waitr

Sophie Sacks (Author), Samantha Lavelle (Author), Dina Razek (Author)

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—**A system capable of utilizing sensors to measure how many people are in a waiting line gives users the opportunity to check an application to see the estimated wait time. Our end product includes one custom-built and one commercial RFID scanner with one at the beginning of the line and one at the end of the line. The user will scan their CMU ID when they enter the line and when they exit the line, which will send timing information to our web application. The web application will use machine learning to predict wait times throughout the day using real-time information from the scanners.

***Index Terms*—Django, inductor coil circuit, machine learning, RFID, web application.**

I. Introduction

Students on campus have a very difficult time knowing how crowded and busy restaurants are in real time. Without a way to know how long the line is at the on-campus eateries, students may miss an opportunity to grab food in between classes. Especially during COVID, students may be concerned about how crowded an indoor area is. If there was an option to check the wait time at a certain restaurant before walking across campus and physically seeing the line, students would be able to budget their time spent getting food or a drink more appropriately. This increased accessibility to food makes it much easier for students to stop skipping meals because they do not have the time. Additionally, a solution to this use case would lead to decreased stress on eateries and staff as well as increased business during slower times.

Although our focus is placed towards one on-campus dining location, this project can be applied to any physical space with a line with a single entry and exit and is scalable to many other businesses. Other applications include, but are not limited to, package pickup in the University Center, other on-campus eateries, and restaurants located elsewhere off campus.

A combined hardware-software solution would help mitigate this issue for students. By having physical sensors to measure how many people are in line in a certain eatery, users can then check an online web application to see the estimated wait time. Our desired end-product consists of two radio-frequency identification (RFID) scanners to track how long each patron is in line. Every user will scan in at the beginning of the line and scan out at the end of the line. This data will be sent to our web application, which will use machine learning (ML) to predict wait times as well as display the current approximate wait time.

While the main goal of this project is to be able to predict an accurate wait time, we also have the goal of gaining user buy-in in order to have enough data for training the ML model. Moreover, the principal advantage of this approach is that the data collected is completely anonymous, especially in comparison to a camera-focused solution using computer vision.

II. Use-Case Requirements

In order to support our predetermined use case, we must define certain use-case requirements that will ensure our users are happy with the product. First, each user wants to have a simple-to-use web application that is not noticeably slow for users and makes it very easy to figure out an accurate wait time once opened. This qualitative requirement results in the quantitative requirement of needing a margin of error for the wait time within 2 minutes or within 10% (whichever is larger) of the actual wait time. Users also want as little interaction as possible with our product, resulting in a total of 2 or less points of interaction (i.e. the scanners) while physically in line at the eatery.

To continue maintaining accurate wait times, our system must be able to properly track ID and keep track of up to 50 patrons who are in line. This number ensures that we will be able to collect and analyze enough data to make an accurate prediction in the time window. On the other hand, our system should be able to show information to users when there are no patrons in line as well. In addition, it must be able to properly handle the case of a user leaving the line after being tracked as entering the line or scanning out of line without scanning into line. Thus, if 3 or more users are tracked as exiting the line who originally came before the patron in question (based on data points of exact time and ID in our entry data table), then that user's data point will be discarded. Similarly, if the user is tracked as exiting the line without entering the line, their data point will also be discarded so that our ML model can still accurately predict wait times. Lastly, our system must have a battery lasting three or more hours. We do not want to add a large burden to the staff who must keep the devices in operating conditions, so three hours gives the staff ample time and opportunity to update the power source in times of low customer count.

III. Architecture and/or Principle of Operation

The diagram in Fig. 1 shows our entire system architecture. On the left side is the hardware subsystem and on the right side is the software subsystem.

Our solution to the previously described problem utilizes RFID scanners to track wait times in line. This data is fed through a machine learning model to then predict wait times in the future. All of this data is stored in a Django SQL database, which then propagates through to the frontend web application where users can view the current and future wait times.

The hardware subsystem consists of three major components: the inductor coil circuit, the Arduino, and the Raspberry Pi. We created one total hardware component to

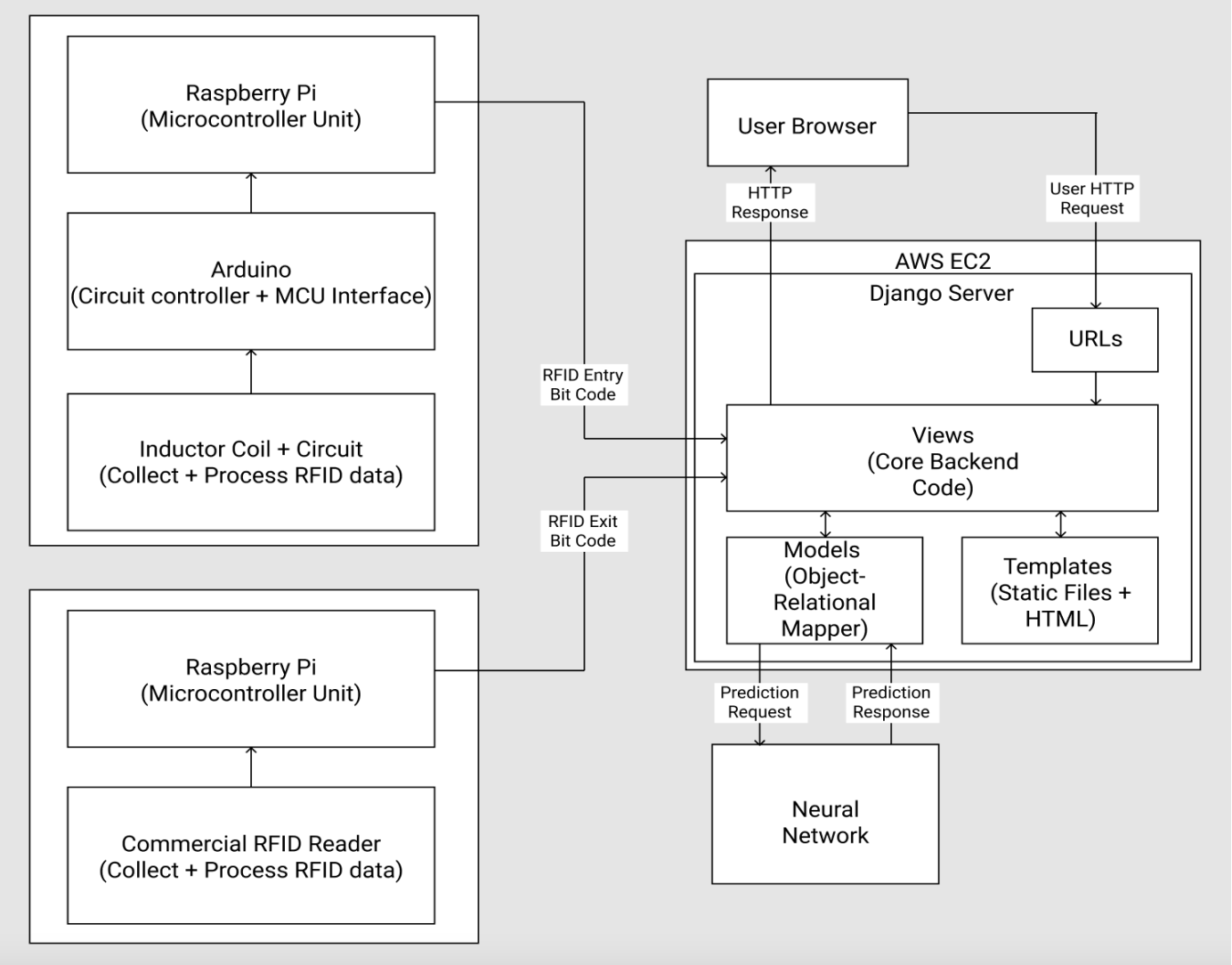

Fig. 1. Block diagram of entire system architecture.

connect to our software to be used as an RFID reader in the line. The other hardware component used is a commercial RFID reader to be used as the other scanner in the line. The main connection between the hardware and software systems are the RPis. The data flows from these hardware pieces to the software backend.

The software subsystem consists mainly of the Django web application server and the machine learning component. These are connected in the backend for communicating information to the user. The data flows from the backend of the server to the frontend where the user can see the predicted wait time.

We originally wanted to have two custom-built RFID readers in our system, but due to issues with the lab equipment, we were only able to construct one. Ideally, if we were to continue developing the project, we would solder the circuit that's on our proto-board twice to make two ready-to-use boards. The custom-built reader and the commercial reader read IDs at the same frequency, so for now we are able to use one at the beginning of the line and one at the end of the line.

Fig. 2 shows the actual implementation of our system. The laptop screen shows our web application. On the right is an ADALM, which powers the op amp in our circuit and provides a 125 kHz sine wave to the scanner antenna portion of the circuit. On the left is our inductor coil and the circuit itself, which scans and demodulates the ID number via RFID. The circuit is connected to an Arduino Uno, which converts the analog bit signal to digital and sends it to the RPi Zero W. The Arduino and the RPi are directly connected to each other with a cable. Finally, the ID number is sent to our web server via the wireless capabilities of the RPi Zero W.

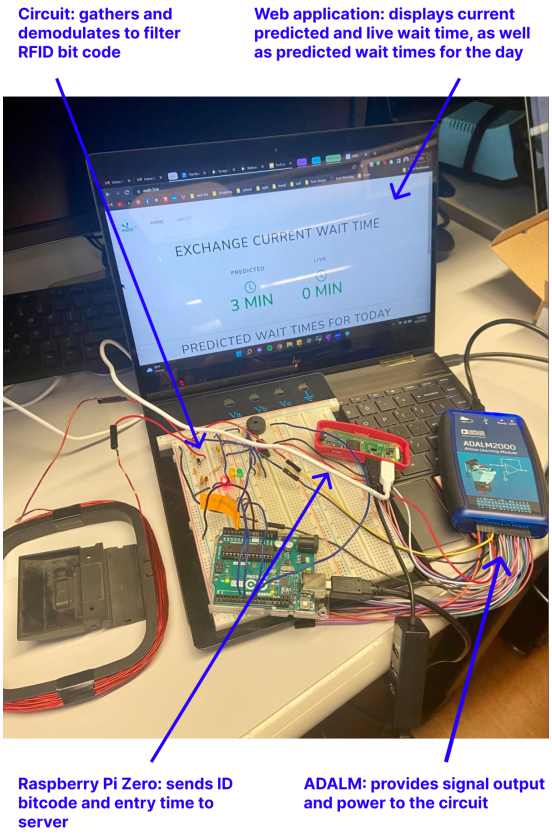

Fig. 2. Overall completed system with one RFID reader.

IV. Design Requirements

Our solution space has many critical constraints that we will use to define success. To satisfy the use case requirement of having a battery last three hours or more, our design requires a 125 kHz power source to power the RFID reader circuit and an additional power source for the op amp, RPi, and Arduino components.

To meet the use case of quick responsiveness of our web application, we must have transmission from the RFID reader to the server within 2 seconds and web application response within 2 seconds. These requirements are based on the size of the data, the channel bandwidth, and general expectation of transmission delay. Users are likely to notice a delay if it is over 2 seconds, and may leave the application entirely as a result. In order to be usable, we have a design requirement of a reasonable level of security in our web application, ensuring corrupt user input does not crash our website and that our data is securely stored and inaccessible to the public. Similarly, we require that relevant data is displayed in a format easily digestible to any possible users.

In order to satisfy the use case requirement of accurate data being displayed, we require a margin of error for wait time to be within 2 minutes or 10% - whichever quantity is larger. This ensures that the predicted wait time closely reflects the actual wait time for the entire range of possible times. Similarly, we must ensure our design has proper error mitigation, catching any cases of incomplete or invalid data to ensure our data does not get affected and remains accurate. To be able to hold up to 50 patrons at a time, we have a design requirement of having a large enough SQL database.

V. Design Trade Studies

A. Hardware Subsystem

As shown in Fig. 1, our hardware subsystem will involve an inductor coil that detects the presence of an RFID card.

$$L = N^2 \mu_0 a \left[ln \frac{8a}{r} - 2 \right] \quad (1)$$

The inductance of our coil is found via equation (1) [1], where L is the inductance (in Henries), N is the number of turns, μ_0 is the permeability of free space ($4\pi * 10^{-7}\ H/m$), a is the radius of the loop (in meters), and r is the radius of the wire (in meters). Because our inductor coil detects and reads the RFID card, it must be larger than the ID card in diameter. This will cause a to increase, resulting in a higher inductance.

$$\omega_0 = \frac{1}{\sqrt{LC}} \quad (2)$$

$$\omega = 2\pi f \quad (3)$$

This inductance is then used to calculate the capacitance required to achieve our resonant frequency. The calculation for this is shown in equation (2), where ω_0 is the resonant frequency and C is the capacitance needed. Since we are operating at 125 kHz, we will use equation (3) to convert this to angular frequency. 125 kHz is used to power the system because that is the RFID frequency used in ID cards [2].

B. Computers

Other design choices include which devices will connect to our circuit and transmit our data. The use of the Arduino Uno allows the system to be easily adaptable, as we can modify how the circuit behaves via the Arduino code. This allows us to customize the LED lights, sound, and more. The use of the Raspberry Pi Zero W as our microcontroller unit allows us to send data wirelessly to our web server thanks to facile connection to the Arduino. We chose to use two separate controllers so that both components of the system (software and hardware) can be developed in parallel, and easily combined at the end. This worked out well; the circuit was tested and debugged with the Arduino attached, and the software component was tested using the RPi hooked up to a commercial RFID reader. These two components were also chosen simply because they were readily available to the project, allowing us to begin work as early as possible.

C. Web Application Frontend Subsystem

For our web application frontend, the main design requirement is being intuitive, responsive, and easy to use. We will use HTML, CSS, and Javascript to build our frontend design. While we considered writing our own CSS for the design, we looked into possible frontend frameworks that have built-in components we could utilize. Bootstrap is a responsive open-source frontend framework that contains CSS and Javascript-based design templates for many types of interface components, such as navigation, forms, and typography, thus satisfying our design needs. There are many existing themes

that we could utilize and easily add to our project. Our team also has experience using Bootstrap in web applications. As a result, implementing a Bootstrap framework can fit in our project timeline while also meeting our design requirements.

D. Web Application Backend Subsystem

For our web application backend, we performed a design review of a few different backend frameworks. The two frameworks that stood out for our application were Express and Django. Express is a minimal and fast framework that provides some core framework functionalities while remaining very flexible, thus suiting some of our design needs. However, there is a lack of definition of a lot of functionality that would make it difficult to ramp up quickly and complete our project within the semester time limit.

On the other hand, Django is a Model-View-Template framework that contains many features we need for our solution, namely security and authentication tools. The models make it simple to deal with the database, the templates set us up to reuse many components, and the views contain all business logic required for our application. The built-in structure is robust and simple, making it easy for developers to write efficient and clean code. Our team also has extensive experience using Django, so we would be able to quickly jump into the code. Through performing this design review, we were able to make the decision to use Django.

E. Machine Learning Backend Component

One of our most important design requirements is accuracy in predicting wait times, which relies completely on our neural network. The main design choices affecting accuracy are the optimizer and loss functions used by the network.

The optimizer function called Adam, or adaptive moment estimation, has been shown to require little memory as well as converge on a model much faster than other types of optimizer functions [5]. Although most of the optimizer functions are similarly simple to implement using `tensorflow`, we decided to utilize Adam in order to make our neural network as robust as possible.

The loss function is used to minimize the error while training the neural network model. We chose to use mean squared error (MSE) since that makes the most sense with regression.

Another component impacting the accuracy and training of the neural network is its architecture. The one we chose is discussed further in the System Implementation section below.

VI. System Implementation

Our overall system solution is a combination of hardware and software. On the hardware end, we built one RFID reader, placed at the entrance of the line, for users to scan in. A commercial RFID reader is placed at the end of the line for users to scan out. This setup allowed us to focus on creating and testing the custom-built RFID reader, which would just be duplicated to replace the commercial reader. A Raspberry Pi sends the time data and ID number to our backend server (`views.py`), which will be analyzed and parsed by our ML model and displayed by our web application frontend.

A. Hardware Subsystem

Our hardware subsystem consists of four components: (1) the inductor coil, which detects and reads the RFID card; (2) a circuit which filters, demodulates, and delivers the signal; (3) an Arduino, which captures the data from the circuit and controls various circuit features; and (4) a Raspberry Pi microcontroller unit (RPi MCU) which connects to the Arduino and transmits the data to our web server.

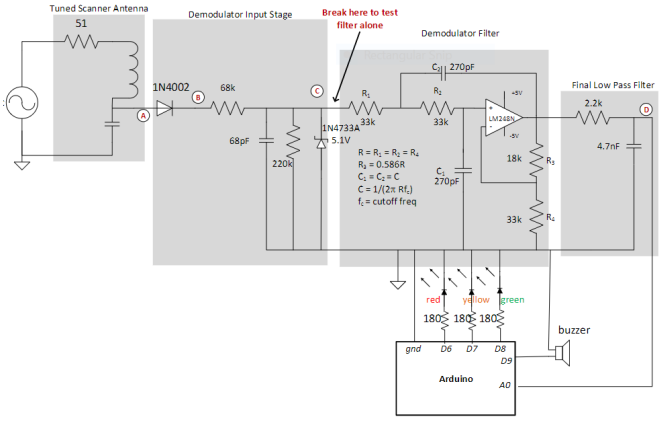

Fig. 3. Schematic of circuit and connected Arduino. Image shown larger in Appendix. This image is taken from the 18-220 Lab 4C handout.

Fig. 3 shows our circuit design. This figure can be found in a larger size at the end of our report. The circuit has four stages: (1) The scanner antenna and tuning components, (2) The input demodulator, (3) The demodulator filter, and (4) a final low pass filter. The circuit is connected to an Arduino, which: powers the op-amp; powers and controls the LED lights and speaker; captures the output information, which contains the bit code of the RFID chip; and transmits this information to our Raspberry Pi.

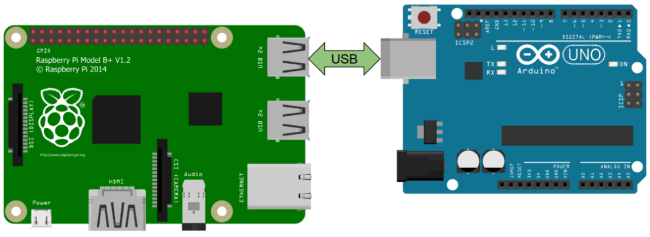

Fig. 4. Connection of Arduino to RPi MCU.

Fig. 4 [3] shows how the Arduino is connected to the Raspberry Pi. A USB connection allows for transmission of data from the Arduino to the Raspberry Pi, which can then send the data to our web server via an internet connection.

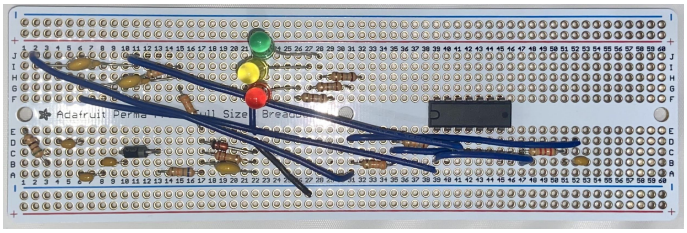

Fig. 5. Completed circuit.

Fig. 5 shows the finished circuit on a Perma-Proto Board, ready to be connected to the inductor coil and Arduino.

B. Web Application Frontend Subsystem

The web application frontend consists of HTML to display each page, CSS to design each page, and Javascript to handle the user interactions. Our HTML has the following structure: a base page that all pages extend (reusable component), a navigation bar that is included in the base page, and the two pages that represent the home page and about page. Our CSS is a combination of our own implementation and a Bootstrap template.

When the web application is opened or reloaded, a request is sent to the backend to retrieve the predicted and live wait times as well as the wait time predictions for the day. The data is then displayed on the home page. The `views.py` file is responsible for rendering each page and retrieving the wait time.

```
def default_home_context_generator():…

def home_page(request):…

def submit_wait_time(request):…

def about_page(request):…

@csrf_exempt
def scan(request):…
```

Fig. 6. `views.py`, which depicts the rendering of HTML pages.

For the home page design, we display: the predicted wait time; the live wait time; a graph of the wait times for the current day, as collected by our ML model; and a form to allow the user to input their entry and exit time in the case that they forget to scan in and out on the RFID readers, as seen in Fig. 7. Graph data points are shown in 15 minute intervals, and a user can hover over the point to see point details.

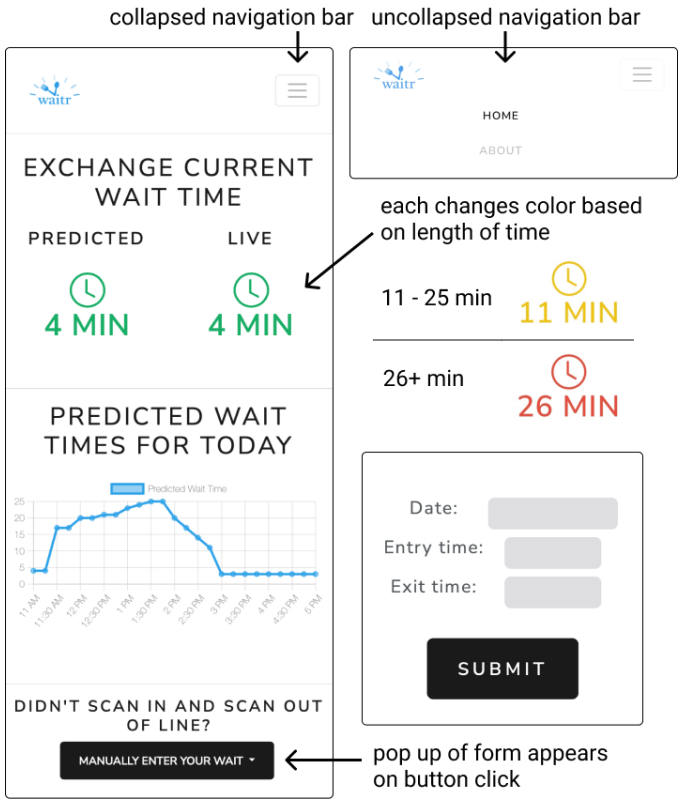

Fig. 7. The final design of our web application home page.

In `views.py`, the data from a form submission is received as a POST request, and it is either validated or rejected. If it is a valid data point, `views.py` sends the success message and the HTML is updated with a success banner. In the case that it is an invalid data point, the error message is sent, and the user is prompted to edit their form response based on the error received. Examples of both success and error messages can be seen in Fig. 8.

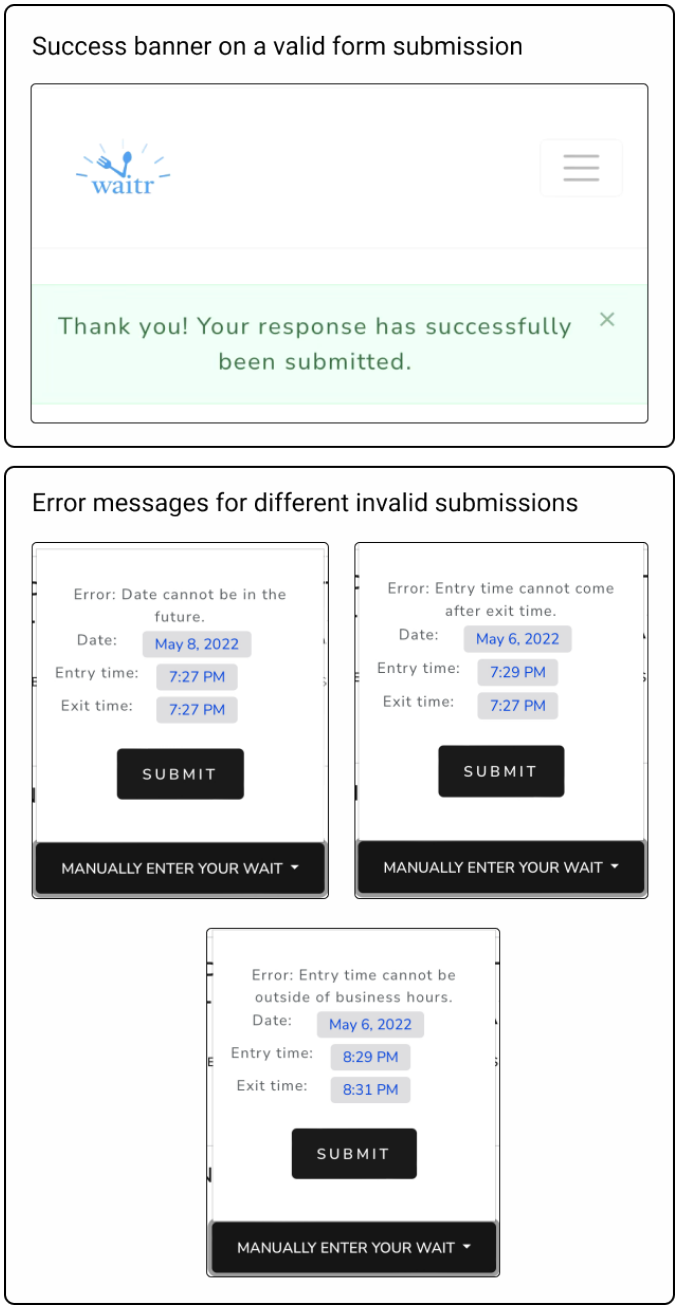

Fig. 8. Possible success and error messages displayed to the user..

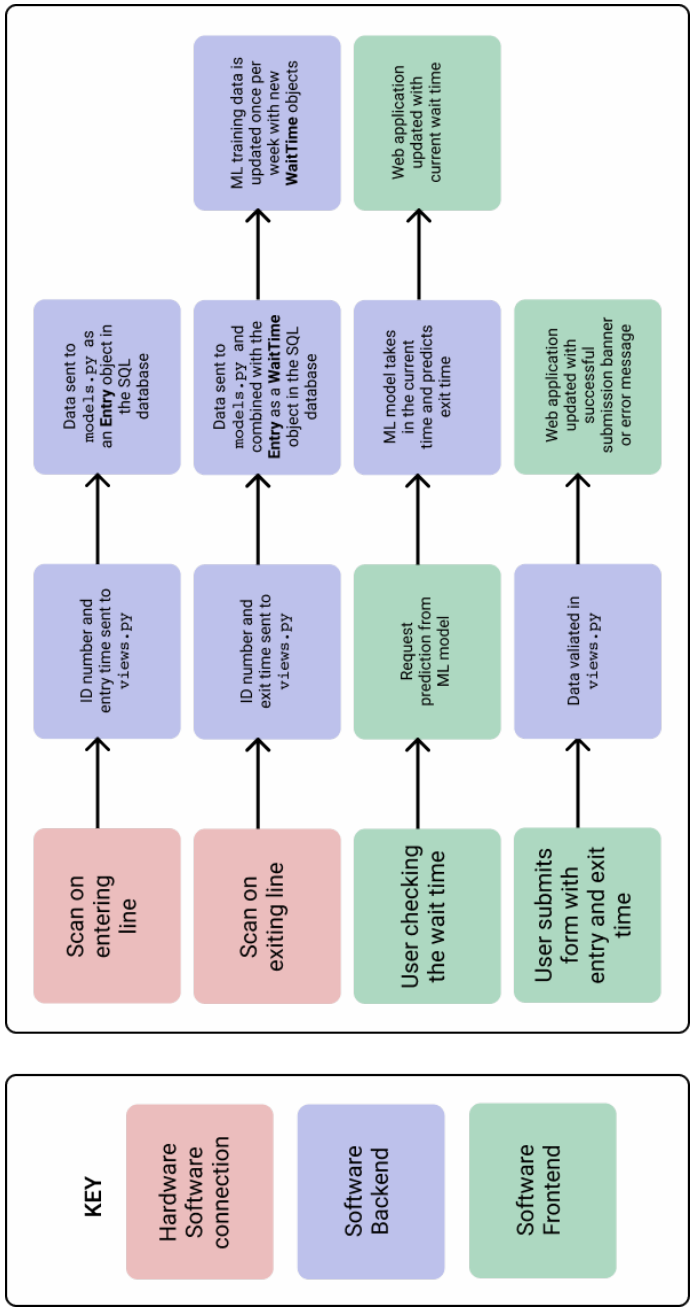

Fig. 9. Software data flow diagram.

C. *Web Application Backend Subsystem*

Our web application software backend consists almost entirely of Django, a Python-based open-source framework that follows the model/template/views architectural framework. As shown in Fig. 9, the system receives information from the RFID scanners, including the ID number that was scanned. This data is stored in the Django SQL database as either an `Entry` model object for the first scan or as a `WaitTime` model object for the second scan (Fig. 10). When data from a second scan comes in, `views.py` searches for the `Entry` model object with the matching ID number, combines it with the current time to create a new `WaitTime` model object, and deletes the used `Entry` model object.

```
waitr > waitr > models.py > ...
1   from django.db import models
2
3   class Entry(models.Model):
4       id_number = models.CharField(max_length=200)
5       entry_time = models.DateTimeField(blank=True, null=True)
6
7       def __str__(self):
8           return str(self.id_number) + ',entry_time=' + self.entry_time
9
10  class WaitTime(models.Model):
11      entry = models.DateTimeField(blank=True, null=True)
12      exit = models.DateTimeField(blank=True, null=True)
13
14      def __str__(self):
15          return str('entry=' + self.entry) + ',exit=' + self.exit
```

Fig. 10. `models.py`, which depicts the Django SQL database structure.

In addition to the data from the RFID scanners, data

collected from the user input form on the web application frontend is also stored in the SQL database. The form inputs are validated using several Django validators in order to implement an allow list. By utilizing both syntactic validation to ensure the user input has the correct syntax as well as semantic validation to enforce correctness of the input values in the specific business context (i.e. time is within operating hours, wait time is under an hour), the allow list protects against both cross-site scripting (XSS) and SQL injection attacks.

When a wait time is requested by the frontend subsystem of the web application, the backend will call upon the neural network (described in the section below) to predict the wait time for the appropriate input time using the most recently trained and saved neural network model.

We will retrain our neural network once each week outside of business operating hours with all of the available data in the SQL database since it takes around an hour to train. The backend code reformats each `WaitTime` model object into the right inputs and output to the neural network as necessary. Any data more than one month old is discarded by the backend system by simply deleting it from the SQL database in order to avoid overfitting our ML model or using any stale data.

Overall, our web application server is hosted on AWS as an EC2 instance. We used a `t2.medium` instance type in order to have enough memory and space for all the necessary libraries and the ML model. In addition, our domain name (`waitr.link`) is mapped to our server using Route53. We were also able to encrypt our web application with SSL using `certbot`, which means that all of our traffic is routed to `https` to make it more secure.

D. Machine Learning Backend Component

To design our neural network, we are using the `tensorflow` Python library. This allows us to choose every piece of the network, including the dense layers, optimization function, learning rate, loss function, and evaluation metrics.

Our neural network model design is as displayed in Fig. 11. We chose to utilize two hidden layers because our mapping is not linear and therefore needs at least one, but it is also not complex enough to require very deep learning. In addition, two or fewer layers is often enough for more simple datasets [4].

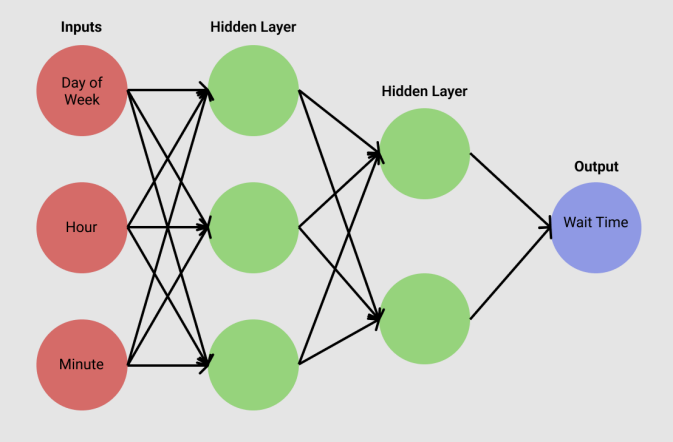

Fig. 11. Neural network design.

Our data flows through the neural network using three inputs: the day of the week (mapped from 1-7), the hour of the day (mapped from 0-24), and the minute of the hour (mapped from 0-60). After passing through the hidden layers, the network will have one output representing the wait time. There are many rule-of-thumb methods to determine the number of hidden neurons [4], so we decided to start with three neurons on the first hidden layer and two neurons on the second hidden layer. After testing, we kept this design as it was the most accurate.

Since we need our neural network to predict a single numerical value, it represents a regression with no activation function. Thus, we chose mean squared error (MSE) as the loss function since it fits best with regression models in terms of guiding the optimizer function based on the loss output. For the optimizer, we chose a gradient descent algorithm derivative algorithm called Adam that adds fractions of previous gradients to the current gradient [5]. We decided to use this optimizer because it is computationally efficient and requires little memory usage. We initially chose a learning rate of 0.001 to start since this is the default value. However, after some initial testing, our loss function yielded a very high value. In the end, we chose a slightly higher learning rate of 0.01 for a quicker pass of the data and for more accurate training since our output is rounded to integer values.

When fitting our neural network model, we need to determine the right number of epochs to avoid overfitting. We started with 100 and checked the loss function output as discussed in testing. We did not end up changing the number of epochs. We chose 32 for our batch size because it typically leads to the best results [6].

VII. Test, Verification and Validation

We performed several testing methods to evaluate the design implementation and make comparisons to theoretical design trade-offs. These tests both validated and verified our final solution.

A. Results for Power Source Design Requirement

As we developed our system, the power source design requirement became obsolete. This is due to the fact that our

circuit input must be powered by a 125 kHz, 5Vpp sine wave and includes an op amp, which requires +/- 5V of DC power. These requirements mean that we must have a custom power unit, which can convert AC power from a standard wall outlet to these two power sources. For demonstration and testing purposes, the circuit is currently powered by one ADALM, plugged into a laptop, which meets these requirements. Because the laptop can be plugged in, this technically puts our battery life at an infinite time, thus making this test null.

B. Results for Extracting RFID Bit Code Requirement

To test how well our system is able to accurately extract an RFID bit code, we broke the connection in two places. First, we broke the connection from the circuit to the Arduino, and instead sent the output to an oscilloscope. At first, the signal could not be seen, but after troubleshooting our circuit and the lab equipment we were able to see the bit code encoded in a square wave while an RFID tag was being scanned (Fig. 12).

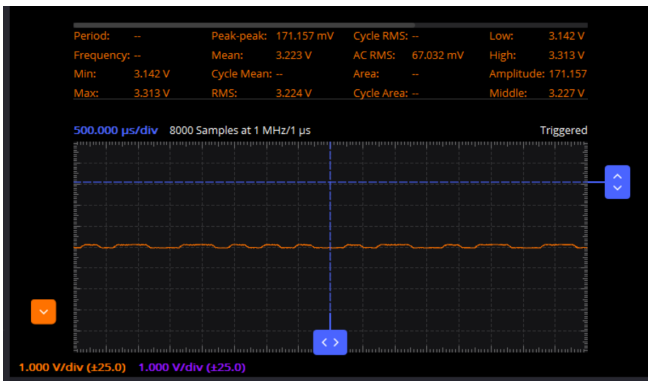

Fig. 12. RFID bit code displayed on an oscilloscope.

We then broke the connection from the Arduino to the RPi, and instead of sending data to the RPi, printed it to the serial monitor (Fig. 13). This showed the bit tag in hexadecimal upon scanning a tag.

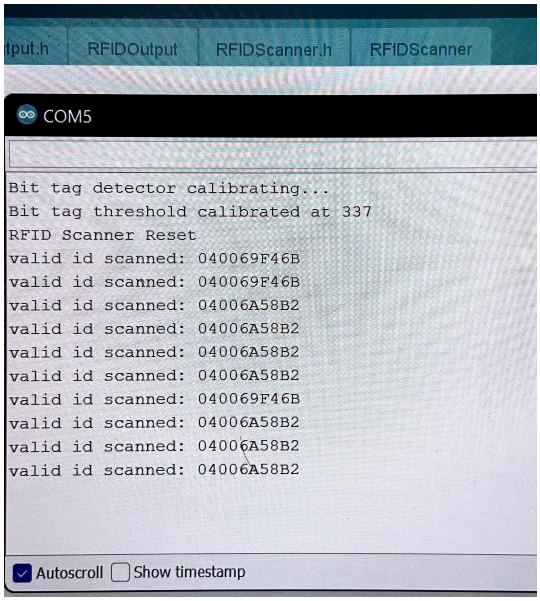

Fig. 13. Bit code, in hexadecimal, displayed on the Arduino serial monitor.

To prove that the circuit was able to successfully and accurately send the code to the Arduino, and the Arduino was able to convert the analog signal to digital, we compared the value to that from a commercial RFID scanner, and found ours to be correct. This shows that we are able to reliably track ID values, which is integral to our system.

C. Results for Transmission from Scanner to Server Requirement

Since we wanted our scanners to be usable almost immediately after each scan, we needed to ensure that the time of transmission from the RFID reader itself to the web server was under 2 seconds. To verify this design requirement, we recorded the start time as the exact time when the ID was scanned by the RFID reader, and we recorded the end time as the exact time when the POST request from the RPi to the web server was completed. Subtracting the start time from the end time yielded the full transmission time.

We performed the above test 50 separate times by printing the transmission time to the console to give us 50 data points to verify our solution. The results can be seen in Fig. 14 below.

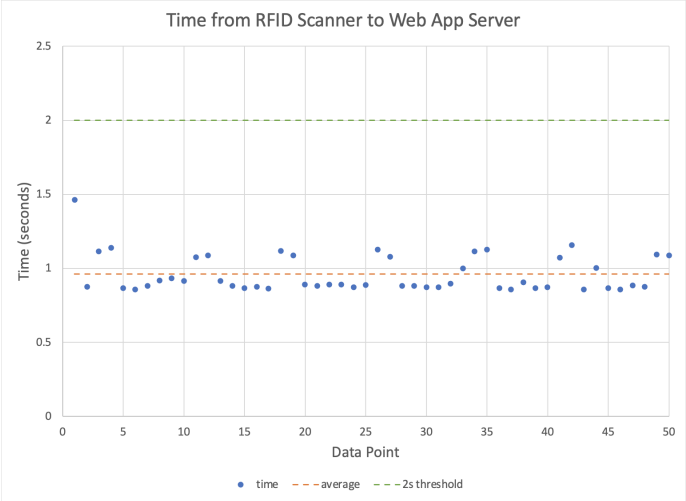

Fig. 14. Graph of 50 separate tests of scanner to server transmission time.

As shown in the graph, every single test resulted in a total transmission time of less than 2 seconds, and the average time was 0.96 seconds. Thus, we were able to meet our design requirement of a quick time of transmission from the scanner to the server, ensuring that our scanner would be readily available after each scan for any user.

Although these results are promising and meet our requirements, there is still potential for breaking through this limitation and lowering the average transmission time by altering our system. Raspberry Pi computers are typically seen as fast, there is still a difference between the models that impacts speed performance. The RPi that we used is an RPi Zero W, which has a 1 GHz single-core processor. In comparison, an RPi 3 B has a 1.2 GHz quad-core processor, which is faster and can handle larger data. Therefore, utilizing

a newer model of RPi would likely make the transmission time to the server faster.

Another potential speed improvement has to do with the hosting location of our web application entirely. Due to prior team experience, we decided to host our server on AWS. However, if the server was hosted on the RPi itself instead of having the RPi send a POST request to the server, the transmission time from the scanner to the server would likely improve since there would be less steps involved to save the scan data to the proper table in the database.

If we were given unlimited time and resources, we would adjust our system to implement the two changes discussed above to break through our original limitation of the transmission time being less than 2 seconds. With these changes, it is possible that the requirement could be changed to always less than at least 1.5 seconds, but maybe even 1 second.

D. Results for Web Application Response Time Requirement

To ensure the web application does not appear slow, we printed the time it took for the page to reload with new data based on updates to the data. Similar to the transmission tests, we recorded the start time as the exact time when the exit scan was performed on the RFID reader, and we recorded the end time as the exact time when the live value changes. This was done by checking the live time from the backend every 25 milliseconds within a 4 second interval. When the live wait time value updates with a different value, the exact time of that event is printed. Subtracting the start time from the end time yielded the full response time.

We performed the above test 50 separate times by printing the response time to the console to give us 50 data points to verify our solution. The results can be seen in Fig. 15 below.

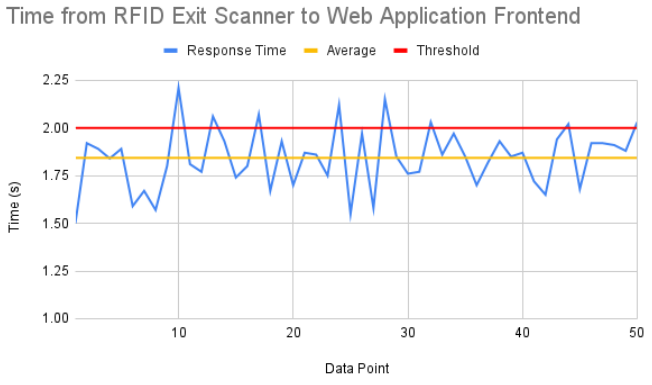

Fig. 15. Graph of 50 separate tests of web application response time.

As shown in the graph, the average time was 1.84 seconds, satisfying the design expectation of under 2 seconds and the use-case requirement of not appearing noticeably slow for users. While there are some spikes above the 2 second limit, it is the average that we care most about, as that shows us overall system performance.

There are a few ways to decrease this time. As mentioned above, there are 2 main changes considered to decrease transmission time, which accounts for almost half of the response time. Beyond that, we could speed up our transmission from the server to the frontend by reducing the number of queries being made, defining more asynchronous processes, and essentially optimizing the way we wrote our code.

E. Results for Input Validation Requirement

To ensure no invalid data entered our database and affected our wait time accuracy, we implemented input validation and conducted validation testing. We ensure form and scan times had already passed in history, that times were during Exchange business hours, and that the entry time occurred before the exit time. We wrote unit tests that considered an extensive possibility of edge cases, and updated our code with any failed tests. We did this process iteratively until we had a 100% pass rate.

F. Results for Database Size Requirement

The maximum capacity for a SQL database is 524,272 terabytes according to the Microsoft specifications, which equates to billions of rows. Even with a limited server size of 16 GB based on the EC2 instance we are using, we are able to store at least 50 rows in our `Entry` table. We validated this requirement of being able to keep track of up to 50 patrons by scanning as many IDs to store them in the database. After populating the table with these values, there were no errors with handling all 50 data points, resulting in the ability of our database to hold all necessary data at once.

VIII. Project Management

This section describes our project schedule, team member responsibilities, bill of materials with budget, AWS usage, and risk management.

A. Schedule

Our Gantt Chart can be found in Fig. 18 at the end of our report. Milestones include completion of the hardware and software components, combining the components, completing final testing, and presenting our final design at the project demos.

Our schedule has changed from the schedule we had in our design report. We replaced the soldering step with the quicker task of assembling the circuit on a Proto-Perma Board, as this would still allow us to show what the final circuit would look like soldered. We also had to remove the exploration of execution in an on-campus eatery, as our system was not as compact as we had hoped due to issues with providing the necessary power to the system. We were able to otherwise complete the assembly and testing of our system on time, resulting in a successful project and demo.

B. Team Member Responsibilities

We essentially had one team member working on each subsystem individually until it was time to connect them together. The hardware subsystem was completed by Sam,

which included planning, creating, and further developing the inductor coil, demodulating and filtering the circuit, and working with the Arduino Uno. The software component will be split between Dina and Sophie. Dina completed the software web application frontend, while Sophie worked on the software web application backend, including the ML component and hosting on AWS. Dina and Sophie also worked together to connect Sam's circuit to the RPi and send POST requests from the RPi to the web server.

As each subsystem was developed, we worked together to ensure that the connection between them functioned properly. All members worked together to complete deliverables, including our reports, presentations, final video, and demo.

C. Bill of Materials and Budget

See Fig. 17 at the end of the report for a table of parts and costs.

Items highlighted in gray were not used in the project; this includes an RFID Power Source (as it did not come in time) and 3D-Printed Hardware Casing (as we did not have time to solder a smaller version of our circuit).

Items highlighted in blue were not originally planned for; this includes an ADALM (which was used in lieu of the RFID Power Source) and a Perma-Proto Board (which was used to create the model of our soldered circuit. We did not know this modern type of soldering board existed before doing research).

D. AWS Usage

Our AWS credits were used to host our web application on an EC2 instance. We needed to utilize a `t2.medium` instance type to fulfill the required amount of space and memory for our database and our ML model. We also used Route53 for hosting our domain name and connecting it to our EC2 instance. By the end of the project demo, all $50 of credits we were given will be used. We would like to thank AWS for these credits and giving us the ability to easily host and manage our web application.

E. Risk Management

Our main project risk was our hardware component; adapting the design of the 18-220 RFID lab to reading a real CMU ID was a tricky task. In the lab, cards encoded with bit tags and power from waveform generators were used. However, for a real-life application, we did not want to have these - we instead wanted to read CMU ID cards that we did not know the resonance frequency of, and AC or DC power. Unfortunately, both of these risks required workarounds. We found that CMU ID cards do not resonate at the most common RFID frequency of 125 kHz, and even after some research, had no real way of ascertaining their resonance frequency. We mitigated this issue by pivoting and instead using the 220 lab cards as our "IDs". This allowed us to continue building and testing our circuit at 125 kHz, and to have many data points - rather than just testing with our three CMU IDs, we now had many cards to really test our system with. The problem of powering our circuit and devices also became an issue as we designed our solution. Although our RPi and Arduino could be powered with standard AC power from an outlet, the circuit had components that required specialized sources. The resonance stage required a 5 Vpp, 125 kHz sine wave, while an op amp in the circuit required +/-5 V DC power. While testing, we relied on lab equipment, but this was not a sustainable solution for even the demo. We mitigated this issue by ordering an ADALM, which can act as an oscilloscope, signal generator, and power supply when connected to a laptop. This allowed us to have a portable system for testing and demonstration purposes. It also allowed us to resolve an anticipated risk: much of the equipment in the labs worked improperly or was not calibrated. This caused our system to fail to read the RFID cards, even when it was constructed properly. The ADALM solved this problem by providing a reliable way of powering and testing the circuit.

Another unanticipated risk we faced was with our software component. As we built our website, we worried about it being hard to tell the difference between live (from RFID/input data) and predicted (from our ML model) wait times. We solved this issue by listing two separate, labeled wait times on the site, so that users can use both forms of data to make the decision that is best for them.

IX. Ethical Issues

One potential ethical issue with our project is the fact that when ID numbers are sent from the RFID reader to the web server and stored in the backend database, they are not encrypted. With our current implementation, this does not really present a problem because we are using ID cards from the 18-220 lab that do not contain any personal information. However, if we were to adjust our circuit to be able to read CMU IDs instead, then our web application would be holding onto unencrypted personal information after the entry scan and before the exit scan. By implementing an encryption scheme, this information could be kept safe from anyone trying to access it since it would no longer be exposed.

Another ethical issue is potentially knowing how many people are in an area based on the wait time shown on our web application. Especially since we display a live wait time on the home page, someone could determine how many people are in line and thus how many are in the restaurant based on the time shown. This is sensitive information, but it would be available to anyone who checks our web application. To mitigate this issue, we could require users to sign in using SSO through CMU to attempt to limit who has access to our web application.

In addition, although we have input validation on the form on our web application, there is still a chance that someone could put false times into the form. Since we do not have any tracking of who uses the web app since we do not make anyone login, we would be unable to tell who put the false times into the form. These inputs would directly impact our ML model since the data would be used to train the neural network, which would then begin to affect our users since the predicted wait times would change. Mitigating this issue could involve validating form input times with the ML model

prediction or the current RFID reader input data to see if they match closely enough.

Finally, one more ethical issue with our project is our ML model predicting noticeably incorrect wait times. This would quickly cause us to lose users since they would no longer trust our product. We could mitigate this issue by continuing to audit the ML model as well as by retraining it with the most accurate to live data that we can.

X. Related Work

Work related to our project includes the 18-220 lab that our circuit design was adapted from. While only experimental in nature, the goal of this lab is to construct a working RFID reader, which is a key feature of our own project's design.

Other related work includes other forms of occupancy sensors; while our project uses RFIDs to count the number of people in line, this can also be achieved through imaging programs like ComputerVision. While other methods may be trickier to implement, they can eliminate the need for crowdsourced data, thus potentially improving accuracy and user experience.

In addition, this project could also be achieved through a purely software implementation where users input their wait time manually or their entry/exit time manually. Although this application could yield somewhat accurate results, having real-time data through the RFID scanners improves the accuracy immensely.

Another piece of related work is a study predicting patient wait time in a queue in the emergency room using deep learning algorithms [7].

XI. Summary

Our system was able to meet our design requirements by providing a solution for the very prevalent problem on campus of unexpected long waits. We successfully built a system that accurately collects wait time information, publishes that information for users to view, and provides estimated wait times based on accumulated data. On both the hardware and web application fronts, our system is easy to use, intuitive, and efficient. The main limit on our system's performance is its lack of use in the real world. As a system whose data is crowdsourced, it is difficult for our ML model to learn trends and build in accuracy without real use in a line. With more time, we could consolidate our hardware by soldering and boxing the circuit in order to have a simple interface for users to scan in on. We would adapt our circuit to the frequency of CMU IDs, after researching which frequency they resonate at. Finally, we would create a custom power unit so that our system could be powered by one standard AC power cord. On the software side, we could improve this project by building a framework that could easily be reused for other waiting lines. They could customize business hours and reuse components of our ML model to easily set up the software for another line.

We learned many lessons throughout the development of our system. For one, we learned the hard way to always test your equipment! Weeks were spent debugging bugs that did not exist simply due to faulty equipment, and this time could have been used to create another custom board, or begin soldering.

On the software side, we learned that training even a simple ML model takes a great deal of time. In addition, it is very important to coordinate well with anyone else involved in writing code so that all components can be connected properly and so that there are no issues with merging.

Glossary of Acronyms

RFID – Radio-frequency identification
ML – Machine learning
RPi – Raspberry Pi
MCU – Microcontroller Unit
Adam – Adaptive moment estimation
SQL – Structured query language
XSS – Cross-site scripting
MSE – Mean squared error

References

[1] EMI Analyst Software, *Circular Loop Inductance*, Accessed on Mar. 2, 2022, [Online]. Available: https://www.emisoftware.com/calculator/circular-loop/

[2] Microchip Technology, Inc., *microID® 125 kHz RFID System Design Guide*, Accessed on Feb. 28, 2022, [Online]. Available: http://ww1.microchip.com/downloads/en/devicedoc/51115f.pdf

[3] Image from The Robotics Back-End: *Raspberry Pi Arduino Serial Communication – Everything You Need To Know*, Accessed on Mar. 2, 2022, [Online]. https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/

[4] Heaton, Jeff. Heaton Research. *The Number of Hidden Layers*, Accessed on Feb. 23, 2022. [Online] Available: https://www.heatonresearch.com/2017/06/01/hidden-layers.html

[5] Kingma, Diederik P. and Jimmy Ba, arXiv, *Adam: A Method for Stochastic Optimization*, Accessed on Feb. 23, 2022, [Online]. Available: https://arxiv.org/abs/1412.6980

[6] Masters, Dominic and Carlo Luschi, arXiv, *Revisiting Small Batch Training for Deep Neural Networks*, Accessed on Feb. 23, 2022, [Online]. Available: https://arxiv.org/abs/1804.07612

[7] Hijry, Hassan and Richared Olawoyin, IJIEOM, *Predicting Patient Wait Time in the Queue System Using Deep Learning Algorithms in the Emergency Room*, Accessed on May 6, 2022, [Online]. Available: https://www.ieomsociety.org/journals/volume3/vol-3-no-1-3.pdf

Supplemental Figures

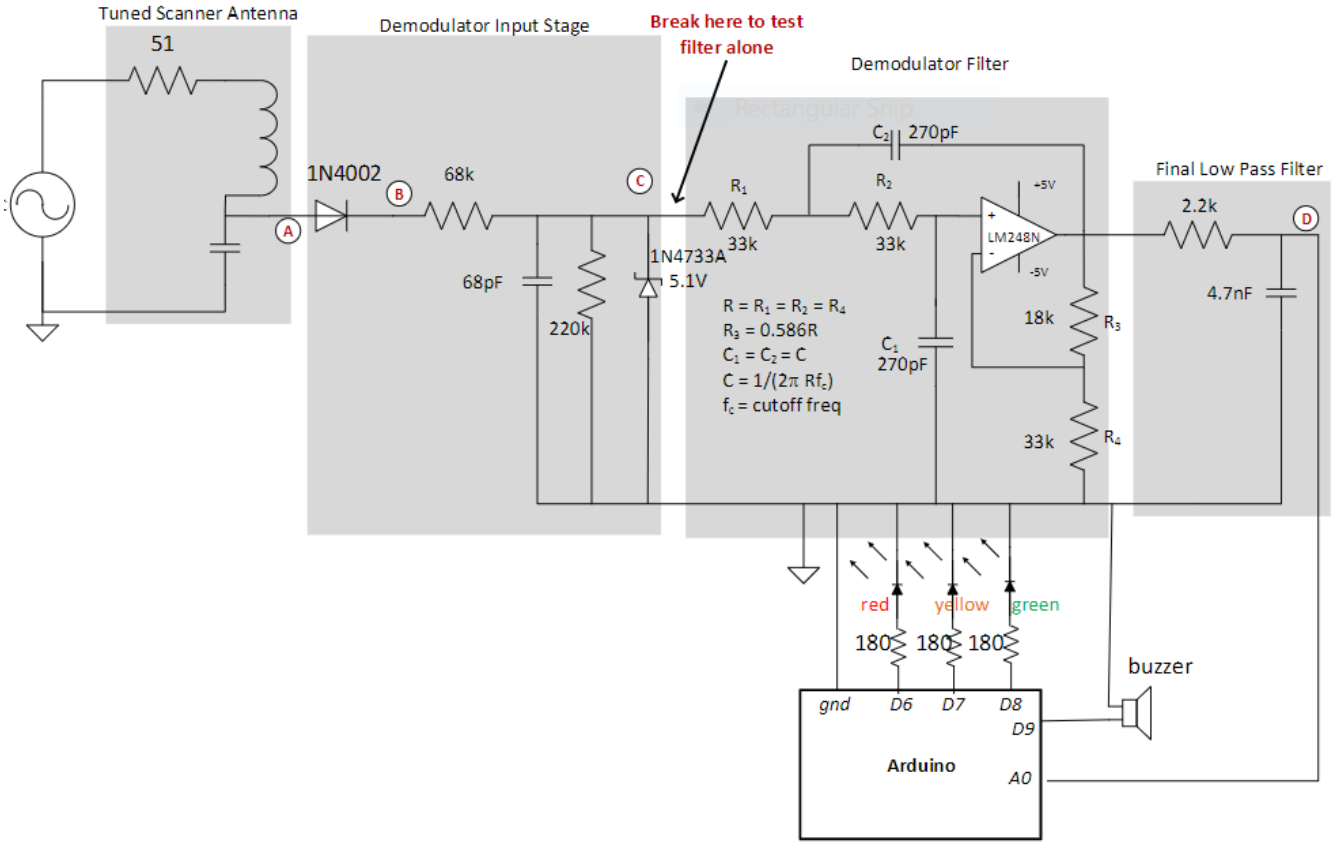

Fig. 16. Schematic of circuit and connected Arduino. Enlarged version of Fig. 3. This image is taken from the 18-220 Lab 4C handout.

Name	Description	Model No.	Manufacturer	Quantity	Cost
Arduino Uno R3 Board	Controls lights, sound on board. Collects bit tag data from circuit.	Uno SMD R3	Arduino	2	$0
Raspberry Pi Zero W 16 GB	Microcontroller unit; transmits data from circuit to web server	Zero W V1.1	CANAKIT	2	$0
RFID Power Source	Converts AC power from outlet to LF 125 kHz 5 VDC 1 amp signal.	PS5PTR	RFID, Inc.	2	$59 each
3D-Printed Hardware Casing	Encloses all circuit components for streamlined design.	N/A	3D printed in Techspark.	2	Variable
LANMU RFID Reader	125 kHz commercial reader used to test circuit & software.	R20XD	Lanmu	1	$0
ADALM	Device that provides oscilloscope, power source functionality	N/A	Analog Devices	1	$0
AWS Credits	Necessary to host our own website, and choose our domain name	N/A	N/A	N/A	$50
Perma-Proto Board	Transition board between the proto-board and soldering	N/A	Adafruit	3	$20

Fig. 17. Bill of materials, including quantities and costs.

Display Week: -5

TASK	ASSIGNED TO	PROGRESS	START	END
Setup				
Set Up Website	Sam, Sophie	100%	1/31/22	2/5/22
Design Solution Implementation	All	100%	1/31/22	2/14/22
Order Parts	Sam	100%	2/14/22	2/21/22
Build Solution				
HW: Circuit on breadboard	Sam	100%	2/16/22	3/28/22
HW: Arduino + Rpi	Sam	100%	3/29/22	4/16/22
HW: Solder (put on PCB board)	Sam	100%	4/20/22	5/4/22
SW: Frontend	Dina	100%	2/14/22	2/27/22
SW: Backend	Sophie	100%	2/28/22	3/13/22
SW: ML	Sophie, Dina	100%	3/14/22	3/20/22
Testing				
HW ability to read ID card	Sam	100%	3/14/22	4/3/22
Ability to transmit from HW to SW	All	100%	4/6/22	4/13/22
Ability to calculate wait times from data	Sophie, Dina	100%	4/14/22	4/21/22
Integration				
Integrate wait times into web app	Sophie, Dina	100%	4/5/22	4/19/22
Present final system	All	100%	4/18/22	5/4/22

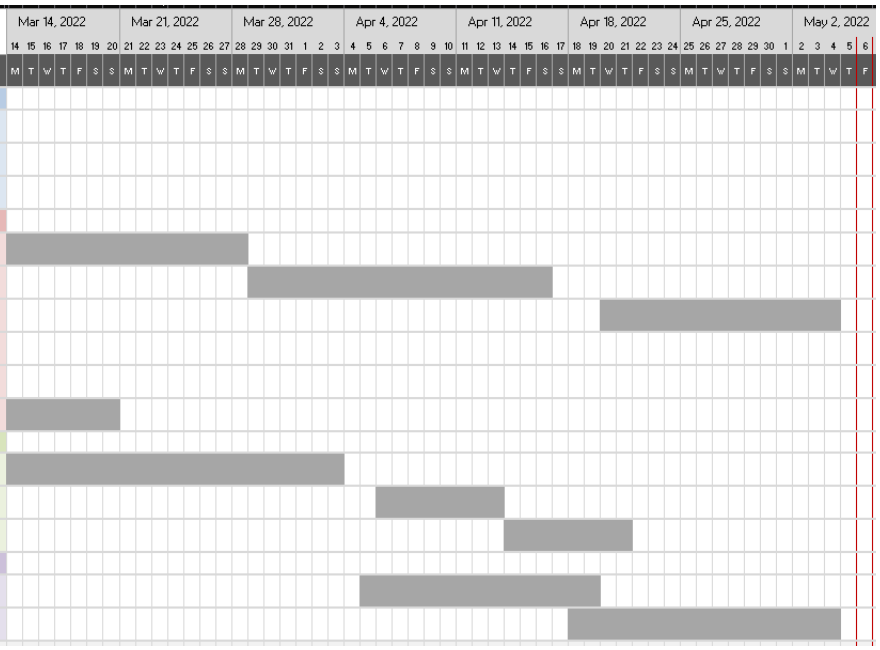

Fig. 18. Gantt chart with all milestones completed.