EEG Controlled Robotics

A Technical Report submitted to the Department of Mechanical and Aerospace Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Joshua Rivas-Zelaya

Spring, 2025

Technical Project Team Members

Hailey Boyd

Cayla Celis

Abigail Dodd

Joshua Rivas-Zelaya

On my honor as University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Sarah Sun, Department of Mechanical and Aerospace Engineering

Problem Statement

There were over 5.4 million people living with paralysis in the United States as of 2013, which often leads to a significant reduction in quality of life (Armour et al., 2016). Many assistive technologies rely on muscle-based inputs, which are unusable for some patients, such as those with neurological conditions such as multiple sclerosis. Our project aims to use electroencephalography (EEG), enabling brain-computer interfacing, and reinforcement learning algorithms to control a robotic arm, offering a more accessible solution.

Research

Convolutional Neural Networks (CNNs) have been researched for EEG analysis, but it can often require large datasets, are prone to overfitting, and often operate in offline or non-real-time modes (Hosseini et al., 2020). Previous research at the University of Virginia focused on using CNNs for EEG-based control in an upper limb rehabilitation exoskeleton, but its implementation resulted in limitations during real-time testing (Zare & Sun, 2024). There was a lack of research in the application of reinforcement learning in EEG-based systems, despite reinforcement learning algorithms being able to adapt over time and not requiring labeled data to be accurate, which is the gap our project aimed to address.

Ideation

At the beginning of the fall semester, we each presented ten ideas for the project. Much of these ideas were more mechanical engineering-related rather than directly machine learning related, as can be seen in Tables 1 and 2. Details about our screening and selection process can be found in the section below.

Table 1: Summary of Pre-Screening Ideas (A-E).

Abbie	**A: Pattern Recognition**	**B: 1 DOF Arm.** This version will have the Headset attached to a robotic sleeve that will allow the user to move their elbow. A one degree of freedom motion.	**C: Distinctly different thoughts.** Thinking of something like music to move left and thinking of a tree to move right.	**D: 2 DOF.** The arm will have 2 degrees of freedom. It will allow for movement at both the Elbow and the shoulder.	**E: Therapy Device. Device.** Moves only in a certain way to provide physical therapy for a user
Hailey	**A: Control headset through microcontroller and emergency stop on fabric shoulder strap.** Put arm through bands, with wristband controlled by a motor with rod connections.	B: **Design A,** but the microcontroller and emergency stop button is on a neck attachment.	**C: Design B,** but the microcontroller turns a circular "joint," that pushes the arm upward or downward at the shoulder joint, which is attached to the user by a secure shirt.	**D: Design B,** but with a pulley system stored in a backpack the user wears.	**E: Design D,** but with a belt with a rod sticking out that supports the user's forearm.
Cayla	**A:** Survey and test at least 5 different subjects.	**B:** Study brain waves while participants move dominant arm moves during sessions. Daily sessions for a few minutes at a random time for a week.	**C:** Study brain waves while participants are sitting still in a designated room for a few minutes. Ask for their mood before, during, and after.	**D:** While studying brain waves, put a camera on a stand to film arm to connect brain waves to arm movements.	**E:** Implement band-pass filtering for better sorting.
Josh	**A: Unsupervised Learning Algorithm**. Finds patterns, similarities, differences in data without labels. Feed EEG data into algorithms to find patterns to figure out user intention.	**B: Reinforcement Learning Algorithm.** Program learns to take actions in an environment by receiving feedback (rewards or punishments) on its actions	**C: Clustering Algorithm.** unsupervised learning technique that groups data into subsets based on similarity. Program would group human intentions based on EEG signals.	**D**: **Semi-supervised Learning Algorithm.** Trains a program with a small amount of labeled data, then trains it further with unlabeled data.	**E: Supervised Learning Algorithm.** Uses labeled training data to predict labels

Table 2: Pre-Screening Ideas (F-J)

Abbie:	**F: Super Strength.** EEG will control an exoskeleton device that is capable of lifting heavy objects	**G: Machine Learning.** This version uses machine learning Artificial intelligence to adapt the code to be more accurate as more tests are completed	**H: Sports Trainer.** Device that helps with sports training	**I: Driving Arm.** This version allows people who have been paralyzed to drive cars.It is set up to function with cars that have hand controls installed so the user can drive with their thoughts.	**J: Extreme Conditions.** Heat resistant full sleeve arm that protects user from activities such as firefighting and welding
Hailey:	**F**: Design B with curved rod from hip and stiff rods attached by armbands and wristbands to rotate the arm.	**G:** Design B with a waistband having a curved rod attachment to move the arm.	**H:** Design B with rods supporting the arm, allowing it to rotate in a limited manner.	**I:** Armband and vest where the arm is controlled by a pulley system.	**J:** Armband and vest where the arm is controlled by a balloon inflation and deflation system.
Cayla:	F: Machine train programs to note maximum forearm side movement. If arm angle surpasses this (which will be observed using a camera), the programs will stop immediately	G: There will be a GUI that the tester can interface with so that in case of emergencies, the program can stop immediately.	H: To avoid using pneumatic actuators to imitate arm movement, use an automated pulley system instead.	I: To avoid using pneumatic actuators to imitate arm movement, tester can enable arm movement by electromagnets.	J: Safety could also be ensured by using touch sensors if using a frame for the electromagnet system.
Josh:	**F: Association Rule.** Finds associations and relationships among large sets of data items	**G: General Adversarial Network.** Autonomously identifies patterns in input data, enabling the model to produce new examples that resemble the original dataset. Could be used to create more test data to train other algorithms	**H: Q-Learning Algorithm.** Maps states to actions. Estimates the expected reward for taking a particular action in a given state. Could be used to have an algorithm examine multiple" intentions" which would result in the biggest reward based on the best match .	**I: State -Action-Reward-State-Action Algorithm.** Updates the expected reward for an action based on the action actually taken rather than the optimal action. Makes the labeling process more fluid.	**J: Anomaly Detection Algorithm**. Identifies outliers or anomalies in data. could be used to identify errors in labels and clean them up in order to have better predictions

Selection and Screening

We decided to first screen initial ideas based on the degrees of freedom allowed by each; how reliable data transmission would be; portability; how easy it is to physically put on; novelty; practicality, or its ease of use and implementation; and applicability to many people. The selection criteria prioritized efficiency and reliability above all else, with practicality being the category with the highest weight. The EEG robot is intended to be an assistive device; therefore it was decided that ease of use, reliable data and signal transmission, and applicability needed to be heavily considered. Additionally, this robot should be able to be portable, otherwise a user would be much more restricted with their movements, and with that, having multiple degrees of freedom is also very important. Novelty is important, as there is no use in reinventing the same technology, therefore implementing methods of innovation during the design process was heavily considered. A summary table listing the initial ideas that passed the screening stage can be seen in Figure 1.

	Concept Variants																	
	Abigail Dodd				Hailey Boyd					Cayla Celis			Joshua Rivas-Zelaya					
	B	D	G	I	A	F	H	I	J	A	E	G	A	C	D	H	I	J
Rank	1	1	3	2	3	1	3	3	3	2	1	1	3	1	1	1	1	1
Continue?	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES

Figure 1: Screenshot of summary table.

After further discussion we further chose ideas best suited for meeting our project objective as can be seen in Table 3.

Table 3: Pre-Scoring Ideas.

Label	Idea Description
A	An emergency stop button is on the top of the microcontroller.
B	Semi-supervised Learning Algorithm. Trains a program with a small amount of labeled data, then trains it further with unlabeled data.
C	Anomaly Detection Algorithm. Identifies outliers or anomalies in data. could be used to identify errors in labels and clean them up in order to have better predictions.
D	State -Action-Reward-State-Action (SARSA) Algorithm. Updates the expected reward for an action based on the action actually taken rather than the optimal action. Makes the labeling process more fluid.
E	Distinctly Different Thoughts. In this version, the user will be asked to think about something like music for left and tree for right. This way, the thoughts are more different in hopes it will be easier to decipher.

We then combined our pre-scoring ideas for further improved ideas, as can be seen in Table 4.

Table 4: Hybrid Ideas.

Label	Idea Description
A2	Semi-supervised algorithm with anomaly detection and an emergency stop
B2	SARSA algorithm with distinctly different thoughts
C2	Distinctly different thoughts with anomaly detection
D2	Semi-supervised algorithm with anomaly detection and distinctly different thoughts with an emergency stop

Our scoring criteria was essentially the same as the screening criteria. The results of our scoring can be seen in Table 5.

Table 5: Scoring Results.

		A.2	B.2	C.2	D.2
Selection Criteria	**Weight**	**Weighted Score**	**Weighted Score**	**Weighted Score**	**Weighted Score**
Degrees of Freedom	9.00%	0	0	0	0
Reliable Transmitting	10.00%	0.3	0.2	0.1	0.5
Portable	10.00%	0	0	0	0
Easiness to put on	10%	0	0	0	0
Novelty	6.00%	0.12	0.18	0.24	0.3
Practicality	13%	0.39	0.325	0.325	0.26
Applicability to many people	12%	0.48	0.24	0.36	0.6
Total Score		1.29	0.945	1.025	1.66
Rank		2	3	2	1
Continue?		NO	NO	NO	**Develop**

Initial Specifications

To guide the development of the project, we decided to establish initial specifications based on our selection criteria and project objectives. This can be seen in Table 6.

Table 6: Initial Specifications.

Rank	Metric	Units	Means of Testing
1	The arm will interface with human thoughts with at least a 65% reliability rate.	%	We will do at least 25 tests of movement and count the number of times the robot correctly interfaces with the human's intention.
2	The total cost of assembly of the robot will not exceed $800.	$	A spreadsheet will be kept of all costs. Planning will be put in place to not exceed this value.
3	The arm will be able to move with at least 1 DOF.	Binary	While testing, it will be checked whether it can move with at least 1 DOF or not.
4	The time to execute arm movements will take no longer than 60 seconds.	Seconds (s)	The duration of arm movement executions will be timed.
5	The transition time between arm movements will take no longer than 60 seconds.	Seconds (s)	Time duration between arm movement executions will be timed.
6	The device will include an emergency stop function.	Binary	The emergency stop function will be tested to determine if it will successfully disable device functions.
7	The device will be portable.	Binary	Will be observed to see if a user can move freely or not.
8	The time from not having the device on to it being in use will be less than 10 minutes.	Minutes (min)	Will time the application and removal time of the device on a user.
9	The device will have a minimum battery life of 15 minutes.	Minutes (min)	The capacity of the battery will be measured by recording its run-time.
10	The sampling rate will be at least 100 Hz.	Hz	Will be measured in code/serial monitor.

The specifications above assumed that the system would consist of the EEG headset and a robotic arm. The 65% reliability rate specification was chosen as it was lower than the

accuracy rate in Zare & Sun (2024) trials and an accuracy rate that seemed both achievable and reasonable for the project to be considered a success. This specification was ranked highly since reliable transmitting was a highly weighted selection criteria. As we were strictly limited to $800, specification 2 was ranked high as well. Specifications 3-6 related to the practicality of the system and were ranked high as practicality was the highest weighted selection criteria. Ease to put on and portability selection criteria were addressed in specifications 7 and 8. Finally, specifications 9 and 10 were ranked last as these details did not affect the overall goal of the project to the same extent.

Final Specifications

Specifications drastically changed between the spring and fall semesters as the project shifted to be more aligned with computer-science principles than mechanical engineering. Specifically, during the spring semester, we quickly realized how some of our specifications were no longer relevant to our project, especially as this project became more focused on improving on the previous model's accuracy rate of 75.3% as based on the results by Zare & Sun (2024). This can be seen with Table 7.

Table 7: Final Specifications.

Rank	Metric	Means of Testing
1	The algorithm will be able to identify when the user is tense or relaxed at a general accuracy rate 80% - 100% in real time.	We will do at least 25 tests of movement and count the number of times the arm correctly interfaces with the human's intention.
2	The arm will be able to fully make a fist or unclench.	While testing, it will be checked whether it can do both of these actions.
3	The total cost of assembly of the robot will not exceed $800.	We will use spreadsheets to track purchases.
4	The time to execute each arm movement will take no longer than 60 seconds.	The duration of arm movement executions will be timed.
5	The transition time between arm movements will take no longer than 60 seconds.	Time duration between arm movement executions will be timed.
6	The device will allow frequencies between 0.5 Hz and 50 Hz	Filtering capabilities will be tested during training and actual deployment.
7	The algorithm will increase the total reward as the number of epochs it undergoes increases.	We will graph the total reward after training has completed.
8	The algorithm will undergo epsilon decay as the number of epochs it undergoes increases.	We will graph the total decay after training has completed.
9	The arm will be able to identify which brain signals in the training data are associated with tensing and relaxing.	We will run each training session with the algorithm and see if the arm grasps or unclenches.
10	The EEG headset will be able to calibrate with the algorithm in less than ten minutes when in a session.	We will test this when setting up for tests of movement.

The importance of the arm's accuracy rate stayed consistent, as seen in specification 1's ranking being similar to the initial rankings; however, the accuracy rate we wanted to target increased as proper implementation of reinforcement learning became more important than a structurally optimal arm. Specification 2 was ranked like so as the whole goal of the project was to have the arm's fist clench or unclench; the arm not being able to do so would delay project completion. While mechanical design was not a central focus in our project, it was important to heed specification 3 as we needed external components, such as a Raspberry Pi, to interface with the arm in the most cost effective way possible. Specifications 4 & 5 were ranked like so because for the device to be utilized in the real world, the headset needs to be able to quickly identify the user's brain signals, and the arm needs to act accordingly; however, we were more concerned with its accuracy regardless of time taken and decided to rank these specifications lower. Specification 6 was listed as this range encompasses the different kinds of brain waves. Specifications 7-9 concerned training of the algorithm, which is conducive for the algorithm to reach our specified accuracy rate. Finally, specification 10 was listed to increase the user's comfortability with the device.

Analysis & Calculations

When analyzing which clusters to utilize as triggers for controlling the prosthetic device, we collected data of various binary mental states, such as relaxed vs focused, eyes closed vs speed reading, thinking left vs right, and many more. Data was collected from the four of us, and the average power spectral density (PSD) from each mental state was compared against its opposite. We looked to find significant differences between each mental state's average PSD against the PSD for its opposite state to determine if any of these mental action sets had distinct output signals for the DQN to detect and train upon. In doing so, we found that most clusters did not have significant enough differences in their average PSD except for when comparing the signals the user had when tensing their body vs being relaxed. This tension could be in the form of a clenched jaw, raised heel, or flexed muscle. The output average PSD from a tensed state vs relaxed state had an average difference of about 240 $\mu V^2/Hz$.

Table 8: Band powers for each condition and power band.

Power Band	**Untense PSD ($\mu V^2/Hz$)**	**Tense PSD ($\mu V^2/Hz$)**	**Difference ($\mu V^2/Hz$)**
Delta	498.87	1585.66	1086.79
Theta	29.86	107.28	77.41
Alpha	14.43	51.20	36.78
Beta	5.09	13.23	8.14
Gamma	6.00	0.60	-5.41
Average	110.85	351.59	240.74

Testing

To evaluate our device, we conducted tests of both the training files and real-time actuation. During the initial development phases of the code and robotic hand, we recorded data that captured spikes in brain activity. This data was then uploaded into the program directly and not as a training file to assess whether the hand could successfully clench and unclench. These evaluations were performed prior to implementing real-time actuation. Thirty sessions of a user completing varying tasks were recorded. These tasks included sitting still with eyes closed, sitting with eyes open, making a fist, bicep curls, and concentrating.

After confirming that the device could reliably perform the intended tasks, the algorithm required the training files to be uploaded into the Q-table. Sessions involving relaxation, movement, muscle tensing, and concentration were recorded from multiple users. These recordings were subsequently imported into an Excel spreadsheet, where the average brainwave data for each activity was analyzed and compared (Figure 2). The activities demonstrating the greatest differences in brainwave patterns, specifically, full-body relaxation and maximal muscle tensing — were selected as the control tasks for the device. For each condition, ten sessions were recorded: one involving the user closing their eyes, relaxing, and remaining motionless, and another involving the user tensing all muscles. Subsequently, the performance of the reinforcement learning algorithm was evaluated by graphing epsilon decay and cumulative rewards as functions of epoch (time slice). As illustrated in Figure 2, the epsilon decay exhibited a negative exponential trend, while the cumulative rewards displayed a positive exponential trend. These results suggest that the algorithm successfully learned from the provided training data over time.

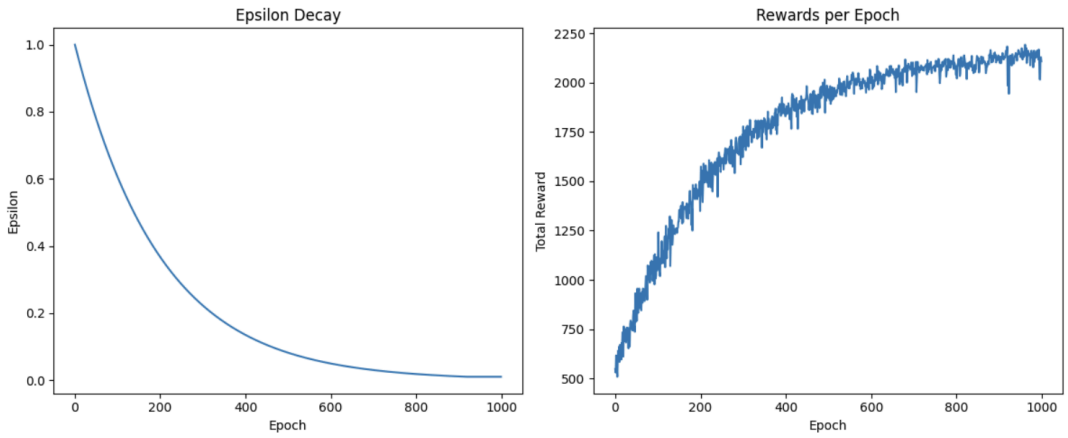

Figure 2: Left graph depicting epsilon decay. Right graph shows rewards per epoch.

After determining that the algorithm could learn effectively from the training data, real-time actuation testing was conducted. In this phase, live brainwave data was fed directly into the system to evaluate the device's ability to respond dynamically. The robotic hand's performance was assessed based on its ability to correctly interpret the user's brain activity in real time and execute the appropriate clenching or unclenching motions. For this test, 25 attempts to clench the robotic hand were made per session. The user was asked to tense their entire body for this test. Whether the hand successfully carried out the user's intention, carried it out with a delay, or not at all was recorded in an Excel spreadsheet and made into a pie graph (Figure 3). As seen from the chart, the model and device successfully carried out the user's intention 80.9% of the time during the first 50 trials. The last 50 trials demonstrated an overall accuracy rate of 96%, with the arm detecting tension instantly 46% of the time or with delay 50% of the time. Only 4% of tension instances were undetected. No false positives were observed. The data for the last 50 trials can be seen in Table 9.

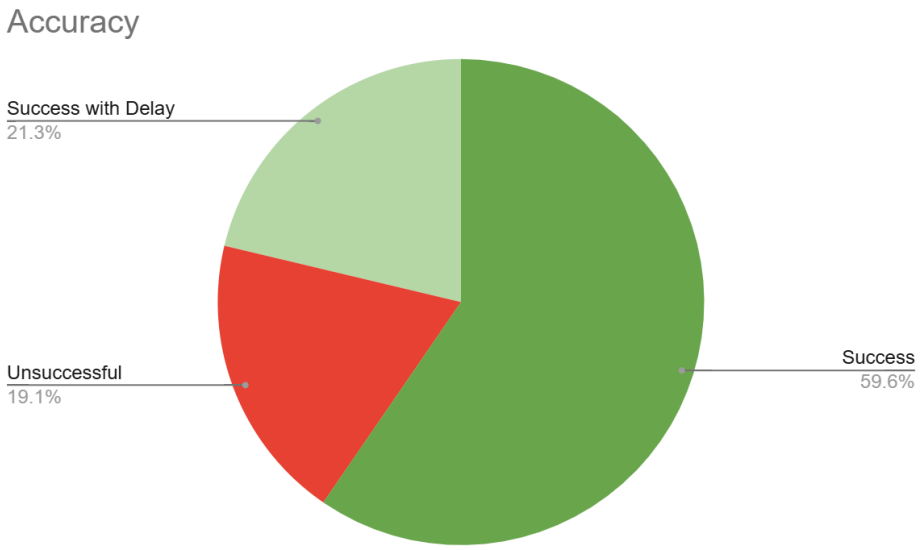

Figure 3: Summary Pie Chart of First Tension Instances.

Table 9: Results Summary of Last Tension Instances.

	Instant Success	**Success w/ Delay**	**No Success**	**False Positive**
Trial Set 3 (51-75)	10	13	2	0
Trial Set 4 (76-100)	13	12	0	0
Total	23	25	2	0
Average	11.5	12.5	1	0
Overall	46.00%	50.00%	4.00%	0.00%

Summary and Conclusions

This project set out to create a system that could pick up EEG signals tied to intentional mental commands, process them with a reinforcement learning model, and use them to move a robotic arm in real time. The final design used an OpenBCI EEG headset, a Raspberry Pi, and a 3D-printed robotic arm with one degree of freedom. A Deep Q-Network (DQN) was trained on both recorded and live brainwave data, allowing the arm to respond to increases in the user's brain activity.

During the project's development, we researched current assistive technologies, many of which still rely on muscle signals like EMG or IMU sensors, which can not be used by all users depending on the severity of their paralysis. Additionally, most previous research projects that did use EEGs, used convolutional neural networks (CNNs) to process the signals. However, CNNs often need a lot of labeled data and do not work well in real time. By using reinforcement learning, we built a system that could learn with less data, handle noisy signals, and adapt over time.

Once the device was created, our team tested the device on both pre-recorded brainwave data and live sessions. Early tests showed that the system could correctly turn mental commands into movements. In real-time testing, the robotic arm followed user commands correctly with an average accuracy rate of 88.45% for full body tensing. Tracking the model's epsilon decay and reward growth showed that the system kept learning and improving during use. Tests were also run to see how little muscle tension was needed to trigger movement. By having users clench just their hand, jaw, or toes, we looked at how much tensing was needed to significantly change the brainwave signals. These results showed that the system could generalize what it had learned and stay reliable even with small differences in how users gave commands.

Despite these promising results, there were still some challenges. The system was trained with few users, as we did not pursue IRB approval to test on human subjects other than ourselves due to lack of time. Additionally, we experienced problems with keeping all of the electrodes in contact with the user's head. This caused the brainwaves to not always be measured accurately and the robotic hand to not always accurately carry out the user's intention. There were also occasional lags in processing live data. Still, the project showed that reinforcement learning is a strong option for controlling prosthetics through EEG.

For the future, there are many improvements to the project that could be made. The robotic hand can be improved by adding more degrees of freedom and designing it to be able to be physically attached to a person. Also, the EEG signal processing can be improved to better handle noise. The program would also benefit from testing and training on a wider range of users. A partnership with healthcare providers to test the system with individuals who have motor impairments would be the most accurate testing of whether the system is an effective aid for motor control. Overall, this project proved that reinforcement learning can be used to interpret brainwaves for real-time control of a robotic arm, offering a new path toward more accessible and adaptable assistive technologies.

Appendix

Bibliography

Armour, B. S., Courtney-Long, E. A., Fox, M. H., Fredine, H., & Cahill, A. (2016). Prevalence and causes of paralysis—United States, 2013. *American Journal of Public Health, 106*(10), 1855–1857. https://doi.org/10.2105/AJPH.2016.303270

Cao, Z. (2020). A review of artificial intelligence for EEG‐based brain−computer interfaces and applications. *Brain Science Advances, 6*(3), 162–170. https://doi.org/10.26599/bsa.2020.9050017

Hosseini, M., Hosseini, A., & Ahi, K. (2020). A review on machine learning for EEG signal processing in bioengineering. *IEEE Reviews in Biomedical Engineering, 14*, 204–218. https://doi.org/10.1109/rbme.2020.2969915

Zare, S., & Sun, Y. (2024). EEG motor imagery classification using integrated transformer-CNN for assistive technology control. In *2024 IEEE/ACM Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)* (pp. 189–190). https://doi.org/10.1109/CHASE60773.2024.00033

Detailed Drawings

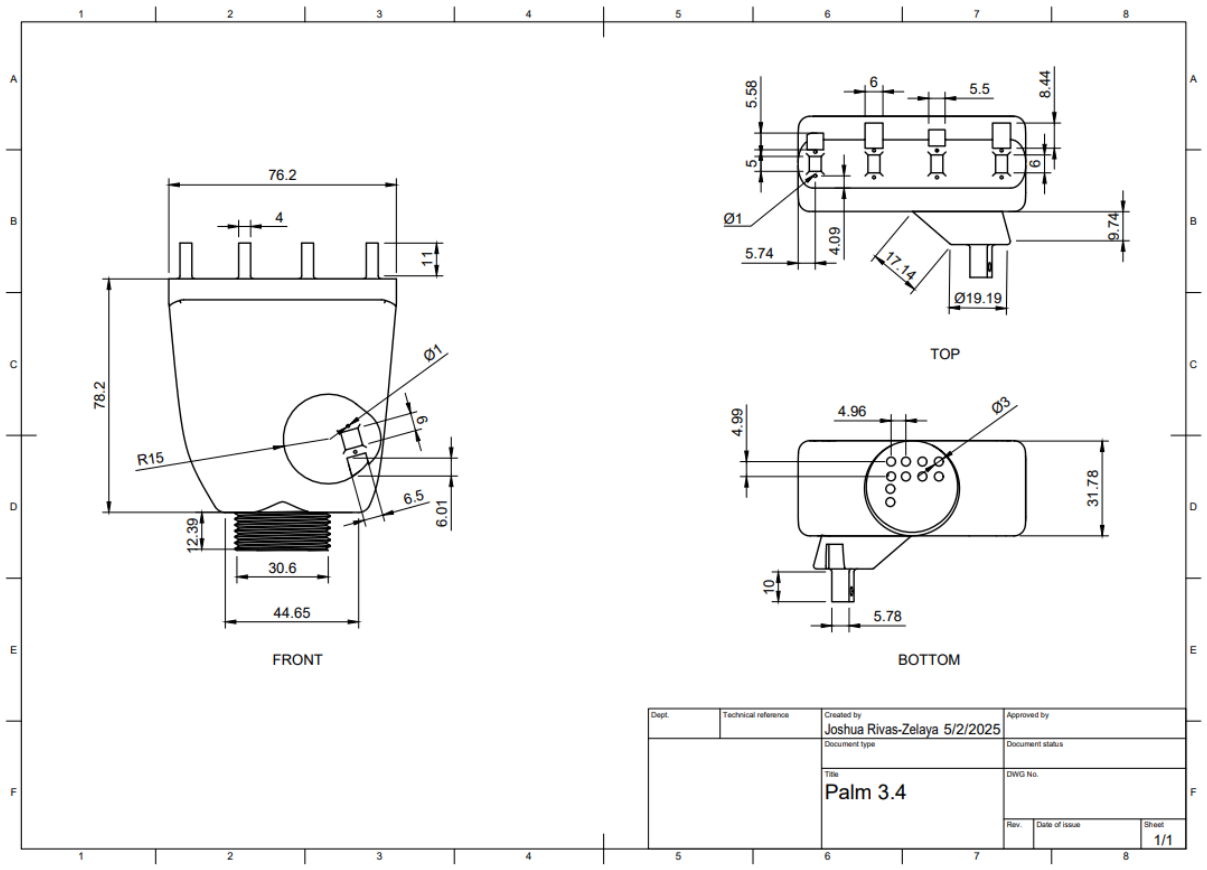

Figure 4: 2D Drawing of Prosthetic Palm

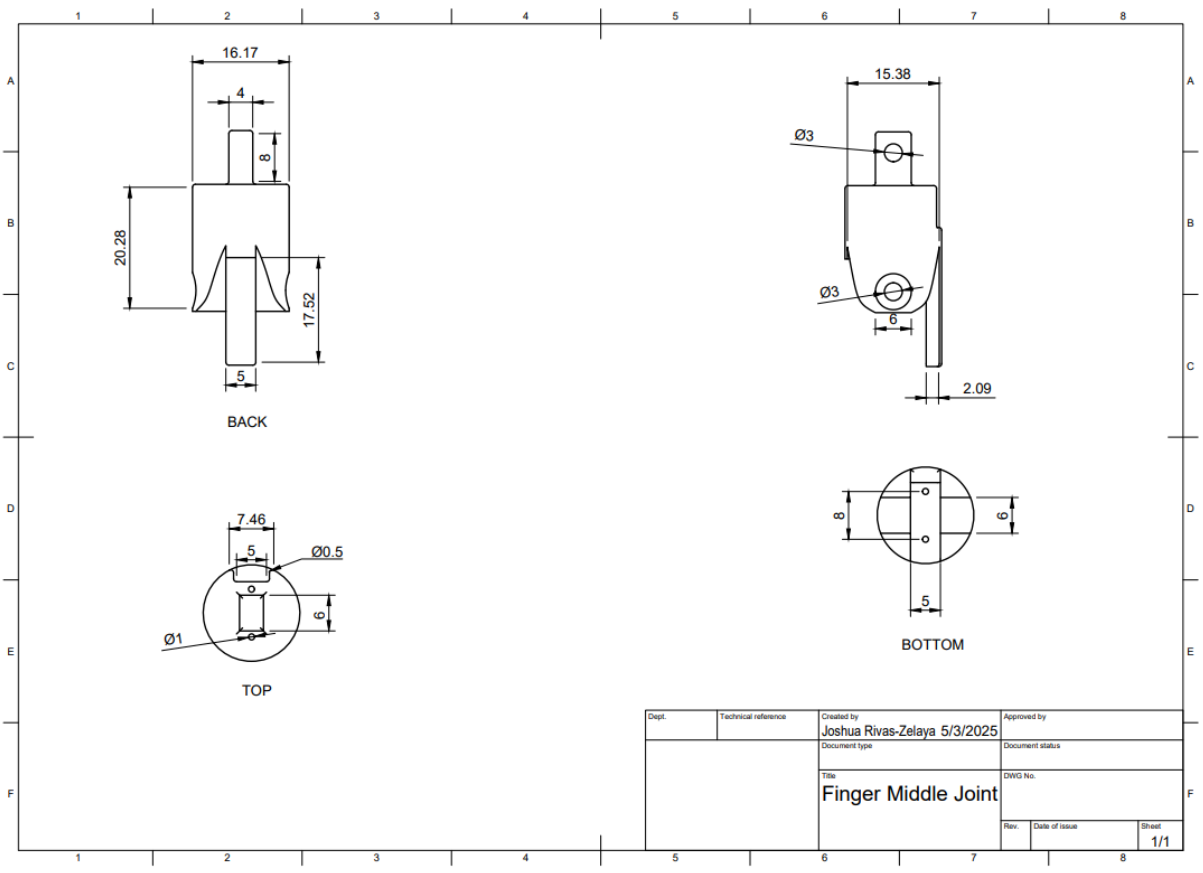

Figure 5: 2D Drawing of Middle Joint of Prosthetic Finger

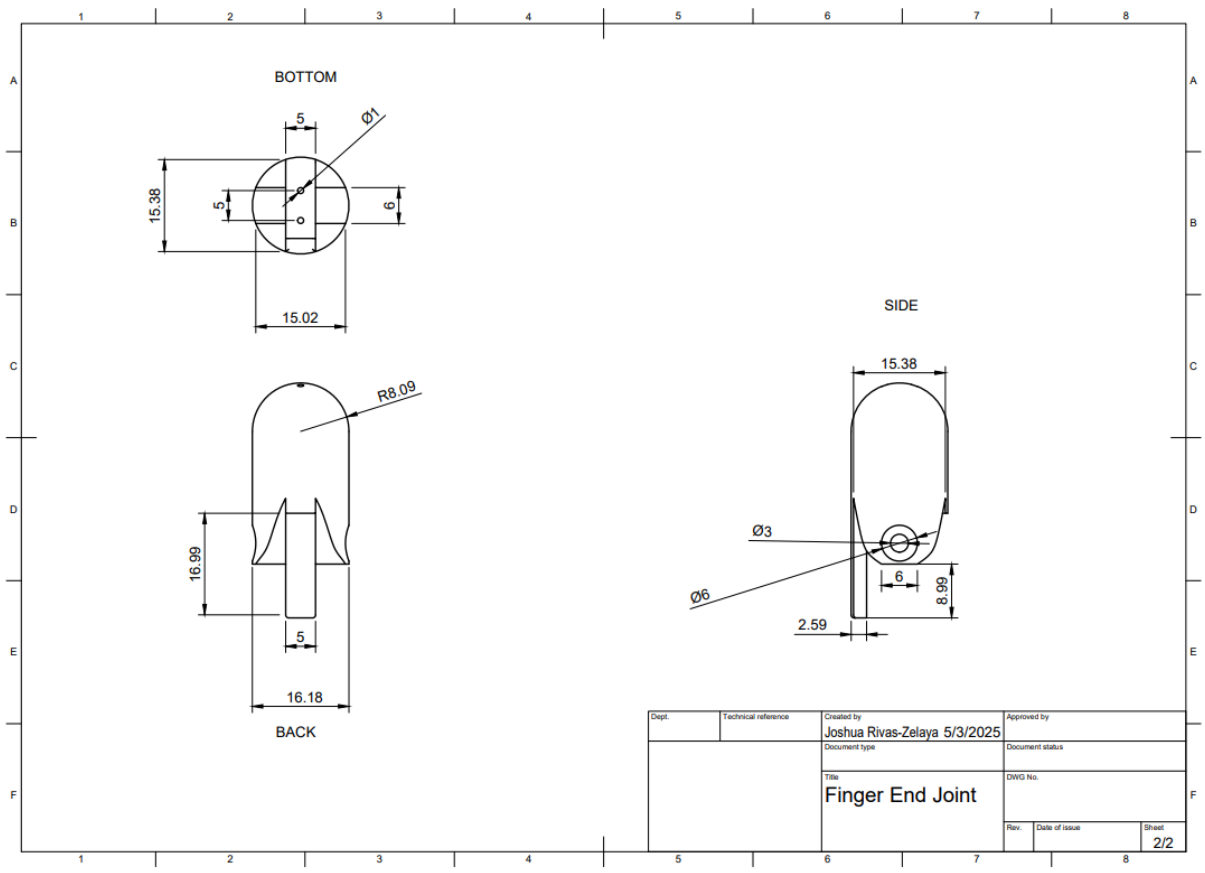

Figure 6: 2D Drawing of Prosthetic Finger Tip Joint

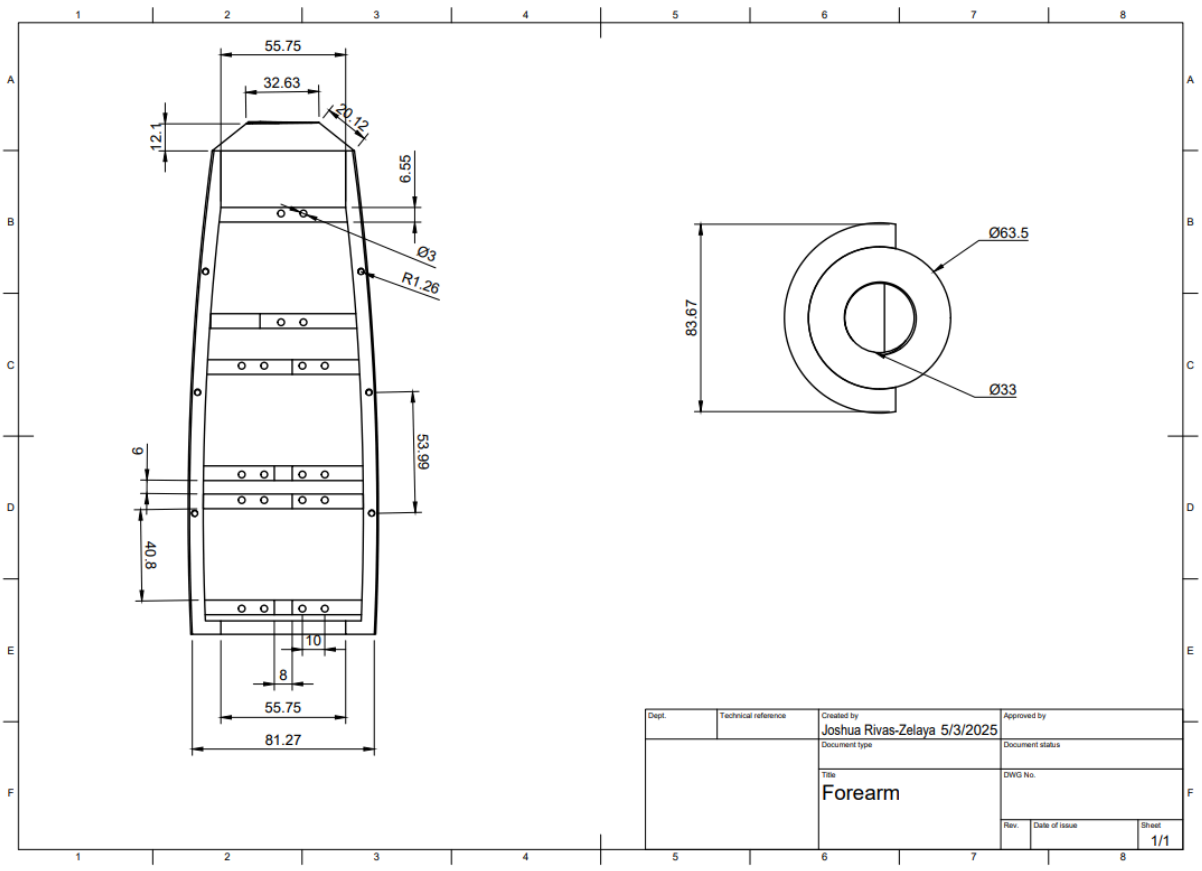

Figure 7: 2D Drawing of Prosthetic Arm

Code

```python
import os
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from scipy import signal
from collections import deque


# === Model Definition ===
class DQN(nn.Module):
    def __init__(self, num_states, num_actions):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(num_states, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, num_actions)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)


# === Feature Extraction ===
def bandpass_filter(data, lowcut=0.5, highcut=50, fs=125, order=4):
    nyq = 0.5 * fs
    b, a = signal.butter(order, [lowcut / nyq, highcut / nyq], btype='band')
    return signal.filtfilt(b, a, data, axis=0)


def compute_fft_features(data, fs=125):
    fft_vals = np.abs(np.fft.rfft(data, axis=0))
    freqs = np.fft.rfftfreq(data.shape[0], d=1 / fs)

    def band_power(low, high):
        idx = np.where((freqs >= low) & (freqs <= high))
        return np.mean(fft_vals[idx], axis=0)

    delta = band_power(0.5, 4)
    theta = band_power(4, 8)
    alpha = band_power(8, 14)
    beta = band_power(14, 30)
    gamma = band_power(30, 50)

    return np.vstack([delta, theta, alpha, beta, gamma]).T  # (channels, 5)


# === Windowing ===
def create_windows(data, window_size):
    return [data[i:i + window_size] for i in range(0, len(data) - window_size + 1, window_size)]


# === Load and Preprocess Data ===
def load_eeg_data(files, window_size):
    scaler = StandardScaler()
    eeg_tensors = {}
    states_flat = {}

    for file in files:
        df = pd.read_csv(file, header=None, delimiter='\t', usecols=range(1, 17), skiprows=250)  # 16 channels
        data = df.values
```

Figure 8a: Page 1/4 EEG DQN Python Program

```
        data = df.values
        data = bandpass_filter(data)

        eeg_windows = create_windows(data, window_size)
        window_features = []

        for window in eeg_windows:
            features = compute_fft_features(window)  # (16, 5)
            features_norm = scaler.fit_transform(features)
            # flat = torch.FloatTensor([features.flatten()])
            window_features.append(features_norm.flatten())  # (80,)

        eeg_states_tensor = torch.FloatTensor(window_features)
        eeg_tensors[file] = eeg_states_tensor
        states_flat[file] = np.array(window_features)

    return eeg_tensors, states_flat


# === Training ===
def train_dqn(eeg_tensors, label_map, epochs=1000):
    num_states = 80
    num_actions = 2
    model = DQN(num_states, num_actions)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    loss_fn = nn.CrossEntropyLoss()

    epsilons = []
    rewards_per_epoch = []
    epsilon = 1.0
    epsilon_min = 0.01
    epsilon_decay = 0.995
    memory = deque(maxlen=10000)
    batch_size = 64

    model.train()
    for epoch in range(epochs):
        total_reward = 0
        correct = 0
        total = 0

        # Collect memory
        for file, tensor in eeg_tensors.items():
            label = label_map.get(os.path.basename(file), 0)
            for state in tensor:
                state = state.unsqueeze(0)
                if random.random() < epsilon:
                    action = random.randint(0, 1)
                else:
                    with torch.no_grad():
                        q_values = model(state)
                        action = q_values.argmax().item()

                reward = 2.0 if action == label else -1.0
                total_reward += reward
                correct += int(action == label)
                total += 1

                memory.append((state, label, reward))

        # Sample and train from memory
        if len(memory) >= batch_size:
            batch = random.sample(memory, batch_size)
            for state, label, reward in batch:
                output = model(state)
                loss = loss_fn(output, torch.tensor([label]))
```

Figure 8b: Page 2/4 of EEG DQN Python Program

```
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

        rewards_per_epoch.append(total_reward)
        epsilons.append(epsilon)
        epsilon = max(epsilon * epsilon_decay, epsilon_min)
        accuracy = 100 * correct / total if total > 0 else 0.0
        print(
            f"Epoch {epoch + 1}/{epochs} | Reward: {total_reward} | Accuracy: {accuracy:.2f}% | Epsilon: {epsilon:.4f}")

    torch.save(model.state_dict(), "Hailey_Tensing_Model_New.pt")
    print("✅ Model saved as Hailey_Tensing_Model_New.pt")

    # === Plot graphs ===
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epsilons)
    plt.title("Epsilon Decay")
    plt.xlabel("Epoch")
    plt.ylabel("Epsilon")

    plt.subplot(1, 2, 2)
    plt.plot(rewards_per_epoch)
    plt.title("Rewards per Epoch")
    plt.xlabel("Epoch")
    plt.ylabel("Total Reward")
    plt.tight_layout()
    plt.show()


# === Main ===
if __name__ == "__main__":
    window_size = 125

    file_paths = [
        "BrainFlow-RAW_Hailey_Not_Tense_3_0.csv",
        "BrainFlow-RAW_Hailey_Not_Tense_4_0.csv",
        "BrainFlow-RAW_Hailey_Not_Tense_7_0.csv",
        "BrainFlow-RAW_Hailey_Not_Tense_8_0.csv",
        "BrainFlow-RAW_Hailey_Not_Tense_9_0.csv",
        "BrainFlow-RAW_Hailey_Not_Tense_10_0.csv",
        "BrainFlow-RAW_Hailey_Not_Tense_12_0.csv",
        "BrainFlow-RAW_Hailey_Not_Tense_13_0.csv",
        "BrainFlow-RAW_Hailey_Not_Tense_14_0.csv",
        "BrainFlow-RAW_Hailey_Not_Tense_15_0.csv",
        "BrainFlow-RAW_Hailey_Tense_1_0.csv",
        "BrainFlow-RAW_Hailey_Tense_2_0.csv",
        "BrainFlow-RAW_Hailey_Tense_3_0.csv",
        "BrainFlow-RAW_Hailey_Tense_4_0.csv",
        "BrainFlow-RAW_Hailey_Tense_5_0.csv",
        "BrainFlow-RAW_Hailey_Tense_6_0.csv",
        "BrainFlow-RAW_Hailey_Tense_11_0.csv",
        "BrainFlow-RAW_Hailey_Tense_12_0.csv",
        "BrainFlow-RAW_Hailey_Tense_13_0.csv",
        "BrainFlow-RAW_Hailey_Tense_14_0.csv",
        "BrainFlow-RAW_Hailey_Tense_15_0.csv"
    ]

    label_map = {
        "BrainFlow-RAW_Hailey_Not_Tense_3_0.csv":0,
        "BrainFlow-RAW_Hailey_Not_Tense_4_0.csv":0,
        "BrainFlow-RAW_Hailey_Not_Tense_7_0.csv":0,
        "BrainFlow-RAW_Hailey_Not_Tense_8_0.csv":0,
        "BrainFlow-RAW_Hailey_Not_Tense_9_0.csv":0,
        "BrainFlow-RAW_Hailey_Not_Tense_10_0.csv":0,
        "BrainFlow-RAW_Hailey_Not_Tense_12_0.csv":0,
        "BrainFlow-RAW_Hailey_Not_Tense_13_0.csv":0,
        "BrainFlow-RAW_Hailey_Not_Tense_14_0.csv":0,
```

Figure 8c: Page 3/4 of EEG DQN Python Program

```
        "BrainFlow-RAW_Hailey_Not_Tense_14_0.csv":0,
        "BrainFlow-RAW_Hailey_Not_Tense_15_0.csv":0,
        "BrainFlow-RAW_Hailey_Tense_1_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_2_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_3_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_4_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_5_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_6_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_11_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_12_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_13_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_14_0.csv":1,
        "BrainFlow-RAW_Hailey_Tense_15_0.csv":1
    }

    eeg_tensors, states_flat = load_eeg_data(file_paths, window_size)
    train_dqn(eeg_tensors, label_map)

```

Figure 8d: Page 4/4 of EEG DQN Python Program

```python
import os
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
import ctypes
import pi_servo_hat
import time
from scipy import signal
from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes

# Load BrainFlow
so_path = "/home/admin/.local/lib/python3.11/site-packages/brainflow/lib/libBoardController.so"
os.environ["LD_LIBRARY_PATH"] = os.path.dirname(so_path)
ctypes.CDLL(so_path)

# Load pHAT Settings
test = pi_servo_hat.PiServoHat()
test.restart()
# Moves servo position to 0 degrees (1ms), Channel 0
test.move_servo_position(0, 0)
test.move_servo_position(1, 0)
test.move_servo_position(2, 0)
test.move_servo_position(3, 0)
test.move_servo_position(4, 0)

# Pause 1 sec
time.sleep(1)

# DQN Model
class DQN(nn.Module):
    def __init__(self, num_states=80, num_actions=2):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(num_states, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, num_actions)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)

# Bandpass Filter
def bandpass_filter(data, lowcut=0.5, highcut=50, fs=125, order=4):
    nyq = 0.5 * fs
    b, a = signal.butter(order, [lowcut / nyq, highcut / nyq], btype='band')
    return signal.filtfilt(b, a, data, axis=0)

# Compute Band-power
def compute_fft_features(data, fs=125):
    fft_vals = np.abs(np.fft.rfft(data, axis=0))
    freqs = np.fft.rfftfreq(data.shape[0], d=1/fs)

    def band_power(low, high):
        idx = np.where((freqs >= low) & (freqs <= high))
        return np.mean(fft_vals[idx], axis=0)

    delta = band_power(0.5, 4)
    theta = band_power(4, 8)
    alpha = band_power(8, 14)
    beta = band_power(14, 30)
    gamma = band_power(30, 50)

    return np.vstack([delta, theta, alpha, beta, gamma]).T  # shape: (16, 5)

```

Figure 9a: Page 1/3 of Live Prosthetic Control Python Program

```
# Set up BrainFlow
params = BrainFlowInputParams()
params.serial_port = "/dev/ttyUSB0"
board_id = BoardIds.CYTON_DAISY_BOARD.value
board = BoardShim(board_id, params)

# EEG Settings
window_size = 125  # 1 second at 125 Hz
num_channels = 16
eeg_channels = BoardShim.get_eeg_channels(board_id)
timestamp_channel = BoardShim.get_timestamp_channel(board_id)

#  Load trained DQN model
model = DQN(num_states=80, num_actions=2)
model.load_state_dict(torch.load("/home/admin/PycharmProjects/PythonProject/Josh_Tensing_Model_New.pt", map_location=torch.device('cpu')))
model.eval()
print("Loaded model Josh_Tensing_Model_New.pt")

# Start streaming
BoardShim.enable_dev_board_logger()
board.prepare_session()
board.start_stream()
print("Streaming EEG from Cyton-Daisy...")

# Live plot setup
plt.ion()
fig, ax = plt.subplots()
plot_data = np.zeros((window_size, num_channels))
lines = ax.plot(plot_data)
ax.set_ylim(-100, 100)
ax.set_title("Live EEG + Prediction")
prediction_text = ax.text(0.01, 0.95, "", transform=ax.transAxes, fontsize=14,
                          verticalalignment='top', bbox=dict(facecolor='white', alpha=0.8))

# Main loop
try:
    while True:
        time.sleep(1)
        data = board.get_current_board_data(window_size)
        if data.shape[1] < window_size:
            print("Waiting for more data...")
            continue

        eeg_data = data[eeg_channels, :]
        filtered = bandpass_filter(eeg_data.T)
        print("EEG range (micro volts): ", np.round(np.ptp(filtered, axis=0), 2))
        if np.mean(np.ptp(filtered, axis=0)) < 5.0:
            print("Weak signal - check electrodes or increase activity")

        fft_features = compute_fft_features(filtered)
        features_flat = fft_features.flatten().astype(np.float32)
        input_tensor = torch.tensor(features_flat).unsqueeze(0)

        with torch.no_grad():
            q_values = model(input_tensor)
            prediction = q_values.argmax().item()

```

Figure 9b: Page 2/3 of Live Prosthetic Control Python Program

```
        action_str = "ON" if prediction == 1 else "OFF"
        print(f"Prediction: {action_str}")
        if action_str == "ON":
            test.move_servo_position(0, 90)
            test.move_servo_position(1, 90)
            test.move_servo_position(2, 90)
            test.move_servo_position(3, 90)
            test.move_servo_position(4, 90)
            time.sleep(1)
            test.move_servo_position(0, 0)
            test.move_servo_position(1, 0)
            test.move_servo_position(2, 0)
            test.move_servo_position(3, 0)
            test.move_servo_position(4, 0)
            time.sleep(1)
        else:
            test.move_servo_position(0, 0)
            test.move_servo_position(1, 0)
            test.move_servo_position(2, 0)
            test.move_servo_position(3, 0)
            test.move_servo_position(4, 0)
            time.sleep(1)

        # Update plot
        plot_data = filtered
        for i, line in enumerate(lines):
            line.set_ydata(plot_data[:, i])
        prediction_text.set_text(f"Prediction: {action_str}")
        fig.canvas.draw()
        fig.canvas.flush_events()

except KeyboardInterrupt:
    print("Stopped by user.")
finally:
    board.stop_stream()
    board.release_session()
    print("Session closed.")
```

Figure 9b: Page 3/3 of Live Prosthetic Control Python Program