Introduction:

Brief:
Concussions are a major health concern, particularly in sports, with approximately 3.8 million sports-related concussions occurring annually in the US. This project aims to improve concussion detection by integrating impact sensors and EEG monitoring into a wearable helmet.

Content:
Watching a college-level football game, I witnessed a jarring moment that left a deep impact on me. A teammate collided with another player and fell to the ground. He seemed fine though—he stood up and continued playing, showing no immediate signs of distress. However, over the next few days, he became dizzy and disoriented, prompting the coach to take him off the field. We later learned he had suffered a concussion from the impact.

What struck me most was how subtle and delayed the symptoms were, making it nearly impossible to detect the injury immediately. If the injury had been immediately noticeable, action could have been taken sooner. This incident highlighted that some injuries might seem minor initially while having serious underlying effects. This realization made me think about the many scenarios where hidden injuries could occur—not just in athletes, but also in children engaged in activities such as riding bikes, scooters, or roller skates.

A few months later, another incident reinforced this idea when two friends crashed into each other while biking and riding an electric scooter. They both seemed shaken up, but we couldn't tell if they were injured or not. The uncertainty made us anxious, and we had to wait until Monday to take them to the doctor for a proper check-up. When we finally saw the doctor, we were relieved to hear that they had suffered no brain injuries. However, the wait time before we could get a definitive answer was painful as we were left wondering about the severity of their injuries.

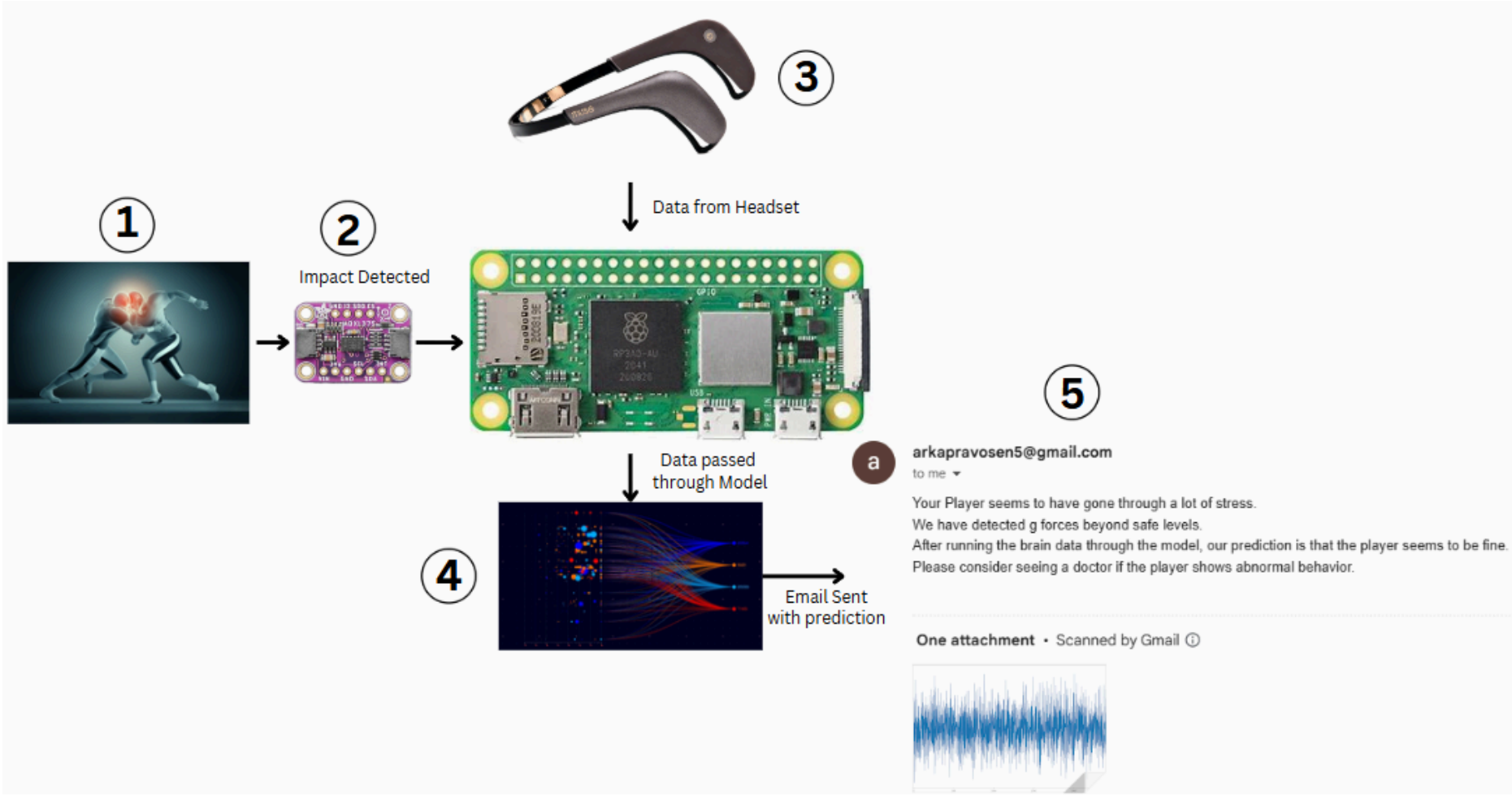
With this project, I hope to make concussion detection more accessible reducing the risk of long-term brain damage caused by undiagnosed concussions in sports, recreational activities, and everyday life.

Problem & Goal

Problem:
Concussions are a major concern in sports and recreational activities, with approximately 3.8 million sports-related concussions occurring annually in the US. These injuries can lead to long-term cognitive, behavioral, and emotional issues if not detected early. Current detection methods rely mainly on visible symptoms or reported impact forces, leaving a critical gap in real-time concussion monitoring. Additionally, studies show that 30% to 50% of sports-related concussions are not reported immediately, often due to a lack of awareness, the fear of losing playtime, or misconceptions about the severity of the injury. This delay in reporting further complicates timely diagnosis and treatment.

Goal:
My goal was to develop a helmet equipped with accelerometers and EEG sensors to detect concussions by measuring impact forces and abnormal brain activity, thereby ensuring timely medical intervention and improved safety. While initially targeted towards athletes, this technology could also benefit children riding bikes, roller skates, or scooters in communities, expanding its application to enhance safety across various demographics.

High-level Working Mechanism



1. **An impact occurs.**
2. **The impact is detected, and the Raspberry Pi is triggered to gather data from the EEG headset.**
3. **The Raspberry Pi collects data from the headset.**
4. **The collected data is processed through a trained model to obtain a prediction.**
5. **The Raspberry Pi sends an email notification with the model's prediction.**

Hardware Used

Items	Current Price
RPi Zero 2W - Used for interfacing with sensors and processing data	$ 15.00
Muse 2 EEG Headset - Gathers data from the brain	$ 199.99
ADXL 375 Accelerometer - Detects Impact forces	$ 15.00
4G USB Dongle	$ 9.05
SD Card 32GB - Stores all code and data	$ 6.99
10000 Mah Power Bank - Powers the system	$ 17.99
TOTAL	**$ 264.03**

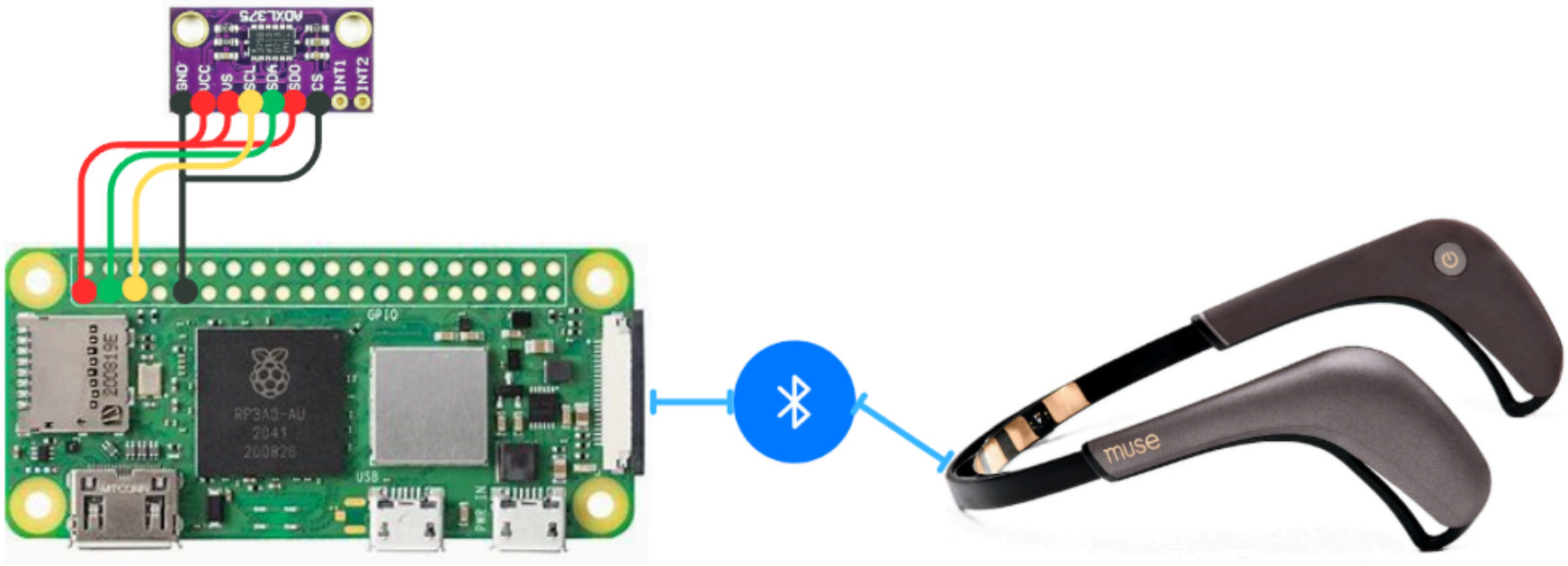
Software Used

Items	Current Price
PyTorch - Used to build and train Deep Learning models	$ 0.00
MuseLSL2 - Used to interface with the Muse 2 headset to gather EEG data	$ 0.00
Visual Studio Code - Used to code	$ 0.00
RaspberryPi Imager - Used to download OS onto Raspberry Pi	$ 0.00

Python - Language of Choice	$ 0.00
TOTAL	**$ 0.00**

Making The Device

Hardware Set-up



Setting up the Raspberry Pi

The Raspberry Pi was set up by first downloading the Raspberry Pi Imager. A MicroSD card was inserted into a computer, and the operating system (OS) was installed onto it using the imager. Once the OS was successfully written to the SD card, it was inserted into the Raspberry Pi.

After powering on the device, the Command Line Interface (CLI) was loaded, and SSH was enabled in the Pi's configuration settings. The Raspberry Pi was then accessed remotely using PuTTY.

Connecting the ADXL375 Accelerometer to the Raspberry Pi
To measure impact forces, the ADXL375 accelerometer was connected to the Raspberry Pi using the I2C interface.

- **Powering the ADXL375:**
  - VCC on the ADXL375 was connected to the Raspberry Pi's 3.3V pin.
  - GND on the ADXL375 was connected to the Raspberry Pi's GND pin.
- **I2C Communication:**
  - SDA (Serial Data) on the ADXL375 was connected to GPIO 2 (SDA) on the Raspberry Pi.
  - SCL (Serial Clock) on the ADXL375 was connected to GPIO 3 (SCL) on the Raspberry Pi.

Connecting the Muse EEG Headset to the Raspberry Pi
The Muse EEG headset was connected to the Raspberry Pi wirelessly via Bluetooth using the MuseLSL2 library to stream EEG data. Bluetooth on the Raspberry Pi was enabled through system settings, and the MuseLSL2 library was utilized to establish the connection with the headset.

Setting up the Python Environment
The Python environment was prepared with the necessary libraries to facilitate sensor integration and data processing:

- **PyTorch** for running machine learning models.
- **I2C libraries** for interfacing with the ADXL375 accelerometer.
- **MuseLSL2** for handling EEG data streaming from the Muse headset.

Powering the System
The Raspberry Pi and connected sensors were powered using a portable USB power bank, ensuring the system remained mobile and fully operational within the helmet.

This setup enabled real-time sensor data collection, processing, and alert notifications, forming a robust foundation for concussion detection and monitoring.

Rationale for Selecting Hardware Components:

When I first started the project, I had a list of hardware in mind to use. I wanted to use the Raspberry Pi 4 Model B because of its processing power, the ADXL345 as the accelerometer to detect impacts, and the Neurosky Mindwave Mobile 2 to gather EEG data because it was affordable at $100. However, after further research, I realized that concussions begin to occur at 60G, which is well outside the 16G range of the ADXL345. Additionally, the Neurosky EEG headset was incompatible with the ARM architecture of the Raspberry Pi as its software was designed exclusively for x86 systems.

These challenges prompted changes in the hardware:

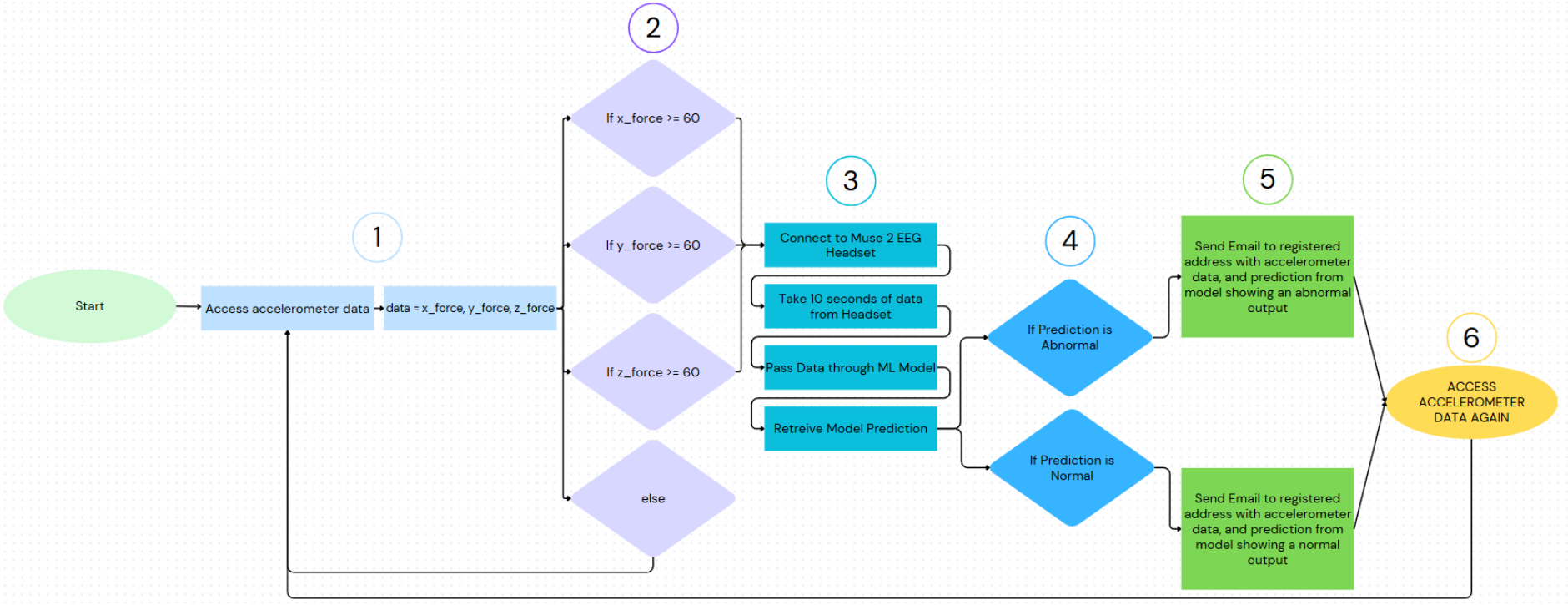
- I chose the ADXL375 accelerometer over the ADXL345 because the ADXL375 is more sensitive and capable of detecting forces up to +/-100G well above the concussion threshold of 60G.
- I switched from the Neurosky headset to the Muse 2 EEG headset, which can work with ARM architecture and the Raspberry Pi using MuseLSL2 to stream brainwave data for real-time analysis.
- To address size constraints, I replaced the Raspberry Pi 4 Model B with the Raspberry Pi Zero 2W, which offers a smaller form factor while maintaining sufficient processing power and memory. This change allowed the hardware to fit inside a wearable helmet.

The hardware needed to be portable, powerful, and affordable as the device must fit inside a wearable helmet while processing real-time sensor data. After evaluating several microcontrollers, I chose the Raspberry Pi Zero 2W due to its balance of processing power, memory, cost-effectiveness, and small form factor.

To measure impacts, I used the ADXL375 accelerometer because it is highly sensitive and capable of detecting forces well above the concussion threshold of 60G. For EEG monitoring, I used the Muse 2 EEG headset connected through MuseLSL2, which streams brainwave data for real-time analysis. The Raspberry Pi's GPIO pins ensure flexibility for future expansions, such as adding Bluetooth modules for wireless data transmission.

Software

Functional Code



Setting up the Code:

1.

Impact Detection:

The first priority was to detect impact forces. Without this, the system wouldn't function. To achieve this, I used the ADXL375 accelerometer library from Adafruit to gather impact data through the I2C bus from the accelerometer itself.

2.

Threshold Check:
The code continuously checks if the impact data exceeds the 60G threshold. If the force does not cross this threshold, the system remains idle waiting for a higher impact value.

3.
Trigger Data Acquisition:
Once the impact force exceeds 60G, the system triggers the acquisition of EEG data from the Muse EEG headset.
Data Collection:
The system collects 10 seconds of EEG data from the headset which is then passed through the machine learning model for analysis.
Prediction Processing:
The model processes the data to predict whether the player has sustained brain damage based on the EEG patterns and impact severity.

4.
Normal Prediction (No Brain Damage):
The model predicts that there is no brain damage from the impact
Abnormal Prediction (Possible Brain Damage):
The model predicts that there is brain damage from the impact

5.
Email
The device will email the emergency contacts registered in the headset with the impact force, model prediction, and the EEG graph of the player.

6.
Continuous Impact Detection:
After sending the email, the system returns to monitoring the accelerometer for any new impacts. If another impact exceeding the 60G threshold occurs, the process repeats, collecting new EEG data, processing it through the model, and sending updated predictions and alerts if necessary.

Software Challenges:

NeuroSky Mindwave Compatibility:
Initially, I faced difficulties gathering data from the NeuroSky Mindwave headset on the Raspberry Pi. The available software layers for the Mindwave headset were incompatible with the Raspberry Pi, which hindered progress.

Switching to Muse 2 EEG Headset:
After switching to the Muse 2 EEG headset, I encountered compatibility issues with the MuseLSL library, which failed to work on the Raspberry Pi despite being advertised as ARM-compatible. This software incompatibility delayed the data acquisition process.

MuseLSL2 Library Integration:
After extensive research, I discovered MuseLSL2, an alternative library compatible with the Raspberry Pi. This resolved the software issue and allowed for successful integration with the EEG headset.

Real-Time EEG Data Processing:
Developing algorithms to process EEG data from the Muse 2 headset in real time presented challenges. The system needed to accurately analyze brain activity for abnormal patterns indicative of a concussion, requiring precise and efficient data handling.

Machine Learning Model Integration:
The model, which was designed to analyze EEG brain data to predict concussion risks, needed to be seamlessly integrated with the real-time data acquisition system. Ensuring the model could quickly process the EEG data and output predictions for timely intervention was a significant challenge.

Model Code:

Training the Model:

```python
# Define the model
class MyFourConvModel(nn.Module):
    def __init__(self):
        super(MyFourConvModel, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=4, out_channels=16, kernel_size= 9, stride=2, padding=4)
        self.conv2 = nn.Conv1d(in_channels=16, out_channels=32, kernel_size=7, stride=2, padding=3)
        self.conv3 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=5, stride=2, padding=2)
        self.conv4 = nn.Conv1d(in_channels=64, out_channels=128, kernel_size=5, stride=2, padding=1)

        self.bn1 = nn.BatchNorm1d(16)
        self.bn2 = nn.BatchNorm1d(32)
        self.bn3 = nn.BatchNorm1d(64)
        self.bn4 = nn.BatchNorm1d(128)

        self.dropout = nn.Dropout(p=0.55)
        self.global_avg_pool = nn.AdaptiveAvgPool1d(1)
        self.fc1 = nn.Linear(128, 2)

    def forward(self, x):
        x = nn.ReLU()(self.bn1(self.conv1(x)))
        x = nn.ReLU()(self.bn2(self.conv2(x)))
        x = nn.ReLU()(self.bn3(self.conv3(x)))
        x = nn.ReLU()(self.bn4(self.conv4(x)))


        x = self.global_avg_pool(x)


        x = torch.flatten(x, 1)
        x = self.dropout(x)
        x = self.fc1(x)
        return x
```

For training the model, I used a combination of convolutional layers (CNN) for both feature extraction and classification. The training was conducted on my personal computer using an RTX 4060 8GB GPU, Intel i7 13700F CPU, and 32GB DDR5 RAM, which enabled fast model training. Each model took approximately one hour to train, and I was able to train up to four models simultaneously thanks to the RAM, allowing

me to compare different architectures and identify the most effective one. The dataset was split into training and evaluation sets, with the evaluation set used to assess model performance.

Initial Architecture and Dataset Split:
For training the model, I utilized a combination of convolutional layers (CNN) for both feature extraction and classification. The dataset was split into a training set and an evaluation set. The evaluation set was used to assess model performance. I trained the model on my personal computer using an RTX 4060 8GB GPU, which allowed for faster training times.

Initially, I used and trained models I found on the internet, however they were VERY slow to train. Chrononet (a model trained on a similar dataset) took four days to train and achieved an accuracy score of 50%. I learnt then that I needed to train my own models.

Training Multiple Models Simultaneously:
My models took approximately one hour to train. I trained up to four models simultaneously, which allowed me to compare different architectures and identify the most effective one.

Early Model Exploration:
Initially, I experimented with 2, 3, and 4-layer convolutional models. The 4-layer deep Convolutional Model yielded the best performance, achieving just over 70% accuracy. This became the starting point for further experimentation.

Fine-Tuning Hyperparameters:
Despite the improvements with the 4-layer model, the accuracy remained barely above 70%. I spent several months fine-tuning hyperparameters such as kernel size, stride, padding, and output channels. I also experimented with adding BatchNorm layers and using ReLU activation functions. After extensive fine-tuning, I managed to push the accuracy up to 72%.

Incorporating the Muse2 Headset's Full Capability:
Initially, I had been working with only 1 channel of data from the EEG headset. However, I later realized that the Muse2 Headset actually provided 4 channels. This opened up new possibilities for improving the model's performance. I retrained the model using all four channels of data, significantly improving the accuracy.

Achieving the Final Model Performance:
By retraining the model with data from all four channels, I was able to achieve a final accuracy of 76.32%. This marked a significant improvement over the initial models, and the results showed that the model was better equipped to handle the increased complexity and information provided by the additional channels.

Final Model Choice:
The 4-layer deep CNN with all four channels from the Muse2 Headset was ultimately chosen as the final model. Its performance surpassed previous iterations, and the fine-tuning of hyperparameters played a crucial role in enhancing its ability to make accurate predictions.

Modeling Challenges:

Existing Models and Accuracy Issues: The initial models I found on the internet achieved poor accuracy, around 50%, when working with only a single channel. I had to explore different approaches to improve the model performance, leading to the decision to make my own model for more accurate analysis.

Learning Deep Learning: I had to teach myself how to build deep learning models from scratch, particularly convolutional neural networks (CNNs). This was a steep learning curve, as I needed to understand both the theory and the practical implementation aspects. Despite extensive experimentation, the CNN model initially got stuck at around 66% accuracy. I tried using multiple CNN layers and after extensive testing, found that four layer models were the best for my use case. I tried two, three, five, six, and eight layer models before settling on four layers.

CNN Model Optimization: After months of fine-tuning hyperparameters and training strategies, I managed to push the model's accuracy from 66% to 72%. This process required careful attention to the model architecture and training parameters, as training the model took several hours and needed to be as efficient as possible.

Model Performance with Four Channels: When I finally gained access to four EEG channels via the Muse 2 headset, the model's accuracy improved significantly, reaching 76%.

Data:

Before deciding on the TUH EEG Dataset for this project, I explored various concussion-specific datasets. Although such datasets do exist, many are not publicly accessible. After reviewing several options, I found that the TUH EEG ABNORMAL/NORMAL CORPUS, which is the largest publicly available EEG dataset, was the most suitable for my needs. This dataset is publicly available and contains labeled instances of normal and abnormal brain activity, making it ideal for EEG classification tasks. It was not collected by me, but rather gathered using EEG sensors from a diverse population, with a roughly equal number of male and female participants.

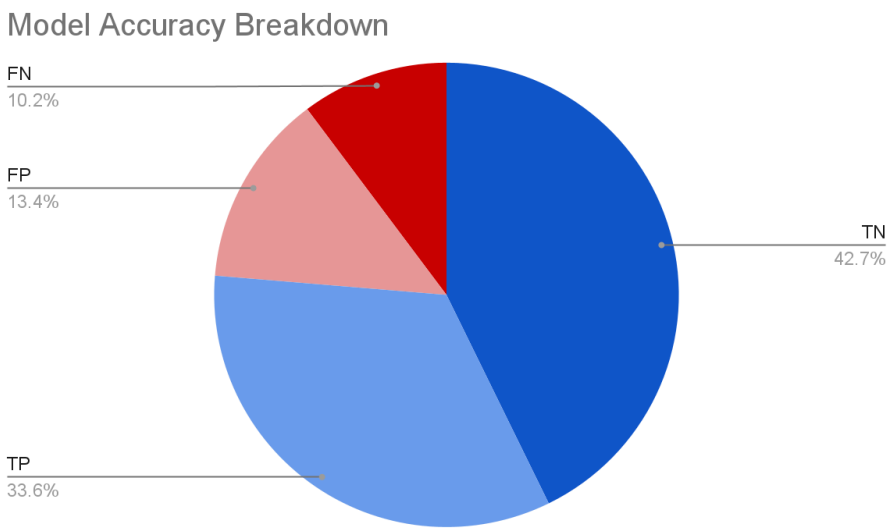
Model Accuracy:

This graph shows how good the model is at predicting abnormal or normal brain patterns - The data for the graph below was taken when I evaluated the model on its performance using the evaluation set.

True Negative (TN): Indicates that the individual was in good health, and the model correctly predicted their condition as fine.

True Positive (TP): Indicates that the individual was not in good health, and the model accurately identified their condition as not fine.
False Positive (FP): Occurs when the individual was in good health, but the model incorrectly predicted their condition as not fine.
False Negative (FN): Occurs when the individual was not in good health, but the model erroneously predicted their condition as fine.

Model Accuracy: 76.32% - The model correctly predicted which category each file would be most likely to fall into 76% of the time. TN and TP represent the correct predictions, which add up to 76.32%.



My goal for the model was to minimize FN as much as possible while maximizing TP and TN. I have used convolutional(CNN) layers to achieve this accuracy score of 76.32%. In trying to better this accuracy score, I created new architectures consisting of recurrent(RNN) layers, gated-recurrent units(GRU), and long short-term(LSTM) layers; however, these new models only gave me accuracy scores approaching 60%. In the future, however, I plan on using RNN-like models to improve the accuracy of the device as it is highly likely that the unsatisfactory results stem from my inexperience and lack of time with these layers.

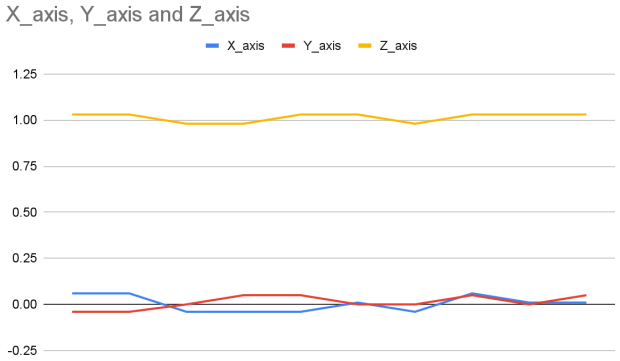
Device testing in Real Life:

To evaluate the device's accuracy, I tested the combined performance of the accelerometer, EEG headset, and deep learning model.

Test Cases:

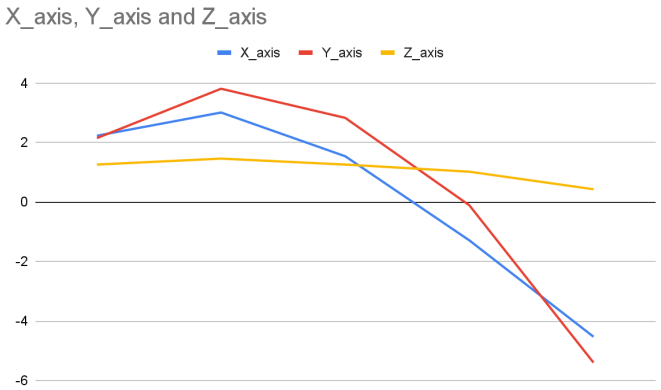
Case 1:
Scenario: The accelerometer does not experience any force, and the threshold is not crossed.
Expected Outcome: No data is requested from the EEG headset.
Explanation: This test ensures that the system does not trigger unnecessary data requests when no significant impact occurs. The threshold is set to 60G to demonstrate that no request will occur unless the threshold is exceeded.

```
X: 0.06 G, Y: -0.04 G, Z: 1.03 G
X: 0.06 G, Y: -0.04 G, Z: 1.03 G
X: -0.04 G, Y: 0.00 G, Z: 0.98 G
X: -0.04 G, Y: 0.05 G, Z: 0.98 G
X: -0.04 G, Y: 0.05 G, Z: 1.03 G
X: 0.01 G, Y: 0.00 G, Z: 1.03 G
X: -0.04 G, Y: 0.00 G, Z: 0.98 G
X: 0.06 G, Y: 0.05 G, Z: 1.03 G
X: 0.01 G, Y: 0.00 G, Z: 1.03 G
X: 0.01 G, Y: 0.05 G, Z: 1.03 G
X: 0.06 G, Y: -0.04 G, Z: 0.98 G
X: -0.04 G, Y: -0.04 G, Z: 1.03 G
X: 0.01 G, Y: 0.00 G, Z: 1.08 G
X: 0.06 G, Y: 0.00 G, Z: 0.98 G
X: -0.04 G, Y: 0.00 G, Z: 0.98 G
```

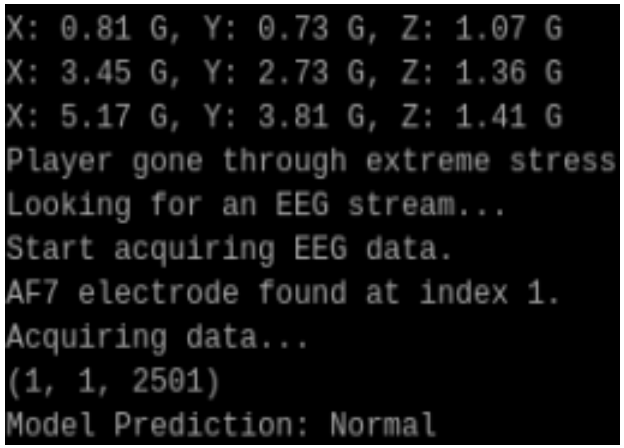


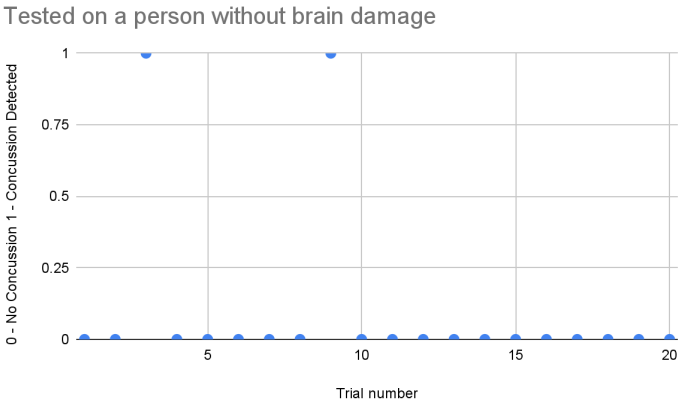
Case 2:
Scenario: The accelerometer experiences some force, but the threshold is not crossed.
Expected Outcome: No data is requested from the EEG headset.
Explanation: This test verifies that the system correctly ignores minor impacts below the threshold. The threshold is set to 60G to show that small forces do not trigger a data request.

```
X: 2.23 G, Y: 2.15 G, Z: 1.26 G
X: 3.01 G, Y: 3.81 G, Z: 1.46 G
X: 1.54 G, Y: 2.83 G, Z: 1.26 G
X: -1.20 G, Y: -0.11 G, Z: 1.02 G
X: -4.53 G, Y: -5.40 G, Z: 0.43 G
```

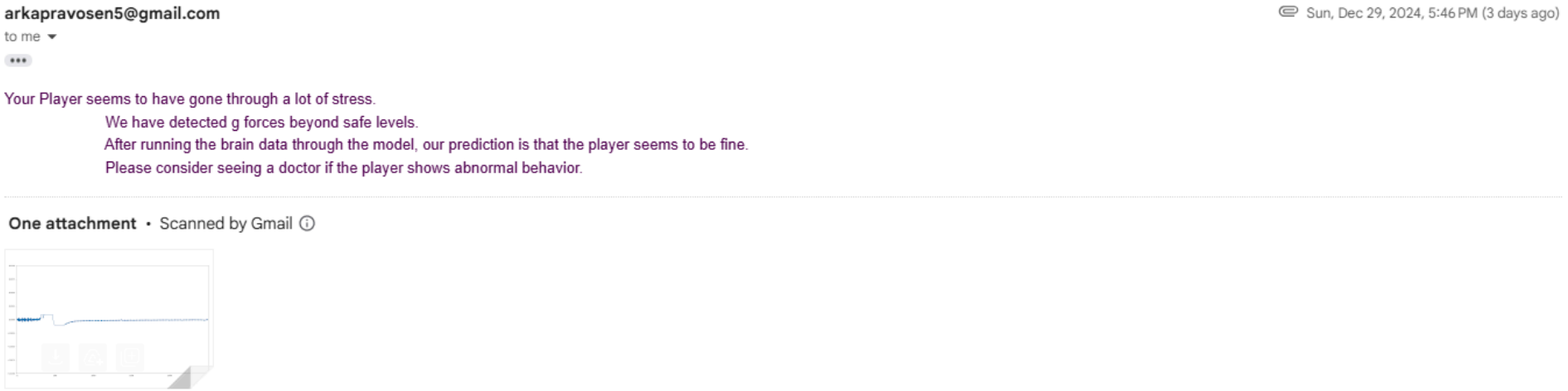


Case 3:
Scenario: The threshold is crossed, but the player shows no signs of brain damage.
Expected Outcome: Data is requested from the EEG headset, and the analysis indicates no abnormal brain activity.
Explanation: To simulate this scenario, the threshold is set to 5G (since physically achieving 60-90G is not feasible in this controlled test). This test confirms that the system can handle cases where significant impacts do not result in injury. This test was conducted 20 separate times to validate accuracy.





Email After Predicted Normal Brain Health:



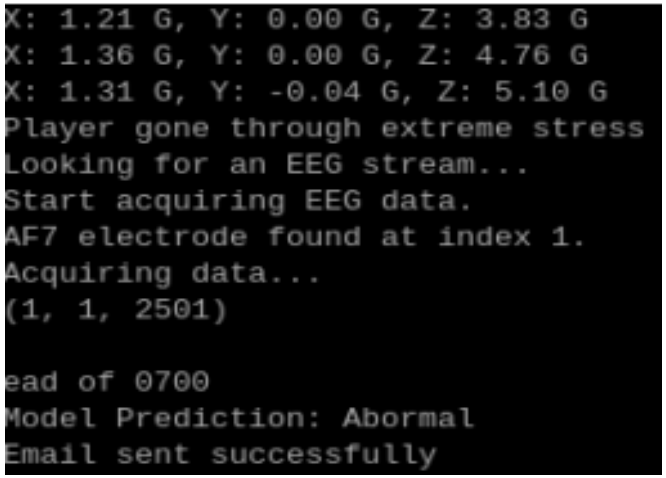
Case 4:
Scenario: The threshold is crossed, and the player shows signs of brain damage.
Expected Outcome: Data is requested from an EEG file, and the analysis detects abnormal brain activity indicative of potential brain damage. I am using a file with abnormal brain data to simulate what would have happened if someone with brain damage had put on the helmet.
Explanation: The threshold is set to 5G for the same practical reasons as Case 3. This test was conducted 20 times to validate accuracy. This test ensures the system responds appropriately to high-impact scenarios and correctly identifies signs of brain damage. Normally data would be taken from the headset and passed through the model using this code:

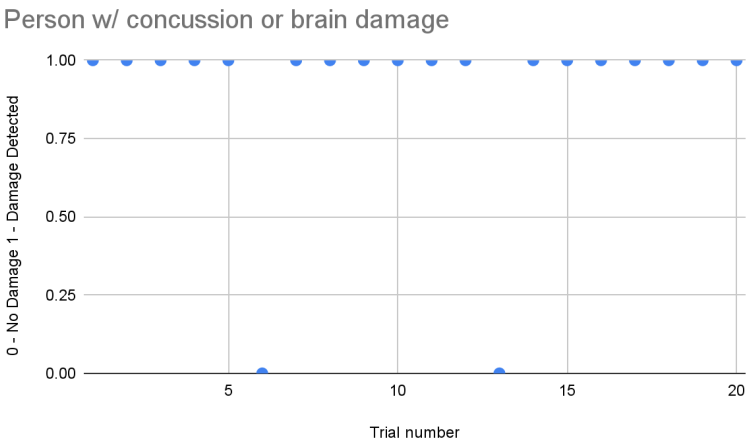
```
data_array = view()
print(data_array.shape)
data_array = torch.tensor(data_array, dtype=torch.float32)
with torch.no_grad():
    output = model(data_array)
    softmaxing = nn.Softmax(dim=1)
    output = softmaxing(output)
```

However because I am taking data from a file, I need to modify the code to be able to do so. I have kept everything the same but instead of “data_array” getting the value returned by the view function, I have it get the value from the file instead.

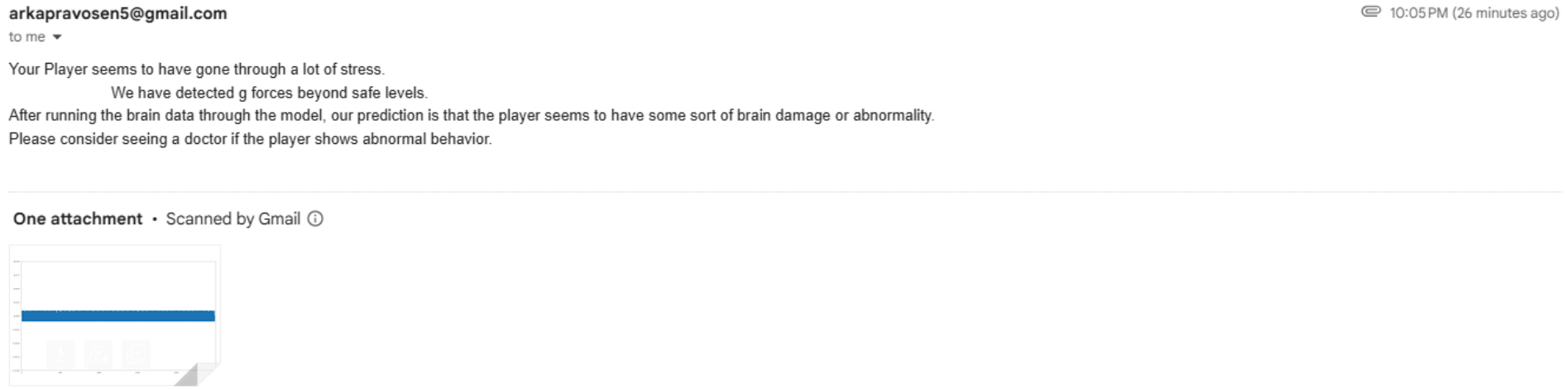
```
fif_file = "path_to_your_file.fif"
raw = mne.io.read_raw_fif(fif_file, preload=True)
data = raw.get_data()
data_array = torch.tensor(data, dtype=torch.float32)
with torch.no_grad():
    output = model(data_array)
    softmaxing = nn.Softmax(dim=1)
    output = softmaxing(output)
```

The first line has the path to the EEG file. The second gets the raw data from the file. The third line extracts only the data points from the raw data. The fourth line converts the data to a PyTorch tensor. After being extracted, the data is passed through the model to get a final prediction.





Email After Predicted Abnormal Brain Health:



Conclusion:

This project has demonstrated the feasibility of a wearable concussion detection system that integrates accelerometers and EEG sensors. By

combining data from the ADXL375 accelerometer and the Muse 2 EEG headset, the system successfully identified abnormal brain activity patterns linked to concussions. While the model achieved an accuracy of 72%, there is room for improvement, particularly in minimizing false negatives to ensure safer outcomes for users.

The testing process confirmed the device's capability to detect potential concussions in real time, making it a promising solution for sports and recreational activities. The accelerometer reliably measures impact forces, and the EEG headset provides critical brain activity data for analysis.

Future developments will focus on refining the machine learning models, incorporating advanced neural network architectures like RNNs, using a different headset to reduce the cost of the device, and improving the system’s hardware portability and affordability. With continued research and iteration, the NeuroGuard system has the potential to become a reliable tool for athletes, coaches, and medical professionals, helping to reduce the risk of long-term brain injuries and ensuring quicker medical interventions.

Bibliography:

Concussion's Impact on the Brain: Affected Areas and Long-Term Consequences
How Hard is Too Hard? Examining the Forces Behind Concussive Impacts | Complete Concussions
Long-Term Effects of a Concussion
https://healthcare.utah.edu/healthfeed/2023/11/concussions-how-they-can-affect-you-now-and-later
Sport’s £161m question – is playing on worth the risk of brain damage?
The Best EEG Headset In 2023