waitr

Sophie Sacks, Samantha Lavelle, Dina Razek

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—**A system capable of utilizing sensors to measure how many people are in a certain space gives users the opportunity to check an application to see the estimated wait time. Our desired end-product includes two RFID scanners 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 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 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 keep track of up to 50 patrons who are in line. 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, then that user's data point will be discarded. Similarly, if the user is tracked as exciting the line without entering the line, their data point will also be discarded so that our ML model can still accurately predict wait times.

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 will be fed through a machine learning model to then predict wait times in the future. All of this data will be 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 will create two total hardware components to connect to our software since there will be two RFID readers in the line. The main connection between the hardware and software systems are the RPis. The data will flow from these hardware pieces to the software backend.

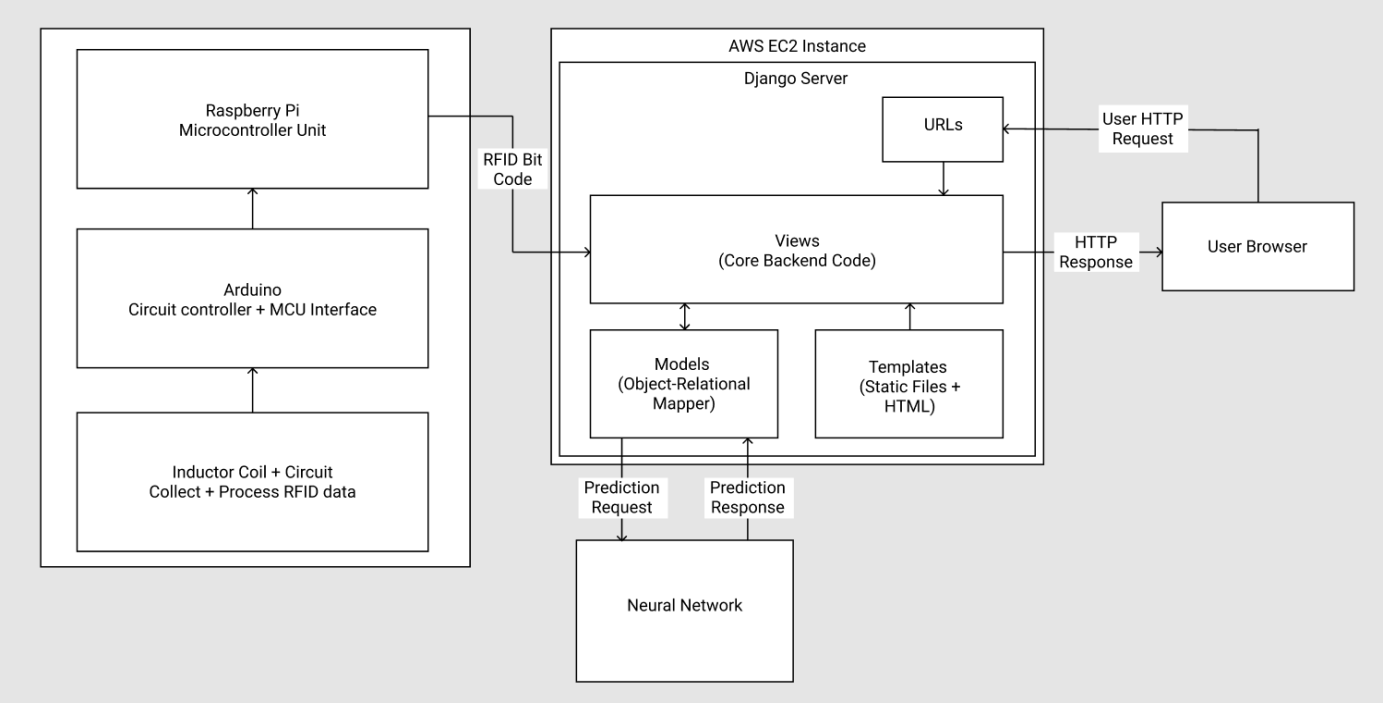


Fig. 1 Block diagram of entire system architecture

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

IV. Design Requirements

Our solution space has many critical constraints that we will use to define success. On the hardware end, we must have a power source that lasts three hours or more. This source may be AC-powered or battery powered, but the design requirement entails a life of at least three hours, as we do not want to add a large burden to the staff who must keep the devices in operating conditions. Three hours gives the staff ample time and opportunity to update the power source in times of low customer count.

We also require transmission from the RFID reader to the server within 200 milliseconds. This requirement is based on the size of the data, the channel bandwidth, and general expectation of transmission delay. On the software end, we require the web application response to be under 2 seconds, as users are not likely to notice the delay. Anything over 2 seconds will be problematic, potentially causing users to leave the application entirely. We also require a reasonable level of security, ensuring corrupt user input does not crash our website and that our data is securely stored and inaccessible to the public. Our web application frontend should be intuitive, responsive, and easy to use.

In terms of solution accuracy, 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. We also require the ability to keep track of at least 50 patrons at a time, meaning that data is stored live for at least 50 patrons in one time window. This number ensures that we will be able to collect and analyze enough data to make an accurate prediction in the time window.

In terms of error mitigation, we want to ensure we disregard incomplete data if at least three other complete data points (entry and exit time) are logged. In other words, we should delete data from our database if we do not receive both an entry and exit time from our user.

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 ($4\pi e - 7$), 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].

Other design choices include which devices will connect to our circuit and transmit our data. The use of the Arduino 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 as our microcontroller unit allows us to send data wirelessly to our web server thanks to facile connection to the Arduino.

B. 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.

C. 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.

D. 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.

VI. System Implementation

Our overall system solution is a combination of hardware and software. On the hardware end, we will be building two RFID readers, placed at the entrance and exit of the line, for users to scan in and scan out. A Raspberry Pi will send 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 will consist 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.

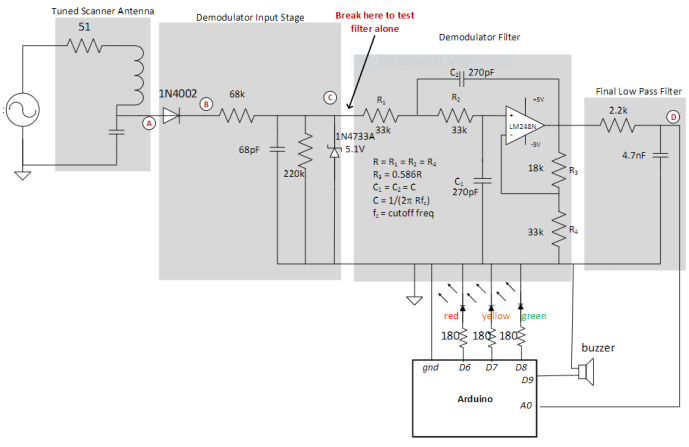


Fig. 2. Schematic of circuit and connected Arduino. Image shown larger in Appendix. This image is taken from the 18-220 Lab 4C handout.

Figure 2 shows our circuit design. 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.

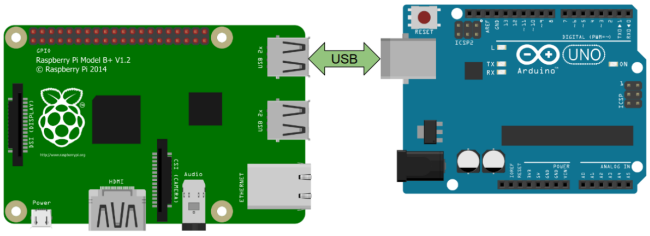


Fig. 3. Connection of Arduino to RPi MCU.

Figure 3 [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.

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 will have the following structure: a base page that all pages will extend (reusable component), a navigation bar that will be included in the base page, and the three pages that represent the home page, about page, and feedback page. Our CSS will be a combination of our own implementation and a Bootstrap template.

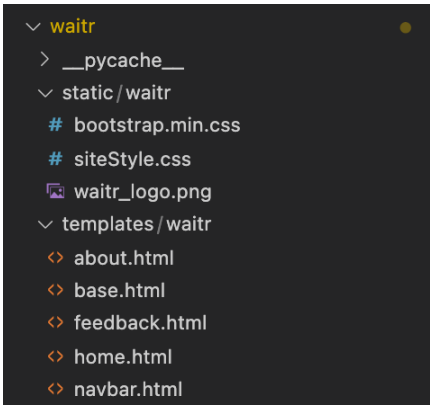


Fig. 4 Depicts the HTML and CSS file structure in our Django framework

When the web application is opened or reloaded, a request is sent to the backend to retrieve the wait time, which is then displayed on the home page. The views.py is responsible for rendering each page and retrieving the wait time.

```python
def home_page(request):
    # render takes: (1) the request,
    #               (2) the name of the view to generate, and
    #               (3) a dictionary of name-value pairs of data to be
    #                   available to the view.
    return render(request, 'waitr/home.html', {})

def about_page(request):
    return render(request, 'waitr/about.html', {})

def feedback_page(request):
    return render(request, 'waitr/feedback.html', {})
```

Fig. 5 `views.py`, which depicts the rendering of HTML pages

For the home page design, we will display the predicted 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.

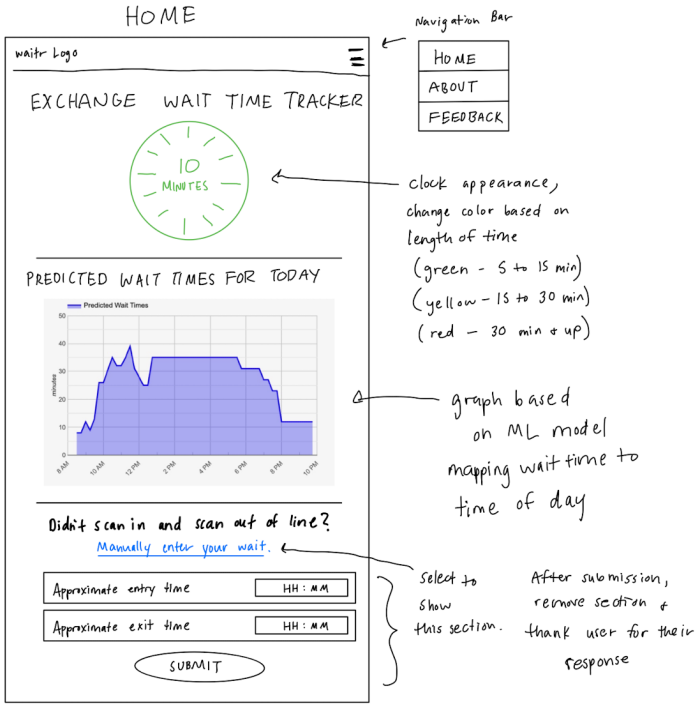


Fig. 6 Depicts the low fidelity design for our web application home page

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. 7, 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. 8). 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.

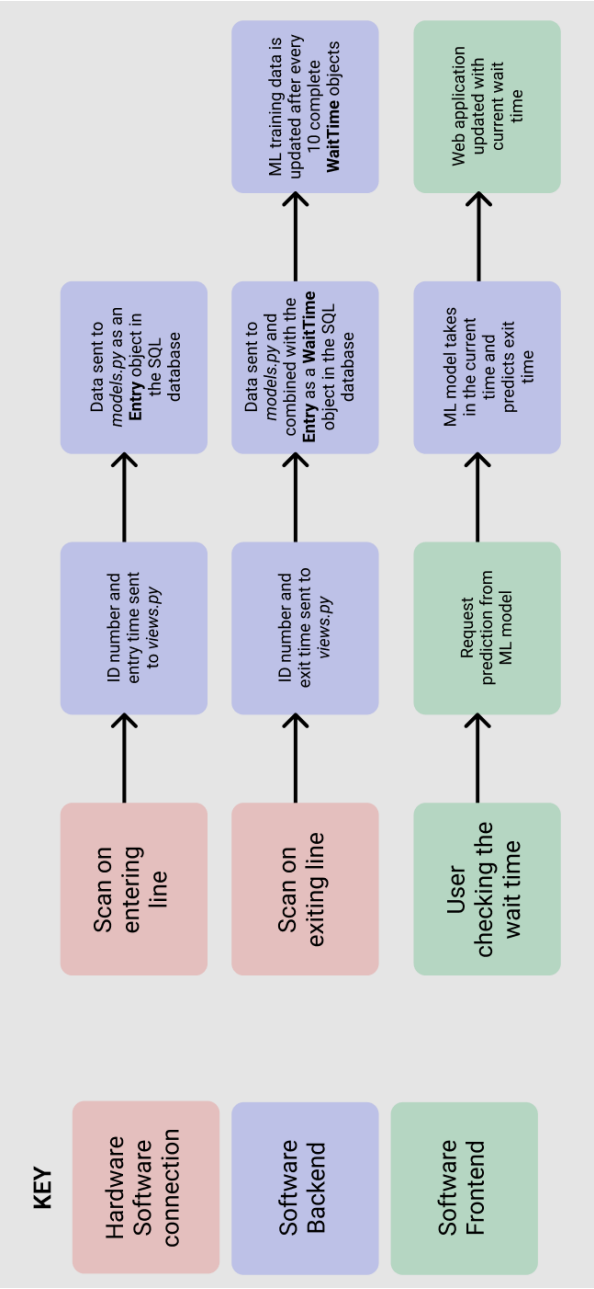


Fig. 7 Software data flow diagram

```
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. 8 `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 will most likely take a few hours 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.

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 will look like Fig. 9. 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].

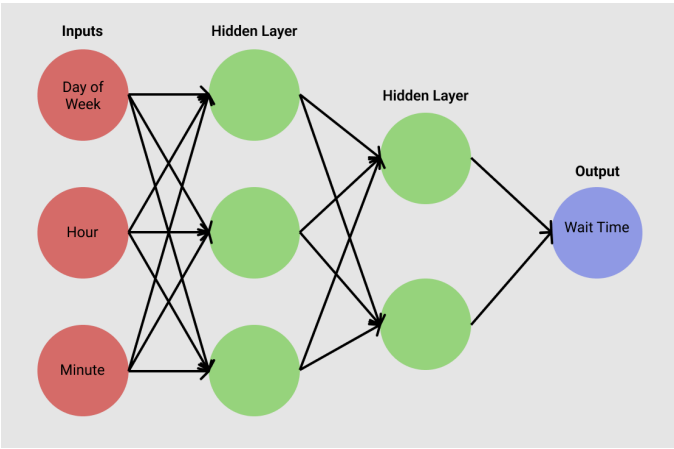


Fig. 9 Neural network initial design

Our data will flow 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. Depending on the accuracy of the model later on, this design could change slightly, but we believe this is a good starting point.

Since we want 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 chose a slightly higher learning rate for a quicker initial pass of the data.

When fitting our neural network model, we need to determine the right number of epochs to avoid overfitting. We will start with 100 and check the loss function output as discussed in testing. We chose 32 for our batch size because it typically leads to the best results [6].

VII. Test, Verification and Validation

We plan to have several testing methods to evaluate the design implementation and make comparisons to theoretical design trade-offs.

A. Tests for Capability of Extracting Bit Code from ID

Our hardware will undergo testing to ensure it is capable of reading an RFID card and extracting its data. We will first test under lab conditions (i.e., with the circuit powered by a waveform generator and using a plain card with a known bit code). Then, one by one, we will introduce real world components: first, powering the circuit with our AC power converter, then testing with a CMU ID card, and finally connecting a Raspberry Pi to the circuit to prepare for wireless data transmission.

B. Tests for Hardware-Software Connection

We need to ensure that the date, time, and ID data from the RPis are properly being transmitted to the Django SQL database. We will need to ensure that we can hold at least 50 patrons at a time. We will perform functional tests to determine if the criteria has been met. We will conduct tests in both laboratory conditions and also in real-life conditions, such as testing within the Exchange, to ensure our data is being submitted. We will perform load tests to ensure our database can hold all patrons.

C. Qualitative Usability Testing of the Web Application

We will perform qualitative usability testing of our web application with a pool of users to ensure the product is beneficial, efficient, and easy to use. We will perform a Think Aloud Protocol where we will ask users to state their thoughts as they use our application. This will give us insights as to which features could be improved.

D. Quantitative Neural Network Testing

We want our machine learning model to be able to predict wait times within 2 minutes of the actual wait time or with a 10% margin of error. To ensure our neural network is within our accuracy bound, we will utilize the loss function to see at what point the MSE is at its lowest. We will also check to see if we can get to our desired accuracy with the current design or if a different design yields better results. Using the `tensorflow` package, we are able to see a variety of metrics by evaluating the model, which will give us more information.

VIII. Project Management

This section describes our project schedule, team member responsibilities, bill of materials with budget, and risk mitigation plans.

A. Schedule

A detailed Gantt chart is labeled as Figure 10 in the Figures section, located at the end of this document. We use this Gantt chart to track milestones and assign them timelines and team members.

Major milestones for our project include: construction and validation of hardware (ability to read RFID bit code), construction and validation of software (ability to calculate wait times from a timed set of ID codes), and construction of the entire system with use-case testing.

B. Team Member Responsibilities

We essentially have one team member working on each subsystem individually until it is time to connect them together. The hardware subsystem will be completed by Sam. The software component will be split between Dina and Sophie. Dina will be completing the software web application frontend, while Sophie will be working on the software web application backend, including the ML component. As each subsystem is developed, we will work together to ensure that the connection between them functions properly.

See Figure 10 at the end of this document for our Gantt chart, which dictates timelines and tasks for each member.

C. Bill of Materials and Budget

See Figure 9 at the end of this document for a table of components and their prices.

All other parts and components are from the team members' personal inventories.

D. Risk Mitigation Plans

For our hardware component, a critical risk is adapting the design of the 18-220 RFID lab to reading a real CMU ID. In

the lab, cards encoded with bit tags and power from waveform generators were used. However, for a real-life application, we will not have these - we instead have to contend with ID cards that we do not know the encoding of, and AC or DC power. We will mitigate this risk by conducting primary testing under lab conditions, and then introducing the real-world components one by one. This will allow us to isolate any sources of error, and find suitable fixes or alternatives. For example, we have acquired an out-of-the-box RFID reader so that we can ascertain what information is encoded on our ID cards before using the cards to test our own custom scanner.

For our software component, we do not foresee any critical risks. We must perform considerable debugging to ensure our web application is performing as expected. We will also need to continuously monitor and recalibrate our ML model to ensure accuracy.

IX. 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.

X. Summary

Waitr is a system that uses a combination of real-time data from RFID readers and knowledge of peak times from a machine learning model to give accurate wait time estimates for on-campus eateries. Thus, it allows the user to make an educated choice when choosing where to get a meal. This is especially useful when the user has only a limited amount of time, as is often the case for busy students.

Upcoming challenges for the system include building the custom circuit needed to read CMU ID cards while being powered by a standard AC outlet. We also must connect all system components - hardware, software frontend, and software backend - and ensure that the data is transmitted quickly and accurately. Our risk mitigation plan will help us reach our goal as we navigate these challenges.

Waitr has great potential to improve the on-campus dining experience for students, faculty, and visitors. The system can be adapted to other locations on campus, such as the UC package pickup window, to create a seamless experience while using campus services.

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

FIGURES

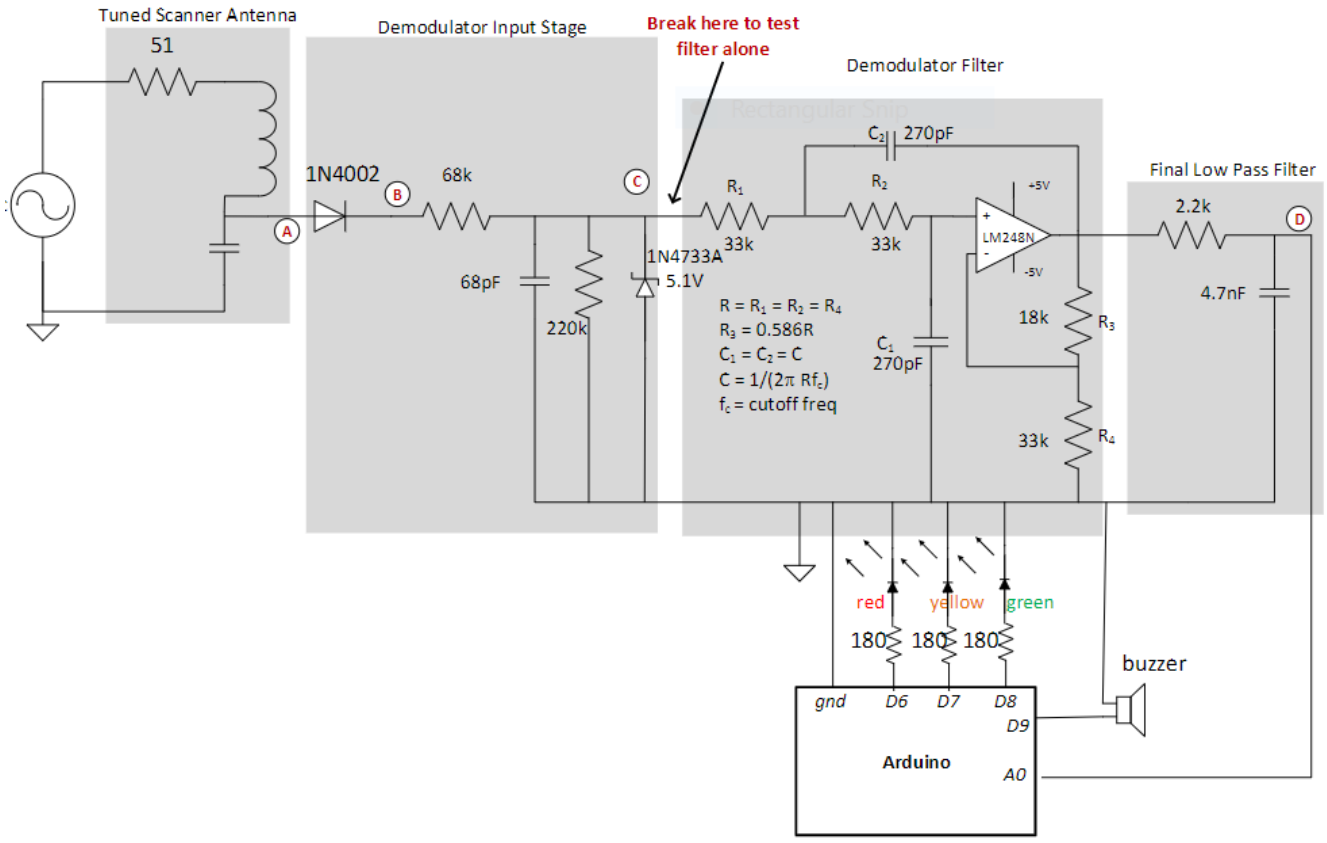
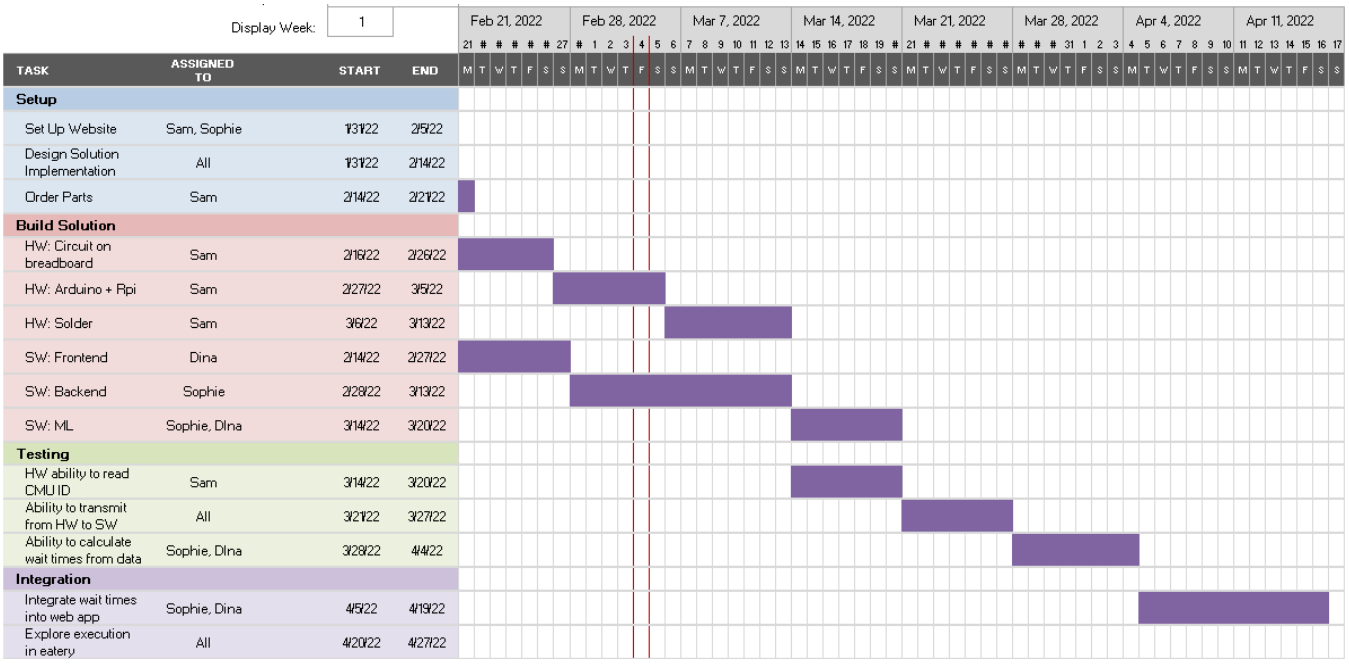


Fig. 2. Schematic of circuit and connected Arduino. This image is taken from the 18-220 Lab 4C handout.

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

Fig. 9. Bill of Materials. Items with a cost of $0 are on loan from 18-500 inventory.



TASK	ASSIGNED TO	START	END
Setup			
Set Up Website	Sam, Sophie	1/31/22	2/5/22
Design Solution Implementation	All	1/31/22	2/14/22
Order Parts	Sam	2/14/22	2/21/22
Build Solution			
HW: Circuit on breadboard	Sam	2/16/22	2/26/22
HW: Arduino + Rpi	Sam	2/27/22	3/5/22
HW: Solder	Sam	3/6/22	3/13/22
SW: Frontend	Dina	2/14/22	2/27/22
SW: Backend	Sophie	2/28/22	3/13/22
SW: ML	Sophie, Dina	3/14/22	3/20/22
Testing			
HW ability to read CMU ID	Sam	3/14/22	3/20/22
Ability to transmit from HW to SW	All	3/21/22	3/27/22
Ability to calculate wait times from data	Sophie, Dina	3/28/22	4/4/22
Integration			
Integrate wait times into web app	Sophie, Dina	4/5/22	4/19/22
Explore execution in eatery	All	4/20/22	4/27/22

Fig. 10 Gantt chart, showing tasks, timelines, and assignments.