IAB330 Assignment 2
Applied IoT Project Report

Gesture Recognition

Alex Achille - 11100842

[Redacted] - xxx

[Redacted] - xxx

Date submitted: September 21, 2025

Online Tutorial Wednesday 6 pm

Table of Contents

1. Introduction

Many individuals with writing disabilities face daily communication challenges due to conditions that affect their ability to write or type. Our system addresses this real-world problem by acting as an assistive technology tool that provides an alternative method for text input. The device is strapped to a user's wrist, allowing them to "write in the air" by performing gestures that are then classified as alphabet characters. This approach is intended to promote independence and inclusivity by enabling a new form of communication for people with writing disabilities.

The scope of this prototype is limited to recognising a subset of uppercase letters (P, R, B, D, G, S, C, L, O, V, Z). This specific group was chosen to balance simple, distinct gestures with more complex, confusable pairs (e.g., P/R, B/D) to provide a realistic challenge for the model. For this initial stage, the system will not include the remainder of the uppercase alphabet, lowercase letters, or numbers. This limited scope allows us to prioritise achieving high accuracy and to build a robust model using high-quality, varied data, which is essential for a proof-of-concept.

Human Activity Recognition (HAR) is central to our chosen application. The technology provides the crucial link between the raw Inertial Measurement Unit (IMU) data collected from the user's wrist gestures and the meaningful letters they are intended to represent. The HAR pipeline, which includes data collection, annotation, segmentation, classification, and evaluation, is what transforms motion data into usable text. Without a robust HAR framework, the raw sensor data would remain meaningless, and the system would be unable to solve the communication problem it was designed to address.

2. Design Objectives

Application context and Purpose

The primary objective of this project is to design and propose a gesture-based HAR system as an assistive technology tool for individuals with limited fine motor control who cannot use traditional input devices like pens or keyboards. By recognising single arm and wrist movements as characters, the system provides an alternative method for text input, promoting independence and inclusivity. The device is a proof-of-concept worn on the wrist, which allows users to "write in the air" with gestures that are classified and transmitted via Bluetooth to a Raspberry Pi for display. The process is guided by an LED cueing system to ensure consistent timing for data capture. This application aligns with gesture control research while directly addressing a real-world communication challenge for people with mobility impairments.

Recognised Activities and Their Significance

The system recognises the wrist gestures corresponding to the alphabet characters. Each character is a distinct activity class in the HAR pipeline. For feasibility, the prototype will focus on uppercase letters (P, R, B, D, G, S, C, L, O, V, Z), with the option to extend further lowercase letters or digits in later expansion. These letters were chosen to balance simple, distinct gestures (L, O V, Z) that the classifier can reliably distinguish, with familiar or confusable gestures (P/R, B/D, G/S, C/O) that provide a realistic challenge. The significance of recognising these activities is that they:

- ❖ They directly map to text input, offering an alternative to typing or handwriting for individuals with limited motor ability.

- They demonstrated the potential of gesture-based HAR systems to enhance human-computer interactions in accessibility contexts (Wang et al., 2019).
- Provide a foundation for future studies on which letters are easier or harder to distinguish, informing improvements in both classifier and user training.

Expected Performance

The following objectives are set to ensure that the system is reliable and useful:

- **Accuracy**: The system should achieve at least 80% classification accuracy when recognising characters across multiple users. Prior HAR studies show this level of performance is feasible with wearable sensor data (Kristoffersson & Linden, 2022).
- **Latency**: Predictions should be generated within 1-2 seconds of completing a gesture. The end-to-end latency, including the 2.5s capture window, should remain 3s, ensuring responsiveness for real-time use. Low latency is essential for interactive systems and communication use cases.
- **Robustness**: The system should perform consistently across different users and natural variation in gesture style. To manage scope, training data will be collected in a sitting posture, with axis remapping applied to stimulate standing and lying posture.
- **Scalability**: While this prototype is limited to 11 uppercase letters, the design should allow expansion to additional characters or symbols without requiring a fundamental redesign (Kristoffersson & Linden, 2022).

3. System Design and Decision Justification

This section describes the full architecture of the prototype, outlining each component with the rationale behind the choices.

3.1. Sensor Node (Arduino Nano 33 IoT)

Arduino and Placement

The project will use four separate Arduino Nano 33 IoT devices. One for each of the four participants. Each device is worn on the user's wrist, similar to a watch. To secure each Arduino to the participant's wrist, a simple, low-cost, and adjustable velcro strap will be used. Specifically, the Velcro Brand Strip. This method provides a reliable way to affix and adjust the device's position without causing discomfort to the user. Justifications for a velcro strap in this prototype are highlighted by its:

- **Ease of use**: An Efficient and time-effective way to wear and remove the prototype.
- **Adjustability**: The velcro strap is easily tightened or loosened to accommodate different wrist sizes. This will ensure consistent sensor placement across the participants.
- **Cost-Effective**: It is a cheap and viable solution that mitigates complex, custom-fabricated alternatives.
- **Stability**: The velcro strap will prevent the Arduino from shifting during gesture training and demonstrations, which is critical for collecting clean and consistent accelerometer data.

This configuration is essential for collecting diverse, multi-user data, critical for training a robust machine learning (ML) model that is less susceptible to bias toward a single user's gesture patterns.

IMU Configuration

The IMU on each Arduino is configured to capture accelerometer data. To balance data resolution with processing and transmission overhead, the following settings have been selected:

- ❖ **Sampling rate**: The accelerometer samples at 50 Hz. The rate is sufficient to capture the dynamics of wrist movements.
- ❖ **Axis orientation**: The accelerometer's axes are oriented to capture motion relative to the wrist. The software will be designed to handle data regardless of the device's absolute orientation in space.
- ❖ **Dynamic range**: The sensor is set to a dynamic range of ±4g. The range is sufficient to capture the maximum accelerations produced by typical wrist gestures associated with air writing without clipping the signal. This will be an optimal balance between avoiding data loss while maintaining the fine-grained resolution needed to distinguish between similar gestures. The raw data is read as 16-bit integers, providing fine-grained resolution within its range. Lower ranges could offer higher resolutions but risk missing data during sharper movements. Conversely, a higher range of ±8g would unnecessarily reduce resolution.

Local Data Processing

The Arduino will perform local data processing to reduce wireless data transmission and conserve battery power. It will operate by continuously reading raw accelerometer data while temporarily storing it in a ring buffer. When the participant completes a gesture, a 2.5-second window of this data will be extracted for transmission. This window, which contains 125 samples (2.5 s * 50 Hz sampling rate), will serve as the fundamental unit of a single gesture. The Arduino then processes this window by segmenting it into smaller, more manageable chunks for efficient transmission over BLE.

Additional Sensors

No additional sensors are used. The project design is intended to focus on the core accelerometer data needed for gesture recognition. This will reduce the complexity, power consumption, and volume of data needed for transmission.

Data Diversity

The system is designed to be robust across different environments. Gestures will be performed with participants in various postures, such as sitting, standing, and lying, to collect subtle variations in wrist movement across different contexts. Data will be collected in an indoor setting. As environmental factors, such as light, do not affect accelerometer data collection results. This inherent diversity in the training data should ensure that the model generalises well to real-world use cases of gesture recognition.

3.2. Wireless Communication

The system uses **BLE** for wireless communication between the Arduino Nano 33 IoT and the Raspberry Pi. This choice is based on BLE's low power consumption, which is critical for a wearable device.

BLE Roles

- ❖ **Arduino Nano 33 IoT**: Acts as the BLE peripheral. Advertising its presence and providing data.
- ❖ **Raspberry Pi**: Acts as the BLE central device. Scanning for and connecting to the Arduino to receive and request data.

Service and Characteristic Setup

A custom BLE service is defined with two characteristics to manage data flow:

❖ **Command Characteristic (Write)**: The Raspberry Pi writes to this characteristic to control the data capture process on the Arduino.
  ➢ Value 0: Signals the Arduino to remain in an idle state.
  ➢ Value 1: Requests the Arduino to begin capturing a new accelerometer data window.

❖ **Sensor Data Characteristic (Notify)**: The Arduino uses the notify characteristic to send the captured accelerometer data to the Raspberry Pi. The Raspberry Pi subscribes to notifications on this characteristic to receive the data in real-time.

Data Packaging and Transmission

The accelerometer data window is 2.5 seconds long, consisting of 125 samples. Each sample consists of three 16-bit integer values (x, y, z), resulting in 6 bytes per sample with a total window size of 750 bytes. With limited payload sizes, the Arduino will divide the window into smaller chunks to ensure reliable transmission over BLE. Each chunk will consist of 3 samples (18 bytes), and a 1-byte sequence number is added to each, bringing the total payload to 19 bytes. This process results in 41 full notifications, each containing 3 samples, and a final notification containing the remaining 2 samples, for a total of 42 notifications required to send a full data window. The 1-byte sequence number, which ranges from 0 to 41, indicates the chunk's position in the window.

Connection Flow and Error Handling

1. **Connection**: The Raspberry Pi connects to the Arduino and enables notifications on the Sensor Data characteristic.
2. **Request**: The Raspberry Pi initiates a gesture capture by writing 1 to the command characteristic. The Arduino gives 1 LED blink to visually indicate capture start.
3. **Capture and send**: The Arduino records a 2.5-second accelerometer data window. It then sequentially sends 42 notifications, each containing a chunk of data and its corresponding sequence number.
4. **Reassembly**: The Pi receives the notifications and reconstructs the full 750-byte data window by checking that all 42 sequence numbers are present and in order.
5. **Completion**: After the final notification is sent, the Arduino writes 0 back to the Command characteristic, indicating it is ready for a new request, and gives a 2 LED blink to visually indicate capture stop. After completion, wait 5 seconds and then go back to 1 LED blink to visually signal the start of the next capture cycle.
6. **Error strategy**: The Raspberry Pi discards the entire window if it detects a missing or out-of-order sequence number. Then re-requests a capture by writing 1 to the command characteristic. This strategy mitigates the complexity of mid-window repair and ensures the machine learning model only receives complete and uncorrupted data.

Latency and Performance

The end-to-end system is designed for minimal latency to ensure a responsive user experience. The total time from the completion of a gesture to the data being available for classification on the Raspberry Pi is approximately 3 seconds. This latency is a result of two key components:

❖ **Data Capture**: The Arduino is recording a 2.5-second window of accelerometer data.

- ❖ **Wireless Transmission**: The transfer time of the segmented data over BLE and its reassembly on the Raspberry Pi is estimated to take 0.5 seconds.

This low-latency pipeline directly supports the project's goal of creating an interactive and responsive system.

3.3. Edge Device (Raspberry Pi)

The Raspberry Pi serves as the primary edge device, managing the real-time processing and data handling for the system. Its responsibilities are broken down into three main categories: BLE communication, local processing and inference, and data management.

BLE Communication

The Raspberry Pi acts as the central hub, connecting to the Arduino over BLE. It subscribes to notifications for continuous IMU data streaming, which maintains a steady, low-latency connection and improves efficiency compared to cloud-only transmission (Shi et al., 2016).

Local Processing and Inference

Upon receiving raw IMU data from the Arduino, the Raspberry Pi performs all necessary preprocessing and inference locally. This ensures low-latency predictions while improving data privacy. The strategy for running the trained model for inference follows a sequential process:

1. **Receive and Preprocess Data**: The Raspberry Pi reassembles the received data chunks into a complete window, applies a moving average filter to smooth and reduce noise, and performs feature extraction to convert the raw data into fixed-length feature vectors.
2. **Run Model Inference**: This cleaned feature vector is then applied to the trained machine learning model, which is stored locally on the device, to perform a real-time gesture prediction.
3. **Output**: The final predicted character is then displayed on the Raspberry Pi's console.

This preprocessing pipeline ensures that the model receives clean data, thereby improving the accuracy of the HAR device's gesture recognition.

Data Management and Upload Strategy to MongoDB

The processed data is uploaded from the Raspberry Pi to a MongoDB database to allow for long-term storage, model refinement, and monitoring.

- ❖ **Frequency**: Since processing occurs on the edge device, real-time uploads are not necessary. Data can be uploaded per gesture or in small batches to conserve bandwidth, reduce system load, improve query performance, and enhance privacy by limiting raw data transmission (MongoDB, 2025).
- ❖ **Structure**: Each entry to the database is a JSON document containing key metadata such as participant_id, gesture_label, and extracted features.
- ❖ **Connectivity Handling**: It also includes a local buffering mechanism to prevent data loss in the event of a network interruption.

3.4. Data Flow and Timing

The end-to-end HAR pipeline is a series of six sequential stages from raw sensor data acquisition to on-device inference.

1. **Sensor Data Acquisition**: Arduino captures accelerometer data at 50 Hz, buffering a 2.5-second gesture window, giving 125 samples as 16-bit integers (x, y, z).
2. **BLE Transmission**: The Arduino sends sensor data to the Raspberry Pi using BLE notifications. Each notification chunk consists of a 1-byte sequence number followed by 3 samples, totaling 19 bytes. Per 2.5-second capture window, 41 full chunks and a final 13-byte chunk with 2 samples are sent, making a total of 42 notifications. This ensures the entire window is transmitted efficiently and in the correct order.
3. **Raspberry Pi Edge Processing**: The Raspberry Pi subscribes to the BLE notifications and reassembles the data chunks into a 125×3 array using their sequence order. It discards any window with a missing sequence number. Once a complete window is received, it applies a short moving average for smoothing and feature extraction, then computes a fixed feature vector for each gesture.
4. **Data Upload**: The collected data, along with the labels and metadata, is batch uploaded to a MongoDB database for storage. To ensure data integrity, this process is designed to cache data locally or retry the upload if the device is offline.
5. **Offline Model Training**: The collected dataset is split into train and test sets to train and evaluate a machine learning model. This stage involves performing cross-validation to tune the model's parameters and feature settings. The model is then exported for deployment.
6. **On-Device Inference**: The final trained model is loaded onto the Raspberry Pi. For each new data window, the system extracts features, predicts the character, and prints the result to the Raspberry Pi terminal. The system uses a confidence threshold to classify new gestures or outputs UNKNOWN when confidence is low.

The full pipeline from sensor data to capturing the final prediction is displayed in Figure 1, below.

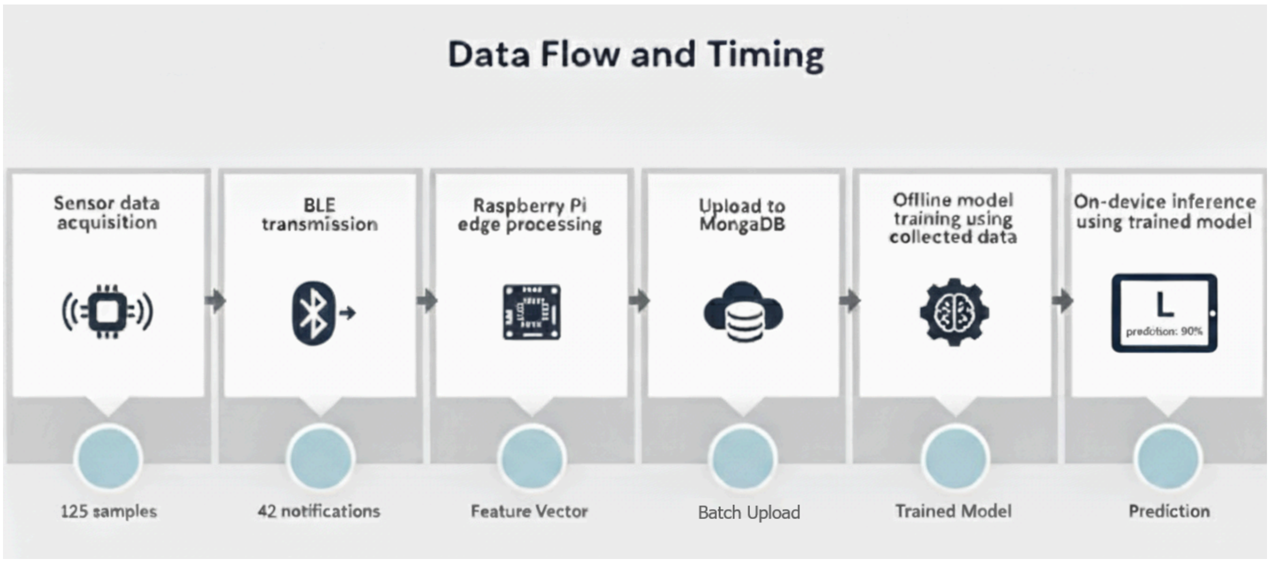

Figure 1: Data Flow and Timing Pipeline Diagram From Capture to Prediction

This design scales because the Arduino treats each capture as an independent window, and the Raspberry Pi processes windows from a small queue, so extra nodes can be added by opening more connections without changing the pipeline. It is responsive because a 2.5-second capture is followed by a half-second for transfer and inference on the Pi, so a character appears about three seconds after the motion ends.

Power use is low because the Arduino only samples during short capture periods and sends compact notification chunks while the Pi performs smoothing, feature extraction, and classification. The choices fit the application because a 50 Hz accelerometer reliably captures in-air lettering, and fixed windows with sequence-checked notifications produce clean training data and consistent live predictions.

4. Machine Learning Model Development

This section details the machine learning approach and training strategy for the project. It outlines the structured process with data collection, feature engineering, model selection protocol, and defined evaluation metrics to validate the model's performance.

4.1. Data Collection Strategy

The data collection strategy approach in this project will focus on meticulous data generation, labelling, and structuring to ensure a comprehensive and high-quality dataset.

Activity Protocols and Data Labelling

The training data is generated by the four group members who will act as participants. Each participant will follow a strict protocol to ensure reliable consistency with data collection.

- ❖ **Protocol**: Participants are instructed to perform each gesture of uppercase letters from (P, R, B, D, G, S, C, L, O, V, Z) while wearing the Arduino device around their dominant writing wrist. Each activity will be repeated 50 times to capture the natural variability regarding gesture speed, amplitude, and individual style.
- ❖ **Postures**: Participants will perform the gestures from a seated position. To streamline the data collection and training process, the model will be trained from this front-facing seated posture. The difference in axis orientation between sitting and standing is minimal, which makes generalisation smoother. Lying poses a slight change in orientation, so the accelerometer x, y, and z axes can be manually adjusted to simulate the data produced in that given position.
- ❖ **Data Labelling**: Each 2.5-second gesture window will be manually labelled with the corresponding gesture character of ‘P’, ‘R’, ‘B’, and so on. This ground truth label will be associated with the raw sensor data before it is stored.
- ❖ **Data Structure**: All of the collected data will be structured and stored in a MongoDB database. Each entry will be a JSON document that the model will take for training, containing the raw accelerometer data array and metadata:
  - ➢ participant_id: Unique identifier for the user
  - ➢ gesture_label: The gesture character performed - example ‘a’
  - ➢ posture: The posture in which the gesture was performed
  - ➢ repetition: Number of times the gesture activity was repeated
  - ➢ arduino_id: Unique ID for the Arduino device

Data Volume and Quality Assurance

The data collection target is to acquire a total of 2200 gesture samples. With four participants each performing the 11 uppercase gestures 50 times (4 * 11 * 50). This strategy is designed to ensure a large and clean dataset for training, validation, and testing. By collecting a high volume of repetitions, the group can account for potential unusable data segments resulting from unforeseen variables such as poorly performed gestures or data corruption during transmission. This will be a key component in ensuring the final model's reliability and accuracy.

4.2. Feature Extraction

The goal of this stage is to transform each 2.5-second accelerometer window (≈125 samples per axis) into a compact, discriminative feature vector. Raw IMU data are noisy and high-dimensional, making them unsuitable for direct classification. Feature extraction reduces dimensionality while retaining information that captures the shape, variability, and dynamics of each gesture, which is essential for distinguishing the 11 uppercase letters in scope P, R, B, D, G, S, C, L, O, V, Z (Wang et al., 2019).

Time Domain Feature (Primary)

Time-domain features are simple statistics derived directly from the signal and are computationally efficient. They are particularly effective for distinguishing wrist gestures because many alphabet shapes can be characterised by their amplitude and variability (Wang et al., 2019). The following features were selected for their direct relevance to the letter set:

- ❖ **Mean and Median (per axis, magnitude)**: capture the average displacement level.
- ❖ **Standard Deviation and Variance**: measure variability, capture spread, loops like O, B, D produce greater variability than linear strokes like L.
- ❖ **Min, Max, Peak-to-Peak Range**: show overall motion scale within the window.
- ❖ **Root Mean Square (RMS) and Signal Energy**: represent gesture intensity, useful for separating small flicks from larger strokes. Curved letters like G, S generate higher energy than angular ones, V, Z.
- ❖ **Zero-crossing Rate**: counts directional reversals, useful for distinguishing multi-axis gestures like "S" from simple linear "L".
- ❖ **First-Difference Statistics** (mean |Δ|, max |Δ|): quantify sharpness; letters with corners like Z, V create higher differences.
- ❖ **Dominant-axis Ratio:** compares variability across axes, one axis letter L, V differs from multi-axis gestures like O, C, S.
- ❖ **Active Time Fraction:** proportion of time above a threshold, helps separate quick stroke-like V from prolonged loops O, B, D.

Optional Frequency Features (Selective use):

If resources allow, a lightweight Fast Fourier Transform (FFT) of the magnitude signal can provide:

- ❖ **Dominant Frequency:** curves and loops are smoother (lower frequency) than jagged strokes like Z.
- ❖ **Spectral Energy Ratio:** useful to distinguish sustained circular motion O, C from more abrupt ones S, Z.

Frequency features are limited to maintain low inference latency on the Raspberry Pi.

Relevance To The Letter Set

These features directly address differences in the selected characters:

- ❖ **Straighted vs Angled:** low variance, high dominant axis ratio like L, V.
- ❖ **Curved vs Open/Closed:** like C vs O, similar spread, but O has a higher active-time fraction and smoother frequency profile.
- ❖ **Multi-lobes vs Single loop:** B vs D, more zero-crossings and energy for B's two lobes.

- ❖ **Similar Pairs:** P and R differ in secondary strokes directionality; G and S differ in axis balance and reversal sequence.
- ❖ **Jagged vs smooth:** Z produces a high first-difference value and multiple zero crossings compared to O.

By targeting these distinctions, the features set to ensure the model can both recognise simple characters with high accuracy and manage confusable pairs by exploiting subtle kinematic differences.

Feasibility

All primary features have a time complexity of *O(N)* over 125 samples, well within the computational limits of the Raspberry Pi. Selective frequency domain features can be included or excluded depending on the performance trade-offs, keeping the system within the real-time constraints defined in earlier sections.

4.3. Model Selection

This project will test a range of classification algorithms to determine the most suitable model for gesture-based HAR. The following algorithms provide a balance of suitability, performance, and interpretability.

K-Nearest Neighbours (KNN)

The KNN algorithm is a straightforward and non-parametric classifier that makes predictions based on the proximity of data points. It classifies a new data point by identifying the majority class of its ‘k’ nearest neighbours in the training dataset (GeeksforGeeks, 2025). It is a great baseline model for this project in terms of its suitability in its simplistic implementation and high interpretability. The algorithm is intuitive as predictions are based on the direct similarity between gestures. Furthermore, its non-parametric nature is advantageous as it makes no assumptions about the underlying data distribution. Making it suitable to effectively model the complex and non-linear patterns found in accelerometer data from wrist motion gestures. However, its ‘lazy’ approach indicates all calculations are performed during prediction. This can lead to higher computational complexity and latency, which are important factors to consider.

Support Vector Machine (SVM)

SVM is a powerful, versatile algorithm that identifies the optimal hyperplane for distinguishing data points into different classes (IBM, 2023). This capability makes it effective in high-dimensional spaces, which is a key characteristic of the multiple sensor readings involved in this project. It is a strong candidate with its robustness and high performance, particularly when dealing with often complex and noisy accelerometer data. Although its decision-making process is less interpretable than simpler models such as the KNN, its ability to achieve high classification accuracy makes it a compelling choice for meeting the project's performance objectives. Furthermore, while its computational complexity is quite high during training, its prediction speed is remarkably fast, which aligns with the project's low-latency objectives.

Decision Tree

A Decision Tree model is structured like a tree where each internal node represents a feature, each branch represents a decision rule, and each leaf node denotes the outcome (Milvus, n.d.). These models are highly interpretable and intuitive, which presents a significant advantage for a proof-of-concept project. In the context of this project, this allows us to easily follow the model's logic to understand why a specific gesture was classified, such as the letter 'C' rather than 'O'. This level of transparency is essential for debugging and validating the efficacy of our sensor data and feature extraction process, a primary goal for proof-of-concept projects. Additionally, Decision Trees are computationally lightweight and provide fast prediction speeds, directly supporting our low-latency objective. Although they are susceptible to overfitting, this can be managed through techniques like pruning or by using more advanced ensemble methods like the Random Forest.

Random Forest

Random Forest is an advanced ensemble learning method that builds upon Decision Trees. It operates by creating multiple Decision Trees during training and outputting the mode of the classes to achieve a more stable and accurate prediction (Najmaei, 2024). It is a strong option for a final model for its ability to mitigate the common overfitting issues with single Decision Trees. By averaging predictions from a diverse set of trees, it significantly improves the model's robustness and generalisation to unseen data. While more computationally complex to train than a single Decision Tree, its prediction latency remains very low, which is essential for meeting the project's real-time objectives. This combination of high accuracy, robustness, and low prediction latency makes it a strong contender for achieving the project's 80% accuracy objective while ensuring a reliable system.

4.4. Evaluation Metrics

Model Performance Metrics

Model performance will be reported using accuracy, precision, recall, and F1-scores. Because the task has 11 classes, one for each accepted character, we will compute these metrics per class, and macro averages will also be provided to ensure each character has equal weight. The main headline numbers will be overall accuracy and macro F1-scores. A confusion matrix over all classes will be included to demonstrate the most common mix-ups and to guide small design tweaks.

Validation and Testing Protocol

Validation will use a standard train-test split of 80/20. Model choice and parameter tuning will use k-fold cross-validation on the training set only, and the final results will be computed once on the held-out test set. If data is collected from multiple people, the test split will be created by participants to check generalisation. Any preprocessing or feature scaling will be applied identically in training and testing.

System Performance Metrics and Targets

For system performance, we will measure end-to-end latency from the window end on the Arduino to the printed character on the Raspberry Pi, reporting the median and 95th percentile. We will also measure the Bluetooth delivery rate as the percentage of windows received with no missing sequence numbers. Initial targets are a macro F1-score between 0.65 and 0.75, a latency under 0.5 seconds, and a delivery rate of at least 95%.

Results Presentation and Error Analysis

Results will be presented as a table of per-class precision, recall, F1-score, and support, with macro averages in the final row, along with a confusion matrix figure. A short error analysis will highlight the highest confusion pairs and any simple adjustments they suggest, such as collecting more samples for weak classes or slightly tuning the window and feature parameters.

5. Testing and Evaluation Plan

To validate the HAR system, testing will cover the hardware and software components, as well as the machine learning model, to ensure stability, accuracy, and reliability across the entire pipeline.

Sensor Accuracy and Data Integrity

IMU readings will be compared against known reference motions to verify accuracy and consistency. These tests will include:

- ❖ Perform the same gesture multiple times to verify sensor responsiveness and accuracy (e.g., raising the arm straight up and down).
- ❖ Comparing sensor output to a known reference motion to detect drift or noise.
- ❖ Comparing raw and processed data to verify preprocessing accuracy.

BLE Stability and Data Transmission Reliability

The BLE connection between the Arduino Nano 33 IoT and Raspberry Pi will be evaluated for:

- ❖ Connection reliability under normal and extended usage.
- ❖ Latency of data transmission to the Raspberry Pi.
- ❖ Packet loss and error handling to ensure reliable data transmission.

Correct Storage and Retrieval from MongoDB

Uploads to MongoDB will be tested to ensure:

- ❖ JSON documents containing the processed data are correctly stored.
- ❖ Data can be retrieved as needed without any corruption.
- ❖ System performance is stable and consistent before, during, and after batch uploads.

Machine Learning Model Testing Using Collected Data

Machine learning models will be tested with the following evaluation strategies:

- ❖ **Metrics**: Accuracy, precision, recall, and F1-score to measure classification performance.
- ❖ **Train-Test Split**: Typically 80/20, training on one portion and testing on unseen data to evaluate generalisation.
- ❖ **K-Fold Cross-Validation**: Training data is divided into (k) folds to tune hyperparameters and validate performance. The best hyperparameters are chosen based on the average across folds.

Real-Time Inference Tests with Unseen Data

The trained model will be deployed on the Raspberry Pi and tested using unseen data to evaluate:

- ❖ Latency from the time of gesture to the time of prediction.
- ❖ Accuracy under variations in gesture speed and posture to assess the model's generalisation to new instances of the same gestures.
- ❖ System stability during continuous operation over an extended session.

Sample Data Collection Plans and Usage Scenarios

Test datasets will be collected from multiple participants performing each gesture in realistic scenarios. This ensures the system performs well across different users, motion speeds, and postures, and supports continuous model refinement and monitoring.

6. Conclusion

This project proposes a wrist-worn HAR prototype that recognises air-written uppercase letters using Arduino Nano 33 IoT and classifies them on a Raspberry Pi via BLE, providing a practical assistive input for users with limited fine motor control. We deliberately scoped the system to 11 uppercase letters (P, R, B, D, G, S, C, L, O, V, Z) to balance feasibility with rigor: it blends distinct/ simple characters (L, O, Z, V) and confusable pairs (P/R, B/D, G/S, C/O) enabling us to demonstrate both strong baseline recognition and stress-test robustness.

Our system design prioritises stable sensing and responsive interaction. Hardware placement is on the dominant wrist with velcro strapping for consistent orientation. The Arduino Nano 33 samples the accelerometers at 50Hz, captures 2.5-second windows (~125 samples/axis), and uses LED cues (1 blink start, 2 blinks stop) to standardise users' timing. Over BLE, windows are sent as sequence-numbered notification chunks (42 per window) to the Raspberry Pi, which reassembles them and discards incomplete windows to protect data integrity. This yields a practical, around ≈3-second end-to-end experience (capture + transfer + inference) while keeping radio payloads small and predictable.

On the edge device, the Pi performs smoothing and feature extraction (time-domain features as primary, with optional lightweight frequency features when resources permit) and the trained model locally for low-latency inference. Collected windows along with labels and metadata are batch-uploaded to MongoDB for long-term storage, analysis, and future retraining, a pattern well-suited to time-series IoT workloads and iterative model improvement. Our data plan collects around 2,200 windows (4 participants x 11 letters x 50 repetitions) in a seated posture; axis remapping will be applied to emulate alternate orientations without expanding the collection burden.

For model development, we will compare KNN, SVM, Decision Tree, and Random Forest classification algorithms to balance interpretability, latency, and accuracy. Evaluation reports per-class metrics and macro-average across the 11 classes, with a confusion matrix to expose error modes. System-level checks cover sensor accuracy, BLE stability, MongoDB storage/retrieval, and real-time inference reliability on unseen data.

In summary, the project demonstrates how wearable IoT and HAR can provide an alternative input method for communication, offering independence and inclusivity for people with mobility impairments. While limiting the scope to eleven characters, the prototype establishes a scalable framework that could be extended to additional letters or symbols in future iterations, strengthening its utility as an accessibility technology.

References

GeeksforGeeks. (2025). *K-Nearest Neighbours (KNN) Algorithm.* https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/

IBM. (2023). What are *Support Vector Machines*(SVMs)?. https://www.ibm.com/think/topics/support-vector-machine

Kristoffersson, A., & Lindén, M. (2022). A Systematic Review of Wearable Sensors for Monitoring Physical Activity. *Sensors*, *22*(2), 573. https://doi.org/10.3390/s22020573

Milvus. (n.d.). *How does a decision tree help with model interpretability?*. https://milvus.io/ai-quick-reference/how-does-a-decision-tree-help-with-model-interpretability

MongoDB. (2025). *Time Series Data And MongoDB: Best Practices Guide*. https://www.mongodb.com/resources/products/capabilities/time-series-best-practices

Najmaei, A. (2024). *What Is Random Forest_ analytics_ IBM.pdf*. https://www.slideshare.net/slideshow/what-is-random-forest-analytics-ibmpdf/267202920

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. https://doi.org/10.1109/jiot.2016.2579198

Wang, J., Chen, Y., Hao, S., Peng, X., & Hu, L. (2019). Deep learning for sensor-based activity recognition: A survey. Pattern Recognition Letters, 119, 3–11. https://doi.org/10.1016/j.patrec.2018.02.010