in terms of accuracy. This could involve adjusting feature extraction methods or algorithm parameters to improve performance for these exercises.

By observing the volunteers' movements during the experiment, we identified several factors that could contribute to the decrease in performance, aside from data transmission and computation accuracy:

1. **Large movement intervals:** Due to fatigue, volunteers sometimes took longer breaks between exercises, which may have impacted state recognition accuracy.
2. **Insufficient movement range:** Volunteers with less fitness experience often performed exercises with a smaller range of motion, not fully completing the standard movements. This could negatively affect classification performance.
3. **Heavy load:** When volunteers selected heavier weights, the slower movement speed and less noticeable acceleration made it harder for the system to detect peaks, reducing the effectiveness of peak detection and affecting both classification and counting accuracy.

These factors suggest that the variability of users' movement characteristics in real-world scenarios could impact the system's performance. This should be taken into account when optimizing the algorithm.

6.3 Summary

1. In the offline Python experiments, the system demonstrated excellent performance in both algorithm recognition accuracy and counting performance. Since the offline experiments directly read the dataset and calculated feature values without considering data transmission, the results were highly stable. The classification accuracy and counting accuracy reached 96.66% and 96.64%, respectively, representing the theoretical maximum performance of the system.

2. In the offline experiments on Android devices, we used the same dataset and algorithm but simulated the real data transmission process using the smartwatch. The experimental results showed a slight decrease in both classification accuracy and counting accuracy compared to the Python offline experiments, mainly due to: a) Precision loss during the data transmission process. b) Subtle differences in floating-point calculations between the Android and Python environments.

Despite these challenges, by optimizing the transmission logic and using Python scripts for feature computation, the system performance remained at a high level, with classification accuracy and counting accuracy reaching 95.03% and 88%, respectively, which is still close to the Python experiment results.

3. In the real-world Android experiment, where volunteers performed actual fitness movements, the results showed a further decrease in classification and counting accuracy compared to the offline experiments. The main issues were: a) Some exercises (e.g., Machine Row and

Shoulder Press_2) exhibited lower classification and counting accuracy. b) Movements with shorter durations or smaller changes in acceleration were more prone to misclassification.

These declines in performance were not only due to data transmission and computational environment differences but were also influenced by practical factors: i. Volunteers experienced fatigue, leading to longer movement intervals, which affected state recognition. ii. Less experienced volunteers did not fully complete standard movements, resulting in reduced classification performance. iii. When volunteers selected heavier weights, movements became slower, and acceleration became less noticeable, which impacted the peak detection algorithm.

The final results showed a classification accuracy of 86.11% and counting accuracy of 81.42%.

Chapter 7

Future work

To further enhance the performance and applicability of the SmartWT system, future research will focus on the following improvements and expansions:

1. **Increasing Data Types**

   The current system primarily uses three-axis acceleration data for activity state recognition and classification. However, to improve the algorithm's robustness and ability to recognize complex movements, we plan to incorporate additional data types, such as angular velocity. During practical testing, we also measured the system's runtime and found that there is still considerable computation time available. Therefore, expanding the types of data used will enhance the system ' s ability to capture posture changes and improve classification and counting performance for various fitness activities.

2. **Using Dynamic Length Sliding Windows**

   Currently, the system uses fixed-length sliding windows to process the data, which may not be well-suited to the rhythm differences in movements among different users. In the future, we plan to explore methods for dynamic length sliding windows that will adaptively adjust the window length based on the user ' s actual movement characteristics. This approach will better capture personalized movement patterns, enhancing the system ' s applicability to a wider range of users.

3. **Improving Long-Term Feedback Mechanisms**

   The current system design focuses on real-time activity recognition and counting. However, the impact of long-term fitness behavior on users has not been thoroughly explored. Future work will improve the long-term feedback system by analyzing users ' historical

fitness data to provide personalized suggestions and motivational incentives. Additionally, we will integrate gamification features, such as achievement systems and ranking mechanisms, to further investigate the role of long-term feedback in promoting user engagement and cultivating fitness habits.

Acknowledgments

This thesis would not have been possible without the support of many people, and I would like to express my deepest gratitude.

First, I sincerely thank my supervisor, Prof. G. Lopez, for his tremendous support and for giving me the opportunity to study and research here. His guidance went beyond academic matters, providing me with encouragement in university life and career development after graduation. I have learned a lot from him—not only in research but also about food, culture, leadership, and kindness. I have decided to continue my studies in the Lopez Laboratory as a doctoral student and hope to contribute even more in the future.

I would also like to express my gratitude to Prof. Y. Tobe, who guided me in interacting with other researchers during conferences and presentations, and who provided valuable insights for my research. Additionally, I am very grateful to Okuma-san from the Lopez Laboratory. She has not only helped me with research but also supported me in daily life, which has been especially meaningful as an international student.

I feel proud to be part of the Lopez Laboratory. Since joining, I have received great support from everyone. They have helped me with research, daily life, and even improving my Japanese, allowing me to gradually feel at home and contribute to the lab. I would also like to acknowledge the rapid development of artificial intelligence, which has greatly improved my research efficiency and allowed me to explore even more knowledge.

Lastly, none of this would have been possible without the support and encouragement of my parents and family.

Thank you all!

January 30th, 2025

Liu Zeyu

References

[1] W. R. Thompson, "Worldwide Survey of Fitness Trends for 2023," *ACSM's Health & Fitness Journal*, vol. 27, no. 1, pp. 9–18, Dec. 2023, doi: 10.1249/fit.0000000000000834.

[2] G. Di Martino *et al.*, "Enhancing Behavioural Changes: A Narrative Review on the Effectiveness of a Multifactorial APP-Based Intervention Integrating Physical Activity," *International Journal of Environmental Research and Public Health*, vol. 21, no. 2, Art. no. 2, Feb. 2024, doi: 10.3390/ijerph21020233.

[3] F. Ozdamli and F. Milrich, "Positive and Negative Impacts of Gamification on the Fitness Industry," *European Journal of Investigation in Health, Psychology and Education*, vol. 13, no. 8, Art. no. 8, Aug. 2023, doi: 10.3390/ejihpe13080103.

[4] I. Cho, K. Kaplanidou, and S. Sato, "Gamified Wearable Fitness Tracker for Physical Activity: A Comprehensive Literature Review," *Sustainability*, vol. 13, no. 13, Art. no. 13, Jan. 2021, doi: 10.3390/su13137017.

[5] "The Role of Steps and Game Elements in Gamified Fitness Tracker Apps: A Systematic Review," Accessed: Jan. 31, 2025. [Online]. Available: https://www.mdpi.com/2414-4088/5/2/5.

[6] H. Xie, A. Watatani, and K. Miyata, "CoreUI: Interactive Core Training System with 3D Human Shape," *arXiv preprint arXiv:2106.09196*, 2021. [Online]. Available: https://arxiv.org/abs/2106.09196

[7] Y. Wu, A. Kankanhalli, and K. Huang, "Gamification in Fitness Apps: How Do Leaderboards Influence Exercise?," in *ICIS 2015 Proceedings*, 2015, no. 14. [Online]. Available: https://aisel.aisnet.org/icis2015/proceedings/IShealth/14

[8] R. Gal, A. M. May, E. J. van Overmeeren, M. Simons, and E. M. Monninkhof, "The Effect of Physical Activity Interventions Comprising Wearables and Smartphone Applications on Physical Activity: A Systematic Review and Meta-analysis," *Sports Medicine – Open*, vol. 4, no. 1, p. 42, 2018. doi: https://doi.org/10.1186/s40798-018-0157-9

[9] E. Southcott and J. Jooste, "Unveiling the Impact of Mobile Fitness Applications on Motivational Orientation in Sustaining Exercise Behaviors: A Qualitative Investigation," *Physical Culture and Sport. Studies and Research*, vol. 103, 2023. doi: 10.2478/pcssr-2024-0008.

[10] D. Jin, H. Halvari, N. Maehle, and A. H. Olafsen, "Self-tracking behaviour in physical activity: a systematic review of drivers and outcomes of fitness tracking," *Behaviour & Information Technology*, vol. 41, no. 2, pp. 242–261, 2020. doi: https://doi.org/10.1080/0144929X.2020.1801840.

[11] B. Soulé, G. Marchant, and R. Verchère, "Sport and fitness app uses: a review of humanities and social science perspectives," *European Journal for Sport and Society*, vol. 19, no. 2, pp. 170–189, 2021. doi: https://doi.org/10.1080/16138171.2021.1918896.

[12] A. Schneider and R. Arnold, "Wearables from head to toe: Are they friend or foe? An empirical landscaping of health and fitness wearables and apps in six countries to identify emerging policy challenges," Jul. 31, 2022. [Online]. Available: https://ssrn.com/abstract=4177215 or http://dx.doi.org/10.2139/ssrn.4177215.

[13] "RepCount," [Online]. Available: https://www.repcountapp.com/. Accessed: 2024.

[14] "Strong Workout Tracker Gym Log," [Online]. Available: https://www.strong.app/. Accessed: 2024.

[15] H. Xie, A. Watatani, and K. Miyata, "CoreUI: Interactive Core Training System with 3D Human Shape," *arXiv preprint arXiv:2106.09196*, 2021. [Online]. Available: https://arxiv.org/abs/2106.09196.

[16] M. Pasula and P. Saha, "Video-based exercise classification and muscle group activation prediction using hybrid X3D-SlowFast network," *arXiv preprint arXiv:2406.06703*, Jun. 2024. [Online]. Available: https://arxiv.org/abs/2406.06703.

[17] I. U. Khan, S. Afzal, and J. W. Lee, "Human activity recognition via hybrid deep learning based model," *Sensors*, vol. 22, no. 1, Art. no. 1, Jan. 2022. doi: https://doi.org/10.3390/s22010323.

[18] S. K. Chaurasia and S. R. N. Reddy, "State-of-the-art survey on activity recognition and classification using smartphones and wearable sensors," *Multimedia Tools and Applications*, vol. 81, pp. 1077–1108, 2022. doi: https://doi.org/10.1007/s11042-021-11410-0.

[19] S. Chung, J. Lim, K. J. Noh, G. Kim, and H. Jeong, "Sensor data acquisition and multimodal sensor fusion for human activity recognition using deep learning," *Sensors*, vol. 19, no. 7, Art. no. 7, Jan. 2019. doi: https://doi.org/10.3390/s19071716.

[20] D. Morris, T. S. Saponas, A. Guillory, and I. Kelner, "RecoFit: using a wearable sensor to find, recognize, and count repetitive exercises," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*, New York, NY, USA: Association for Computing Machinery, 2014, pp. 3225–3234. doi: https://doi.org/10.1145/2556288.2557116.

[21] C. Shen, B. -J. Ho, and M. Srivastava, "MiLift: Efficient smartwatch-based workout tracking using automatic segmentation," *IEEE Transactions on Mobile Computing*, vol. 17, no. 7, pp. 1609–1622, Jul. 2018. doi: https://doi.org/10.1109/TMC.2017.2775641.

[22] A. Soro, G. Brunner, S. Tanner, and R. Wattenhofer, "Recognition and repetition counting for complex physical exercises with deep learning," *Sensors*, vol. 19, p. 714, 2019. doi: https://doi.org/10.3390/s19030714.

[23] K. Skawinski, F. Montraveta Roca, R. D. Findling, and S. Sigg, "Workout type recognition and repetition counting with CNNs from 3D acceleration sensed on the chest," in *Advances in Computational Intelligence. IWANN 2019*, I. Rojas, G. Joya, and A. Catala, Eds. Cham: Springer, 2019, vol. 11506, *Lecture Notes in Computer Science*. doi: https://doi.org/10.1007/978-3-030-20521-8_29.

[24] S. Ishii, A. Yokokubo, M. Luimula, and G. Lopez, "ExerSense: Physical exercise recognition and counting algorithm from wearables robust to positioning," *Sensors*, vol. 21, p. 91, 2021. doi: https://doi.org/10.3390/s21010091.

[25] A. Dehghani, O. Sarbishei, T. Glatard, and E. Shihab, "A quantitative comparison of overlapping and non-overlapping sliding windows for human activity recognition using inertial sensors," *Sensors*, vol. 19, p. 5026, 2019. doi: https://doi.org/10.3390/s19225026.

[26] V. A. Cornelissen, R. H. Fagard, E. Coeckelberghs, and L. Vanhees, "Impact of resistance training on blood pressure and other cardiovascular risk factors," *Hypertension*, vol. 58, no. 5, pp. 950–958, Nov. 2011. doi: https://doi.org/10.1161/HYPERTENSIONAHA.111.177071.

[27] J. Kristensen and A. Franklyn-Miller, "Resistance training in musculoskeletal rehabilitation: a systematic review," *British Journal of Sports Medicine*, vol. 46, no. 10, pp. 719–726, Aug. 2012. doi: https://doi.org/10.1136/bjsm.2010.079376.

[28] A. H. E. Akpa, M. Fujiwara, H. Suwa, Y. Arakawa, and K. Yasumoto, "A smart glove to track fitness exercises by reading hand palm," *Journal of Sensors*, vol. 2019, no. 1, p. 9320145, 2019. doi: https://doi.org/10.1155/2019/9320145.

[29] W.-Y. Chung, A. Purwar, and A. Sharma, "Frequency domain approach for activity classification using accelerometer," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug. 2008, pp. 1120–1123. doi: https://doi.org/10.1109/IEMBS.2008.4649357.

[30] D. Jayakumar, M. Krishnaiah, S. Kollem, S. Peddakrishna, N. Chandrasekhar, and M. Thirupathi, "Emergency vehicle classification using combined temporal and spectral audio features with machine learning algorithms," *Electronics*, vol. 13, no. 19, Art. no. 19, Jan. 2024. doi: https://doi.org/10.3390/electronics13193873.

[31] K. Bhangale and M. Kothandaraman, "Speech emotion recognition based on multiple acoustic features and deep convolutional neural network," *Electronics*, vol. 12, no. 4, Art. no. 4, Jan. 2023. doi: https://doi.org/10.3390/electronics12040839.

[32] W.-Y. Chung, A. Purwar, and A. Sharma, "Frequency domain approach for activity classification using accelerometer," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug. 2008, pp. 1120–1123. doi: https://doi.org/10.1109/IEMBS.2008.4649357.

[33] O. D. Lara and M. A. Labrador, "A Survey on Human Activity Recognition using Wearable Sensors," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1192–1209, 2013. doi: https://doi.org/10.1109/SURV.2012.110112.00192.

[34] M. Wang *et al.*, "A comprehensive evaluation of dual-polarimetric Sentinel-1 SAR data for monitoring key phenological stages of winter wheat," *Remote Sensing*, vol. 16, no. 10, Art. no. 10, Jan. 2024. doi: https://doi.org/10.3390/rs16101659.

[35] E. C. Ketola, M. Barankovich, S. Schuckers, A. Ray-Dowling, D. Hou, and M. H. Imtiaz, "Channel reduction for an EEG-based authentication system while performing motor movements," *Sensors*, vol. 22, no. 23, Art. no. 23, Jan. 2022. doi: https://doi.org/10.3390/s22239156.

[36] W.-Y. Chung, A. Purwar, and A. Sharma, "Frequency domain approach for activity classification using accelerometer," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug. 2008, pp. 1120–1123. doi: https://doi.org/10.1109/IEMBS.2008.4649357.

[37] Y. Meng, N. Yang, Z. Qian, and G. Zhang, "What makes an online review more helpful: An interpretation framework using XGBoost and SHAP values," *Journal of Theoretical and Applied Electronic Commerce Research*, vol. 16, no. 3, Art. no. 3, Jun. 2021. doi: https://doi.org/10.3390/jtaer16030029.

[38] S. Knapič, A. Malhi, R. Saluja, and K. Främling, "Explainable artificial intelligence for human decision support system in the medical domain," *Machine Learning and Knowledge Extraction*, vol. 3, no. 3, Art. no. 3, Sep. 2021. doi: https://doi.org/10.3390/make3030037.

[39] M. A. Khan *et al.*, "Smart Android Based Home Automation System Using Internet of Things (IoT)," *Sustainability*, vol. 14, no. 17, Art. no. 17, Jan. 2022. doi: https://doi.org/10.3390/su141710717.

[40] R. Luque, E. Casilari, M.-J. Morón, and G. Redondo, "Comparison and characterization of Android-based fall detection systems," *Sensors*, vol. 14, no. 10, Art. no. 10, Oct. 2014. doi: https://doi.org/10.3390/s141018543.

[41] A. R. Anwary, H. Yu, and M. Vassallo, "An automatic gait feature extraction method for identifying gait asymmetry using wearable sensors," *Sensors*, vol. 18, no. 2, Art. no. 2, Feb. 2018. doi: https://doi.org/10.3390/s18020676.

[42] R.-A. Voicu, C. Dobre, L. Bajenaru, and R.-I. Ciobanu, "Human physical activity recognition using smartphone sensors," *Sensors*, vol. 19, no. 3, Art. no. 3, Jan. 2019. doi: https://doi.org/10.3390/s19030458.

[43] T. R. Mauldin, M. E. Canby, V. Metsis, A. H. H. Ngu, and C. C. Rivera, "SmartFall: A smartwatch-based fall detection system using deep learning," *Sensors*, vol. 18, no. 10, Art. no. 10, Oct. 2018. doi: https://doi.org/10.3390/s18103363.

[44] S. Chung, J. Lim, K. J. Noh, G. Kim, and H. Jeong, "Sensor data acquisition and multimodal sensor fusion for human activity recognition using deep learning," *Sensors*, vol. 19, no. 7, Art. no. 7, Jan. 2019. doi: https://doi.org/10.3390/s19071716.

[45] R. Szabo, "Implementing computer vision in Android apps and presenting the background technology with mathematical demonstrations," *Technologies*, vol. 13, no. 1, Art. no. 1, Jan. 2025. doi: https://doi.org/10.3390/technologies13010027.

[46] A. Sehgal and N. Kehtarnavaz, "Guidelines and benchmarks for deployment of deep learning models on smartphones as real-time apps," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, Art. no. 1, Mar. 2019. doi: https://doi.org/10.3390/make1010027.

[47] A. Ghaffari and Y. Savaria, "CNN2Gate: An implementation of convolutional neural networks inference on FPGAs with automated design space exploration," *Electronics*, vol. 9, no. 12, Art. no. 12, Dec. 2020. doi: https://doi.org/10.3390/electronics9122200.

[48] Google, “Data Layer API Overview,” *Android Developers*. Available: https://developer.android.com/training/wearables/data/overview. [Accessed: 08-Jan-2025].

[49] ONNX Runtime Developers, *ONNX Runtime*, 2021. [Online]. Available: https://onnxruntime.ai/. [Accessed: Jan. 29, 2025].

質疑応答

戸辺　義人　情報テクノロジー学科　教授

Q	リアルタイムというのがあいまいですが，本研究におけるリアルタイムとは？
A	ご質問ありがとうございます．本研究における「リアルタイム」とは、スマートウォッチで収集した加速度データをスマートフォンに送信し、TLSW（三層スライディングウィンドウ）アルゴリズムを用いて即時に処理し、運動状態の識別、運動種別の分類、および回数のカウントを行うことを指します。

浦垣　啓志郎　情報テクノロジー学科　助手

Q	Python と Android の誤差は利用しているライブラリなどの影響を確認しましたか？
A	ご質問ありがとうございます．本研究では Python 環境と Android 環境の違いによる誤差を確認し、特にフーリエ変換や特徴量計算の精度統一のために Chaquopy を導入し、Python スクリプトを直接実行することで誤差を最小限に抑えました。