Brain Activity Monitor

ELEC 498 - Capstone Final Report

Department of Electrical and Computer Engineering
Queen's University

Faculty Supervisors
Dr. Ning Lu
Dr. Paul Hungler

Date of Submission
Monday, April 07, 2024

Group 08
Nicholas Seegobin (20nss, 20246787)
Robin Farber (19rf21, 20215846)
Rodrigo Del Aguila Velarde (20rdav, 20275528)
Samhith Sripada (20ss18, 20232740)

Executive Summary

This project demonstrates a proof-of-concept Brain-Computer Interface (BCI) system that translates non-invasive electroencephalography (EEG) signals into real-time control commands for an RC car. By leveraging a consumer-grade Muse 2 headset, the system acquires high-fidelity EEG data at a 256 Hz sampling rate, which is then processed through a robust machine-learning pipeline.

The pipeline, which is comprised of a filter selector and an action classifier, efficiently maps facial gestures such as blinking, jaw clenching, biting, and eyebrow raising into discrete control commands. These commands are communicated over WiFi using an Arduino Uno WiFi and a motor shield to drive the RC car, achieving an average command acknowledgment latency of approximately 49.5 ms, well within the target of 200 ms.

The design and implementation of the system were guided by a modular approach that facilitated the integration of four key subsystems: data collection, machine learning, user interface, and hardware control. Emphasis was placed on ensuring responsiveness, accuracy, and safety while maintaining a low-cost and non-invasive solution.

Extensive testing and evaluation confirmed the system's ability to reliably sample EEG data, process commands with minimal latency, and perform precise motor control under varied conditions. Additionally, the design addresses critical factors such as stakeholder needs, safety, privacy, and manufacturability. Overall, the project validates the potential of accessible BCI technology for assistive applications and lays the groundwork for future developments in affordable, user-centred control systems.

Table of Contents

Executive Summary i
List of Figures iv
List of Tables v
1. Introduction 1
  1.1. Background and Motivation 1
  1.2. Problem Statement 2
  1.3. Project Scope 2
2. Design 2
  2.1. System Overview 3
    2.1.1. Data Collection 4
    2.1.2. Machine Learning Pipeline 5
    2.1.3. User Interface 5
    2.1.4. Hardware Components 5
  2.2. Functional Requirements and Constraints 6
  2.3. Design Approach 7
    2.3.1. Initial Plan and Project Changes 7
    2.3.2. System Architecture 8
    2.3.3. Tools and Libraries 8
    2.3.4. Evolution of Filter Selector 9
    2.3.5. Evolution of Action Classifier 9
    2.3.6. Evolution of Hardware Design 10
3. Implementation 11
  3.1. Data Collection 11
    3.1.1. Recording Data 11
    3.1.2. Data Preprocessing 13
  3.2. Machine Learning 15
    3.2.1. Data Annotation 15
    3.2.2. Model Configuration 17
    3.2.3. Filter Selector Model 18
    3.2.4. Action Classification Model 19
  3.3. User Interface 21
    3.3.1. Main Dashboard 21
    3.3.2. Brainwave Visualization 21
    3.3.3. Real-Time Interface 22
  3.4. Hardware 22
    3.4.1. Hardware Components 22
    3.4.2. Arduino Communication 23
    3.4.3. RC Car Logic 23

3.5. Bill of Materials and Budget 24
4. Testing and Evaluation 24
4.1. Classifier Performance 24
4.1.1. Evaluation Metrics 24
4.1.2. Filter Selector Model 25
4.1.3. Action Classifier Model 27
4.2. RC Car Testing 29
4.2.1. WiFi Reliability Testing 29
4.2.2. Motor Dynamics and Responsiveness 29
4.2.3. Thermal Performance and Hardware Integrity 30
4.2.4. Motor Locking and Wiring Integrity 30
5. Results 31
5.1. Software Results 31
5.1.1. Data Collection 31
5.1.2. Machine Learning 32
5.1.3. User Interface 34
5.2. Hardware Results 35
5.2.1. Performance Metrics 35
5.2.2. Performance Corroboration 35
5.2.3. Data Analysis and Insights 37
6. Important Factors 37
6.1. Stakeholder Needs 37
6.2. Safety 37
6.3. Privacy 38
6.4. Codes/Standards 38
6.5. Manufacturability 38
6.6. Ethics and Cost 39
7. Conclusions and Recommendations 39
7.1. Technical Lessons Learned 39
7.2. Lasting Impact 40
7.3. Further Research and Alternative Approaches 40
7.4. Commercialization and Manufacturing Considerations 41
7.6. Market Size and Growth 42
7.7. Societal and Cultural Impacts 43
8. Effort Distribution 44
9. References 44

List of Figures

Figure 1: The Muse 2 headset used for collecting EEG signals. 3
Figure 2: High-level system architecture showing how data flows within the four subsystems. 3
Figure 3: The 10-20 electrode placement system showing the electrodes on the Muse 2. 4
Figure 4: Comparison of raw (left) and filtered (right) EEG signals illustrating a blink gesture. 4
Figure 5: External view of the modified RC car. 6
Figure 6: Internal view of the RC car showing the Arduino, motor shield, and dual battery setup. 6
Figure 7: Snippet of the rule-based function for filter selection based on signal features. 9
Figure 8: Function that connects to muselsl stream and records a 2.5-second EEG snippet using pylsl. 11
Figure 9: Raw EEG signals for each gesture, shown over a 2.5-second window. 12
Figure 10: Implementation of the bandpass_filter() function used to clean EEG signals. 13
Figure 11: Filtered EEG signals of each gesture using a Butterworth bandpass filter. 14
Figure 12: Signal smoothing and threshold-based detection used in the original labeling script. 15
Figure 13: Labeled version of the blink signal shown in Figure 4. 16
Figure 14: JSON output from the original labeling script. 16
Figure 15: Training Random Forest model using extracted features and gesture labels. 17
Figure 16: The features extracted for the filter selector model. 19
Figure 17: The features extracted for the action classifier model. 20
Figure 18: Wiring between Arduino, motor shield, motors, and power source 23
Figure 19: Initial filter selector model (left) and model with improved frequency features (right). 25
Figure 20: Final filter selector performance showing 95% accuracy after parameter tuning. 26
Figure 21: Initial classifier (left) and classifier with undersampled blink data (right). 28
Figure 22: Performance of the third model (left) and final model (right). 28
Figure 23: Muse 2 headset detected and connected via Bluetooth in the main dashboard. 31
Figure 24: Real-time EEG visualization from all four Muse 2 channels with live signal updates. 32
Figure 25: End-to-end processing latency for all four gestures, each under 14 ms. 33
Figure 26: Live brainwave monitor displaying Delta, Theta, Alpha, Beta, and Gamma bands. 34
Figure 27: Task Manager showing zero percent GPU usage during real-time system operation. 35
Figure 28: Blink data sample confirming 256Hz sampling frequency 36
Figure 29: Connection between Arduino and WiFi network. 36
Figure 30: Commands received by Arduino with timestamps 36
Figure 31: Top-down view of car electronics. 36
Figure 32: Wheels moving forward. 36
Figure 33: Wheels moving backward. 36
Figure 34: Wheels steering left. 36
Figure 35: Wheels steering right. 36

List of Tables

Table 1: Software specifications outlining the software requirements of the BAM system. 6
Table 2: Hardware specifications outlining the hardware requirements of the BAM system. 7
Table 3: Key design constraints that influenced system implementation. 7
Table 4: Summary of all the tools used and their purpose in the project. 8
Table 5: Sample raw EEG data showing the header and two rows captured using the Muse 2. 12
Table 6: Bill of materials for the system and associated costs. 24
Table 7: Distribution of effort expended by each group member. 44

1. Introduction

1.1. Background and Motivation

Brain-Computer Interfaces are systems that allow users to interact with machines through electrical brain activity, most captured using EEG. These interfaces have gained significant traction in various applications, including assistive technologies, medical rehabilitation, gaming, and advanced human-computer interaction systems [1]. The increasing demand for hands-free control mechanisms has positioned BCIs at the forefront of accessible technology development, particularly for individuals with motor impairments who require alternative means of device interaction [2].

The Brain Activity Monitor (BAM) project addresses the critical need for affordable, functional BCI technology. Current BCI solutions often present significant barriers to widespread adoption, including prohibitive costs, complex setup requirements, and limited accessibility. According to recent market analyses, commercial-grade EEG devices can cost thousands of dollars, placing them beyond the reach of many potential users and researchers [2]. BAM proposes a novel approach that combines consumer-grade hardware with sophisticated machine learning techniques to create an accessible BCI system that translates brainwave signals into tangible device control commands.

Our implementation focuses explicitly on the translation of muscle-activated brain signals which are reliably detectable through non-invasive EEG. These signals present an ideal entry point for BCI technology as they are voluntarily controllable by users and produce distinctive electrical patterns that can be classified accurately. By enabling control of devices through these natural movements, BAM serves as a proof-of-concept for accessible BCI development and a foundation for future applications in assistive technology, particularly for individuals with limited mobility [3]. Furthermore, the non-invasive nature of our approach addresses significant ethical considerations surrounding brain data collection, prioritizing user safety and privacy while still delivering meaningful functionality.

1.2. Problem Statement

The core challenge addressed by the BAM project is the development of an affordable, reliable system that can translate brainwave signals into real-time device control commands. While consumer-grade EEG headsets like the Muse 2 have made brain activity monitoring more accessible, these devices remain underutilized for practical control applications due to significant technical barriers in signal processing, feature extraction, and real-time classification [4].

Specifically, facial movement related EEG signals provide an accessible entry point for BCI control but require sophisticated filtering and classification techniques to be reliably detected amid the considerable noise inherent in EEG data [1]. Current approaches often struggle with achieving both the accuracy and speed required for intuitive device control, particularly when using affordable consumer hardware with limited electrode coverage. Our project addresses this gap by developing a comprehensive pipeline that transforms raw EEG data from the Muse 2 headband into distinct, reliable control commands for external hardware, demonstrating how accessible BCI technology can be implemented without specialized medical-grade equipment.

1.3. Project Scope

The BAM project focuses on developing a complete brain-computer interface system that translates specific facial muscle movements detected via EEG signals into directional controls for an RC car. The system encompasses the full BCI pipeline, including signal acquisition through the Muse 2 headband, preprocessing and feature extraction of raw EEG data, development and training of classification models, and the real-time translation of detected actions into wireless hardware commands. The project operates within constraints of limited budget, consumer-grade hardware capabilities, time constraints of the academic term, and the inherent limitations of the Muse 2's electrode placement.

2. Design

This section describes the design of the Brain Activity Monitor system. It begins with an overview of the system and its key components. It then outlines the functional requirements and design constraints that guided development. Finally, it explains the design approach, including the major decisions that shaped the final implementation.

2.1. System Overview

The Brain Activity Monitor is a brain-computer interface that allows users to control an RC car using EEG signals. These signals are produced by facial gestures, including blinking, jaw clenching, biting, and eyebrow raises. Each gesture triggers a specific directional movement, allowing users to drive the car without physical input.

Figure 1: The Muse 2 headset used for collecting EEG signals.

To enable this functionality, the system uses the Muse 2 headset, a non-invasive, consumer-grade EEG device that captures brain activity associated with each gesture (see Figure 1). The captured EEG signals are then processed through four main subsystems: data collection, machine learning (ML), user interface, and hardware components (see Figure 2). The following sections describe how these components work together to enable device control using EEG signals.

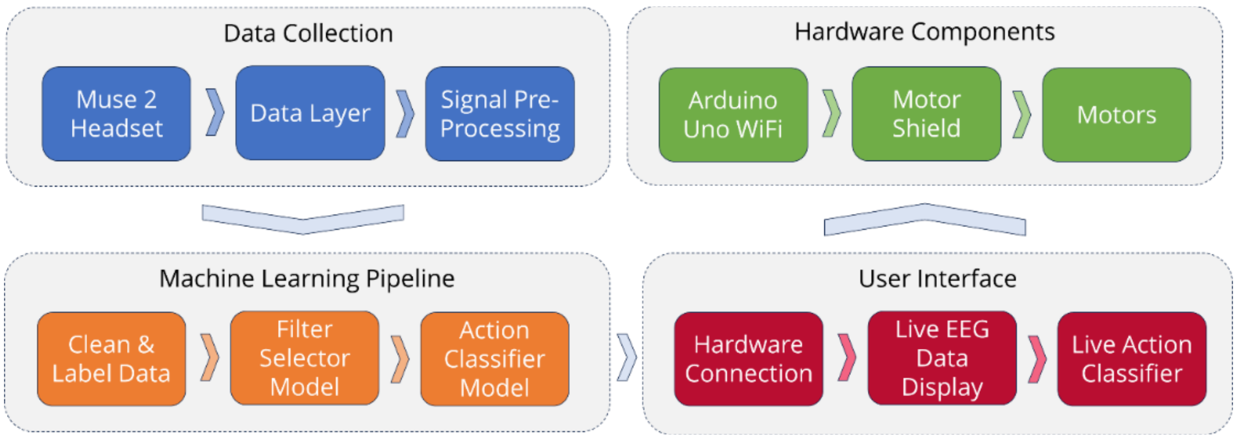


Figure 2: High-level system architecture showing how data flows within the four subsystems.

2.1.1. Data Collection

The data collection subsystem captures EEG signals using the Muse 2 headset. The headset has four electrodes (TP9, AF7, AF8, and TP10) that capture electrical activity from the frontal and temporal areas of the head, as shown in Figure 3.

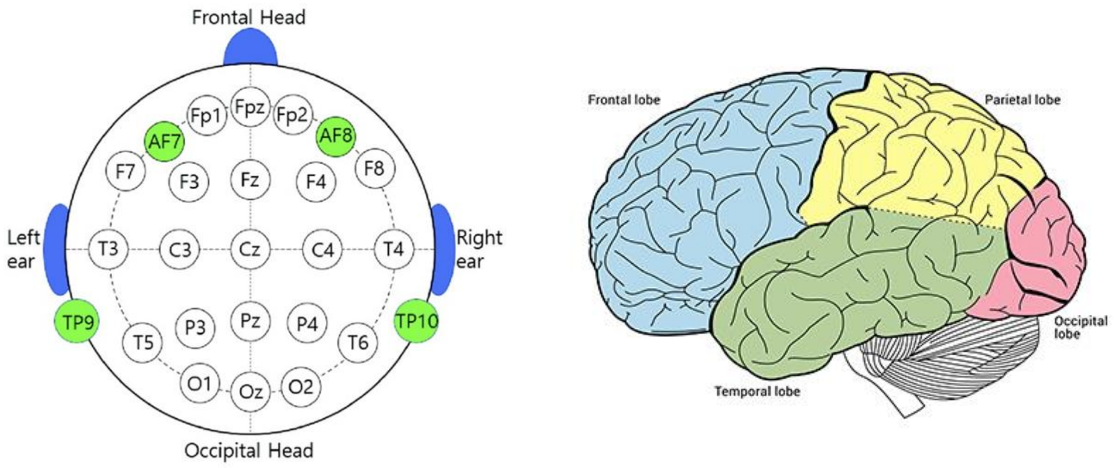


Figure 3: The 10-20 electrode placement system showing the electrodes on the Muse 2.

During training, the data collection subsystem recorded samples for each gesture and saved them in CSV files. Bandpass filters were applied based on the typical frequency ranges of each action to isolate the most relevant signal features (see Figure 4). Blinking was filtered within the delta band (0.1 to 4 Hz), while jaw clenching and biting were filtered in the beta band (20 to 50 Hz). Eyebrow raises were isolated using a slightly narrower beta band (25 to 40 Hz). This filtering step was initially applied during training and later handled dynamically by the filter selector model.

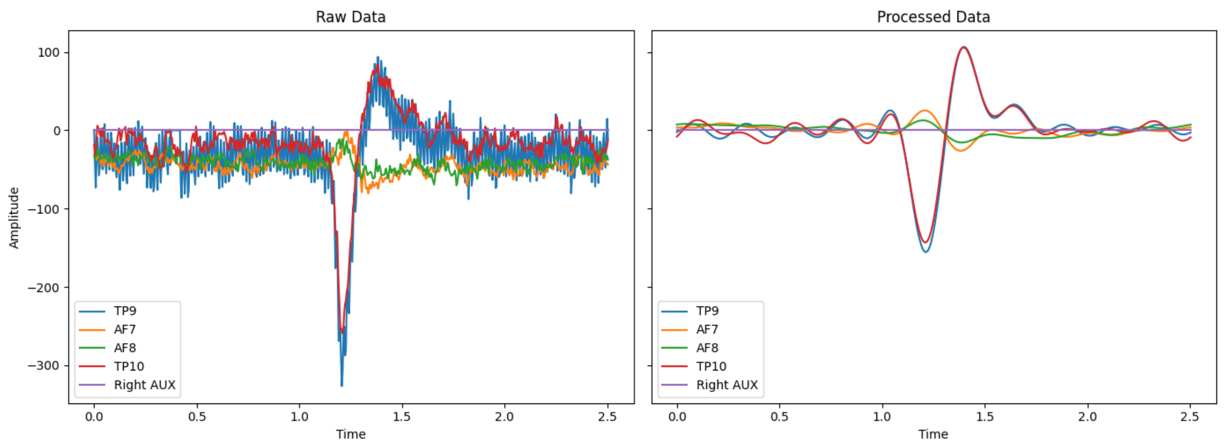


Figure 4: Comparison of raw (left) and filtered (right) EEG signals illustrating a blink gesture.

In the final product, the data collection subsystem streams EEG data in real time. The incoming signals are continuously batched into CSV files and stored in a buffer. At this stage, the batches are passed to the ML pipeline for further processing and classification.

2.1.2. Machine Learning Pipeline

The machine learning subsystem identifies gestures from EEG signals. It consists of two components: a filter selector and an action classifier. The filter selector dynamically applies an appropriate frequency filter to incoming EEG signals. The action classifier then determines the specific gesture based on the filtered signal. During real-time operation, EEG signals are processed by the filter selector, and the resulting prediction from the action classifier is transmitted to the hardware components subsystem.

2.1.3. User Interface

The user interface subsystem manages communication among all system components. It primarily handles the flow of data between the Muse 2 headset, the ML models, and the hardware components subsystem. Alongside communication, the interface also offers real-time visualization of the EEG signals. Users can view the live EEG waveforms from all four electrodes, and the interface features a breakdown of each channel into its individual brainwave bands (delta, theta, alpha, beta, and gamma). The UI also presents the most recent predicted action.

2.1.4. Hardware Components

The hardware components subsystem receives commands from the user interface and translates them into physical movement. It consists of an Arduino Uno WiFi board connected to a motor shield that drives the RC car's motors (see Figure 6). Using network protocols, the Arduino can receive commands over WiFi from another device on the same network. Upon receiving a command, the Arduino activates the motors, producing real-time motion in the desired direction. This subsystem is designed to respond quickly, ensuring smooth, responsive car control.

Figure 5: External view of the modified RC car.

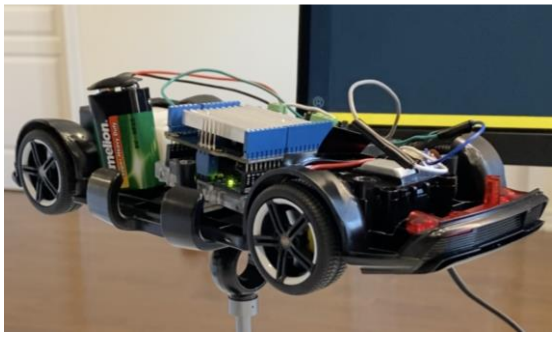


Figure 6: Internal view of the modified RC car.

2.2. Functional Requirements and Constraints

This section outlines the key software and hardware requirements that guided the design of the BAM system and the main constraints that shaped development. The requirements presented here are adapted from the initial blueprint design document to reflect the updated needs of the finalized system [5]. The software table outlines functional, interface, and performance requirements with corresponding test results. The hardware table details specifications for data streaming, communication, and control. The constraints table summarizes real-world limitations, including hardware, budget, and timeline. Table 1 shows the software specifications.

Table 1: Software specifications outlining the software requirements of the BAM system.

ID	Specification	Specification Met?
1	**Functional Requirements**	**-**
1.1	Acquire EEG signals from the Muse 2 headset in real time	Yes
1.2	Apply preprocessing (filtering) before classification	Yes
1.3	Detect four actions (blink, jaw clench, eyebrow raise, bite)	Yes
1.4	System classifies actions in real time	No
1.5	Arduino controls car's motors based on received command	Yes
1.6	Each action mapped to a unique directional control command	Yes
2	**Interface Requirements**	**-**
2.1	Provide a GUI for starting/stopping EEG streaming	Yes
2.2	Show live EEG signals from each of the four electrodes	Yes
2.3	Display brainwave bands (delta, theta, alpha, beta, gamma)	Yes
2.4	Display the current predicted action	Yes
2.5	Use WiFi to send commands from PC to Arduino	Yes
2.6	Muse 2 must stream EEG data to the system via Bluetooth	Yes
3	**Performance Requirements**	**-**
3.1	Classifies gestures with at least 85% accuracy	Yes
3.2	Must respond to a gesture in 2.5 seconds or less	Yes
3.3	Signal processing and ML model latency ($\leq$110 ms $\pm$ 10 ms)	Yes
3.4	Must run in real time on a standard laptop (no GPU)	Yes
3.5	Live EEG display must update with less than 300 ms latency	Yes

Overall, the team met most of the software requirements listed in Table 1. The system successfully acquires EEG signals, applies filters, and detects all gestures. The only unmet requirement is continuous real-time classification. Although data streams continuously and both models run in the background, classification only occurs when the user manually triggers it. This limits the system to semi-real-time operation.

Table 2 presents the hardware specifications. These focus on data transmission, response time, power usage, and motor control. Most hardware requirements were met within acceptable tolerances, allowing the system to operate effectively within its design constraints.

Table 2: Hardware specifications outlining the hardware requirements of the BAM system.

ID	Specification	Target Value	Tolerance	Achieved Value
4	**Hardware Requirements**	**-**	**-**	**-**
4.1	Data Transmission Rate (Muse 2 Headband)	256 Hz	± 10 Hz	256 Hz
4.2	Arduino receives command within 200 ms	≤ 200 ms	± 20 ms	~ 150 ms
4.3	RC car runs on onboard battery	5 hours	±5%	3 hours
4.4	Wireless communication from PC to Arduino	≤ 10% loss	±5%	≤ 5% loss
4.5	Motor shield controls 2 DC motors	≥ 2 motors	-	2 motors

Most hardware targets were met. The only exception was battery life, which remained within acceptable limits due to design trade-offs. Key constraints influencing these decisions are summarized in Table 3.

Table 3: Key design constraints that influenced system implementation.

Constraint	Description
5	**Constraints**
5.1	Hardware Limitation: Muse 2 headset only has 4 EEG channels (TP9, AF7, AF8, TP10)
5.2	Non-Invasive System: All components must be consumer-grade and non-invasive
5.3	Budget Limit: Total system cost must not exceed $600
5.4	Local Processing: Processing and classification must occur locally on a laptop (no cloud)
5.5	Limited Time: Project must be fully completed within eight months
5.6	Power Constraints: Hardware components must operate on portable battery power

2.3. Design Approach

2.3.1. Initial Plan and Project Changes

The original plan for the BAM system was built around a phased development approach with three main stages. The first phase aimed to toggle a light using relaxed and focused states as a binary switch. The second phase would introduce four-directional control of an RC car, and the final phase proposed integrating both systems to control a claw machine.

The Muse 2 headset was selected early in the project for its accessibility, low cost, and support through the muselsl library. However, early testing showed it could not reliably distinguish between complex thought patterns due to its limited electrode placement, which does not cover motor cortex regions like C3, Cz, and C4. With coverage only at TP9, AF7, AF8, and TP10, it was better suited to detecting muscular activity from facial gestures. As a result, the team shifted from thought-based classification to gesture-based control, narrowing the system's focus to controlling an RC car using four distinct actions.

2.3.2. System Architecture

The system architecture was designed to support modular development, with each team member working on a separate subsystem. This separation allowed parallel development and simplified debugging by isolating issues within specific modules. It also made the system more flexible, enabling the replacement of components without affecting the rest of the system. For example, the rule-based logic in the filter selector was eventually replaced with a machine learning model without requiring changes to the other subsystems.

2.3.3. Tools and Libraries

A variety of software and hardware tools were used to support the BAM system, summarized in Table 4.

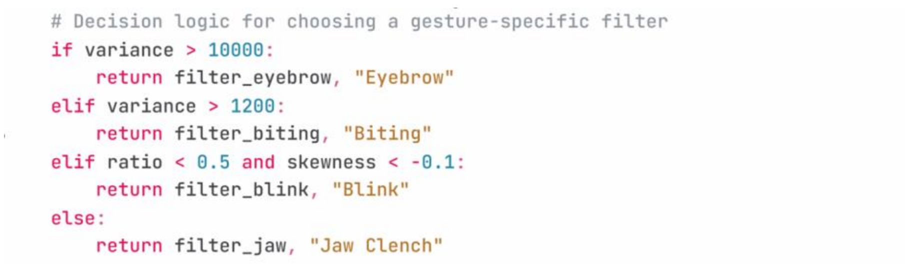
Table 4: Summary of all the tools used and their purpose in the project.

Tool	Purpose
Visual Studio Code	Primary development environment
Arduino IDE	Secondary development environment
Python	Programming language used for most software components
Arduino	Used for writing the code that runs on the Arduino
muselsl	Streams EEG data from Muse 2 using LSL interface
pylsl	Records EEG snippets and pulls raw EEG samples
NumPy	Numerical operations and feature extraction
Matplotlib / Seaborn	Visualization of signals and model evaluation
SciPy	Signal preprocessing (Butterworth filter, filtfilt)
scikit-learn	Trained Random Forest architecture for both models
PyQt5	GUI for EEG visualization, gesture predictions, and controls
Hardware Tools	Soldering tools, jumper wires, multimeter

2.3.4. Evolution of Filter Selector

One of the early design challenges was determining how to filter EEG signals before sending them to the action classifier. While each gesture had distinct frequency characteristics, choosing the right bandpass filter wasn't straightforward. The team explored a few different ideas, such as sending four separately filtered versions of the raw EEG data to the classifier, running a separate classifier for each gesture, or finding a way to select the best filter dynamically. The final design followed the third approach, where only one filtered stream is sent to the classifier.

The initial solution was a hardcoded logic function that checked for features like variance, skewness, and power ratios, as shown in Figure 7. While this method worked for certain samples, it was too rigid to handle variations across different recordings or users. The hand-tuned conditions often failed to generalize and became difficult to maintain as the system grew. To improve flexibility and accuracy, the logic was replaced with a machine learning model called the filter selector.



```python
# Decision logic for choosing a gesture-specific filter
if variance > 10000:
    return filter_eyebrow, "Eyebrow"
elif variance > 1200:
    return filter_biting, "Biting"
elif ratio < 0.5 and skewness < -0.1:
    return filter_blink, "Blink"
else:
    return filter_jaw, "Jaw Clench"
```

Figure 7: Snippet of the rule-based function for filter selection based on signal features.

The filter selector was trained on labeled EEG data using a compact set of features that captured key signal characteristics. Instead of relying on fixed thresholds, it learned to recognize patterns in the signal and select the most suitable filter for each input. This made the system more dynamic and reliable during real-time operation.

2.3.5. Evolution of Action Classifier

The action classifier determines which gesture the user performed based on preprocessed EEG data. While the system was originally intended to use a machine learning model for this task, the team briefly explored a threshold-based approach after it showed strong early performance during the development of the automatic labeling system.

This labeling system identified gestures by applying a moving average filter to selected EEG channels and detecting spikes that crossed predefined thresholds. Each gesture was associated with a specific electrode and threshold, and when the smoothed signal exceeded that threshold, the system marked a gesture. The method worked well on isolated samples, offering fast, lightweight performance.

However, the rule-based approach proved unreliable in more realistic conditions. It relied on only one channel per gesture and did not account for signal variability across users or sessions. As gesture strength and baseline levels changed, the system often missed or misclassified inputs. Due to these limitations, the team moved forward with a machine learning model for classification, which offered greater flexibility and accuracy during real-time use.

2.3.6. Evolution of Hardware Design

The original hardware design planned for the Arduino Uno WiFi to communicate directly with the PC over WiFi. It also used a single 9V battery to power both the Arduino and the motor shield. This setup was simple for prototyping but caused overheating, unstable performance, and potential hardware damage during testing. WiFi communication also proved unreliable due to missing error handling.

To address these issues, the 4×AA battery pack powering the Arduino was first replaced with a dedicated 5V, 2-amp adapter. This provided a constant current and resolved the WiFi instability caused by inconsistent voltage. The 9V battery powering the motor shield was then replaced with a 4×AA pack, which delivered less power but significantly reduced motor heating. Communication reliability was also improved by adding error-handling routines on the PC to detect and resend lost messages.

The final system used a Telnet interface over WiFi to send classification updates. The Arduino processed incoming labels using a switch statement and sent acknowledgments back to confirm receipt, helping maintain response times under 200 milliseconds.

Additional improvements included soldered wiring for durability, dual power supplies for stability, and compact component placement within the RC car. Bluetooth and hosted network options were tested but dropped due to recurring power and connection issues. The final WiFi setup offered the best balance of speed, reliability, and ease of integration.

3. Implementation

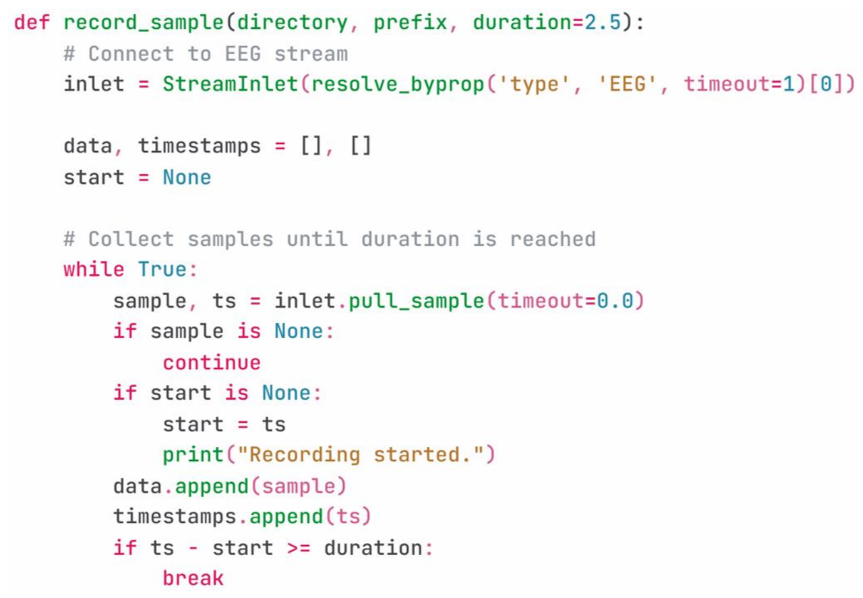
This section details how the BAM was implemented based on the design decisions described earlier.

3.1. Data Collection

3.1.1. Recording Data

Recording EEG data is a fundamental part of the BAM system. The original script was designed to collect training data and was later adapted for real-time classification. Much of the core functionality remained the same across both versions.

During training, data collection was performed using a standalone GUI tool built with Tkinter. The interface featured four buttons, each corresponding to a different gesture. Pressing a button triggered the record_sample() function (see Figure 8), which used the pylsl library to collect a 2.5-second EEG snippet from the active Muse 2 stream. Each recording consisted of 640 samples per channel and was saved to one of four gesture-specific folders based on the button pressed.



```python
def record_sample(directory, prefix, duration=2.5):
    # Connect to EEG stream
    inlet = StreamInlet(resolve_byprop('type', 'EEG', timeout=1)[0])

    data, timestamps = [], []
    start = None

    # Collect samples until duration is reached
    while True:
        sample, ts = inlet.pull_sample(timeout=0.0)
        if sample is None:
            continue
        if start is None:
            start = ts
            print("Recording started.")
        data.append(sample)
        timestamps.append(ts)
        if ts - start >= duration:
            break
```

Figure 8: Function that connects to muselsl stream and records a 2.5-second EEG snippet using pylsl.

The function recorded voltage readings from all four Muse 2 channels along with their timestamps. These raw signals, seen in Figure 9, illustrate the typical output of the recording system for each gesture.

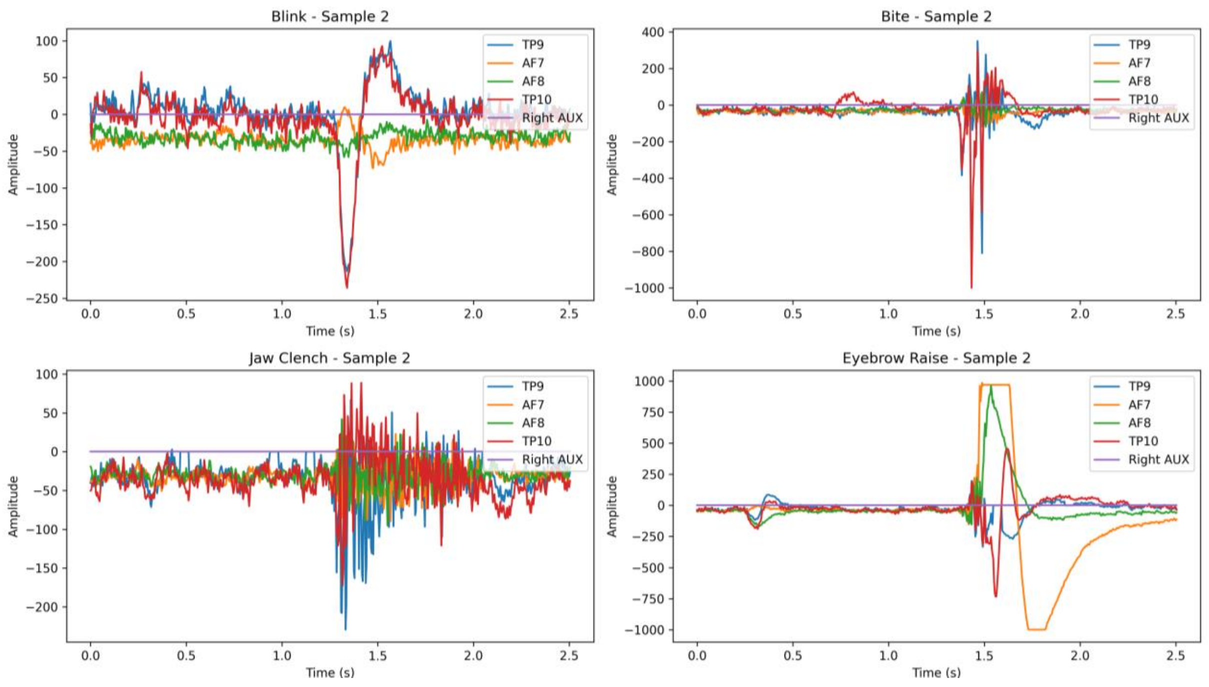


Figure 9: Raw EEG signals for each gesture, shown over a 2.5-second window.

Once the snippet was collected, it was passed to the save_to_csv() function. This function saved the EEG and timestamp data to a new CSV file named according to the gesture and stored it in the appropriate directory. The same recording function was reused in the real-time system. A single "Record Action" button was used to save incoming data into a shared buffer directory, as classification would be handled later by the machine learning pipeline. All recordings are saved to this buffer directory using generic filenames like buffer_01.csv. A sample of the recorded EEG data is shown in Table 5.

Table 5: Sample raw EEG data showing the header and two rows captured using the Muse 2.

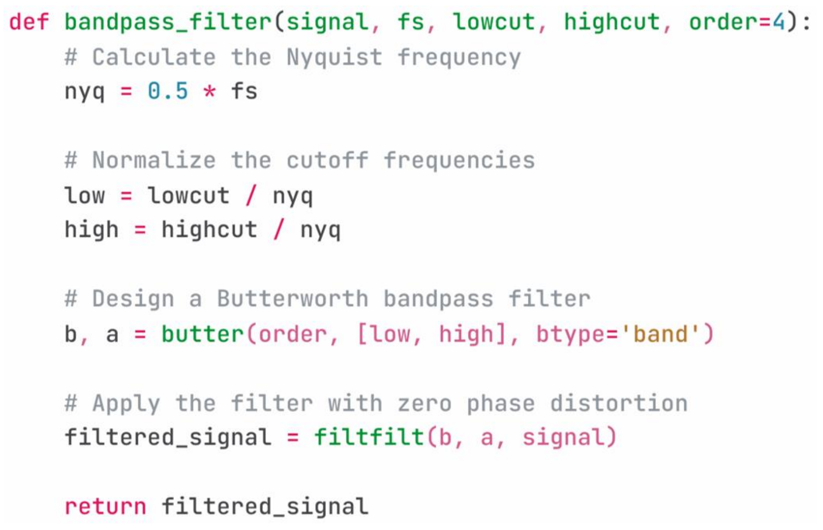
Timestamps	TP9	AF7	AF8	TP10
1.74E+09	-0.48828	-34.1797	-31.25	-18.5547
1.74E+09	-44.9219	-30.7617	-36.6211	-31.25

The voltage values in the table are expressed in microvolts (μV) and represent the electrical potential at each electrode relative to a reference point. Negative values are normal and indicate the electrode was momentarily lower in potential than the reference.

3.1.2. Data Preprocessing

To prepare EEG data for machine learning, a preprocessing script was developed to clean the raw signals using a bandpass filter. This isolated frequency ranges relevant to each gesture and reduced noise from movement and external artifacts.

The script processed each CSV file by loading the EEG values, applying a bandpass filter to each channel, and saving the cleaned signal to a new CSV file. The bandpass_filter() function was responsible for applying the filter (see Figure 10).



```python
def bandpass_filter(signal, fs, lowcut, highcut, order=4):
    # Calculate the Nyquist frequency
    nyq = 0.5 * fs

    # Normalize the cutoff frequencies
    low = lowcut / nyq
    high = highcut / nyq

    # Design a Butterworth bandpass filter
    b, a = butter(order, [low, high], btype='band')

    # Apply the filter with zero phase distortion
    filtered_signal = filtfilt(b, a, signal)

    return filtered_signal
```

Figure 10: Implementation of the bandpass_filter() function used to clean EEG signals.

The bandpass_filter() function starts by calculating the Nyquist frequency, which is half the sampling rate. This value is important because it defines the highest frequency that can be represented in a digital signal without distortion. Next, the low and high cutoff frequencies are divided by the Nyquist frequency to scale them between 0 and 1. This scaling is necessary because the filter design function, scipy.signal.butter(), expects the cutoff values to be in this format to correctly define the range of frequencies to keep. Without this step, the filter would not work as intended. By normalizing the cutoff frequencies this way, the function ensures that the filter behaves consistently, even if the sampling rate changes.

Once the cutoff values are normalized, the function uses the scipy.signal.butter() function to create a fourth-order Butterworth bandpass filter. This filter order was selected because it balances separating the desired frequencies and keeping the signal stable. Lower-order filters may not remove enough noise, while higher-order filters can become unstable or distort the signal. After the filter is created, it is applied using the filtfilt() function. This function processes the signal twice, first in the forward direction and then in reverse. Filtering in both directions cancel out any timing shifts in the signal, which are known as phase distortions. Preventing phase distortion is especially important for EEG data because the timing and shape of the signal are essential for detecting short gestures like blinks or jaw clenches. Using filtfilt() helps preserve the original waveform while still removing unwanted frequency components, as shown in the filtered signals in Figure 11.

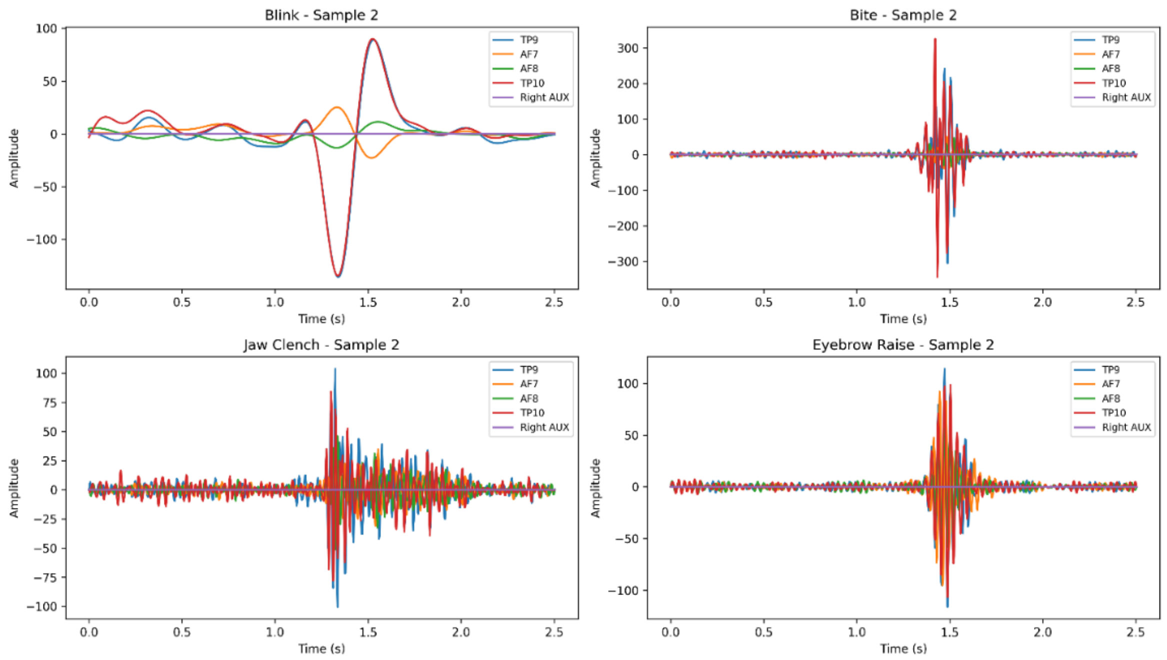


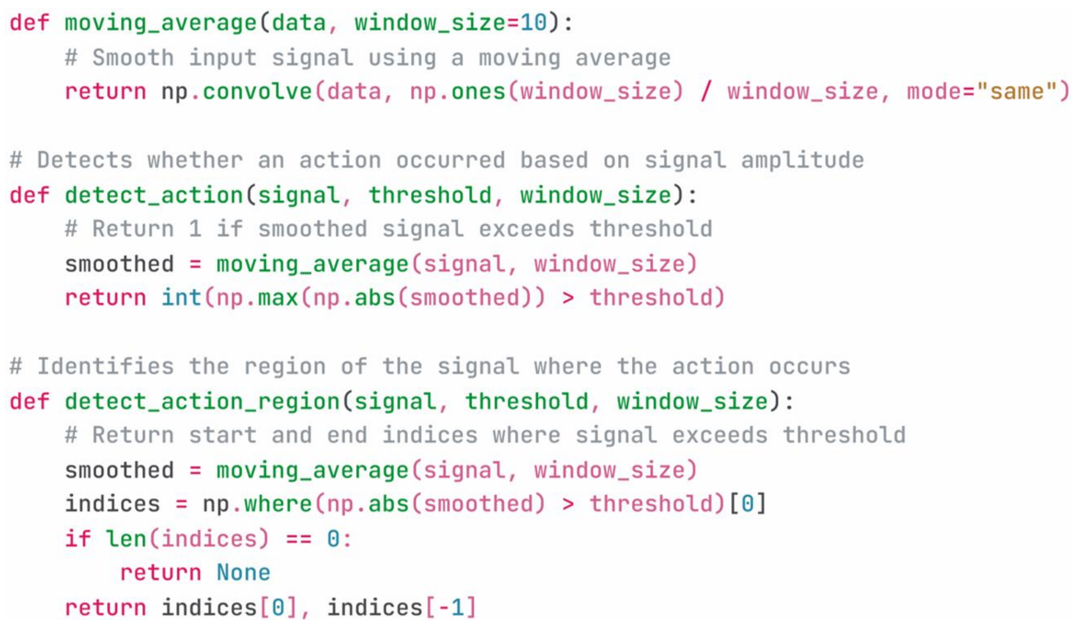
Figure 11: Filtered EEG signals of each gesture using a Butterworth bandpass filter.

As introduced in Section 2.2.1, each gesture used a predefined bandpass range based on its known frequency characteristics. Reusing this same filtering logic in both the training pipeline and the real-time system ensured consistency. By applying the same signal transformation before classification, the model could reliably interpret gestures in live operation using the patterns it had seen during training. With this preprocessing step in place, the system was ready to move into the ML phase.

3.2. Machine Learning

3.2.1. Data Annotation

Before any machine learning models could be trained, the EEG data first had to be labelled. Initially, this was done using a threshold-based detection script that applied a moving average filter across all EEG channels (see Figure 12).



```python
def moving_average(data, window_size=10):
    # Smooth input signal using a moving average
    return np.convolve(data, np.ones(window_size) / window_size, mode="same")

# Detects whether an action occurred based on signal amplitude
def detect_action(signal, threshold, window_size):
    # Return 1 if smoothed signal exceeds threshold
    smoothed = moving_average(signal, window_size)
    return int(np.max(np.abs(smoothed)) > threshold)

# Identifies the region of the signal where the action occurs
def detect_action_region(signal, threshold, window_size):
    # Return start and end indices where signal exceeds threshold
    smoothed = moving_average(signal, window_size)
    indices = np.where(np.abs(smoothed) > threshold)[0]
    if len(indices) == 0:
        return None
    return indices[0], indices[-1]
```

Figure 12: Signal smoothing and threshold-based detection used in the original labeling script.

The script was then used to label all the filtered EEG recordings individually. It smoothed the selected channel using a moving average filter and checked whether the signal exceeded a predefined threshold. If the threshold was crossed, the script identified the start and end points of the gesture and marked the corresponding region in the data. This threshold-based region was later applied across all EEG channels to generate training labels. Figure 13 shows a labeled blink event, with the red shaded region indicating the detected action window. For space, only one EEG channel is shown.

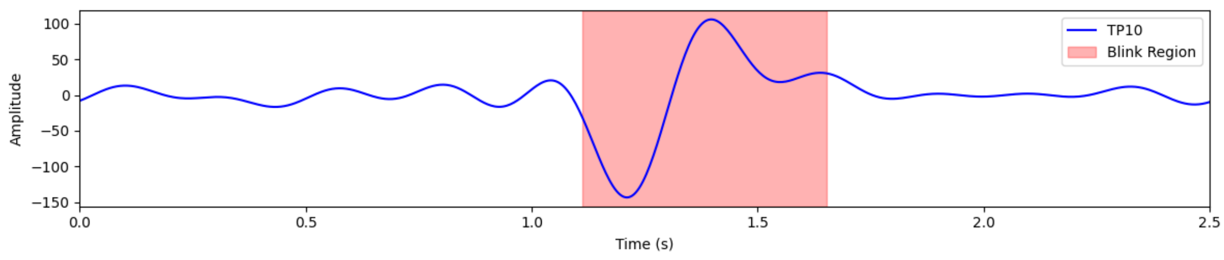


Figure 13: Labeled version of the blink signal shown in Figure 4.

If an action was detected, the script assigned an actionLabel to indicate the type of gesture. It also recorded an actionRegion, which marked the start and end row indices in each CSV file where the signal exceeded the threshold. The final output was a JSON file containing the actionLabel, the actionRegion, and the full EEG data under the museHeadsetData field. Labels were encoded numerically as follows: 0 for biting, 1 for blinking, 2 for eyebrow raises, and 3 for jaw clenches. An example of the JSON file format can be seen in Figure 14.

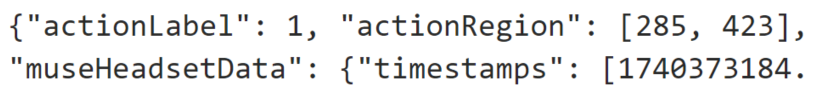

```
{"actionLabel": 1, "actionRegion": [285, 423],
"museHeadsetData": {"timestamps": [1740373184.
```

Figure 14: JSON output from the original labeling script.

The results of the first model trained using the labeled data were disappointing. While some gestures like blinking were detected with high accuracy, others showed very poor performance. The issue came from how the data was labeled. The original method relied on detecting the exact region in the EEG signal where the gesture occurred, but this region detection step often produced inconsistent results.

After reviewing the approach, it became clear that knowing the exact start and end of the gesture in the signal was not necessary. The entire 2.5-second recording already focused on a single gesture, so the whole sample could be treated as one labeled example.

To fix this, the team switched to a simpler labeling method. Instead of using the detection script, the gesture label was taken directly from the filename of the recording. The full 2.5-second EEG snippet was used as the training input, and its filename provided the correct label. This new method removed the need for region detection, reduced labeling errors, and led to much better performance across all four gestures. In total, the final dataset included 50 samples per gesture, resulting in approximately 200 total labeled training examples.

3.2.2. Model Configuration

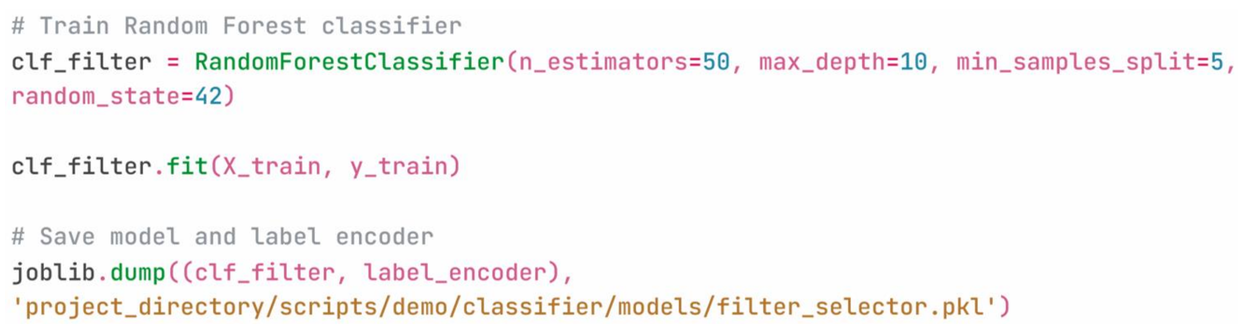
This section describes the shared configuration used for both the filter selector and action classifier models. While the models use different inputs and extract different features, they rely on the same data loading process and Random Forest model architecture.

3.2.2.1. Dataloader

The dataloader() function prepares EEG recordings for training by loading, formatting, and labeling multichannel data from CSV files. It scans data directories for recordings, removes timestamps, and transposes the EEG data so each channel is a row. This structure helps simplify feature extraction by ensuring a consistent format across all samples. Additionally, each sample is labeled based on its filename (e.g., blink_01.csv → blink). After labeling, the function returns a NumPy array of EEG samples, an array of corresponding labels, and a dictionary specifying the sampling frequency.

3.2.2.2. Model Parameters

Both the filter selector and action classifier models were implemented using scikit-learn's RandomForestClassifier (see Figure 15). Each model was trained with 50 trees, a maximum tree depth of 10, and a minimum of 5 samples required to split an internal node. These parameters were selected to balance model complexity, generalization, and execution time, which are all factors that are important when working with limited data.



```
# Train Random Forest classifier
clf_filter = RandomForestClassifier(n_estimators=50, max_depth=10, min_samples_split=5,
random_state=42)

clf_filter.fit(X_train, y_train)

# Save model and label encoder
joblib.dump((clf_filter, label_encoder),
'project_directory/scripts/demo/classifier/models/filter_selector.pkl')
```

Figure 15: Training Random Forest model using extracted features and gesture labels.

Using 50 trees provided enough diversity to make stable, accurate predictions without unnecessary slowdown. Fewer trees risked underfitting by missing important EEG patterns, while too many could reduce real-time responsiveness without added benefit.

Limiting tree depth to 10 helped prevent overfitting on the small, noisy EEG dataset. With only ~200 samples, deeper trees risked capturing noise instead of meaningful patterns, while constrained depth promoted more generalizable decision boundaries.

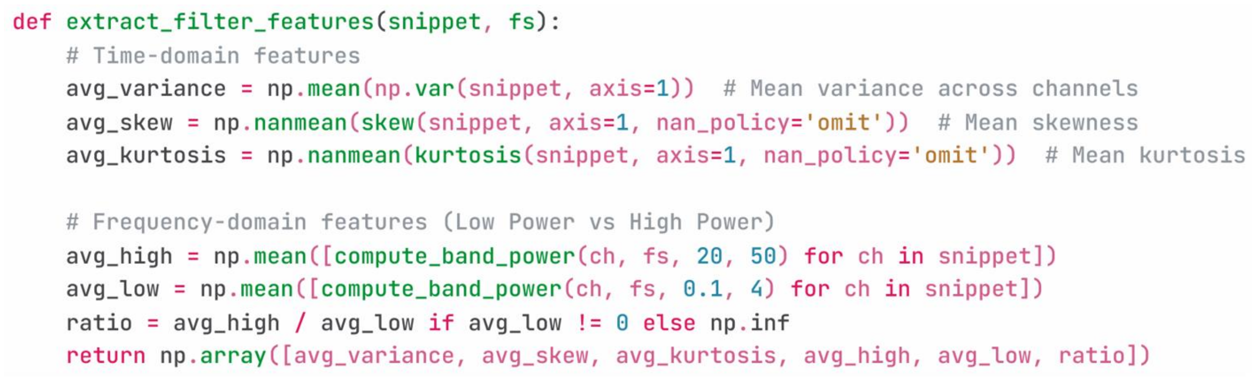
Requiring at least 5 samples to split an internal node further reduced sensitivity to noise. This constraint ensured that splits were only created when there was enough data to support meaningful decisions, preventing the trees from overreacting to small fluctuations in the signal.

Random Forest was selected for its strong performance on small, noisy, multiclass datasets and its ability to model nonlinear relationships. It also performs internal feature selection, helping the model focus on the most informative features. Both models, along with their label encoders, were saved using joblib for efficient real-time reuse.

3.2.3. Filter Selector Model

The filter selector model assigns one of three bandpass filters to each EEG signal. It takes raw EEG signals as input and outputs a filtered signal, which is then passed to the action classifier. This section outlines the features and training setup used to develop the model.

As shown in Figure 16, each EEG snippet is passed through a custom feature extraction function. This function creates a compact version of the signal using time and frequency-based statistics. The time-based features describe how the signal changes over time. The frequency-based features show which frequency ranges are most active.



```python
def extract_filter_features(snippet, fs):
    # Time-domain features
    avg_variance = np.mean(np.var(snippet, axis=1))  # Mean variance across channels
    avg_skew = np.nanmean(skew(snippet, axis=1, nan_policy='omit'))  # Mean skewness
    avg_kurtosis = np.nanmean(kurtosis(snippet, axis=1, nan_policy='omit'))  # Mean kurtosis

    # Frequency-domain features (Low Power vs High Power)
    avg_high = np.mean([compute_band_power(ch, fs, 20, 50) for ch in snippet])
    avg_low = np.mean([compute_band_power(ch, fs, 0.1, 4) for ch in snippet])
    ratio = avg_high / avg_low if avg_low != 0 else np.inf
    return np.array([avg_variance, avg_skew, avg_kurtosis, avg_high, avg_low, ratio])
```

Figure 16: The features extracted for the filter selector model.

The extract_filter_features() function calculates six features for each EEG sample. The time-domain features include variance, skewness, and kurtosis. These describe the shape and intensity of the signal. Variance helped identify eyebrow raises, which often cause large shifts in signal level. Skewness detected uneven rise and fall patterns, also common in eyebrow raises. Kurtosis highlighted sharp peaks, making it useful for detecting quick, high-amplitude spikes from bite gestures.

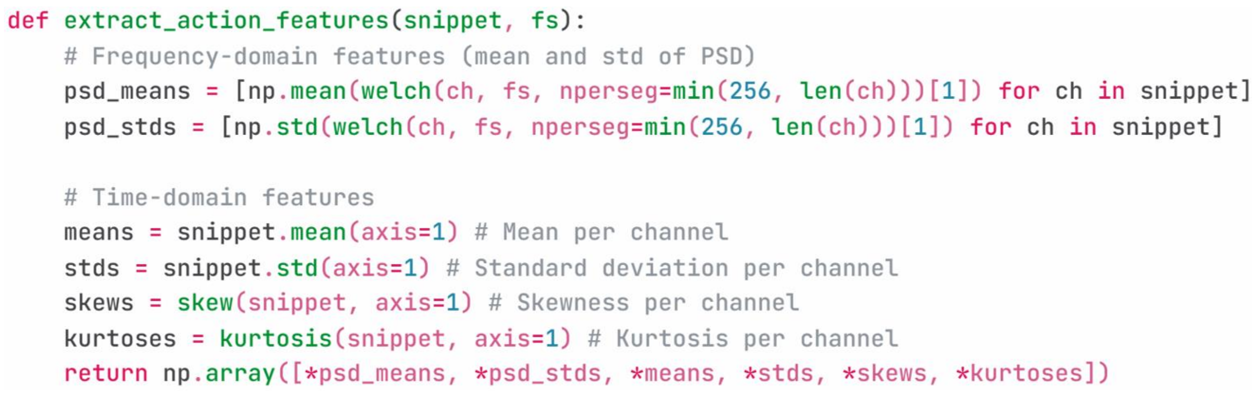
The frequency-domain features include low-band power (0.1–4 Hz), high-band power (20–50 Hz), and their ratio. Low-band power helped detect blinks, which are dominated by slow wave activity. High-band power captured fast, repetitive signals from gestures like jaw clenches and bites. The ratio of low to high power helped distinguish gestures with similar signal strength but different frequency patterns, such as blinks and jaw clenches.

The model was trained using a 60/40 train-test split, with stratified sampling to maintain class balance. After training, the filter selector could reliably choose the best filter for each raw EEG sample, improving the performance of the action classifier that followed.

3.2.4. Action Classification Model

The action classifier predicts which gesture is being performed based on the filtered EEG signal. It receives input from the filter selector and classifies it as one of four gestures. This section outlines the features and training setup used to build the model.

As shown in Figure 17, the extract_action_features() function applies the same feature extraction approach described in the filter selector section to generate a compact feature vector.



```python
def extract_action_features(snippet, fs):
    # Frequency-domain features (mean and std of PSD)
    psd_means = [np.mean(welch(ch, fs, nperseg=min(256, len(ch)))[1]) for ch in snippet]
    psd_stds = [np.std(welch(ch, fs, nperseg=min(256, len(ch)))[1]) for ch in snippet]

    # Time-domain features
    means = snippet.mean(axis=1) # Mean per channel
    stds = snippet.std(axis=1) # Standard deviation per channel
    skews = skew(snippet, axis=1) # Skewness per channel
    kurtoses = kurtosis(snippet, axis=1) # Kurtosis per channel
    return np.array([*psd_means, *psd_stds, *means, *stds, *skews, *kurtoses])
```

Figure 17: The features extracted for the action classifier model.

The extract_action_features() function calculates six features per EEG channel. The time-domain features include mean, standard deviation, skewness, and kurtosis. Mean measures the average signal level and helps detect baseline shifts from eyebrow raises. Standard deviation reflects signal variability and was useful for gestures with large fluctuations, like jaw clenches. Skewness captures signal asymmetry, common in eyebrow raises. Kurtosis highlights sharp peaks, helping detect the spike-like activity of bite gestures.

The frequency-domain features include the mean and standard deviation of each channel's power spectral density (PSD). The PSD mean reflects the average signal power across frequencies, while the PSD standard deviation captures how much that power varies. These features helped the model distinguish between gestures with different frequency characteristics. Jaw clenches and bites typically showed higher and more variable frequency content, while blinks were more stable and concentrated in lower frequencies.

The model was trained using a 70/30 stratified train-test split to maintain class balance. After training, the action classifier reliably identified the user's intended gesture based on the filtered EEG signal.

3.3. User Interface

3.3.1. Main Dashboard

The main dashboard, built with PyQt5, serves as the primary interface for real-time interaction. It integrates EEG streaming, data recording, visualization, and classification into a single window. A logging panel provides live feedback, while buttons and dropdowns allow users to control key system actions.

Device discovery and connection are handled through system calls to muselsl using QProcess. When the "List Muse Devices" button is pressed, the dashboard runs the muselsl list command, parses the output, and populates a dropdown with device names or IPs. The "Start Stream" button launches the muselsl stream command to initiate the EEG stream. Once a signal is detected, the dashboard enables the rest of the interface.

The classification process runs in the background using a separate QProcess that executes the real_time.py script. This script monitors a buffer directory for new EEG snippets. When a new sample is found, it is passed through the filter selector and action classifier. The predicted action is printed to the main log and displayed in the format "Predicted Action: <action>."

The dashboard also initiates communication with the RC car by passing the predicted action to a command-sending function. It handles mapping the prediction to a control signal and triggering the transmission over the network to the Arduino.

3.3.2. Brainwave Visualization

The dashboard includes a brainwave visualization tool that displays real-time EEG activity. When the "Launch Brainwave Monitor" button is clicked, a separate CombinedViewWindow opens with two main views: a scrolling plot of raw EEG signals and a live display of brainwave frequency bands.

The top half shows real-time Muse EEG data using the embedded_view() function and Matplotlib. The bottom half uses LiveBandsWidget and PyQtGraph to plot power levels across Delta, Theta, Alpha, Beta, and Gamma bands. Users can toggle specific channels and bands, with plots updating every 250 milliseconds. Additional controls allow the layout to be reset and views to be hidden or shown as needed.

3.3.3. Real-Time Interface

The real-time interface continuously processes EEG data and generates predictions as new input becomes available. It operates independently from the main dashboard and is implemented in the real_time.py script. This script runs as a background process and monitors a shared buffer directory for new EEG snippet files.

Each time a new snippet is detected, the script uses the filter selector model to choose the appropriate filter. This filter is applied using the same bandpass_filter function used during training, and the resulting signal is passed to the action classifier to generate a prediction.

All processing occurs automatically, with no additional user input required. The pipeline is lightweight and optimized to run efficiently on standard hardware, without requiring a GPU. Keeping the real-time logic separate from the main GUI keeps the system responsive even during continuous operation.

3.4. Hardware

3.4.1. Hardware Components

The hardware system consists of an Arduino Uno WiFi, a motor shield, and two DC motors. The motor shield is mounted directly on top of the Arduino, providing a compact and modular control stack. Power is supplied separately to both the Arduino and the motor shield through barrel jacks, and the motor shield delivers power directly to the DC motors. Soldering was used to create stable connections and improve the durability of the wiring and overall assembly. A diagram of the wiring can be seen in Figure 18. The exact motor shield model used in the final build was not available in the CAD software, so a substitute with similar functionality was used in the diagram to illustrate the wiring layout.

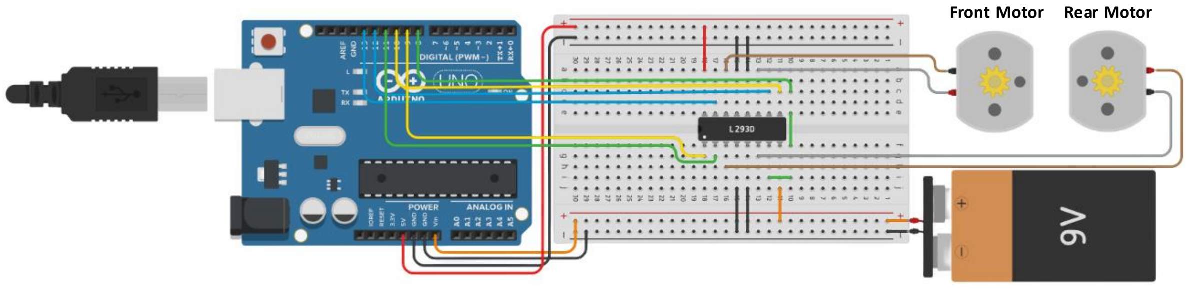


Figure 18: Wiring between Arduino, motor shield, motors, and power source.

3.4.2. Arduino Communication

Network communication was established using an Arduino library called WiFiNina to connect to WiFi. The team developed a function on the PC that retrieves the Arduino's IP address by querying the Address Resolution Protocol (ARP) table. The ARP table maps IP addresses to their corresponding physical MAC addresses using the Arduino's known MAC address. Once the Arduino's IP address was obtained, a socket was created using the Python socket library. The Arduino WiFiNina library was utilized to implement a Telnet server on the Arduino, with the PC functioning as the Telnet client to send messages. Robust error-handling routines were integrated to ensure that if a connection timed out, the system would automatically retry the connection and wait for the Arduino's acknowledgment, preventing the application from crashing due to communication errors.

3.4.3. RC Car Logic

The RC car's logic was implemented by processing incoming command labels received as strings. These labels were mapped to specific numerical values using conditionals, and the resulting numbers were passed through a switch statement to execute corresponding motor actions (forward, backward, left, right, or no movement). Control of the motors was achieved by utilizing the Arduino's digital pins; six designated pins enabled or disabled channels on the motor shield, each dedicated to the drive or steering motor. Additional pins controlled the direction of current flow to the DC motors. The team established a standard time step of 500 milliseconds for each movement. However, extended durations were allowed for steering commands based on testing outcomes to enhance the vehicle's responsiveness to turning instructions.

3.5. Bill of Materials and Budget

Table 6 lists the hardware components, tools, and costs. The system was developed on a budget of $600 using non-invasive, consumer-grade hardware. Items marked with a dash (-) were contributed from personal supplies. The total cost was approximately $245 CAD.

Table 6: Bill of materials for the system and associated costs.

Item	Description	Price (CAD)
Muse 2 Headset	Consumer-grade EEG headset with 4 electrodes	$150
Batteries	8 D batteries and 36 AA batteries	$50
RC Car Chassis	Contains DC motors, wheels, and steering mechanism	$45
Arduino Uno WiFi Rev2	Microcontroller board with integrated WiFi	-
Arduino Motor Shield	Motor shield compatible with Arduino Uno	-
Power Adapter	External power supply for Arduino (5V 2A)	-
Soldering Materials	Solder, flux, iron	-
Miscellaneous Tools	Multimeter, screwdrivers, wires	-

4. Testing and Evaluation

This section outlines testing of the filter selector, action classifier, and hardware. Each component was evaluated using classification metrics or real-time tests to verify system performance and reliability.

4.1. Classifier Performance

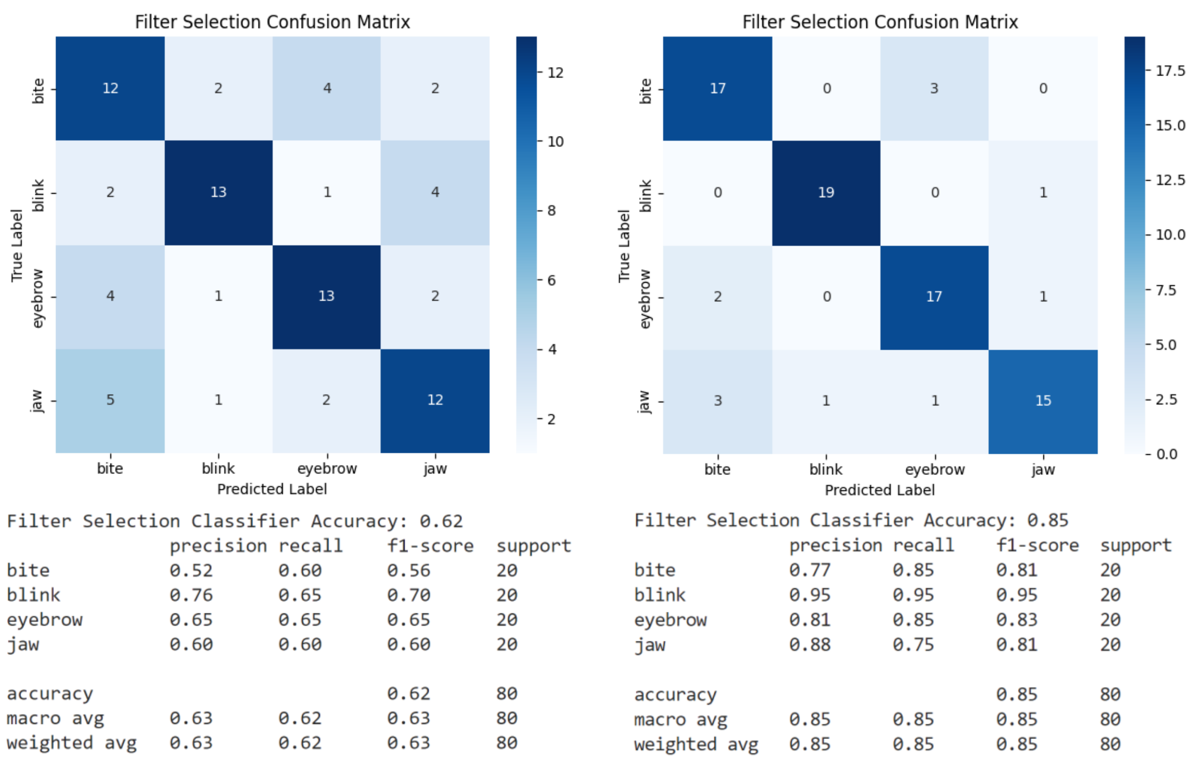
4.1.1. Evaluation Metrics

To evaluate the performance of our models, we used several key metrics that helped us understand how well the predictions matched the actual data. Accuracy tells us how many predictions the model got right out of all the predictions it made. It's a quick way to assess overall performance, but it doesn’t indicate whether certain classes are being overlooked. Precision examines how many predictions for a specific class were correct. For example, if the model predicted “biting” ten times but only six were correct, the precision would be 60%. Recall concentrates on how many of the actual examples of a class the model was able to find. If there were ten real “biting” samples and the model only identified six of them, the recall would also be 60%. The F1-score represents the balance between precision and recall. It provides a single number indicating whether the model is accurate when making predictions and effective at finding all the right examples. We also utilized confusion matrices, which are simple tables displaying exactly which classes were correctly identified and which were misclassified. Together, these metrics helped us understand not just how often the model was correct but how and where it was making mistakes.

4.1.2. Filter Selector Model

The filter selector was tested over three iterations using a 60/40 stratified train-test split. Metrics from Section 4.1.1 were used to track improvements in accuracy and misclassification patterns. This section outlines how the model evolved through adjustments to feature selection and parameters.

The first filter selector model achieved 62% accuracy, with frequent misclassifications between bite and jaw filters. As shown in Figure 19, gestures with similar high-frequency content were often confused, especially jaw clench being predicted as bite. This was likely due to an underdeveloped feature set that relied mostly on variance, skewness, and kurtosis, which failed to capture the key spectral differences between gestures. The model also lacked a power ratio feature, making it harder to distinguish between low-frequency blinks and high-frequency clenches, ultimately limiting generalization.

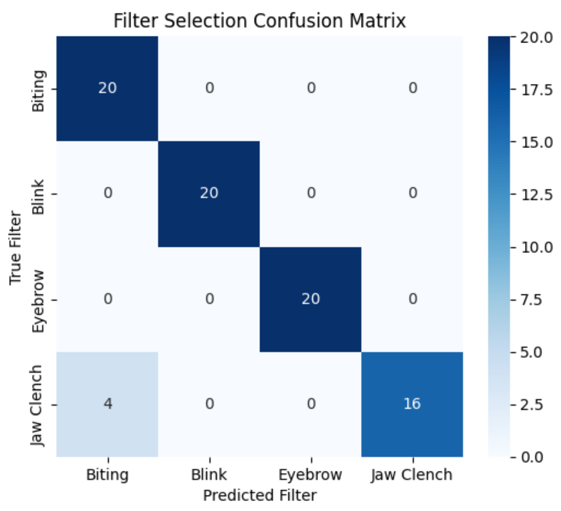


```
Filter Selection Classifier Accuracy: 0.62
               precision recall   f1-score  support
bite           0.52      0.60     0.56      20
blink          0.76      0.65     0.70      20
eyebrow        0.65      0.65     0.65      20
jaw            0.60      0.60     0.60      20

accuracy                          0.62      80
macro avg      0.63      0.62     0.63      80
weighted avg   0.63      0.62     0.63      80
```

```
Filter Selection Classifier Accuracy: 0.85
               precision recall   f1-score  support
bite           0.77      0.85     0.81      20
blink          0.95      0.95     0.95      20
eyebrow        0.81      0.85     0.83      20
jaw            0.88      0.75     0.81      20

accuracy                          0.85      80
macro avg      0.85      0.85     0.85      80
weighted avg   0.85      0.85     0.85      80
```

Figure 19: Initial filter selector model (left) and model with improved frequency features (right).

The second filter selector model showed substantial improvement, reaching 85% accuracy overall. This gain was largely due to the introduction of frequency-domain features, including low and high band power and their ratio, which better captured gesture-specific frequency patterns. As shown in Figure 19, classification accuracy improved across all gesture types, particularly for blinks and eyebrows, which saw near-perfect precision and recall. However, the model still showed some confusion between bite and jaw clench, likely due to overlap in their high-frequency content. While the updated feature set improved generalization, the model's fixed hyperparameters may have limited further gains, especially for gestures with subtle distinctions.



```
Filter Selection Classifier Accuracy: 0.95
              precision    recall  f1-score   support

      Biting       0.83      1.00      0.91        20
       Blink       1.00      1.00      1.00        20
     Eyebrow       1.00      1.00      1.00        20
  Jaw Clench       1.00      0.80      0.89        20

    accuracy                           0.95        80
   macro avg       0.96      0.95      0.95        80
weighted avg       0.96      0.95      0.95        80
```

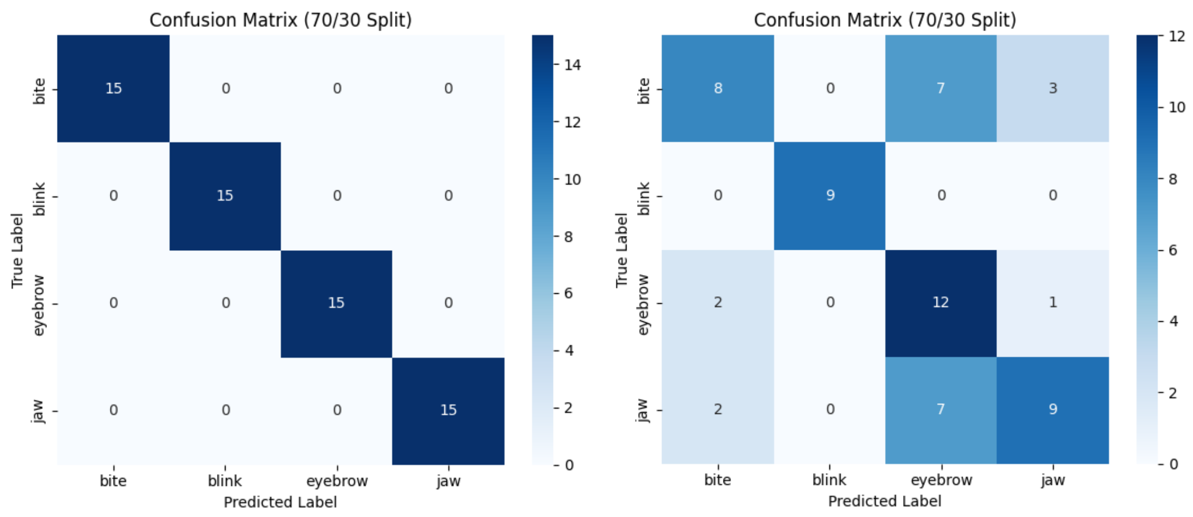
Figure 20: Final filter selector performance showing 95% accuracy after parameter tuning.

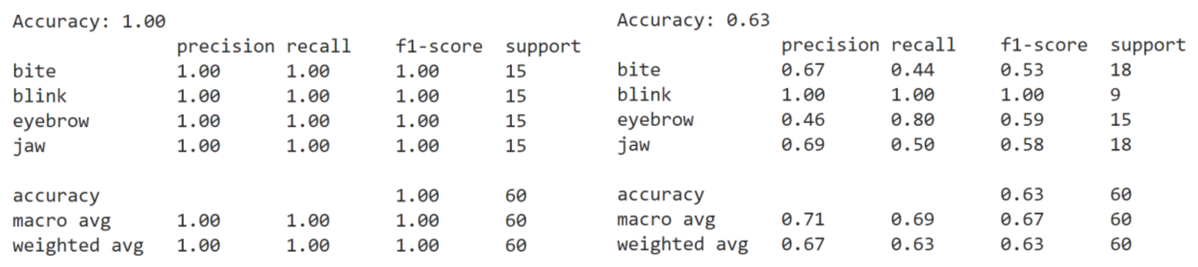
The final version of the filter selector model achieved 95% accuracy, with strong precision and recall across all gesture types, as shown in Figure 20. This improvement came from carefully tuning the Random Forest hyperparameters, including reducing the maximum tree depth to 10, increasing the number of trees to 50, and setting a higher minimum sample split threshold. These adjustments helped the model generalize better by reducing overfitting and increasing decision stability. Most gestures were classified correctly, with perfect recall for bite, blink, and eyebrow filters. The only remaining issue was a slight confusion between jaw clenches and bites, likely due to their overlapping frequency profiles.

4.1.3. Action Classifier Model

The action classifier was tested over four iterations using a 70/30 stratified train-test split. Using the metrics described in Section 4.1.1, we tracked how the model performed with each change and where it struggled. Testing focused on improving overall accuracy, reducing class-specific errors, and identifying misclassification patterns. This section describes how the model evolved through adjustments to data balance, feature selection, and model parameters, with confusion matrices and classification reports guiding each stage of improvement.

Initially, the first model appeared to perform perfectly, achieving 100% accuracy with a confusion matrix showing that every prediction was correct. However, real-world testing revealed that the model severely overfitted the blink class, predicting it for nearly all inputs. This illusion of perfect accuracy stemmed from a lack of meaningful class separation in the feature space. Additionally, the model was biased toward the distinctive characteristics of blink signals. Once blink was undersampled to reduce its influence, the model's accuracy dropped to 63%, reflecting its actual performance better. Although blink was now correctly classified, over 50% of bite and jaw gestures were misclassified as eyebrow, indicating that eyebrow features were overly dominant in the training distribution. This shift in performance is clearly illustrated in the comparison between the two models shown in Figure 21.



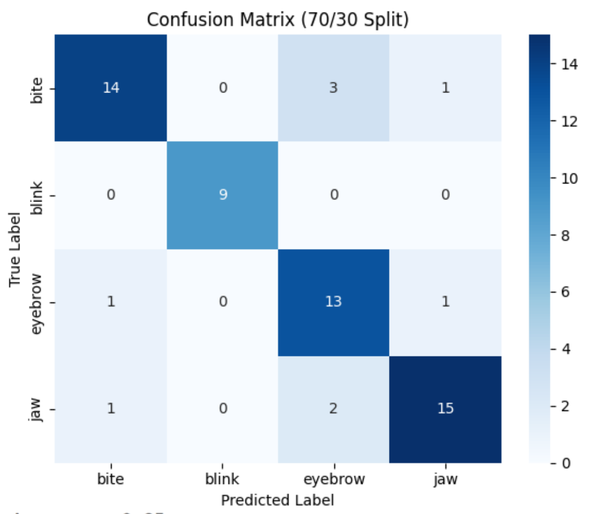


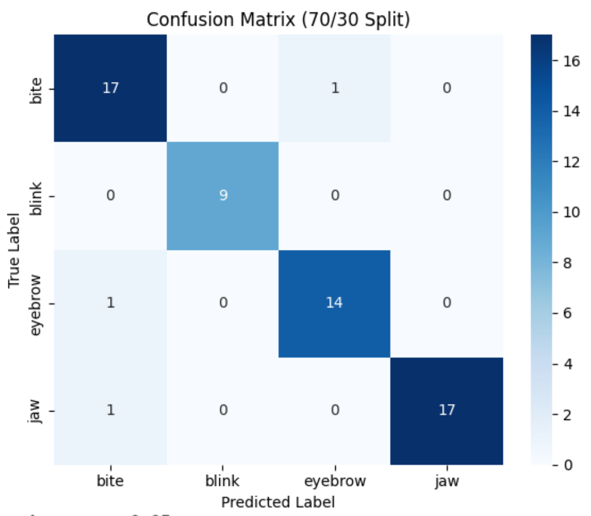
```
Accuracy: 1.00
              precision recall    f1-score   support
bite          1.00      1.00      1.00       15
blink         1.00      1.00      1.00       15
eyebrow       1.00      1.00      1.00       15
jaw           1.00      1.00      1.00       15

accuracy                          1.00       60
macro avg     1.00      1.00      1.00       60
weighted avg  1.00      1.00      1.00       60
```

```
Accuracy: 0.63
              precision recall    f1-score   support
bite          0.67      0.44      0.53       18
blink         1.00      1.00      1.00       9
eyebrow       0.46      0.80      0.59       15
jaw           0.69      0.50      0.58       18

accuracy                          0.63       60
macro avg     0.71      0.69      0.67       60
weighted avg  0.67      0.63      0.63       60
```

Figure 21: Initial classifier (left) and classifier with undersampled blink data (right).

In the third iteration, the eyebrow class was moderately undersampled to improve balance while still preserving its representation. This adjustment significantly improved, with accuracy rising to 85%. Misclassifications across all classes were reduced, and F1-scores became more consistent, particularly for bite and jaw. This demonstrated that both blink and eyebrow had been introducing bias into the model and that controlled undersampling could restore more equitable class behavior.





```
Accuracy: 0.85
              precision recall    f1-score   support
bite          0.88      0.78      0.82       18
blink         1.00      1.00      1.00       9
eyebrow       0.72      0.87      0.79       15
jaw           0.88      0.83      0.86       18

accuracy                          0.85       60
macro avg     0.87      0.87      0.87       60
weighted avg  0.86      0.85      0.85       60
```

```
Accuracy: 0.95
              precision recall    f1-score   support
bite          0.89      0.94      0.92       18
blink         1.00      1.00      1.00       9
eyebrow       0.93      0.93      0.93       15
jaw           1.00      0.94      0.97       18

accuracy                          0.95       60
macro avg     0.96      0.96      0.96       60
weighted avg  0.95      0.95      0.95       60
```

Figure 22: Performance of the third model (left) and final model (right).

In the fourth iteration, we focused on refining model performance by adjusting key Random Forest parameters and eliminating underperforming features. Earlier versions of the classifier used a higher number of estimators and default settings for tree depth and split conditions. This resulted in slower inference and signs of overfitting, particularly on training data with subtle gesture overlap. To address this, we reduced the number of estimators to 50, limited the maximum depth of each tree to 10, and increased the min_samples_split to 5. These changes were made to simplify the model structure, reduce overfitting, and improve execution speed for real-time inference.

We also reevaluated the feature set. The initial version included around ten features, such as signal variance, peak-to-peak range, zero crossing rate, and interquartile range. After analyzing feature importance using Gini-based rankings from the classifier, we found that many of these contributed minimally to classification accuracy. Features with high redundancy or poor separability were removed, and the set was reduced to six core features that consistently had the highest impact on performance. This combination of parameter tuning and feature reduction significantly improved model consistency, resulting in 95% test accuracy and strong per-class F1-scores.

4.2. RC Car Testing

4.2.1. WiFi Reliability Testing

To assess the system's network stability, the team utilized the Arduino's built-in capability to measure Received Signal Strength Indicator (RSSI) values, which consistently ranged from -38 to -48 dBm. This range aligns with strong to moderate signal quality, indicating that the Arduino maintained a relatively stable connection when linked to the local area network. However, the team found Bluetooth integration infeasible due to compatibility constraints. In contrast, when the Arduino attempted to broadcast its own network, testing revealed latencies of one to three seconds and frequent disconnections under higher request volumes. These issues stemmed from the onboard WiFi chip's limited ability to host a network while simultaneously processing incoming labels, exacerbated by the system's overall power constraints.

4.2.2. Motor Dynamics and Responsiveness

Initial trials highlighted an oversight regarding motor inertia: after each 500 ms movement interval, the drive motors retained sufficient momentum to continue moving forward, which reduced turning effectiveness. To address this, the team deactivated the drive motors once the interval elapsed, allowing the steering motors to operate for an additional half-step. This modification increased the turning angle from approximately 10 to 15 degrees to 45 degrees when powered by battery alone and up to 90 degrees with a stable power adapter. Introducing brief delays in steering also diminished the frequency of user inputs required, thereby lowering the cognitive load associated with maneuvering the car.

4.2.3. Thermal Performance and Hardware Integrity

During continuous operation with a 9V power supply for the motor shield and a 6V supply for the Arduino, high voltage and prolonged runtime resulted in significant heat accumulation. The plastic moulding around the drive axles started to melt, creating internal resistance that hindered the wheels from rotating freely. Upon diagnosing the issue, the team trimmed the plastic around the axles to increase airflow and lowered the voltage supply to prevent overheating. These measures were further supported by replacing the 9V battery with a 5V, 2-amp power adapter. Subsequent extended tests confirmed that these thermal management strategies effectively reduced heat buildup and maintained axle integrity.

4.2.4. Motor Locking and Wiring Integrity

The motors would sometimes lock even though the system’s serial monitor indicated that the correct commands had been received and executed. Observing the motor shield’s LEDs showed that the enable and direction pins remained active beyond the designated movement interval, requiring a manual reset of each channel to restore normal operation. Furthermore, the team discovered that the original stranded copper wires were prone to breakage, particularly when the car collided with obstacles or underwent frequent handling. As a result, all motor connections were upgraded to single-strand 22-gauge wiring, which provided greater mechanical durability and reduced the risk of short circuits between motor shield channels.

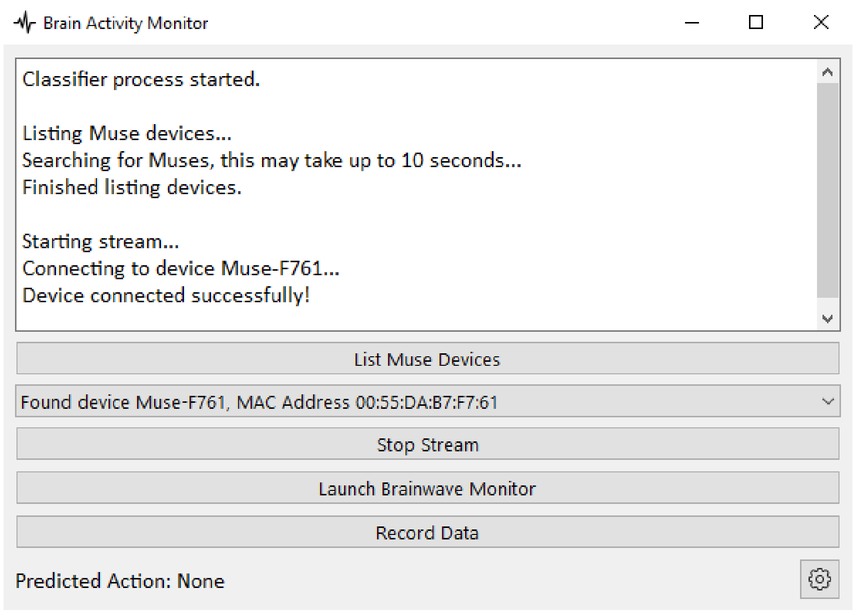
5. Results

The final system was evaluated against the software and hardware specifications outlined in the design blueprint [5]. As summarized in Table 1 and Table 2, the system successfully met 20 out of 22 total specifications. These results reflect strong alignment between the project's design goals and its implementation. The following sections present evidence showing how each requirement was satisfied.

5.1. Software Results

5.1.1. Data Collection

The system successfully acquires EEG signals from the Muse 2 headset in real time and displays them with low latency. Figure 23 shows the main dashboard interface providing GUI controls to start and stop EEG streaming as required by Specification 2.1. It also provides functionality for detecting a connected Muse 2 device via Bluetooth in accordance with Specification 2.6.



```
Classifier process started.

Listing Muse devices...
Searching for Muses, this may take up to 10 seconds...
Finished listing devices.

Starting stream...
Connecting to device Muse-F761...
Device connected successfully!
```

Found device Muse-F761, MAC Address 00:55:DA:B7:F7:61

Predicted Action: None

Figure 23: Muse 2 headset detected and connected via Bluetooth in the main dashboard.

Once the stream is initiated, the brainwave monitor displays live EEG signals from all four Muse electrodes (see Figure 24). This verifies the requirement for real-time acquisition, seen in Specification 1.1. Additionally, this satisfies the requirement for multi-channel visualization, as outlined in Specification 2.2.

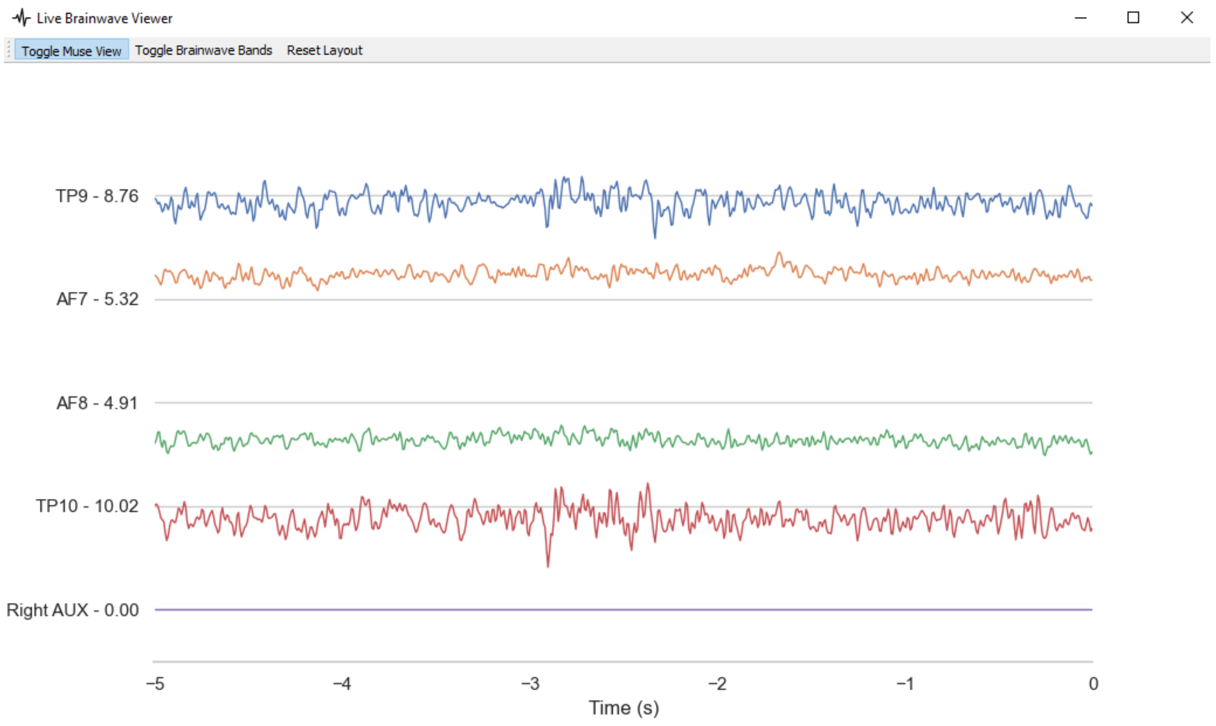
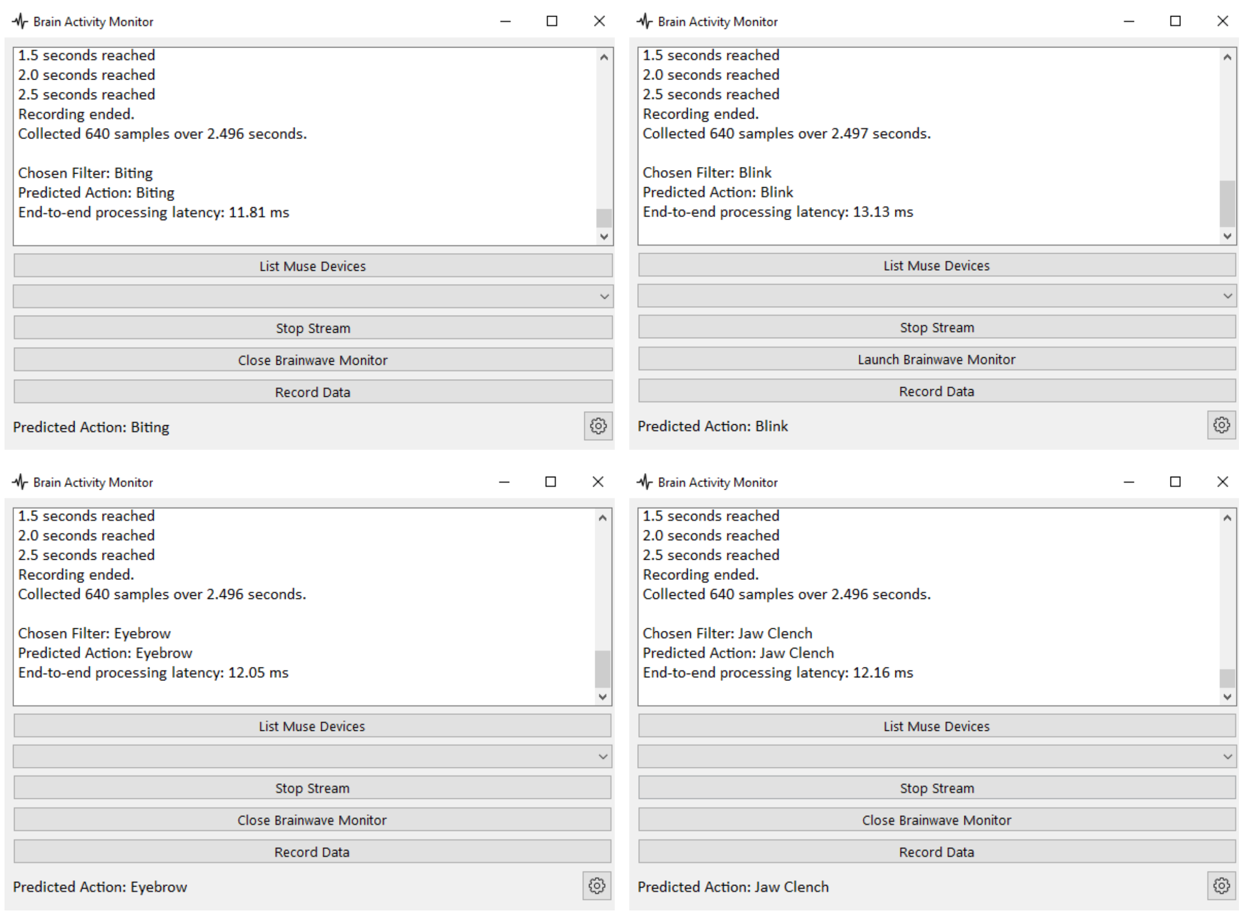


Figure 24: Real-time EEG visualization from all four Muse 2 channels with live signal updates.

The scrolling plots in Figure 24 update continuously, providing clear visual feedback that reflects current EEG activity with no visible delay. This confirms that the live display updates with less than 300 milliseconds of latency, as required for responsive user feedback in Specification 3.5.

5.1.2. Machine Learning

Figure 25 demonstrates a complete prediction cycle for each of the four gestures supported by the system. In each case, the correct filter was selected based on the raw EEG input, which satisfies Specification 1.2. The corresponding gesture was accurately classified by the model, fulfilling Specification 1.3. The predicted action is clearly displayed in the user interface, meeting the requirement for visual feedback described in Specification 2.4.



```
1.5 seconds reached
2.0 seconds reached
2.5 seconds reached
Recording ended.
Collected 640 samples over 2.496 seconds.

Chosen Filter: Biting
Predicted Action: Biting
End-to-end processing latency: 11.81 ms
```

Predicted Action: Biting

```
1.5 seconds reached
2.0 seconds reached
2.5 seconds reached
Recording ended.
Collected 640 samples over 2.497 seconds.

Chosen Filter: Blink
Predicted Action: Blink
End-to-end processing latency: 13.13 ms
```

Predicted Action: Blink

```
1.5 seconds reached
2.0 seconds reached
2.5 seconds reached
Recording ended.
Collected 640 samples over 2.496 seconds.

Chosen Filter: Eyebrow
Predicted Action: Eyebrow
End-to-end processing latency: 12.05 ms
```

Predicted Action: Eyebrow

```
1.5 seconds reached
2.0 seconds reached
2.5 seconds reached
Recording ended.
Collected 640 samples over 2.496 seconds.

Chosen Filter: Jaw Clench
Predicted Action: Jaw Clench
End-to-end processing latency: 12.16 ms
```

Predicted Action: Jaw Clench

Figure 25: End-to-end processing latency for all four gestures, each under 14 ms.

The entire process, from the start of EEG recording to the appearance of the predicted label, was completed in less than 2.5 seconds for all gestures, satisfying Specification 3.2. Additionally, the printed latency logs, calculated using Python's time module, show that signal processing and model inference consistently completed in under 14 milliseconds. This confirms that the system meets the real-time performance constraint defined in Specification 3.3.

Figure 22 shows the confusion matrix and classification report for the final model. It achieved an overall accuracy of 95 percent, with strong performance across all gesture classes. This confirms that the system meets the requirement for gesture classification accuracy of at least 85 percent, as outlined in Specification 3.1.

5.1.3. User Interface

Figure 26 shows the brainwave monitor displaying live plots for five frequency bands: Delta, Theta, Alpha, Beta, and Gamma across all four EEG channels. This satisfies the requirement for multi-band brainwave visualization as described in Specification 2.3.

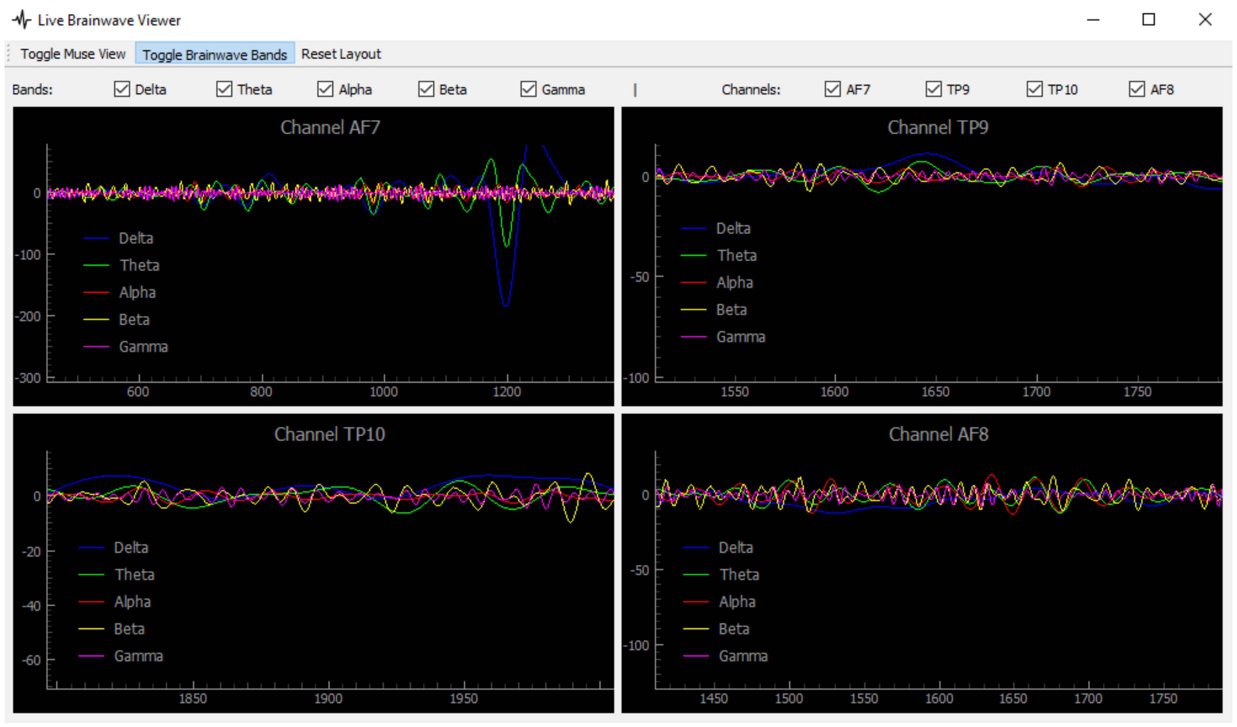
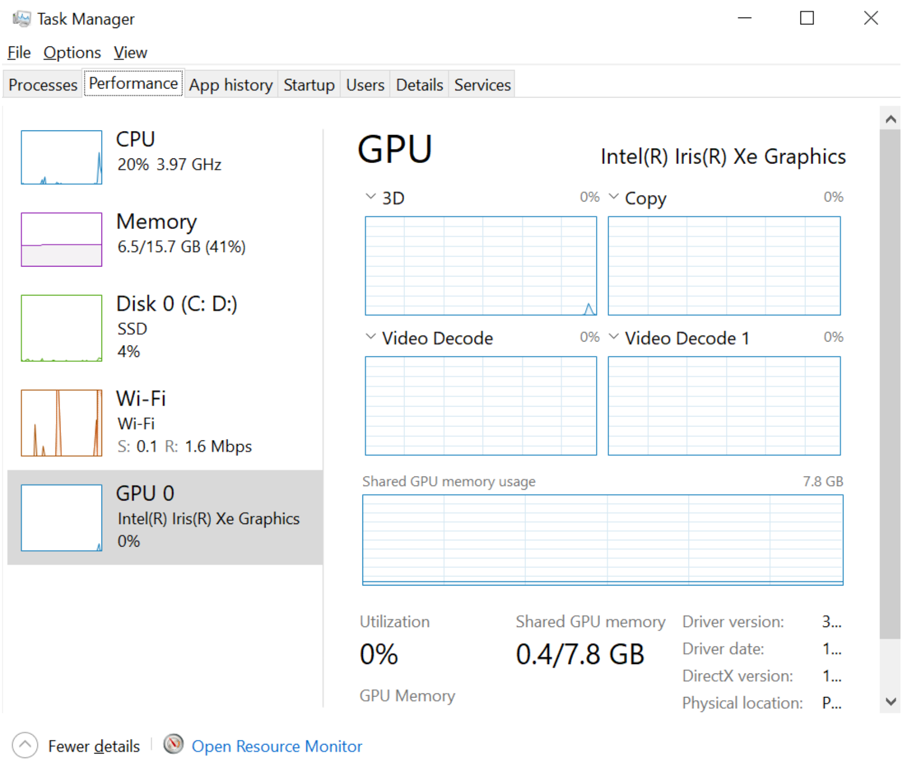


Figure 26: Live brainwave monitor displaying Delta, Theta, Alpha, Beta, and Gamma bands.

Figure 27 confirms that the system runs in real time on a standard laptop without using any dedicated graphics hardware. The task manager output shows zero percent GPU utilization during operation, meeting the performance requirement defined in Specification 3.4.



CPU
20% 3.97 GHz

Memory
6.5/15.7 GB (41%)

Disk 0 (C: D:)
SSD
4%

Wi-Fi
Wi-Fi
S: 0.1 R: 1.6 Mbps

GPU 0
Intel(R) Iris(R) Xe Graphics
0%

GPU
Intel(R) Iris(R) Xe Graphics

Utilization	Shared GPU memory	Driver version:	3...
0%	0.4/7.8 GB	Driver date:	1...
		DirectX version:	1...
GPU Memory		Physical location:	P...

Figure 27: Task Manager showing zero percent GPU usage during real-time system operation.

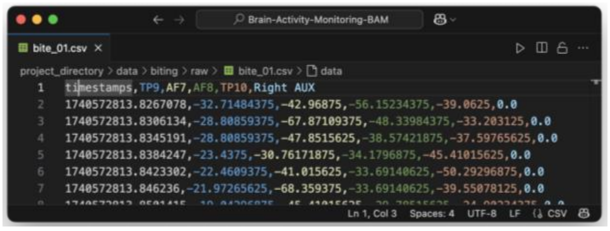
5.2. Hardware Results

5.2.1. Performance Metrics

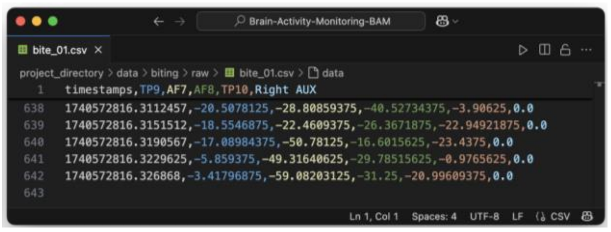
The Muse 2 headband consistently sampled data at a frequency of 256 Hz, generating approximately 640 samples for each 2.5-second recording interval, as indicated in the raw CSV data. The Arduino's command processing times were notably swift, with recorded acknowledgments showing a mean latency of 49.5 ms, well below the specified requirement of 200 ms. Additionally, wireless communication between the PC and Arduino maintained a loss rate of under 10%; for every 200 commands transmitted, only 1 to 3 needed retransmissions, ensuring robust connectivity overall.

5.2.2. Performance Corroboration

A series of images supports these performance metrics. Screenshots of the CSV data confirm that over 640 samples were consistently recorded during each 2.5-second interval, as illustrated in Figure 28. The first row serves as a header, while the data body contains timestamps and recorded measurements from the headsets at a specific instant.



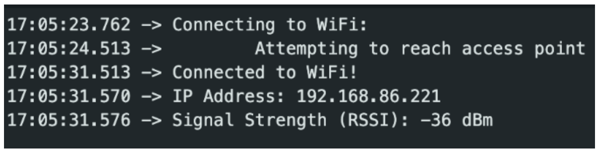
```
bite_01.csv
project_directory > data > biting > raw > bite_01.csv > data
1    timestamps,TP9,AF7,AF8,TP10,Right AUX
2    1740572813.8267078,-32.71484375,-42.96875,-56.15234375,-39.0625,0.0
3    1740572813.8306134,-28.80859375,-67.87109375,-48.33984375,-33.203125,0.0
4    1740572813.8345191,-28.80859375,-47.8515625,-38.57421875,-37.59765625,0.0
5    1740572813.8384247,-23.4375,-30.76171875,-34.1796875,-45.41015625,0.0
6    1740572813.8423302,-22.4609375,-41.015625,-33.69140625,-50.29296875,0.0
7    1740572813.846236,-21.97265625,-68.359375,-33.69140625,-39.55078125,0.0
```



```
bite_01.csv
project_directory > data > biting > raw > bite_01.csv > data
1    timestamps,TP9,AF7,AF8,TP10,Right AUX
638  1740572816.3112457,-20.5078125,-28.80859375,-40.52734375,-3.90625,0.0
639  1740572816.3151512,-18.5546875,-22.4609375,-26.3671875,-22.94921875,0.0
640  1740572816.3190567,-17.08984375,-50.78125,-16.6015625,-23.4375,0.0
641  1740572816.3229625,-5.859375,-49.31640625,-29.78515625,-0.9765625,0.0
642  1740572816.326868,-3.41796875,-59.08203125,-31.25,-20.99609375,0.0
643
```

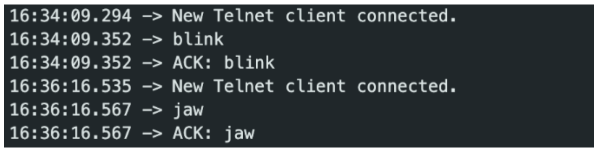
Figure 28: Blink data sample confirming 256Hz sampling frequency.

Serial monitor outputs show that the Arduino received commands and sent acknowledgments within the target time, while also illustrating stable RSSI readings, which verify effective WiFi communication. As seen in Figure 30, the ACK-latency calculated as time difference between the "New Telenet" and "ACK:__" statements.



```
17:05:23.762 -> Connecting to WiFi:
17:05:24.513 ->          Attempting to reach access point
17:05:31.513 -> Connected to WiFi!
17:05:31.570 -> IP Address: 192.168.86.221
17:05:31.576 -> Signal Strength (RSSI): -36 dBm
```

Figure 29: Connection between Arduino and WiFi network.



```
16:34:09.294 -> New Telnet client connected.
16:34:09.352 -> blink
16:34:09.352 -> ACK: blink
16:36:16.535 -> New Telnet client connected.
16:36:16.567 -> jaw
16:36:16.567 -> ACK: jaw
```

Figure 30: Commands received by Arduino with timestamps.

Photographs of the wiring setup validate that the motor shield is successfully controlling two DC motors, and sequential images capture the RC car performing its four primary maneuvers: forward, reverse, left turn, and right turn, as seen between Figure 31 and Figure 35. Note in Figure 31, the yellow boxes show the DC motor connections, and the orange box shows where the motor shield distributes powers from.

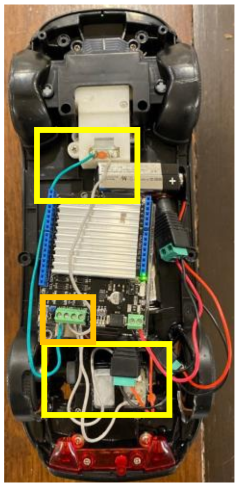


Figure 31: Top-down view of car electronics.

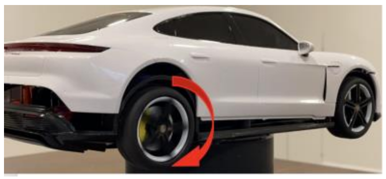


Figure 32: Wheels moving forward.

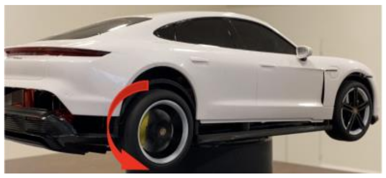


Figure 33: Wheels moving backward.

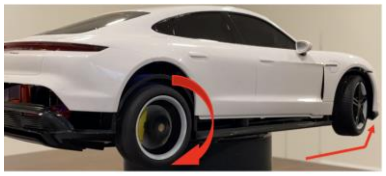


Figure 34: Wheels steering left.

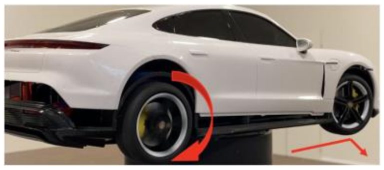


Figure 35: Wheels steering right.

5.2.3. Data Analysis and Insights

The experimental results affirm that the hardware system meets its intended functional requirements. The significantly low mean command processing time indicates a highly responsive control mechanism. The consistent data sampling from the Muse 2 headband ensures precise monitoring of brain signals. Although occasional command losses were observed, these were effectively managed through retransmission protocols without causing any application crashes. While no comprehensive performance graphs were generated, the captured data rows, each containing a timestamp and analog readings from the electrode channels, demonstrate a stable and continuous data stream. Collectively, these findings substantiate that the design delivers robust performance in real-time control, aligning well with the specified hardware requirements.

6. Important Factors

6.1. Stakeholder Needs

The BAM focused on accessibility for individuals with physical disabilities. Stakeholders included end users, BCI researchers, data ethics practitioners, and engineers. They expected a system that is accurate, responsive, non-invasive, portable, and user-friendly. This feedback shaped the system's design: the non-invasive Muse 2 headset provides high-fidelity EEG signals at a 256 Hz sampling rate for precise input tagging. Rapid command processing, with average acknowledgment times of 49.5 ms, along with ergonomic design, makes the system reliable and accessible. User feedback emphasized the need for dynamic input tagging and multi-action commands, like interpreting multiple rapid blinks as a U-turn, to further enhance the user experience.

6.2. Safety

Safety was central to the project. The system defaults to a "stop" state when it encounters an unrecognized command, preventing unintended actions. All wiring connections were soldered and insulated with tape, while barrel connectors ensured stable, low-resistance power. Using low-voltage power sources and separate supplies for the Arduino and motor shield lowered the risks of overheating and failure. Testing showed that hazards like thermal buildup and motor locking were reduced by replacing the 9V battery with a stable 5V, 2-amp adapter and implementing robust error handling in PC-Arduino communication. These steps minimize hazards, ensuring user safety.

6.3. Privacy

Privacy was central to the project design. The system captures only EEG artifacts and action labels, without collecting personally identifiable information (PII). Data is stored locally in CSV files, enhancing security by keeping sensitive information on the user's device. Model knowledge distillation could further enhance this approach in clinical settings, allowing complex models to operate on resource-constrained devices while maintaining data security. The absence of PII collection simplifies regulatory compliance and manufacturing processes, while the use of widely accessible technology ensures that the system is inclusive and can be adapted for broader use. This commitment to privacy guarantees that user data remains confidential and meets strict privacy standards.

6.4. Codes/Standards

Although the project did not explicitly target specific regulations, adhered to recognized electrical safety and data handling best practices. The system meets industry guidelines for safe and reliable electronic and medical devices by using consumer-grade components and robust error-handling protocols.

6.5. Manufacturability

The proof of concept was developed using readily available, off-the-shelf components to demonstrate feasibility while keeping costs within the allocated budget of $600, as detailed in the bill of materials. The design approach prioritized modularity and scalability, with the understanding that the current circuitry could eventually be replaced by a custom-designed printed circuit board (PCB) for mass production. This strategic choice of components supports both current functionality and future enhancements, making the design economically viable and scalable for further development.

6.6. Ethics and Cost

Ethical considerations were central to the project's development, respecting user autonomy and data privacy with a non-invasive EEG headset while avoiding PII collection. The commitment to accessibility ensured the technology is affordable and usable by diverse users, including those with disabilities. Cost constraints guided the design process, resulting in a system that balances performance and affordability. Using consumer-grade hardware kept costs low and demonstrated that effective BCI applications can be developed without expensive equipment, facilitating broader adoption and further research.

7. Conclusions and Recommendations

This section presents the key insights and future potential of the Brain Activity Monitor project. By combining consumer-grade EEG hardware with machine learning, BAM successfully enabled real-time, hands-free control of an RC car using facial gestures. The system proved to be low-cost, non-invasive, and highly responsive, demonstrating the viability of accessible BCI technology for assistive applications. Its strong performance, even with limited resources, highlights the promise of expanding this approach to broader use cases and future development.

7.1. Technical Lessons Learned

A major lesson from this project was the importance of aligning system design with hardware capabilities. The Muse 2 headset, while accessible and affordable, lacks the electrode coverage needed for accurate detection of complex mental states. Recognizing this limitation early allowed us to pivot toward facial gesture detection, which produced more reliable and repeatable EEG patterns. This shift was crucial in achieving accurate real-time control.

The modular design of the system proved highly effective. Dividing the project into distinct subsystems enabled team members to work independently while maintaining system cohesion. This approach simplified integration and allowed individual components to evolve without disrupting the entire pipeline.

Transitioning from rule-based filtering and labeling to machine learning was another key advancement. Initial attempts using hardcoded logic and threshold-based labeling were too rigid and error-prone. By introducing data-driven models for both filtering and classification, we greatly improved adaptability and accuracy across different recordings and users.

Finally, we learned that real-time performance requires careful trade-offs between model complexity and efficiency. Through iterative tuning and selective feature engineering, we achieved high classification accuracy without sacrificing speed. This balance was essential for delivering a responsive system capable of operating on a standard laptop without GPU acceleration.

7.2. Lasting Impact

The Brain Activity Monitor project has strong potential to make a lasting impact beyond this course. As the need for accessible and assistive technologies continues to grow, systems like BAM can significantly improve the lives of individuals with physical disabilities. By enabling hands-free control through consumer-grade EEG devices and machine learning, this project demonstrates how brain-computer interfaces can become practical tools in everyday settings.

The approach taken in this project emphasizes reliability, affordability, and adaptability. These qualities make the system well-suited for expansion into areas such as wheelchair control, smart home interaction, and communication aids. With continued development, this work can serve as a foundation for future research and real-world applications, supporting broader innovation in healthcare, education, and interactive technologies.

7.3. Further Research and Alternative Approaches

There are many opportunities to expand and improve upon the work completed in this project. A key area for further research lies in improving the quality and quantity of data collected. Using EEG hardware with more electrodes and higher signal resolution would allow the system to capture a more detailed view of brain activity. This would provide higher-quality input for analysis, allowing for more effective feature extraction and leading to improved classification accuracy.

Alongside hardware improvements, expanding the dataset to include recordings from multiple users and varied environments would strengthen the system's ability to generalize. This would help ensure reliable performance across a wider range of real-world conditions. Additionally, future research could investigate the use of more sophisticated classification techniques, such as deep learning. This could lead to the development of adaptive models that adjust to individual users over time, further enhancing performance and personalization.

With these improvements in data quality and model capabilities, the system could move beyond facial gestures toward more intuitive and direct control methods. This includes interpreting actual brainwave patterns related to thought and intention, such as motor imagery, where users imagine specific movements to trigger commands. These approaches could eliminate the need for physical gestures entirely, offering a seamless and natural interface between the brain and external devices.

As the field of brain-computer interfaces continues to grow, supported by a wide range of tools, datasets, and ongoing research, exploring these directions could lead to more powerful, adaptable, and accessible BCI systems capable of serving a wider range of real-world applications.

7.4. Commercialization and Manufacturing Considerations

The Brain Activity Monitor system shows strong potential for commercialization, particularly in the assistive technology space. Its hands-free, non-invasive interface addresses a growing demand for accessible solutions that allow individuals with physical disabilities to interact more easily with their environment. By leveraging affordable consumer-grade components, the system lowers the barrier to entry for users, making it attractive for widespread use in rehabilitation, healthcare, and education settings.

To serve these diverse use cases, the system should be designed as a flexible core BCI unit. This central processing module would handle EEG signal acquisition, filtering, classification, and communication. By remaining independent of any single output device, the unit could be configured to control a range of technologies, from mobility aids to home automation systems. Communication can be handled through standard protocols such as WiFi, Bluetooth, or USB. This would ensure compatibility with existing platforms and make integration straightforward for users and developers.

A key step toward commercial viability is the development of a custom EEG headset. Replacing the current third-party solution with a purpose-built design would lower production costs, remove licensing barriers, and allow for optimization of comfort, signal quality, and aesthetics. An in-house headset could also be tailored to specific use cases or demographics, improving user experience while maintaining control over manufacturing.

The remaining hardware can be kept simple and cost-effective. Core components such as the microcontroller, printed circuit board, and power supply can be sourced at scale and assembled using standard manufacturing techniques. Enclosures could be either injection molded, or 3D printed depending on production volume. This approach would help keep costs low while supporting modularity and scalability, enabling the system to meet the needs of a wide range of users and applications.

7.6. Market Size and Growth

Brain computer interface technology and assistive solutions are two sectors that are experiencing significant global growth. In 2024, the global BCI market was valued at approximately USD 2.62 billion and is projected to reach around USD 12.40 billion by 2034, with a compound annual growth rate of 17.35 percent [6]. This growth is supported by rising interest in non-invasive neural technologies and the increasing use of brain computer interfaces in healthcare, communication, and interactive systems.

The assistive technology market is also experiencing continued growth. According to IMARC Group, the assistive technology market is projected to grow from USD 26.8 billion in 2024 to USD 41.0 billion by

2033, with a compound annual growth rate of 4.33 percent [7]. Contributing factors include an aging global population, increased demand for accessibility tools, and ongoing advancements in user centered design.

With its low cost, modular structure, and non-invasive design, the BAM system is well suited to meet the needs of both markets. Its ability to support applications such as assistive mobility, smart home control, and hands-free communication makes it a promising platform for future development and commercial use.

7.7. Societal and Cultural Impacts

The development of accessible brain computer interface systems like the Brain Activity Monitor has the potential to deliver meaningful societal benefits. By providing hands free control to individuals with physical disabilities, this technology promotes greater independence, improves quality of life, and enhances participation in daily activities. As adoption increases, it can contribute to more inclusive environments in homes, workplaces, and public spaces.

Beyond accessibility, brain computer interfaces also offer new ways to interact with digital systems. They could reshape how people engage with technology in areas such as communication, education, and entertainment. This shift could help redefine human computer interaction and reduce reliance on traditional input methods.

However, there are also potential concerns. As brain computer interfaces become more advanced, issues related to user privacy, data security, and psychological wellbeing may arise. There is also a risk of overdependence on technology or unequal access if affordability and availability are not addressed. To ensure positive cultural and societal outcomes, future development must prioritize ethical design, transparency, and equitable access for all users.

8. Effort Distribution

Table 7: Distribution of effort expended by each group member.

Name	Overall Effort Expended (%)
Nicholas Seegobin	100 %
Samhith Sripada	100 %
Robin Farber	100 %
Rodrigo Del Aguila Velarde	100 %

9. References

[1] J.-J. Lin and Z.-Y. Jiang, "An EEG-Based BCI System to Facial Action Recognition," *Wireless Personal Communications,* vol. 94, no. 4, p. 1579–1593, 23 September 2016.

[2] J. LaRocco, M. D. Le and D.-G. Paeng, "A Systemic Review of Available Low-Cost EEG Headsets Used for Drowsiness Detection," *Frontiers in Neuroinformatics,* vol. 14, p. 553352, 15 October 2020.

[3] J. d. R. Millán, R. Rupp, G. R. Müller-Putz, R. Murray-Smith, C. Giugliemma, M. Tangermann, C. Vidaurre, F. Cincotti, A. Kübler, R. Leeb, C. Neuper, K.-R. Müller and D. Mattia, "Combining Brain–Computer Interfaces and Assistive Technologies: State-of-the-Art and Challenges," *Frontiers in Neuroscience,* vol. 4, p. 161, 07 September 2010.

[4] S. Lee, M. Kim and M. Ahn, "Evaluation of consumer-grade wireless EEG systems for brain–computer interface applications," *Biomed Eng Lett,* vol. 14, no. 6, p. 1433–1443, 13 August 2024.

[5] N. Seegobin, R. Farber, R. D. A. Velarde and S. Sripada, "Brain Activity Monitoring: Capstone Project Blueprint Document," pp. 1-13, 07 November 2024.

[6] P. Research, "Brain Computer Interface Market Size and Forecast 2025 to 2034," 11 March 2025. [Online]. Available: https://www.precedenceresearch.com/brain-computer-interface-market. [Accessed 06 April 2025].

[7] I. Group, "Assistive Technology Market Report by Product Type, End User, and Region 2025–2033," March 2024. [Online]. Available: https://www.imarcgroup.com/assistive-technology-market. [Accessed 06 April 2025].