Backpack Buddy

Authors: Joon Cha, Aaron Li, Janet Li: Electrical and Computer Engineering, Carnegie Mellon University

Abstract—**Backpack Buddy is a backpack-based smart inventory system designed for on-the-go students. The system utilizes Bluetooth Low Energy tags attached to items to actively track what items are in the user's backpack. Unlike existing commercial products, which only focus on tracking individual items, Backpack Buddy manages collections of items in relation to the user's schedule, and notifies them when they are missing items for their next event.**

Index Terms—**Asset-tracking, Bluetooth Low Energy, Image recognition, Raspberry Pi Zero, RSSI, Smart inventory, Web application, Django**

1 INTRODUCTION

One of the most common struggles students face is trying to keep track of their items between all the classes, meetings, sports practices, or other events they may have. Students can lose things like their wallets, keys, water bottles, and even skateboards (as can be seen in the Facebook group "CMU Lost and Found"). It's clear that there needs to be a better way for students to keep track of their items with their busy schedules.

Existing solutions involve social groups like "CMU Lost and Found" – however, this relies on the kindness of strangers, the luck that your missing items are found, and the fact that students are only notified retrospectively after their item has been lost – and the Tile tags or Apple AirTags, where tags are attached to each item and are tracked individually. However, this often entails a large burden on the user and isn't manageable for large collections of items. Our project, Backpack Buddy, offers a better asset-tracking solution by focusing on *where* these items are stored to be moved around: the backpacks and bags of students.

Backpack Buddy is a smart inventory system that involves three components: (1) BLE tags on items to-be-tracked (the same technology that Tile uses in their tags), (2) A backpack-based scanner which keeps track of which tagged items are inside the backpack, and (3) A mobile-accessible web application which allows users to manage their items in relation to their schedules. This application will also have an image recognition component to streamline the item registration process.

Driven by our application area, we have specific design requirements. At a high level, the main requirements of our project are:

1. Detect what tags are in the backpack
2. Display the item list to the user and report of missing or list items

For our specific, quantitative requirements, the BLE tags should be easy to attach and unobtrusive, no larger than 40 x 40 x 10mm. The scanner should have a less than 1 second delay for tag detection per tag and be able to accurately detect 10 items within a 0.5-meter range (± 10cm) with 100% accuracy and last 18 hours without recharge while fully operating. For the phone application, the interface delay between when an item is added/removed to the backpack should be less than 3 seconds per item, and the item recognition built into the application should have at least 80% accuracy (i.e. out of the three names suggestion, the correct name should be one of the options at least 80% of the time).

2 DESIGN REQUIREMENTS

Based on the use case and application area for our product, we determined quantitative metrics which our system should meet. Whether or not our product meets these requirements will illustrate how effective our product is and demonstrate how good our work has been.

2.1 Hardware Requirements

For tag communication and detection, we require that the scanner have a delay of less than 1 second between when a tag is removed or placed into the backpack and when the system detects the tag's changed status. This requirement ensures that a user will have updated information about their backpack's contents in a prompt manner at all times. Additionally, the scanner must accurately detect and identify 10 tags placed within a 50 (± 10) centimeter sphere of the backpack device with 100% accuracy. Our system is designed to help users track their items, so it must itself be capable of accurately determining what items are in the backpack. We have chosen to test with 10 items, as that is the maximum number of major items that we believe a user will need to track (we do not expect users to tag individual pencils, for example). We have also chosen 50cm as the cut-off distance because it is the size of an average backpack, and we gave ourselves a ±10cm error range (a tag could be included up to 60cm away, and excluded up until 40cm away) as backpacks vary in shape and size, and therefore it is ultimately up to the user to ensure that there are no items near the backpack but not in the backpack when they use the system. Finally, the backpack system must last 18 hours without recharge while fully operating, as we have determined this to be the maximum amount of time that a student will spend away from their home before returning on a typical school day. We will test the latency for tag detection by placing a tagged item into the backpack and timing when the system updates the item's status as in the backpack. We will do the same for removing the item from the backpack, and time when the item's status is updated to being outside of the backpack. We will test the 10 items within 50 (± 10) cm by placing 10 items within the backpack itself and checking whether the item list produced by the system displays the ID's of all 10 tags correctly. To test the battery life of the backpack device, we will place 10 tags in the backpack to maximize the power consumption of the system, and we will also alternate between removing a tag from and replacing that tag into the backpack every 30 minutes. If the system is still on after 18 hours, then it will pass the test.

2.2 Machine Learning Requirements

During registration of items with our web application, the item recognition component of our system will suggest three names for each item based on the three highest ranked options produced by our CNN model. We require that, for 80% of items, at least one out of the three suggested names will be the correct category for that item. This requirement ensures that our item recognition component actually proves useful to the user. To test this requirement, we will input images from a dataset of 21 common items a student would carry and see what percentage of the images the model is able to correctly identify within its top 3 highest classifications. Initially, our requirement for item recognition accuracy was only 60%. However, after training the ML classification model on thousands of images, we discovered that the image recognition component is able to exceed our initial requirement of 60% accuracy. Thus, to further decrease the user's burden of manually typing all the item information when registering the item, and to improve our system, we have instead decided that 80% accuracy is a more appropriate requirement than 60% accuracy.

2.3 Web Interface Requirements

In addition to the scanner tag detection delay requirement, we also require that the web interface delay be less than 3 seconds from when the scanner reports a change in item status. To test this requirement, we connected the web application to our backpack-mounted system, and placed and removed items from the backpack. We then timed how long it took after the scanner item list is updated for the web interface to update. The total update delay will be the sum of both the scanner tag detection delay and this web interface update delay.

We also required our web interface application to be able to manage scheduling of up to 25 weekly events. We chose this requirement of 25 weekly events because that is the maximum number of events a student may need to attend per week. To test this requirement, we created 25 events through the web application. We then closed the web page and reopened the web page and checked that all 25 events were still displayed correctly on the web interface.

Finally, we required that notifications must appear at the correctly specified notification time for each event, notifying the user if they are currently missing any items in their backpack that are required for their event. We tested this by creating events with notification times on trials where:

1. All items required for the event were already in the backpack
2. The user was missing at least one item from their backpack required for the event

and checking that each notification properly notified the user of the status of their items for each event at the correct notification time.

Our justification for this requirement is that it's one of our critical requirements for our use case – that users are able to keep track of items in their backpack for their events, and that our system will notify them should they be missing any necessary items.

3 ARCHITECTURE OVERVIEW

There are three main modules to our project: the backpack-mounted Raspberry Pi scanner system, the web application to manage items and events, and the machine-learning-based image recognition module for item registration.

The processes involved on the hardware side are illustrated in Figure 1. First, The BLE tags transmit their MAC address information via Bluetooth to the scanner, which will be an RPi Zero attached to a backpack or bag. The RPi Zero uses a distance-pruning algorithm (which we wrote ourselves) to determine which MAC addresses are coming from tagged items within our 50±10cm range, to determine whether or not an item is actually inside the backpack. The list of items inside the backpack is then sent to the Python-based Bluetooth GATT server which is also running on the RPi Zero. This server handles connecting to the web app, and sends the list of items to the web app once connected.

The web application design and its interaction with the item recognition component are illustrated in Figure 2. On the software side, our Web Application receives the list of MAC addresses from the RPi Zero and uses it to create the checklist interface where users are able to track which items are currently inside their backpack. This list of MAC addresses also helps feed information about which items are missing for a specified event. As shown by our entity-relationship diagram (see Figure 3), every user has multiple events and multiple tagged items, and each event contains a subset of all tagged items. This means that multiple or no items are associated to each event that the user has, as shown by the intersection table "EventItems". Users are able to register each MAC address as an item and assign it a readable name via the item registration process. This process also features the image recognition component, where users are able to take a photo of their item, and have 3

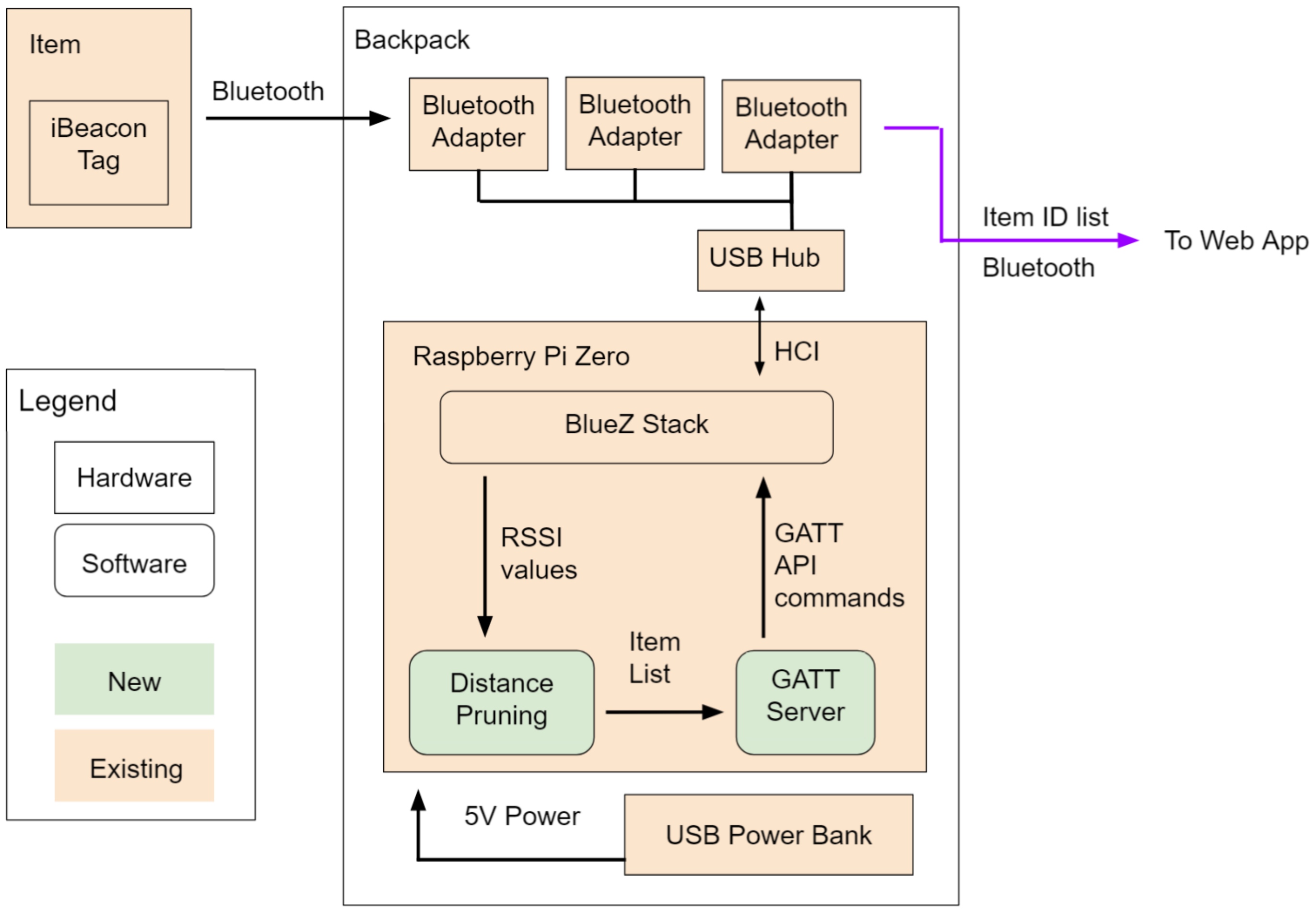


Figure 1: Hardware Block Diagram

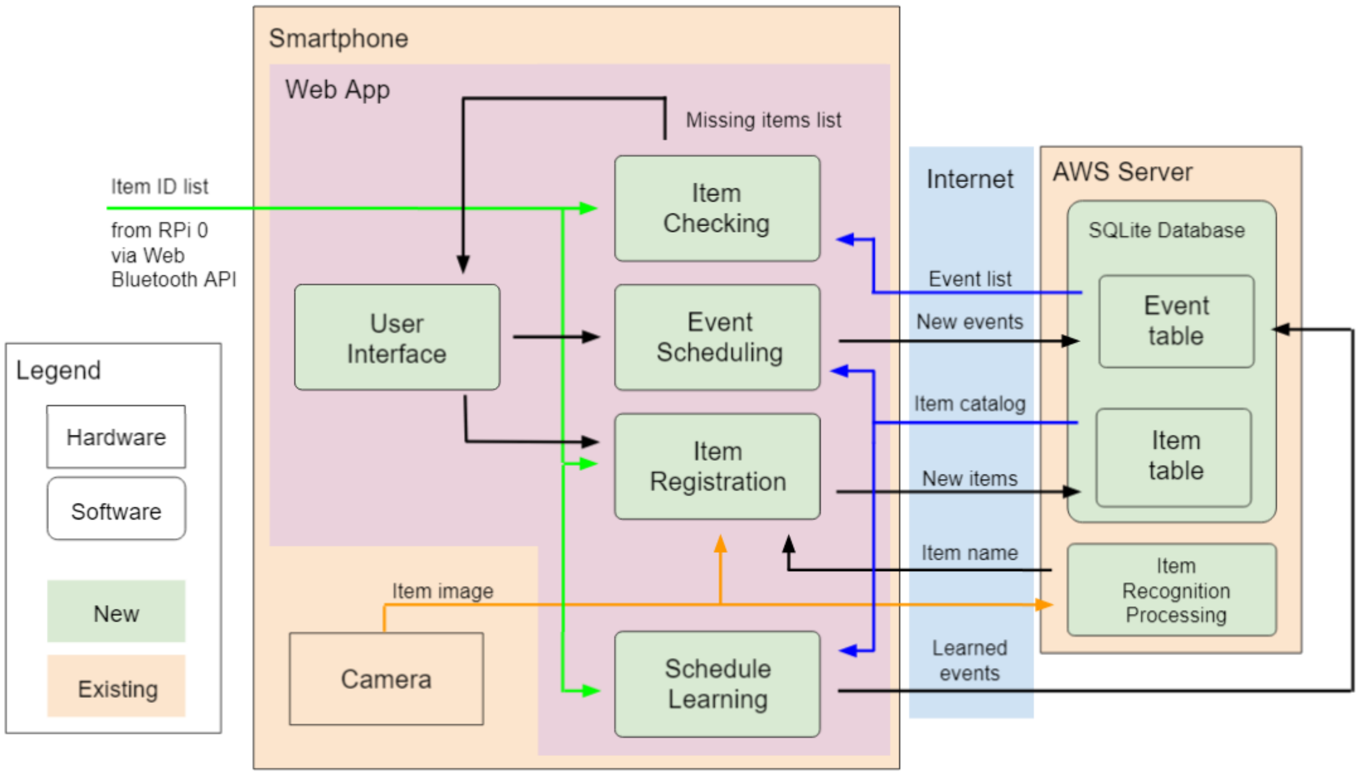


Figure 2: Software Block Diagram

names suggested for that item which they can use, instead of manually typing out the name of the item. In addition to registering items, the user will create and schedule events in the calendar. The user then assigns registered items to each of these events. Alternatively, the user can utilize the schedule learning feature to automatically assign items to their events, based on their activity.

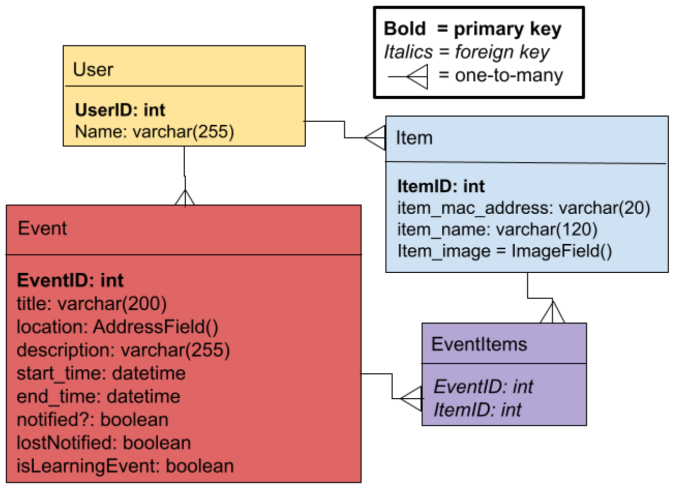


Figure 3: Entity-relationship diagram

To learn the user's schedule, the schedule learning feature collects data about which tagged items are inside the backpack at what times. By correlating these times with events the user has scheduled, the schedule learning system can determine which items are most often located in the backpack during each event. After obtaining data in the form of timestamps and item lists, the schedule learning system automatically assigns items to the events based on which items appeared with at least 85% appearance frequency. The system continues to refine the item lists as the user attends more events. This automatic assignment of items saves the user the burden of having to assign items to events manually. Our image recognition component is built into the web application during the onboarding process when users are first registering their tagged items. Using a phone's camera, the user can take a picture of the item with a physical tag. Figure 4 describes the image recognition and classification process using Python. A Convolution Neural Network (CNN) model, which is trained with scraped images from a search engine, will identify each item. These scraped images are augmented by image processing techniques to increase the size of the training data. Through the CNN model, the classification of the input image is given. The three highest ranked classifications will become the top three suggestions for the item's name.

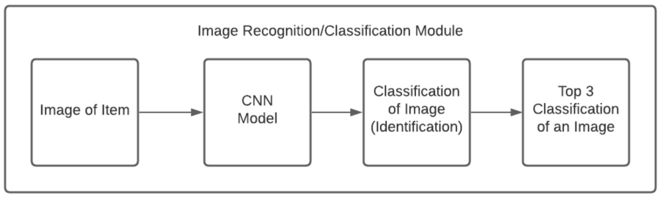


Image Recognition/Classification Module
Image of Item
CNN Model
Classification of Image (Identification)
Top 3 Classification of an Image

Figure 4: Overall Image Recognition Block Diagram

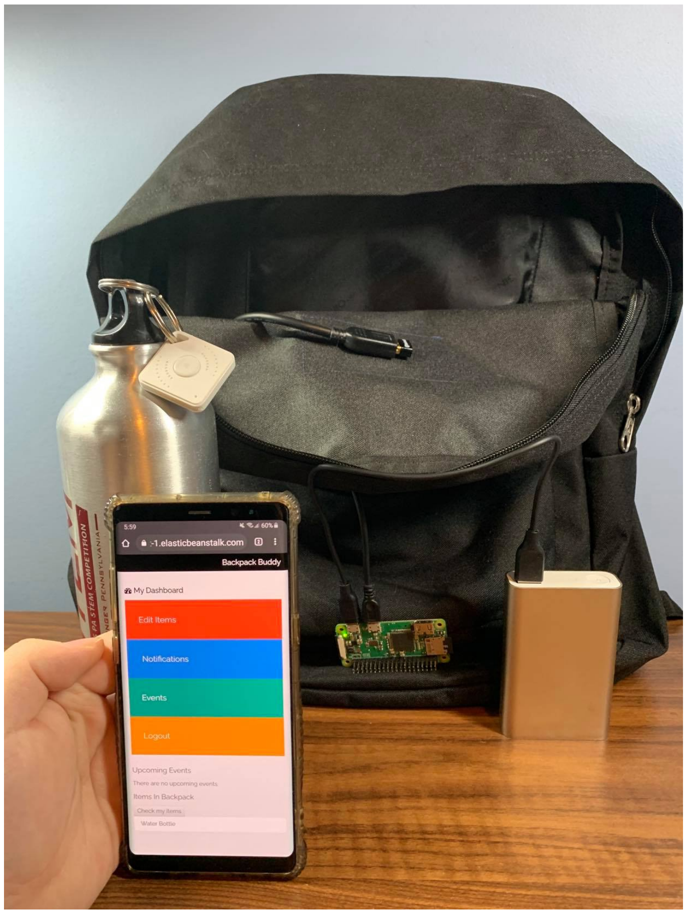


Figure 5: Overall System

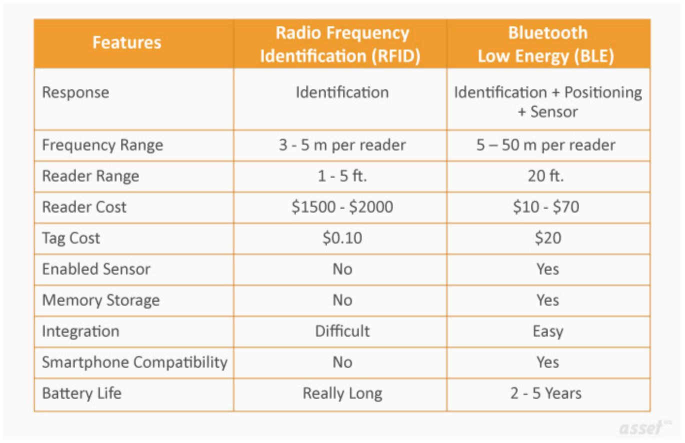
4 DESIGN TRADE STUDIES

4.1 Controller Comparison

We chose to use a Raspberry Pi Zero W for Backpack Buddy rather than other alternatives, such as an Arduino or an FPGA. This was because writing a Bluetooth communication protocol program for either the Arduino or FPGA was infeasible for our project. On the other hand, the Raspberry Pi Zero W runs on the Linux-based Raspbian operating system. We can use the built-in BlueZ Bluetooth protocol stack for Linux to handle Bluetooth communications, including scanning for tags and connecting to the user's smartphone. We chose the Raspberry Pi Zero W over other Raspberry Pi models for its small form factor, low power consumption, and built-in WiFi/Bluetooth antenna.

4.2 BLE vs. RFID

Once our team decided to build a smart-inventory system, we had to compare existing asset-tracking technologies. Two of the most important considerations we made were budget constraints and integration with other components of our system, such as the web application. As shown in Figure 6, BLE best fits our needs in terms of budget and integration, particularly for smartphone compatibility [13]. Although RFID tags are quite cheap, their reader costs far overextend beyond our allotted budget (1 would cost anywhere from $800-$1600, which much overextends our allotted budget). Individual BLE tags are more expensive than RFID tags, and also have a limited battery life (2-5 years), however the tags can be read with any common Bluetooth scanner, and BLE technology is already very commonly used in many asset-tracking commodities such as Tile. With its simple integration, smartphone compatibility, and budget-friendly reader cost, BLE is the correct asset-tracking technology for our project.



Features	Radio Frequency Identification (RFID)	Bluetooth Low Energy (BLE)
Response	Identification	Identification + Positioning + Sensor
Frequency Range	3 - 5 m per reader	5 – 50 m per reader
Reader Range	1 - 5 ft.	20 ft.
Reader Cost	$1500 - $2000	$10 - $70
Tag Cost	$0.10	$20
Enabled Sensor	No	Yes
Memory Storage	No	Yes
Integration	Difficult	Easy
Smartphone Compatibility	No	Yes
Battery Life	Really Long	2 - 5 Years

Figure 6: BLE vs. RFID

4.3 BLE Distance Analysis

To determine which items are located within our backpack, we are using a distance-based gating algorithm. The approximate distance of each item to the Raspberry Pi is calculated from the received signal strength indicator value associated with the item's Bluetooth signal. To do the conversion from power level in decibels of milliwatts (dBm) to meters, we are using a formula published by David Young in a blog post about contact tracing for COVID-19:[18]

$$d = 10^{\frac{p-s}{10n}} \tag{1}$$

In this equation, d represents the distance in meters, p is the measured power at 1 meter distance in dBm, s is the received signal strength in dBm, and n is a calibration constant which represents how quickly the signal decays in air. The constant p can be experimentally determined by averaging the signal strengths of multiple beacons at 1 meter distance from the Bluetooth sensor. The constant n can then be determined by measuring the signal strength at varying distances from 0.5m to 2m in 10cm increments, and fitting the curve using least-squares.

Bluetooth distance estimates are not accurate enough to provide centimeter-level precision, and the precision worsens as the distance between the beacon and Bluetooth sensor increases. For our purposes, we do not need precise distances, and the distance between the beacon and Bluetooth sensor (the Raspberry Pi Zero) is short (less than 1 meter) for the beacons whose distance we care about. Therefore, Bluetooth distance estimation is suitable for our application and eliminates the need to use triangulation or other localization techniques to determine which items are in the backpack.

4.4 Web App vs. Native Phone App

In both our proposal presentation and design review presentation, we planned on developing a phone app for Android phones in Kotlin using Android Studio. We chose to develop specifically for Android devices for ease of testing, as all three team members use Android phones. We chose Kotlin over Java for its coroutines, code safety, and other features that would be advantageous to our project.

However, we quickly ran into issues with Android Studio. Our two main concerns were the clunkiness of collaboration on Android Studio as well as the numerous errors with Gradle, Android Studio's builder.

Since our main focus was building our MVP as quickly in preparation for the interim demo, we decided that the best course of action would be switching to developing our interface application as a web application. Not only does this streamline collaboration between our team members, it also offers more accessibility from more platforms instead of Android phones.

4.5 Integrated Device

Our system incorporates a web app loaded on the user's phone to display the items present inside of the backpack. Another potential approach was to have an integrated LCD panel in the backpack which would display the items present, rather than a web app. The panel would have a set of arrow and control buttons next to it to enable navigation and configuration of the system. However, this integrated system would have a number of disadvantages versus a smartphone-based app.

First, the usability of our system would be reduced with an integrated system. With a smartphone app, the user can easily pull their phone out and quickly view what items are in their backpack. Additionally, users are more accommodated to navigating and using smartphone apps, versus a novel integrated system where the user would have to learn how to use the arrow keys to navigate the options on the LCD panel. Although touchscreen LCD panels are available, they are more expensive than non-touchscreen LCD panels, and often have worse touch detection than common smartphones. Since most users already own a touchscreen in the form of their smartphone, it was more sensible for our system to integrate into the smartphone rather than have its own dedicated touchscreen LCD panel. Moreover, it would be difficult to design an integrated system that would allow the user to interact with the system while wearing the backpack. Such a design would require having the LCD panel and controls on a separate arm that would extend from behind the user (where the backpack is) to the front. Having a smartphone-based app means the user can simply look at their smartphone while wearing the backpack and still have view and control of the system.

Second, the smartphone app enables more features than an integrated system. The smartphone app allows the user to schedule events and register new items which are tasks that would be difficult to do with an integrated system. Additionally, most smartphones have built-in cameras, so a smartphone app allows the user to take images of new items for the registration process. To implement this functionality on an integrated system, we would have to include a camera in the backpack, which would likely be lower quality than a smartphone camera and increase cost. Smartphones usually have cellular data, meaning that the app will have constant access to the cloud server where the events and items are stored. Although an integrated system could have WiFi access, the integrated system's internet access would be cut as soon as the user left their home WiFi network, rendering the system unable to communicate with the cloud server while the user is travelling.

Given these factors, we decided to use a smartphone-based web app as the user interface for our system, rather than having a system fully integrated into the backpack.

4.6 Image Recognition Model

For the image recognition algorithm, a suitable machine learning model needs to be trained. Among various machine learning models, including Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression, we chose to use a Convolutional Neural Network (CNN) to accomplish our image classification.

CNN's are the best choice for image classification for two main reasons. First, whereas other machine learning models first require a separate step of feature engineering before the model can be trained, CNN's automatically extract features from the image datasets and therefore do not require feature engineering. Using the learned features, CNN's use multiple layers extract the feature weights through downsampling and convolution.[14] Finally, the last prediction layer uses the feature weights to output the classification result of the image.

The second reason the CNN model is superior to other models is that CNN's are scalable for large datasets. In order to produce an accurate classifier, the model needs to be trained on a large number of labeled student items. Normally, the large number of images leads to more parameters in a neural network, which makes the training of the model exponentially more computationally heavy. However, the multiple convolution filters present in the convolution layers of a CNN effectively reduce the dimensions of each the image, reducing computational load. Even though the image dimensions are reduced, the images still retain the information from the original image, resulting in efficient but accurate image classification.[14]

4.7 Image Recognition Datasets

To train the CNN model for the image classifier, the dataset of images needs to be generated. In machine learning, we need to increase the size of the training data to achieve higher accuracy. One option of choosing an image dataset is to use a pre-existing dataset such as ImageNet. ImageNet is used for various CNN research papers and is an appropriate dataset for the image classification task. However, since we only need to identify 21 common items that students would carry in their backpacks, a dataset containing a variety of images will decrease the classification accuracy for the item recognition. Therefore, we decided to scrape the images of student items from the search engine, DuckDuckGo, using an Image Dataset Tool (IDT) image scraper library.

There are several reasons that the IDT library is superior to other software libraries. First, the library is designed around creating datasets specifically for training machine learning models such as the CNN model for image classification. IDT supports separation of training data and test data, which prevents the model from training on test images and potentially skewing the test results.[11] Second, the library supports standardization of the images, as images from search engine results are of different sizes and aspect ratios. Using the IDT library, images can also be downscaled and compressed for more efficient storage on our AWS server while we are training the CNN model. Finally, the library allows for multiple searches with different keywords. For example, when searching images for a reusable water bottle, a user will search it by typing different keywords such as “water bottle,” “reusable water bottle,” or “plastic bottle” for the same results. The IDT library enables the user to put multiple keywords so that the user can collect various images to better train the machine learning model.

5 SYSTEM DESCRIPTION

The hardware portion of our system consists of three parts: 10 iBeacon tags attached to individual items, a backpack-mounted Raspberry Pi Zero W with a rechargeable battery and power controller, and the user's smartphone. The control flow for our system starts with the iBeacons. Each iBeacon tag broadcasts its UUID over Bluetooth every 500ms. Then, the Raspberry Pi scans for tag UUID's and determines which tags are close enough to be considered inside the backpack based on the received signal strength of each broadcast. The Raspberry Pi then transmits the list of UUID's of the tags inside the backpack to the user's smartphone via a Bluetooth connection. Finally, The user's smartphone handles events, item information, and notifying the user when they are missing items.

5.1 iBeacon Tags

The iBeacons used by our system are the H1 Beacons from Moko Technology Ltd.[10] These beacons are based on the NORDIC nRF52 chipset and have adjustable transmission power and broadcast intervals. For our project, the H1 Beacons will be set to transmit at the lowest power level (-12 dBm) and longest interval (1000ms) available. This provides the maximum possible battery life for the beacons, which is approximately 15 months as specified by the datasheet. The dimensions of the H1 Beacons are 32.5 x 32.55 x 7mm with a weight of 7.8g. This is small enough and light enough to be securely attached to most user items. The beacons also feature a keyring hole, which allows the beacon to be attached via a keyring to an item if the user so desires. The beacons also have flat backs, and can be attached via double-sided adhesive tape to items for which a keyring is not suitable.

5.2 Raspberry Pi

The Raspberry Pi Zero in our system is powered by an Anker 10000mAh portable USB power bank. The user will be responsible for plugging in the power bank to charge every night so that the power bank will have full charge the next day. RPi Zero is running the Raspbian Lite operating system. This OS only has a command-line interface instead of a graphical desktop, to reduce the computational load on the Raspberry Pi. At startup, the Raspberry Pi runs two Python programs: the distance pruning algorithm, and the Python Bluetooth GATT server. The distance pruning algorithm utilizes a modified version of the Python aioblescan package created by François Wautier.[16] Using this package, the algorithm continuously scans for Bluetooth beacon advertisement packets. For each packet, the distance to the beacon which sent the packet is determined by converting the received signal strength indicator (RSSI) value into a distance using Equation 1 found in section 4.3. The constant n in the equation is set beforehand via calibration tests done with power level measurements of multiple tags at a set one meter distance. The distances are averaged over an interval of 3 seconds, and any beacon whose average distance over this interval is within 0.5m is included in the item list. The algorithm sends this item list to the second program, the Python Bluetooth GATT server, once every second. The Python GATT server is based off of an example server created by Github user Douglas6 to report Raspberry Pi CPU temperatures.[8] The GATT server advertises a GATT Service called *ItemListService*, which is discoverable by Bluetooth-enabled devices. Upon receiving a request to connect, the GATT server produces the necessary packets to establish a Bluetooth GATT connection. As a BLE protocol, the GATT connection does not require an pin code verification to connect. Once connected, the *ItemListService* has a Bluetooth "characteristic" named *ItemListCharacteristic*. This characteristic has a string value, with the string being a semi-colon delimited version of the full item list. Thus, by reading this characteristic, the web app instance running on the connecting device is able to retrieve the item list off of the Raspberry Pi Zero via the GATT server. The item list is transmitted once per second via this method, and the web app updates its stored version of the item list accordingly.

5.3 Smartphone Web App

Our web application was built using the Django framework and hosted on the AWS Elastic Beanstalk service. We used the Web Bluetooth API [17], which allows us to communicate with other Bluetooth devices via JavaScript, to receive the list of MAC addresses of items currently inside the backpack from the RPi Zero. This API retains the full Bluetooth functionality of a native phone app, including requesting and connecting to nearby BLE devices, reading/writing Bluetooth characteristics & descriptors, and receiving GATT notifications when devices get disconnected. The web application uses this list of MAC addresses to create the checklist interface, where users can track which items are currently inside the backpack.

The web application also manages (allows for the creation, editing, and display of) events and displays them on a calendar-like interface. Code was referenced from Huiwen's blog post on a Django Calendar system.[6] By design of our system, users will have multiple events in their schedule, where each event contains a subset of items from the entire list of tagged items (i.e. each event can have multiple or no items associated with it). This is represented as a ManyToMany relationship between Events and Items. Users first create events manually in the web app. Then, in a separate interface, the users register their items and assign item names to tag MAC addresses. The CSS used for this step is referenced from Petia's CodePen project for a MultiStep Form.[3] The user is then asked to take an image of each new item. This image is then sent to the image recognition algorithm, which suggests a name for the item to the user. The user can choose to either accept the suggested name or manually override the name. Once the items and events are created, the user can choose to either assign items to events manually, or have the schedule learning feature automatically assign the items to events. All events, items, and assignments of items to events are then stored on the AWS server database for the web application to retrieve later.

In order to make our web application mobile-accessible, we had to make the server HTTPS compatible, as the Web Bluetooth API only enables Bluetooth connections for secured websites. To do so, we generated an SSL certificate locally using openssl.[9] We then used the AWS Certificate Manager to upload this certificate to AWS, then configured the HTTP listener of our server to use the certificate to support HTTPS connections. Finally, we then installed the SSL certificate on the smartphone we used for testing. To deploy our web application to the public, we would need to obtain a true SSL certificate issued by a certified authority, however obtaining one would have added cost to our budget and was unnecessary for our small-scale project.

5.4 Notifications

Our system sends notifications for two scenarios – missing items and lost items – using web push notifications. For our project, we used the Django-Webpush package to send web push notifications.[7]

For missing items, the system scans through all events tied to a user. If the notification time of an event is less than or equal to the current time and if the event has not yet been notified for, the system will begin drafting a notification to send to the user. Two scenarios exist for the kind of notification sent:

1. At the time of notification for an event, the user already has all the necessary items for the event in their backpack
2. At the time of notification for an event, the user is missing at least one item necessary from their backpack that is necessary for the event

In the first scenario, the system will simply send a notification to the user letting them know of their upcoming event and that they already contain all the necessary items within their backpack. In the second scenario, the system will send a notification to the user notifying them of their upcoming event as well as a list of all items missing for that event.

For lost items, the system scans through all events tied to a user. Both at the start time of an event and the end time of an event, the system takes a snapshot of the item list (i.e which items are currently inside the backpack at that time). If the list of items at the end of the event is shorter than the list of items at the beginning of the event, the system notifies the user, letting them know that they might have left an item behind at an event.

5.5 Schedule Learning

An additional feature of our software component is the Schedule Learning feature, which automatically assigns items to events. This is accomplished by continually taking snapshots of the user's item list (i.e. the items currently inside their backpack) at various timestamps. When a user chooses to opt an event into schedule learning, the system begins scanning through all the snapshot pairs. If any pair has a timestamp that falls within a schedule learning event's time window (i.e. between the start time and end time), then the items associated to that time stamp have their counter for this event incremented by 1. The system continually assigns items with the counter values greater than 85% of all counter values for a specific event. The models used for schedule learning can be seen in the pseudo-code below:

```
class TimeItemListPair:
    user = ForeignKey(User)
    timestamp = datetime
    item_list = ManyToMany(Item)

class ItemCounter:
    event_parent = ForeignKey(Event)
    counter = int
    item_parent = ForeignKey(Item)
```

The automatic assignment of events reduces user burden by eliminating the initial registration and manual assignment of items to events.

5.6 Image Recognition

Our image recognition was primarily developed using Python. We chose Python because it is the most commonly used language in machine learning development and can be easily integrated with the web application, also written in Python. Within Python, the CNN model was built using the Tensorflow and Keras library. Among various CNN models, the image recognition module used the VGG16 model provided by the Keras library.[15] Code for the CNN model that specifically used the VGG16 model was referenced from Iftekher Mamun's image classifier model.[1] The overall CNN model was trained on an AWS server.

The CNN model was trained using a dataset of images from the search engine DuckDuckGo scraped with an Image Dataset Tool (IDT) library. Through the IDT library, images of the 21 identified student items were collected. 250 images per item were collected for the training data and 150 different images per item were collected for the test data. The images used for the test dataset were scraped and saved in different directories for testing purposes. Then the training data images were augmented with image processing techniques such as image distortions and mirroring to create up to 1000 images per item (from 250) using Python and MATLAB. This image augmentation was performed to create multiple variants of a single image. This also accounted for the user taking distorted or slanted photos of items during registration. Additionally, validation data images of 150 images per item, which were also separate from the training and test dataset, were collected. The main purpose to add the validation dataset was to evaluate the performance of a classification model and to tune the hyperparameter (number of epochs to train the model) from the validation accuracy. The training image was then resized to 224x224 pixels to feed into the VGG16 model. When the image of any size is inputted from the web application, the image is then resized to 224x224 pixels to fit to the VGG16 CNN model, and the CNN model classifies the image into one of 21 classes of items. The model outputs the top 3 highest matching classifications and communicates these to the web application. The user can select the correct classification result or manually input the information if the classification results are inaccurate.

6 TEST & VALIDATION

Following the tests described in the design requirements, we tested our final product to determine whether or not it met our specifications. Each component was tested separately, as none of the tests tested the entire system at once. A summary of each of the requirements and whether or not they were met can be found in Figure 11 in the appendix. Overall, our system met the requirements on the software side, however the system fell behind in latency and item accuracy on the hardware side. With more time, our system could be improved with more sensors (Bluetooth adapters), a more powerful processor (Raspberry Pi 4 instead of Raspberry Pi Zero), and further development of the distance pruning algorithm to use advanced techniques. For example, one technique we wanted to utilize was Kalman filtering, as demonstrated by Wouter Bulten in his project which also attempted localization using RSSI values.[5] However, we did not have the time to refine the filter enough for it to perform better than simple averaging of RSSI values.

6.1 Results for Image Recognition Module

Our requirement for the item recognition accuracy was 80%. We tested the accuracy of image recognition module with 3150 test images. The test images were randomly collected from the original pool of student item images, with 150 images collected for each of the 21 categories. The item recognition accuracy was calculated by the following equation:

$$\text{Accuracy}(\%) = \frac{\text{Number of correct label predictions}}{\text{Total number of test images}} \times 100$$

Specifically, the number of correct label prediction was incremented when the correct naming of an image was found out of top 3 highest classifications. After running the prediction of the CNN model and comparing those predictions with the actual labels of the student item images, we found that the CNN model achieved 88.08% accuracy, which meets our item recognition accuracy requirement.

To clearly show the performance of the image classification model and to provide a general overview of the classification results for 3150 test dataset, the confusion matrix is shown in Figure 7. The purpose of the confusion matrix is to compare the true labels with the labels predicted by the machine learning model. Therefore, the confusion matrix provides a holistic view of how well the machine learning classification model performs.

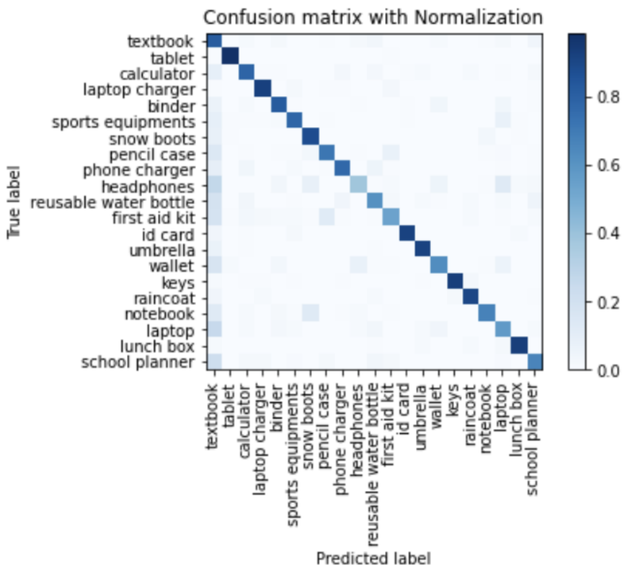


Figure 7: Confusion Matrix

Figure 7 shows the normalized confusion matrix of our CNN model, providing the percentage of the classification of the items. The x-axis of the confusion matrix represents the labels predicted by the CNN model, and the y-axis of the matrix corresponds to the actual label of the test images. Therefore, the diagonal from the top left corner to the bottom right corner represents the correct image classification percentages. From the heatmap on the right of the confusion matrix, we see the model correctly classifies 21 student items as intended greater than 80% of the time.

After implementing the CNN model, we also optimized the number of epochs to train the model to result in the highest classification accuracy. Therefore, we utilized a validation dataset separate from the test dataset to evaluate the performance of a classification model and tune the hyperparameter (number of epochs to train the model). For machine learning models, more epochs to train the model correlates to higher classification accuracy and lower loss. However, more epochs means the model can overfit to the training dataset. In other words, as the training time increases, the model learns more of the details of the outliers and the noise in the training dataset, therefore resulting in lower classification accuracy.

To find the number of epochs to train the model that result in the highest classification accuracy, we trained the CNN model for a total of 50 epochs, and monitored the validation accuracy over the epochs.

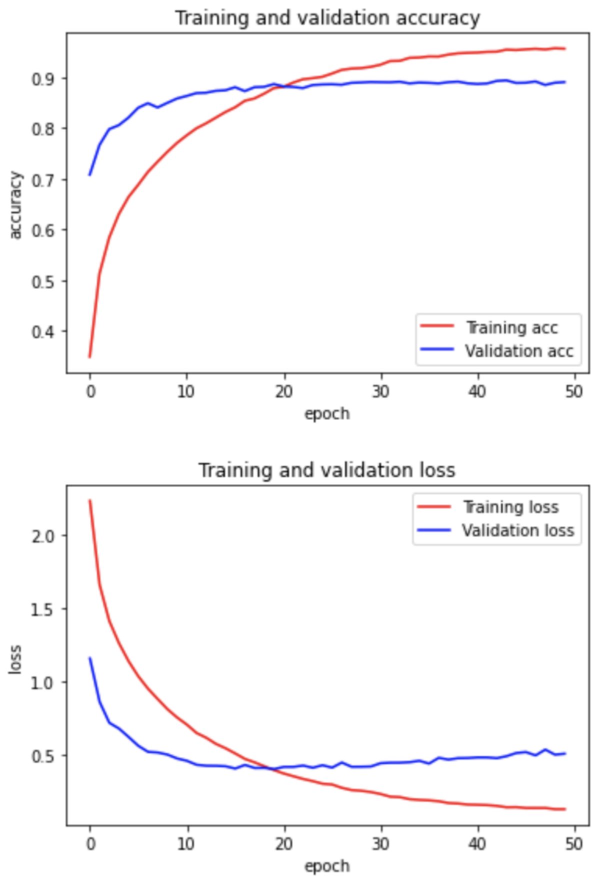
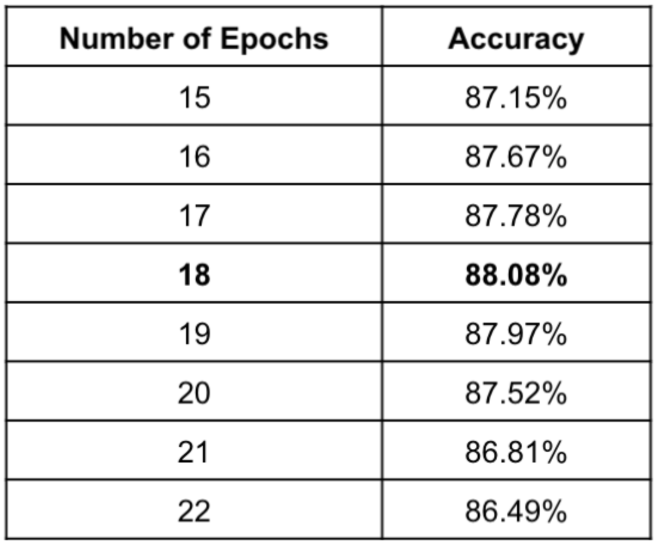


Figure 8: Training and Validation Accuracy and Loss of Image Recognition Model over 50 Epochs

From Figure 8, we see that the training and validation accuracies tend to increase as the number of training epochs increases, and that the training and validation loss decreases as the number of training epoch increases. However, we see that the validation accuracy converges and the validation loss actually increases in the epoch range of 15-22 epochs. This means that the model starts to overfit in this range, and the training needs to be stopped around 15-22 epochs. To further refine the number of epochs to train for, we calculated the accuracy of the CNN model with the test image dataset for each of the number of training epochs from 15 to 22.



Number of Epochs	Accuracy
15	87.15%
16	87.67%
17	87.78%
18	**88.08%**
19	87.97%
20	87.52%
21	86.81%
22	86.49%

Figure 9: Test Accuracy of Image Recognition Model for 15-22 Epochs

Figure 9 shows the test accuracy from 15 epochs to 22 epochs. From the table, we find that the test accuracy increases from 15 to 18 epochs, with 18 epochs reaching the maximum test accuracy. After 18 epochs, the test accuracy starts to decrease. Therefore, we decided to train the model for 18 epochs, which resulted in 88.08%, the maximum classification accuracy.

6.2 Results for Item Delay Testing

Our initial requirements for item delay of our system involved two components:

1. <1 second delay between an action (i.e. adding or removing a tagged item to the backpack) and detection of the tag on the RPi Zero
2. <3 second delay between detection of a tagged item on the RPi Zero and display of the item on the web application interface

The first of these requirements will be discussed in another subsection below. However, for the second of these requirements, we conducted 10 timed trials between when the RPi Zero detected a tag and when the tagged item was displayed on the web application's interface and found that the average time for this delay was 0.684, significantly below the 3 seconds delay requirement we had initially set.

6.3 Results for Battery Testing of Backpack Device

We tested the system battery life for 5 trials over the course of 7 days, recharging the battery to full in between each trial. For each trial, the system was mounted to the backpack, to simulate the thermal environment that it would be expected to run in. 10 tags were then placed in the backpack with the system, to simulate the maximum item load and to place the maximum amount of stress on the system. The system was also connected via Bluetooth to a desktop computer running the web application. The system was then timed until it stopped reporting the item list to the web application. Over the course of 5 trials, the minimum time the system lasted for was 18 hours and 43 minutes, while the average was 19 hours and 42 minutes. Thus, even under the most strenuous of conditions, the system lasted for longer than our 18 hour requirement on battery.

6.4 Results for Distance Pruning Algorithm Testing

We utilized 2 different tests for testing the accuracy of the distance pruning algorithm, and 1 test to determine the latency of the algorithm. The first test we conducted was our inclusion/exclusion distance measurements. For this test, for each trial, we turned the system on, and placed the tag at a point 100cm from the center of the backpack. We then left the tag for 10 seconds. We then moved the tag closer by 5cm, and left the tag sit for another 10 seconds. This process was repeated, bringing the tag closer to the backpack, until the tag was detected by the system, and remained detected for the full 10 second duration. This distance was recorded as the inclusion distance for the trial. Then, the tag was moved away from the backpack in increments of 5cm, waiting 10 seconds at each distance. The distance at which the tag was no longer detected by the system was then recorded as the exclusion distance. We conducted a total of 20 trials (20 inclusion distances + 20 exclusion distances), and got an average inclusion distance of 47cm with a standard deviation of 10.0cm, while the average exclusion distance was 61cm with standard deviation 8.0cm. Although these average inclusion and exclusion distances were close to our requirement range of 50±10cm, 4 of the 20 trials had inclusion distances which not within range, and 7 of the 20 trials had exclusion distances not within range. Thus, our system is not very consistent in the distance at which items are included/excluded from the item list. For our second test, we placed a tag inside the backpack, and left the tag in the backpack for 10 minutes. The system then logged every time the item list changed. We then counted the number of times the tag was erroneously changed, meaning its status changed from being in the item list to not being in the item list. We did not count the subsequent change of status from not being in the item list to being in the item list as part of the count of erroneous status changes. We ran this test 5 times, placing the item at the bottom, middle, top, and two sides of the backpack. Over the 5 trials, we got an average of 1 erroneous status change per 10 minutes, with the worst position being the bottom of the backpack. Although we did not have an official requirement for erroneous status changes, an average of 1 per 10 minutes is not ideal, but means our system is still usable and only rarely leaves items out of the backpack.

For the latency testing, each trial consisted of placing a tag into the backpack, and timing how long it took for the system to update the item list. The tag was then removed, and the system was again timed for how long it took to update the item list. For 20 trials, the average latency after placing the item into the backpack was 4.93 seconds, while the average latency after removing the item was 6.04 seconds. This latency does not meet the 1 second requirement we set during the design report. Although the latency is higher than we expected, and our original requirement was not met, an average latency of about 5 seconds is still responsive enough to be usable. Users will still be able to have the web interface updated with the correct item list in a reasonable amount of time, just not as short of a time as to be unnoticeable.

6.5 Results for Web Application

Additionally, we also had requirements for our web application, and while these are less quantitative and more focused on the functionality working correctly, these requirements are critical to our product and its use case. Two requirements of our web application were that it be able to handle up to at least 25 weekly events, and that notifications sent to the web application would both appear at the correct time and notify the user if they are missing any items in their backpack for the event. To test the first requirement, we created 25 events in the web application throughout one week, and each event was successfully stored in the calendar. To test the second requirement, we conducted 10 trials: 5 trials involved the user already having all items necessary for an event in the backpack, and 5 trials involved the user missing one or more items in their backpack for an event. On all 10 trials, the system correctly notified the user at the correct notification time for the event and properly notified the user if they were missing any items with the list of missing items.

7 PROJECT MANAGEMENT

7.1 Schedule

Our schedule (see Figures 7 and 8 in Appendix A) has remained largely the same since the design report. However, two changes to our schedule did occur:

1. Due to a lack of time, we did not end up implementing Bluetooth persistence (creation of an Android thin client + a WebView of our web application)

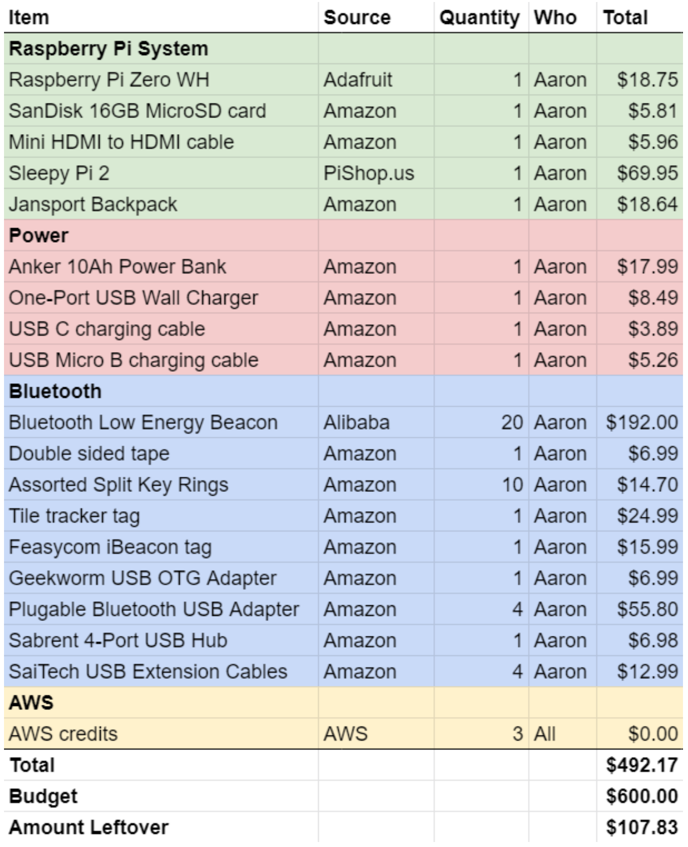
2. Also due to a lack of time, we decided not to implement the sleeping protocol for the RPi Zero. From the user's perspective, not implementing this protocol means that the user does not need to wait for the system to "wake-up" each time they want to use our system. Because our backpack device's battery life already lasts 18+ hours, this change does not affect the user too much.

Additionally, we had 2 weeks of slack time built into the beginning and end of our schedule, and we largely consumed all of our slack time towards the end of our schedule to catch up to implementing the necessary components. This also meant that our system wasn't as refined as it could have been. For example, our image recognition module that was incorporated into our web application accepted image files from the user's device rather than prompting them to take a picture during the registration process. This is simply because we ran out of time for development at the end of our project schedule, as we were preparing for end-of-project deliverables such as the final presentation, poster, video, and final demo.

7.2 Team Member Responsibilities

We divided up each team member's primary responsibilities based on the separate modules of our project as well as the ECE area strengths of each member. Aaron was responsible for the tags and Bluetooth scanner, Janet developed the web application, and Joon implemented the image recognition component. For each member's secondary responsibilities, Janet worked on the Schedule Learning feature (this feature was originally assigned to all members, but Janet completed her necessary tasks on the web application and so was available to work on this), Aaron worked on deployment of the web application to AWS Elastic Beanstalk, and all three members worked on integrating the image recognition component into the web application. Janet and Aaron also worked together on the communication between the Bluetooth scanner and the phone application.

7.3 Budget



Item	Source	Quantity	Who	Total
Raspberry Pi System				
Raspberry Pi Zero WH	Adafruit	1	Aaron	$18.75
SanDisk 16GB MicroSD card	Amazon	1	Aaron	$5.81
Mini HDMI to HDMI cable	Amazon	1	Aaron	$5.96
Sleepy Pi 2	PiShop.us	1	Aaron	$69.95
Jansport Backpack	Amazon	1	Aaron	$18.64
Power				
Anker 10Ah Power Bank	Amazon	1	Aaron	$17.99
One-Port USB Wall Charger	Amazon	1	Aaron	$8.49
USB C charging cable	Amazon	1	Aaron	$3.89
USB Micro B charging cable	Amazon	1	Aaron	$5.26
Bluetooth				
Bluetooth Low Energy Beacon	Alibaba	20	Aaron	$192.00
Double sided tape	Amazon	1	Aaron	$6.99
Assorted Split Key Rings	Amazon	10	Aaron	$14.70
Tile tracker tag	Amazon	1	Aaron	$24.99
Feasycom iBeacon tag	Amazon	1	Aaron	$15.99
Geekworm USB OTG Adapter	Amazon	1	Aaron	$6.99
Plugable Bluetooth USB Adapter	Amazon	4	Aaron	$55.80
Sabrent 4-Port USB Hub	Amazon	1	Aaron	$6.98
SaiTech USB Extension Cables	Amazon	4	Aaron	$12.99
AWS				
AWS credits	AWS	3	All	$0.00
Total				**$492.17**
Budget				**$600.00**
Amount Leftover				**$107.83**

Figure 10: Bill of Materials

The bill of materials for our project can be found in Figure 10. The items are separated by which component of the system they are for. The green and red items are part of the backpack system, with green being supporting components for the Raspberry Pi Zero and red being power-related items. The blue items are the Bluetooth-related items for tracking and mounting to individual items. Finally, the yellow item is the AWS credits which we are using for our AWS server database. Since the Design Report, we have also purchased a backpack, 4 USB Bluetooth adapters, USB extension cables, and a USB hub. These were purchased as a result of changes to our design. We mostly chose parts supplied by Amazon, which reduced shipping costs as Amazon Prime has free shipping. We also elected to purchase a Tile Bluetooth tracker tag to compare the performance of our system with that of a similar existing product. We only purchased one copy of each of the hardware items, since Aaron was the only team member responsible for the hardware of our project. For software, we requested AWS credits for all three team members as we are all contributing to the software portion of the project.

7.4 Risk Management

We knew that our team had to be vigilant when it came to communication and risk management plans, especially as the three of us are located in three different time zones (US Eastern Time, US Pacific Time, and Korean Standard Time). Because our system is implemented in three distinct modules, we also knew that many issues could appear during integration, and this is why we set aside time specifically dedicated to integration tasks.

One of the largest risk areas of concern in our project was the ability to accurately determine which items are in the backpack. With a gating algorithm based on a single distance to a single Bluetooth sensor (the Raspberry Pi Zero), the shape of the included volume is approximately a sphere centered on the sensor. We believed that, by placing the Raspberry Pi at the center of the backpack, the spherical shape will be close enough to the backpack's shape to determine which items are in the backpack. However, if the single-distance-based gating algorithm is unable to exclude items that are near the backpack but not inside the backpack, our risk mitigation plan was to utilize a trilateration approach using a total of 4 Bluetooth sensors spaced around the backpack to obtain a 3D location for the item. The trilateration approach and algorithm is discussed in Pu, Pu, and Lee's "Indoor Location Tracking using Received Signal Strength Indicator" paper.[12] We will use the trilateration approach as opposed to the triangulation approach, as it is easier for our system to determine the distances between each of the 4 sensors than to determine the angles between them.

Another risk element was the issue of using Android Studio to build our phone application. After discussing amongst ourselves as well as with our TA, we decided to switch to building a mobile-accessible web application after the design review presentation for multiple reasons. First, the group collaboration for the development process would be much more streamlined compared to development in Android Studio, as Visual Studio Code (which we would user for web application development) has better Git integration than Android Studio. Second, a web application offers greater accessibility to different platforms such as iOS than an app designed only for Android. Additionally, 2 out of 3 of our team members already have prior experience with web application development, as opposed to Kotlin which none of us have experience with. We also did extensive research and found the Web Bluetooth API, which will allow us to communicate with Bluetooth devices over JavaScript, so our system would still maintain the Bluetooth functionality. Finally, if we found that we needed a component of native functionality that was missing from this web application, we could use the WebView functionality to build any necessary native features in Android Studio port them deliver to our web application.[4]

7.5 AWS Credit Usage

Our project would not have been possible without Amazon Web Services. We used the AWS Elastic Beanstalk service to host our Django-based web app, as it provided a quick and simple way to get our project up and running. We did encounter some roadblocks while setting up the server, however the AWS documentation provided us with the necessary solutions to solve our problems and get past these roadblocks. We would like to thank AWS for providing us with free AWS credits. Although our usage ended up falling within the free usage tier, and therefore no credits were used, having the credits gave us the assurance that we could still work on our project even if our usage exceeded the free limits.

8 ETHICAL ISSUES

Possible ethical issues regarding our project include data privacy of users, possible security vulnerabilities, and lack of accessibility. For example, one possible edge case involves a scenario where hackers might be able to steal data from our app. Since our app collects data regarding what events users might attend and what items they have in their backpack, revealing this data would be a large breach of privacy. Additionally, if our project became an actual commercial product, one ethical issue (that's become quite common these days) is that we, the owners of the product, could voluntarily sell this data to advertisers for marketing purposes. For example, knowing the events a person might attend or the items they frequently carry can give a large advantage to online advertisers looking to target their ads to each individual user.

The people that might be affected adversely by these edge cases include any user of our product. Since our product is targeted towards students, specifically college students, they become a group that would be vulnerable to data leaks or target advertising.

Possible approaches to mitigating potential foreseen adverse impacts include improving the security of our web application (for example, depending on tokens rather than links for our forms), getting a real SSL certificate from a certified authority, and including a user agreement for users who sign up that we intend to collect data about the items they have in their backpacks. This user agreement would inform users about the fact that their item data is being collected, and warn them about the potential dangers of this data being released.

9 RELATED WORK

There are currently several other existing asset-tracking products which help users keep track of their items. One popular existing technology in this space is Tile, a small Bluetooth-based tracking tags that allows the user to locate the missing items so long as they are attached to a tag. Backpack Buddy and Tile share some common features. For instance, both require physical tags to be attached to the items to track the items, and both use Bluetooth Low Energy (BLE) technology to communicate with the tags. Both products also utilize the smartphone app to interface with the user. However, while Backpack Buddy and Tile share some similarities, Backpack Buddy differs from Tile in two major ways. First, Backpack Buddy allows students to manage collections of items in relation to their schedule, whereas Tile only tracks and locates individual items. Second, Backpack Buddy has reduced user burden with regards to item registration. Backpack Buddy uses computer vision to automatically determine item names, saving the user effort when naming items. Backpack Buddy also learns the user's schedule, such as what items a student might carry in the backpack when and where, saving the user from needing to manually assign items to events.

Additionally, Apple recently released their new Apple AirTags, which also utilize Bluetooth Low Energy to connect to an app on your phone. However, AirTags are specialized in that they focus on tracking individual tagged items (again, similar to Tile) using ultra-wideband technology and using Apple's network of existing devices as crowd-sourced beacons to ping each other and determine the location of your tagged item.[2] Thus, AirTags also lack the item collection management or scheduling capability which Backpack Buddy has. AirTags do, however, have the advantage that tags can be located virtually anywhere on the globe, unlike Backpack Buddy, which only locates tags in range of a backpack.

One CMU ECE Capstone project from the Spring 2020 semester called Sous-Chef also shares similarities to Backpack Buddy. Sous-Chef is a smart pantry storage unit that keeps a list of groceries currently inside a pantry. Sous-Chef is similar to Backpack Buddy in that it registers a food item to a database using computer vision and image processing of barcodes on food products. Sous-Chef also provides a streamlined user experience by displaying the list of food items on a web application. However, Sous-Chef has a different use case than Backpack Buddy, as it is intended for tracking of food items in a pantry, whereas Backpack Buddy is designed for tracking schoolwork- or extracurricular-activity-related items in a backpack.

10 SUMMARY

With their busy schedules and plethora of items to track, students often lose or forget important items for their events. A smart inventory system is therefore extremely valuable for managing tracking items, saving students from the burden of having to remember items and the anguish from forgetting them. Backpack Buddy not only informs students what items are inside their backpack, but also suggests what items to bring according to the students' schedules and notifies them of their missing items before each event. We intend to improve the lives and mental well-being of students with our easy to use inventory system.

10.1 Lessons Learned

We have learned a lot about teamwork and communication through working on Backpack Buddy. We also learned a lot about the value of planning ahead, as we found our Gantt Chart to be one of the most important sources of organization and tracking for our project's progress. Because our teammates were all spread across the globe, development was difficult, and we learned that detailed communication and frequent pushes to GitHub for version control were important for effective teamwork.

Glossary of Acronyms

- API – Application Programming Interface
- BLE – Bluetooth Low Energy
- CNN – Convolutional Neural Network
- GATT – Generic Attributes Profile
- MAC – Media Access Control
- MVP – Minimum Viable Product
- RFID – Radio Frequency Identification
- RPi – Raspberry Pi
- SSL – Secure Sockets Layer
- UUID – Universally Unique Identifier

References

[1] *Animal Image Classification using CNN.* https://github.com/imamun93/animal-image-classifications. Apr. 2019.

[2] *Apple AirTags: Everything You Need to Know.* https://www.pcmag.com/how-to/apple-airtag-tips. May 2021.

[3] *Bootstrap MultiStep Form.* https://codepen.io/designify-me/pen/qrJWpG.

[4] *Building web apps in WebView.* https://developer.android.com/guide/webapps/webview. Mar. 2021.

[5] Wouter Bulten. *Kalman filters explained: Removing noise from RSSI signals.* https://www.wouterbulten.nl/blog/tech/kalman-filters-explained-removing-noise-from-rssi-signals/. Oct. 2015.

[6] *django-calendar.* https://github.com/huiwenhw/django-calendar. July 2018.

[7] *Django-Webpush.* https://github.com/safwanrahman/django-webpush. July 2020.

[8] Douglas6. *Python GATT server example for the Raspberry Pi.* https://github.com/Douglas6/cputemp. June 2019.

[9] OpenSSL Software Foundation. *OpenSSL Cryptography and SSL/TLS Toolkit.* https://www.openssl.org/. 2018.

[10] *H1 Beacon Datasheet.* http://doc.mokotechnology.com/index.php?s=/page/28. Dec. 2020.

[11] *IDT - Image Dataset Tool.* https://pypi.org/project/idt/.

[12] Chuan Chin Pu, Chuan-Hsian Pu, and Hoon-Jae Lee. "Indoor Location Tracking Using Received Signal Strength Indicator". In: *Emerging Communications for Wireless Sensor Networks* (Feb. 2011).

[13] *RFID vs BLE: How Are They Different in Terms of Asset Tracking?* https://www.assetinfinity.com/blog/rfid-vs-ble-how-are-they-different-in-terms-of-asset-tracking. Mar. 2021.

[14] Farhana Sultana, Abu Sufian, and Paramartha Dutta. "Advancements in Image Classifications using Convolutional Neural Network". In: *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)* (May 2019).

[15] *VGG16 Model in Keras.* https://keras.io/api/applications/vgg/.

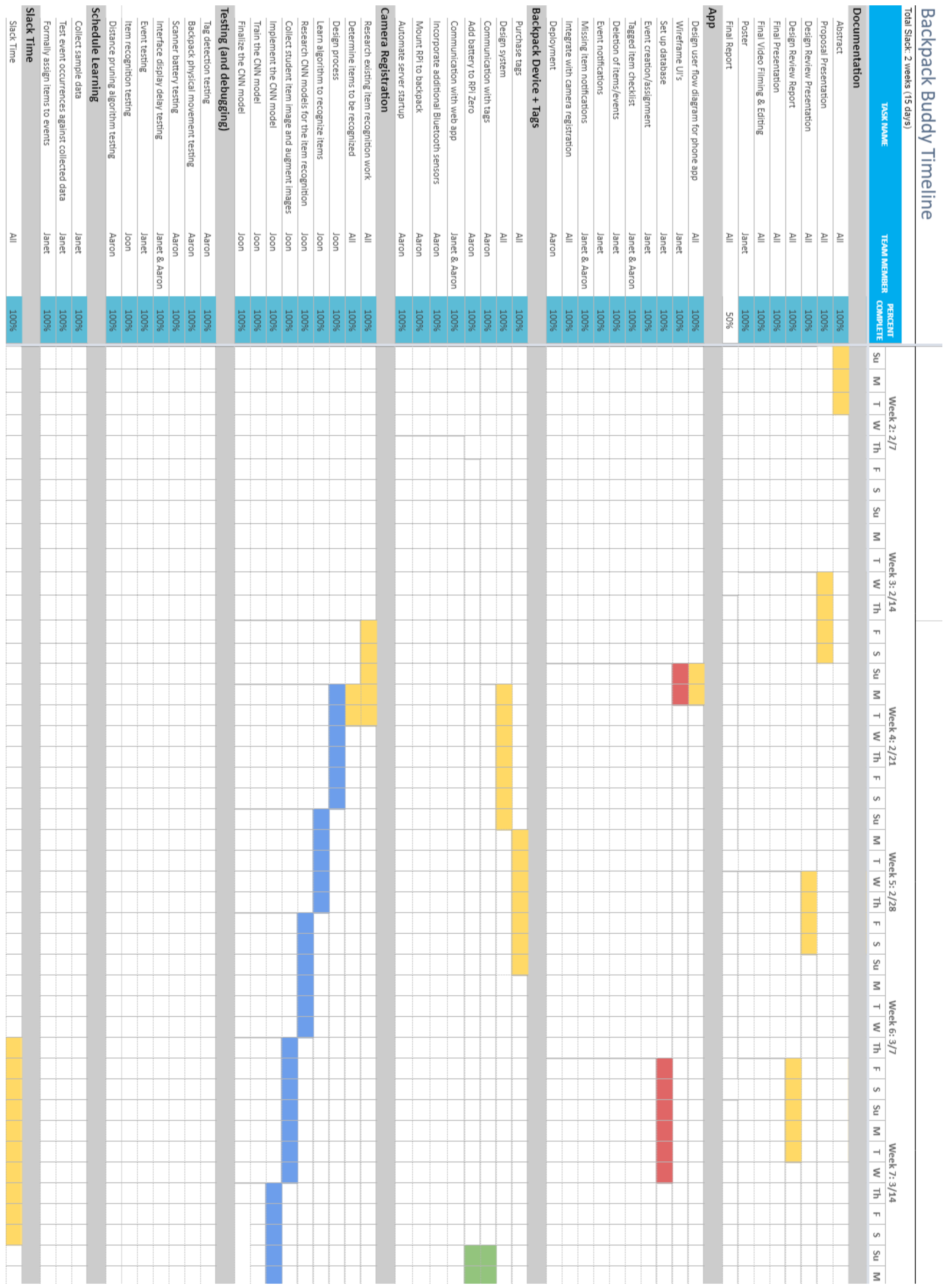
[16] François Wautier. *Python only library to scan and decode advertised BLE info.* https://github.com/frawau/aioblescan. Feb. 2021.

[17] *Web Bluetooth API Documentation.* https://webbluetoothcg.github.io/web-bluetooth/#introduction. July 2020.

[18] David G. Young. *How Far Can You Go?* http://www.davidgyoungtech.com/2020/05/15/how-far-can-you-go. May 2020.

Requirement	Expected Result	Results	
< **1s** delay for tag detection	Item list updates within 1s	✗	Updated between 3.86s to 8.19s
< **3s** delay for interface update	Webpage updates within 3s	✓	Webpage updates < 1s after system update
Detect **10** items within a 0.5-meter range ± 0.1m with **100%** accuracy	Item list displays all 10 correct ID's	✓	10 tags were attached to various objects (some made of metal) and all 10 ID's showed up
Last **18 hours** w/o recharge	System still on after 18 hours	✓	System lasted 18+ hours on 5 trials
Handle **25** weekly events	All events appear in calendar	✓	Web app successfully stored and displayed 25 events
Notification appears at specific time before event which indicates **exactly** which items are missing	Notification with correct list of missing items appears on phone screen	✓	Notification appeared on desktop in Windows and also appeared in Android
80% item recognition accuracy	Correct name of item 80% of the time	✓	84.36% with total 3150 test image datasets (150 images per student items)

Figure 11: System Requirements and Results



TASK NAME	TEAM MEMBER	PERCENT COMPLETE
Documentation		
Abstract	All	100%
Proposal Presentation	All	100%
Design Review Presentation	All	100%
Design Review Report	All	100%
Final Presentation	All	100%
Final Video Filming & Editing	All	100%
Poster	Janet	100%
Final Report	All	50%
App		
Design user flow diagram for phone app	All	100%
Wireframe UI's	Janet	100%
Set up database	Janet	100%
Event creation/assignment	Janet	100%
Tagged item checklist	Janet & Aaron	100%
Deletion of items/events	Janet	100%
Event notifications	Janet	100%
Missing item notifications	Janet & Aaron	100%
Integrate with camera registration	All	100%
Deployment	Aaron	100%
Backpack Device + Tags		
Purchase tags	All	100%
Design system	All	100%
Communication with tags	Aaron	100%
Add battery to RPi Zero	Aaron	100%
Communication with web app	Janet & Aaron	100%
Incorporate additional Bluetooth sensors	Aaron	100%
Mount RPi to backpack	Aaron	100%
Automate server startup	Aaron	100%
Camera Registration		
Research existing item recognition work	All	100%
Determine items to be recognized	All	100%
Design process	Joon	100%
Learn algorithm to recognize items	Joon	100%
Research CNN models for the item recognition	Joon	100%
Collect student item image and augment images	Joon	100%
Implement the CNN model	Joon	100%
Train the CNN model	Joon	100%
Finalize the CNN model	Joon	100%
Testing (and debugging)		
Tag detection testing	Aaron	100%
Backpack physical movement testing	Aaron	100%
Scanner battery testing	Aaron	100%
Interface display delay testing	Janet & Aaron	100%
Event testing	Janet	100%
Item recognition testing	Joon	100%
Distance pruning algorithm testing	Aaron	100%
Schedule Learning		
Collect sample data	Janet	100%
Test event occurrences against collected data	Janet	100%
Formally assign items to events	Janet	100%
Slack Time		
Slack Time	All	100%

Figure 12: Gantt Chart First Half

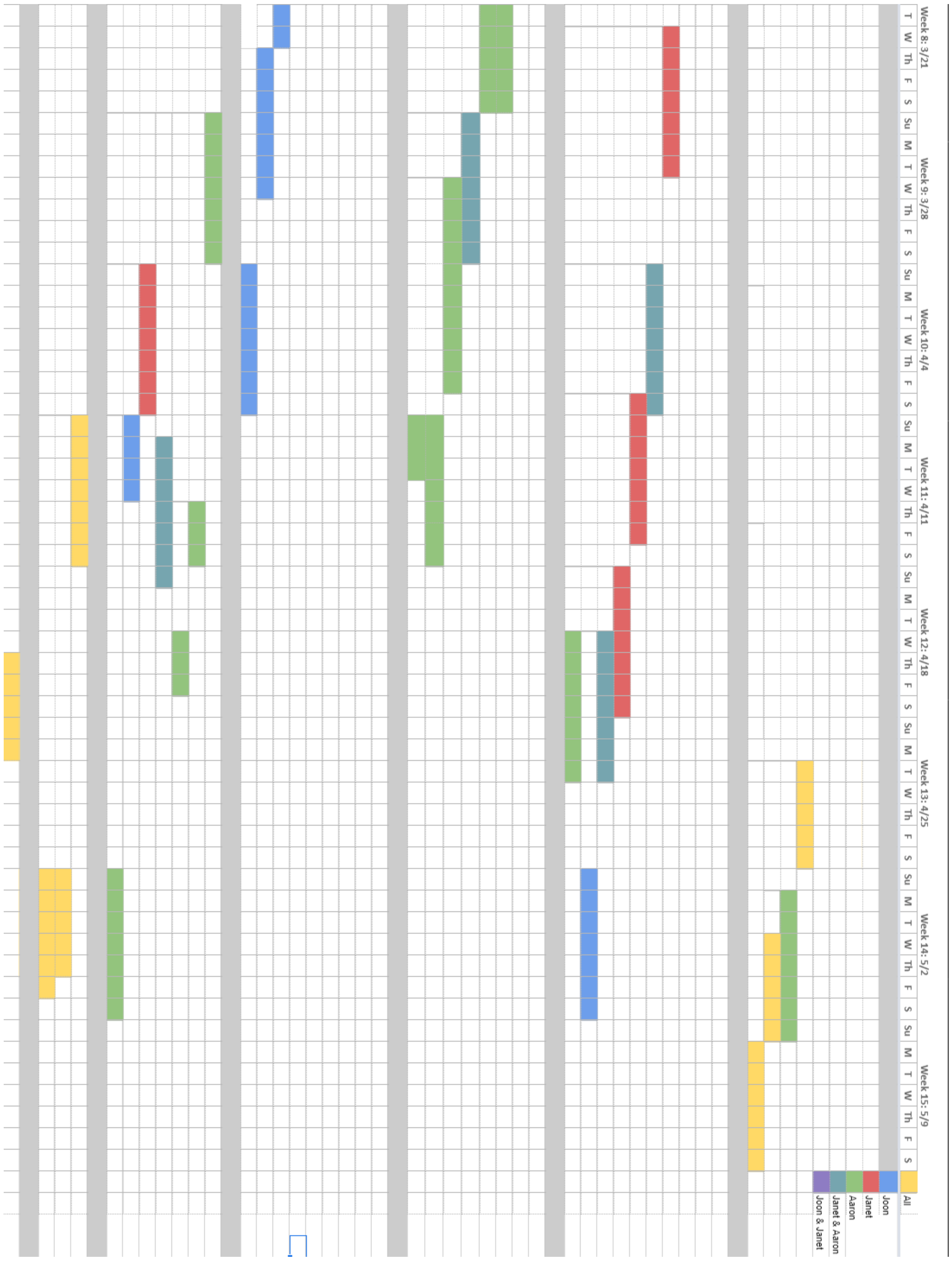


Figure 13: Gantt Chart Second Half