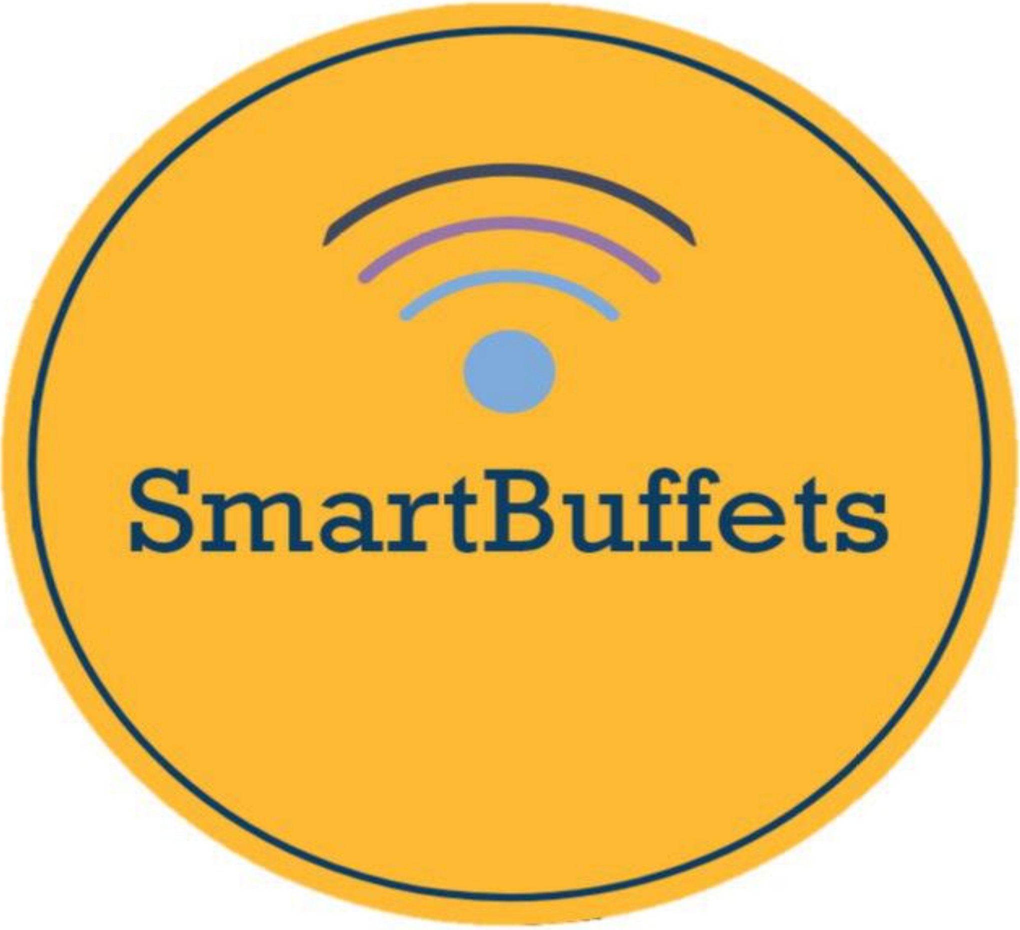
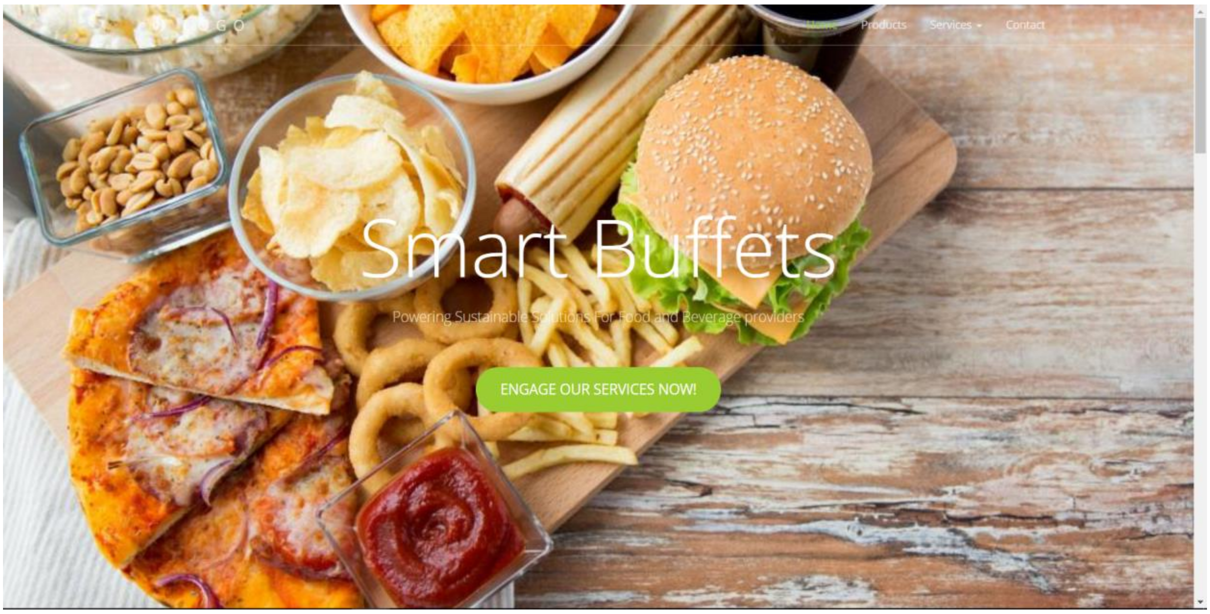
ATOS IT Challenge 2019 Smart Buffets





Smart Buffets

Powering Sustainable Solutions For Food and Beverage providers

Brief Description

An application designed for tablets in kitchens to use machine learning to predict when to start food preparations and hence save food waste. This document is to show how to set it up and the technical aspects of the application.

Hardware

Modules

There are a few libraries or headers used in the Arduino WeMos for it to function properly.

1) <ESP8266WiFi.h> enables the Arduino WeMos to connect to other devices or in this case our server on a local area network.
2) <ESP8266HTTPClient.h> enables the Arduino WeMos to connect to the Internet and websites hosted on it for future scalability.
3) <ArduinoJson.h> enables the creation and transport of JSON packets to and from the device.
4) <HX711_ADC.h> enables the load sensor to function with the WeMOS.
5) <EEPROM.h> enables you to read and write those bytes on the memory in the WeMOS microcontroller which are kept when the board is turned off (EEPROM).

Load Sensor

Our hardware collects the weight of the tray played on it and sends this data to the backend server via Wi-fi or Local Area Network (LAN)

Components:

1) Arduino WeMos D1 R1 (built in ESP8266 Wi-fi module)
2) 20kg load sensor with HX711 load cell amplifier
3) Hardware hookup with addition of acrylic and wood platforms

Load sensor platform: (see images below)

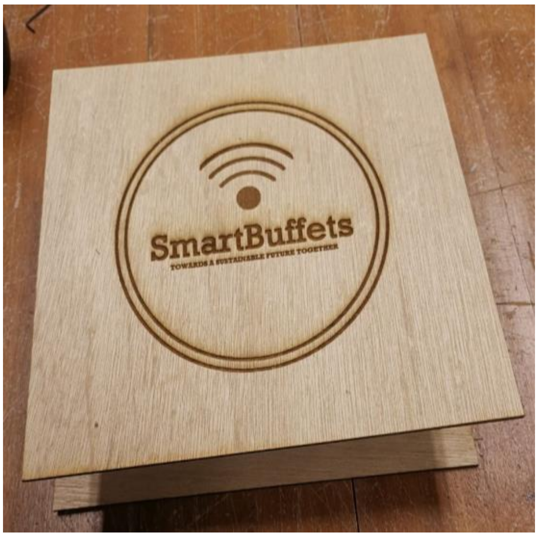


Fig 1: Top View of the Platform

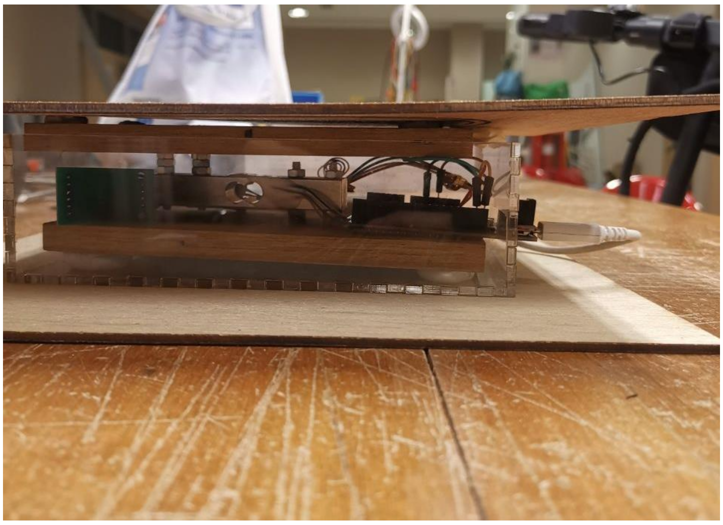


Fig 2: Side View of the Platform showing the Load Sensor

The acrylic and wood platform were cut using a laser printer and then mounted onto and below the load sensors and Arduino WeMos. It should be noted that this platform is for aesthetic purposes and we would be using metal trays to hold the food and heat-insulated boxes to hold the wiring and electronic components.

Upon placing a tray on the platform, a signal is generated by the load cell. It is then converted and amplified by the HX711 module before sending to our Arduino WeMos. Our Arduino WeMos, connected to the internet, takes this input, transforms it into a JSON packet and sends it via the internet or LAN (Local Area Network) to an API URL on the backend server.

Getting Started: Backend Development

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

Python version should be below 3.7.0 as Tensorflow is lacking support for Python versions at or above 3.7.0. Tensorflow is required to deploy our machine learning model and generate predictions. If need be, try downloading Anaconda (Python 3.7 version) and installing it on your computer, depending on its OS.

Creating the Anaconda Environment

This is only necessary if your Python version is 3.7.0 or above (Run the command “python –version” on your terminal to check). A guide can be found here.

1) Install Anaconda and make sure it is in your PATH (System Path in Environment Variables settings). Check by executing “conda -V”. If command found to not exist, manually add the system path to Anaconda. A guide can be found here.
2) Use the command “conda create – n yourenvironame python==x.x anaconda” to create a virtual environment with a Python version below 3.7.0 (Our environment is 3.6.8). A guide can be found here.
3) Navigate to the directory where you downloaded the folder for the next step.

Virtual Environment

This is to create the environment where you will be downloading packages needed to execute the program.

1) Ensure your Python is installed with the virtualenv package. If not, execute the command “python -m pip install virtualenv”.
2) Navigate to the directory using the terminal and go to env/Scripts and execute the command “activate”. For example, if you created your virtual environment with the name “venv” using the command “python virtualenv venv” as seen in the next step, you can execute the command “cd venv/Scripts” and then “activate” to activate the virtual environment. You may use the default virtual environment already created which is under the name “**env”**.
3) If need be, create a virtual environment in the directory by executing the command “virtualenv environ_name” and install required packages with “pip install -r requirements.txt”.

Running the Server

1) Ensure that your virtual environment is already set up.
2) To run the Flask Server, assuming the environment is in Windows, execute the command “python main.py” to execute the main file.
3) The server should be up and running after receiving messages from importing Tensorflow.

Getting Started: Frontend Development

Choice of Device

In choosing our device, the key consideration was down to the size of the device. Given that this device is meant to be deployed in kitchens for chefs, we chose tablets due to its large size, making it accessible and convenient for chefs to access the real-time tracking of dishes and notifications.

In 2018, the iPad was the leading tablet of choice for consumers, achieving the highest market share in 34.9% in the tablet market, more than twice its nearest competitor. One of the key advantages of the iPad was its simple user interface and smooth learning curve. Hence, we chose the iPad as the device of choice. This application has been beta tested on an iPad 6th Gen Model.

Setting up the Node Environment

From the terminal, Nodejs and NPM (Node Package Manager) are required to be run the React Native Application.

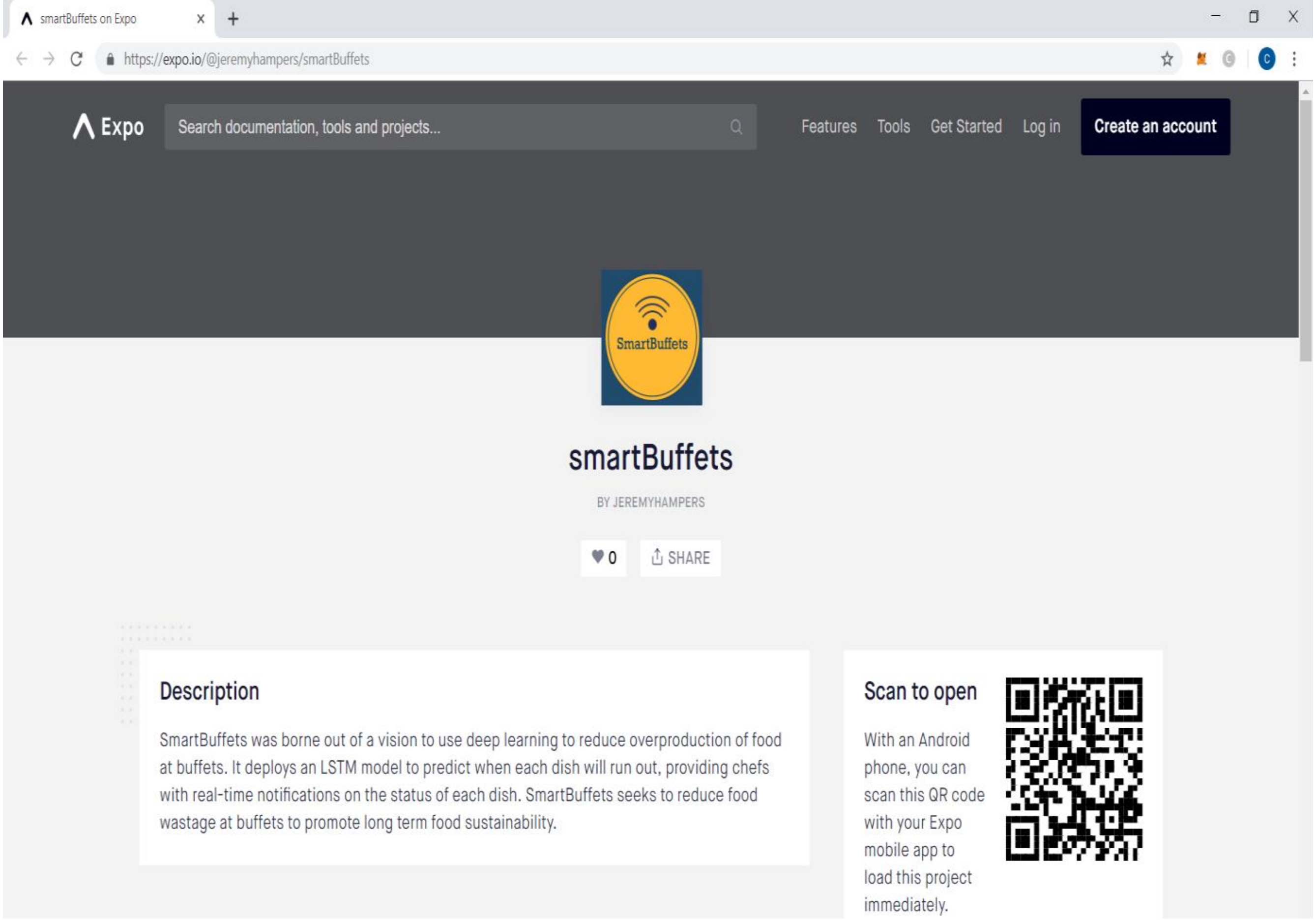
Running the React Native Application

Ensure that your virtual environment is already set

1) Ensure that you are in the directory labeled React-Native
2) Install the dependencies via the “npm install” command
3) To start the application, run the command “npm run start. You should be able to see a new popup window in your browser as shown below.
4) The application can either be deployed on an Xcode Simulator (Select 6th Generation iPad). To deploy it, select the “Run on iOS Simulator” Option
5) Alternatively, the application can be deployed on a physical iPad device by scanning the QR code on the bottom left hand corner. If the device is not connected on the same Wifi Network, the Tunnel option on the left of the Blue LAN button should be selected.

Deploy the Front-End for Testing

1) For user-experience testing, the front-end application independent with simulated data has been published on the Expo App Store at the following link: https://exp.host/@jeremyhampers/smartBuffets as shown in the first image in the next page.
2) To launch the application, the expo app must be downloaded on the iPad. Following that, scan the QR code using the camera scanner to launch it as shown in the second image in the next page.
3) The following images show how the application should look like.



smartBuffets

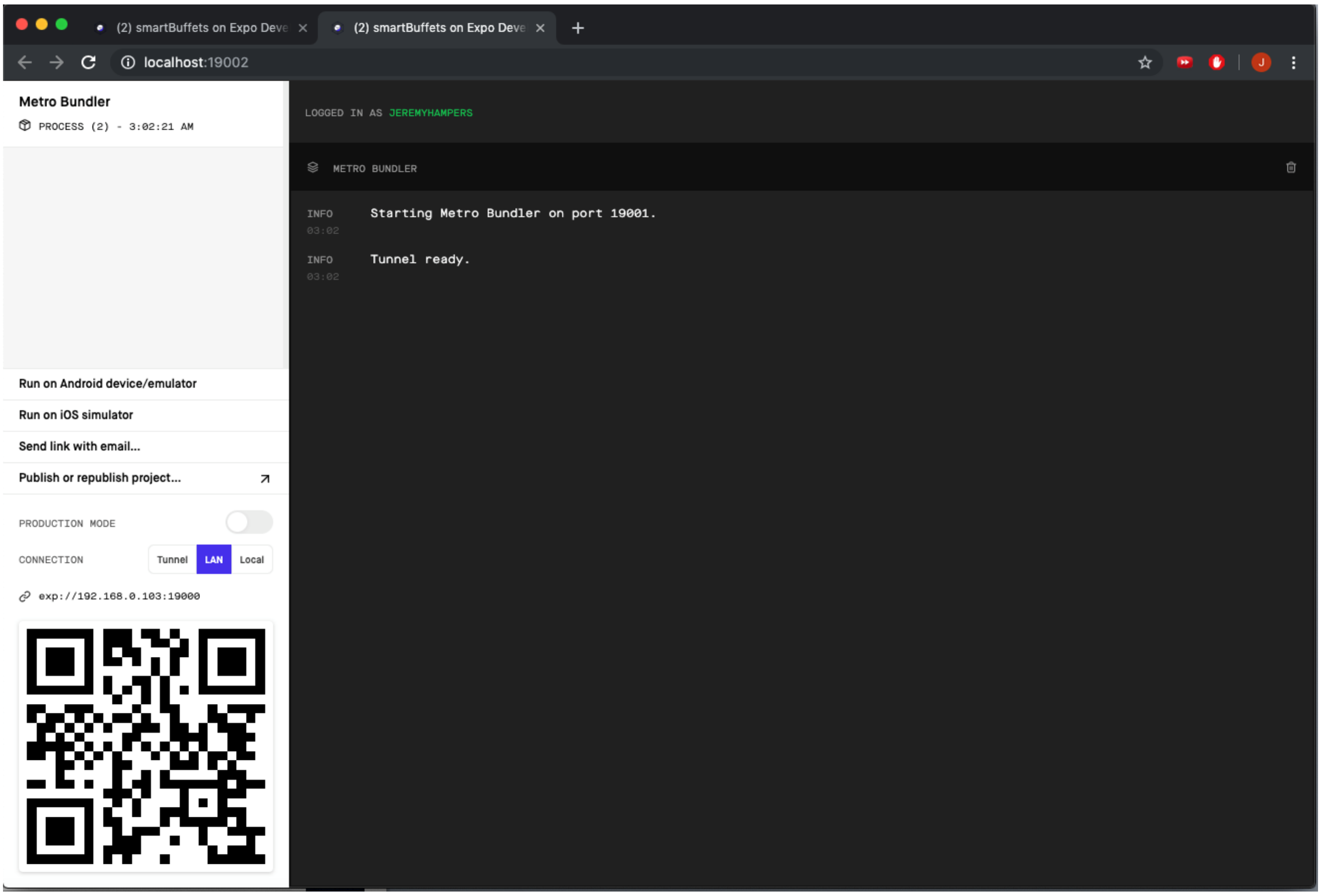
BY JEREMYHAMPERS

Description

SmartBuffets was borne out of a vision to use deep learning to reduce overproduction of food at buffets. It deploys an LSTM model to predict when each dish will run out, providing chefs with real-time notifications on the status of each dish. SmartBuffets seeks to reduce food wastage at buffets to promote long term food sustainability.

Scan to open

With an Android phone, you can scan this QR code with your Expo mobile app to load this project immediately.



(2) smartBuffets on Expo Deve
(2) smartBuffets on Expo Deve
localhost:19002
Metro Bundler
Run on Android device/emulator
Run on iOS simulator
Send link with email...
Publish or republish project...
Tunnel
LAN
Local

PROCESS (2) - 3:02:21 AM

LOGGED IN AS JEREMYHAMPERS

METRO BUNDLER

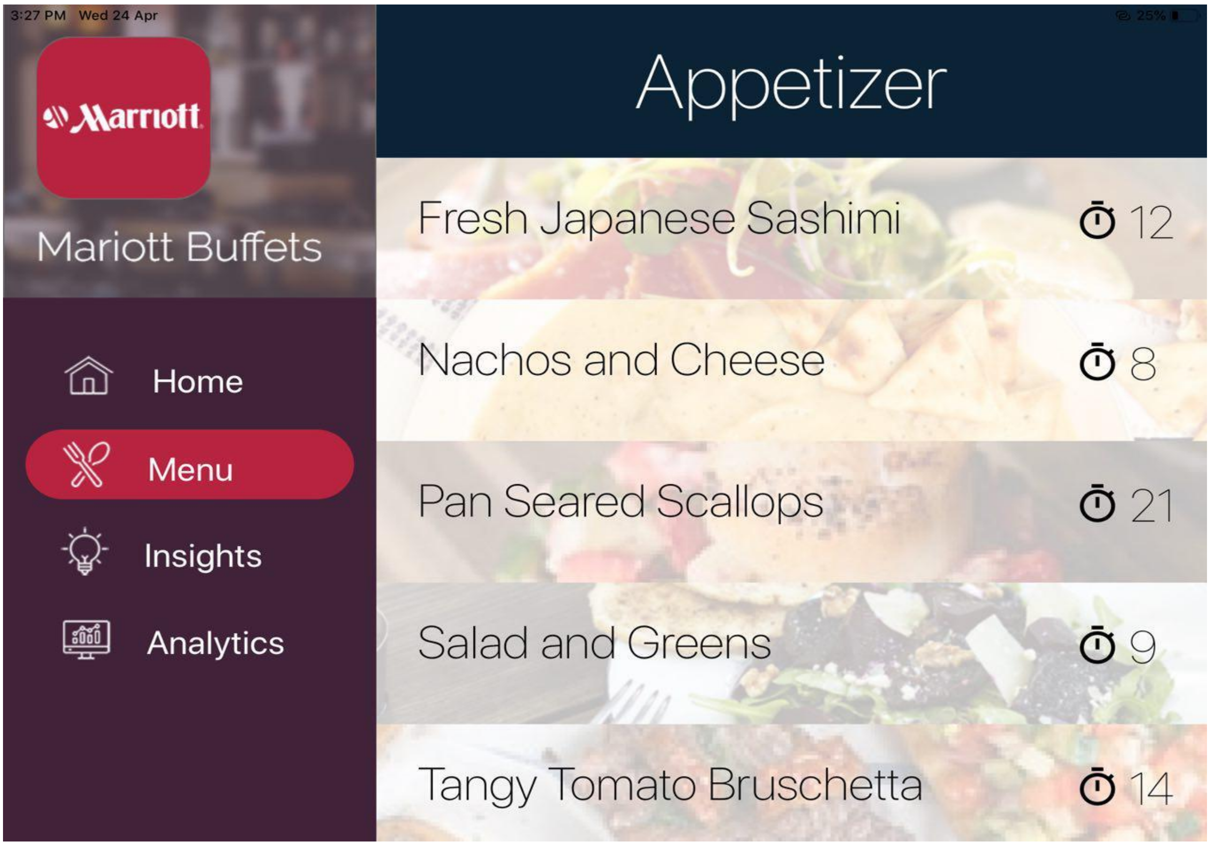
```
INFO     Starting Metro Bundler on port 19001.
03:02

INFO     Tunnel ready.
03:02
```

PRODUCTION MODE

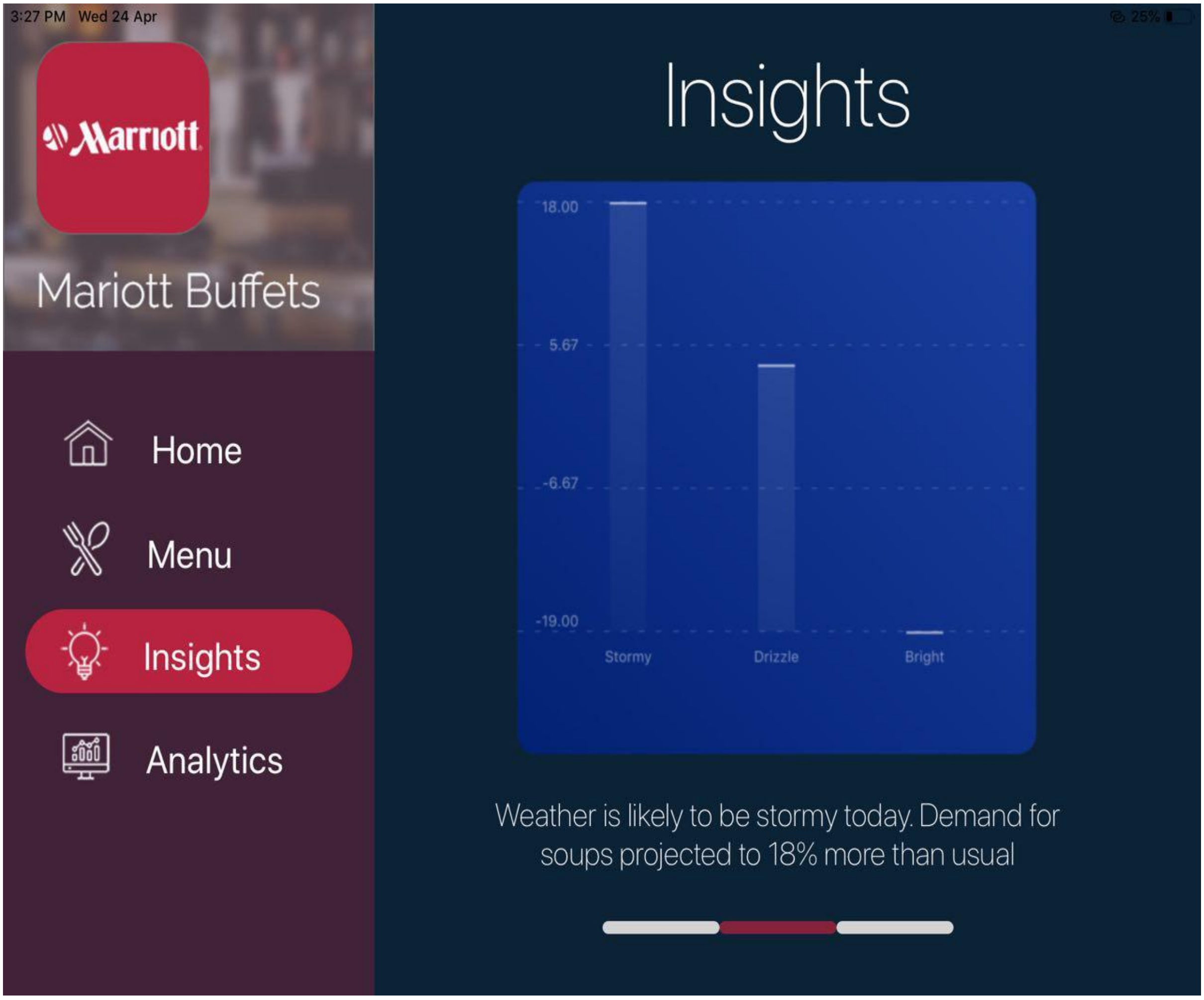
CONNECTION: LAN

exp://192.168.0.103:19000

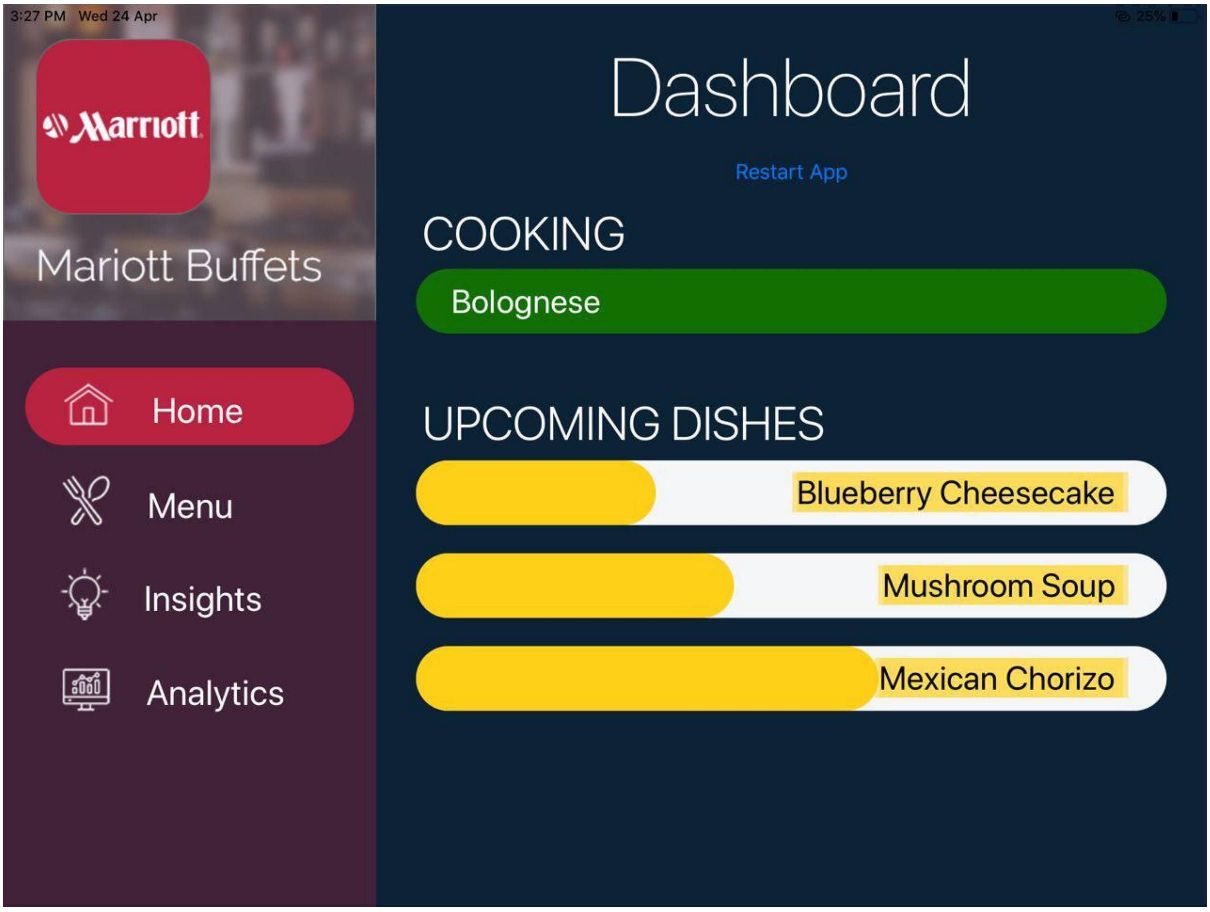


Appetizer

Fresh Japanese Sashimi	⏱ 12
Nachos and Cheese	⏱ 8
Pan Seared Scallops	⏱ 21
Salad and Greens	⏱ 9
Tangy Tomato Bruschetta	⏱ 14



Weather is likely to be stormy today. Demand for soups projected to 18% more than usual



Dashboard

Restart App

COOKING

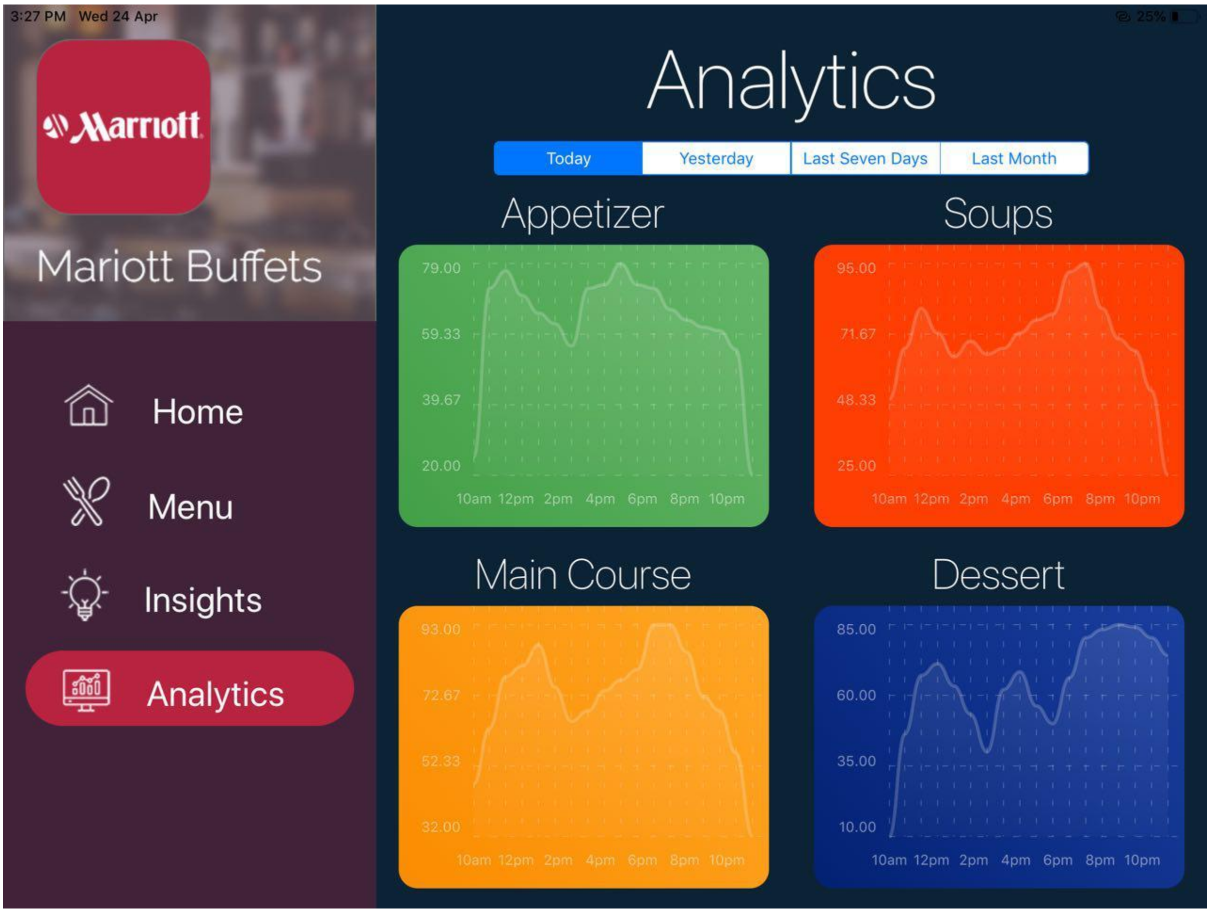
Bolognese

UPCOMING DISHES

Blueberry Cheesecake

Mushroom Soup

Mexican Chorizo



Analytics

Today | Yesterday | Last Seven Days | Last Month

Appetizer
79.00
59.33
39.67
20.00
10am 12pm 2pm 4pm 6pm 8pm 10pm
Soups
95.00
71.67
48.33
25.00
10am 12pm 2pm 4pm 6pm 8pm 10pm
Main Course
93.00
72.67
52.33
32.00
10am 12pm 2pm 4pm 6pm 8pm 10pm
Dessert
85.00
60.00
35.00
10.00
10am 12pm 2pm 4pm 6pm 8pm 10pm

Software Architecture

This section focuses on the back-end server and how it trains the machine learning model, handles the API calls from the front-end and the hardware application.

Modules

Required modules can be found in the list at the end of this document or in the requirements.txt in the directory. The modules imported generally serve three main purposes.

Machine Learning

1) Sklearn was used to split the dataset using the train_test_split function and explore the data in the Jupyter Notebook.
2) Tensorflow or more accurately the high-level API built on it, Keras, was used to build the model using LSTM and GRU.

Data Manipulation and Processing

1) Pandas were used to read the csv files and generate data-frames to be fed into the machine learning model.
2) Other modules like Numpy and Math were imported for data transformations.

Server Running and Processes

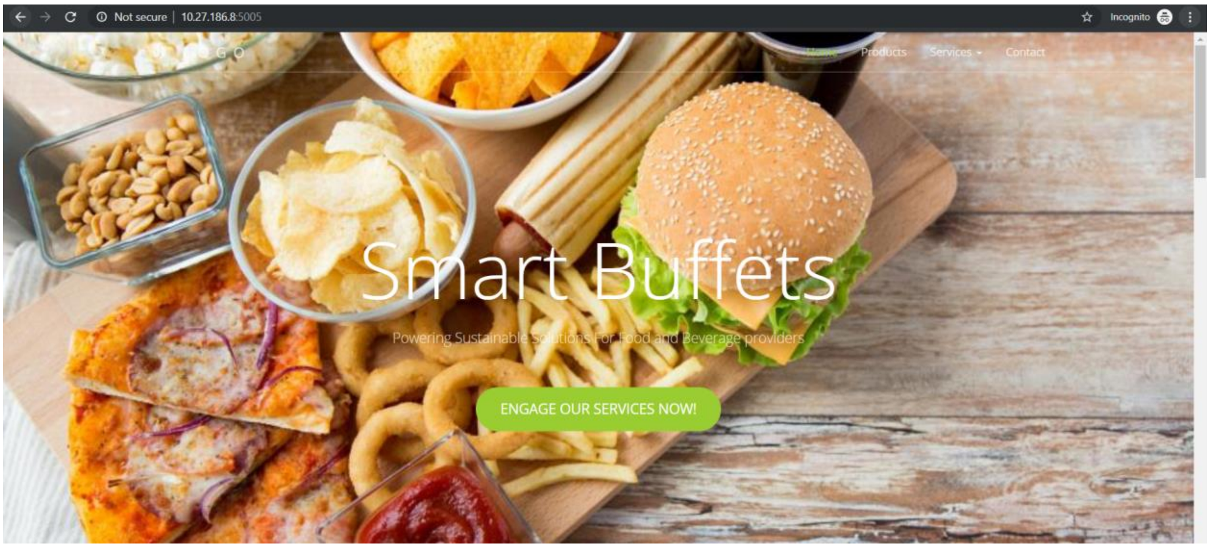
1) Datetime was used to obtain the current time and generate the time for the predicted sequences in the future.
2) Socket was used to obtain the IPv4 address for the Flask Server to run on.
3) Glob and OS were used to access files and check whether directories were empty to ascertain whether files needed to be created for receiving data.
4) JSON was used to receive and transform data into JSON packets to be sent.
5) CSV was used to write and append data to csv files.

Files and Directories

Directories

1) The data directory contains the data files used to process inputs from the weight sensors for different dishes.
2) The env directory contains the virtual environment needed to be set up for the server to run.
3) The predictions directory contains the predictions generated by the model for different dishes.
4) The model_checkpoint directory is where the best model weights are saved from the training.

5) The static and templates directories are where the indexes, CSS and HTML elements of the main web page are stored. This webpage can be accessed by default at http://ip_address:port but it is basically a bootstrapped skeleton which only serves aesthetic purposes as shown below.


Not secure | 10.27.186.8:5005
Incognito
Home
Products
Services
Contact
Smart Buffets
Powering Sustainable Solutions For Food and Beverage providers
ENGAGE OUR SERVICES NOW!

Files

1) Attention.py contains the functions needed to build an Attention-based Neural Network.
2) Prediction.py generates predictions using the model.
3) Training.py contains the functions needed to create and build the model.
4) Requirement.txt contains the list of modules in the virtual environment and needed to run the Flask Server properly.
5) Main.py is the most important file containing the API URLs of the Flask Server.

Training of Model

1) Jupyter Notebooks were used for data exploration and testing of models. Feel free to explore and run them.
2) Attention.py contains the machine models for creating the Attention Neuron and incorporating the Attention Neuron into the neural network.
3) To train the model, execute the training_model function in training.py. Either way, the server will execute a command to train the model after each tray has been refilled.
4) During operation or deployment, the model will be trained on the newest data file from the buffet. It will continue running on the previous model whose weights will be saved in the directory “model_checkpoint”. Once the best model weights are saved from the training, it will begin loading the model with those weights and update the model in operation.

Workflow

1) **Hardware:**

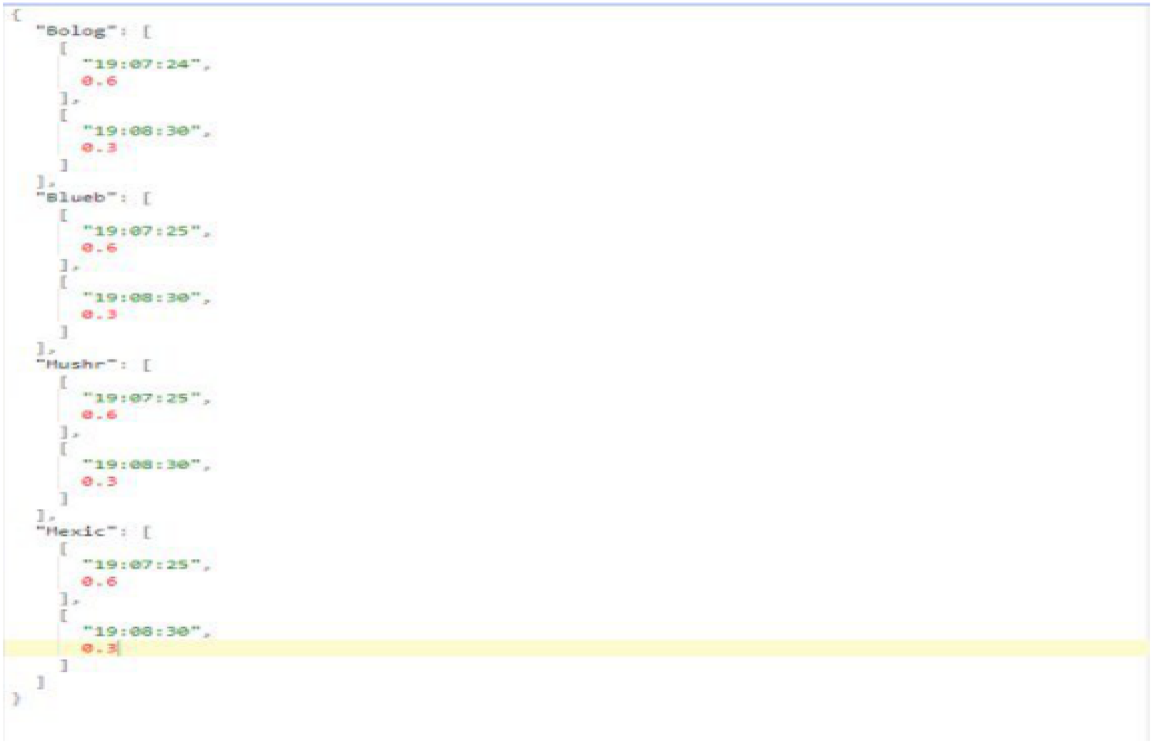
Sample JSON Packet Format: {“Weight”: 100, Food: “Mushr”}

The weight sensor will send POST request to the API URL which is in the format of “http://ip_address:port/post”. The IP address refers to the local machine’s IPv4 address and the port is defined as 5005 by default. The file running the Flask Server, main.py, uses Python’s built-in module socket to automatically obtain the IPv4 address and host the server on it.

The Arduino will send a JSON Packet with the weight detected and food type while the server automatically obtains the time using the datetime module and appends it to the data file in the directory of the food type. If no data file exists in the directory, the server will automatically create a file for it and append the necessary columns ([“Time”, ”Weight”]) and the data.

2) **Front-End:**

Sample JSON Packet Format:



```
{
    "Bolog": [
        [
            "19:07:24",
            0.6
        ],
        [
            "19:08:30",
            0.3
        ]
    ],
    "Blueb": [
        [
            "19:07:25",
            0.6
        ],
        [
            "19:08:30",
            0.3
        ]
    ],
    "Mushr": [
        [
            "19:07:25",
            0.6
        ],
        [
            "19:08:30",
            0.3
        ]
    ],
    "Mexic": [
        [
            "19:07:25",
            0.6
        ],
        [
            "19:08:30",
            0.3
        ]
    ]
}
```

The JSON packet is basically a dictionary. Its keys are the names of the food and the values are arrays containing secondary arrays with the time and weight of the food at that time.

The mobile app will send a GET request to the API URL which is in the format of “http://ip_address:port/retrieve”. If there is not enough data posted by the weight sensor for the model to use to predict (the default size is 1750 inputs which is the size used to train the model), it will simply output the data received. However, once the number of rows in the data file exceed 1750 inputs, it will begin predicting. It will combine the data already obtained and predictions together and append it to a predict_data.csv in the “predictions” directory according to the food type. Then it will create the format as shown in the image above and send it to the front-end.

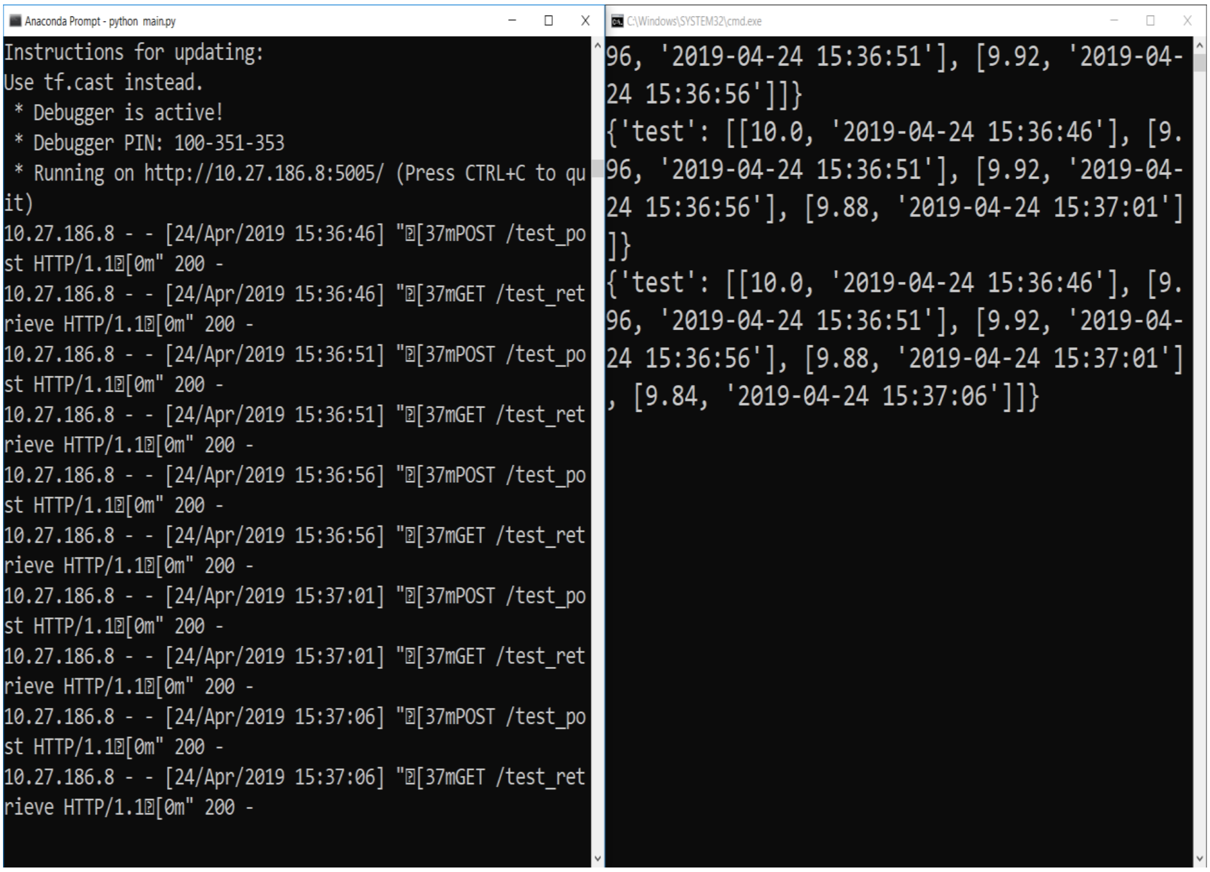
3) **Back-End:**

Anytime the food is topped up, the back-end will automatically create a new file to process the input of data from the weight sensor and use the old data file to train the machine learning model. Since there is some time before the model will get enough data to generate predictions, this time can be used to train the model.

Unit Testing

Running of Backend Tests

1) Unittests.py provides several unit tests for testing out the important functions needed for the Flask Server to function.
2) The post function will send a POST Request to the API URL, which is by default the server’s URL for receiving inputs from the weight sensor.
3) The get function will send a GET request to the API URL, which is by default the server’s URL for sending predictions to the front-end.
4) The reset function will delete the CSV file which is used for testing and create a new CSV file.
5) The most important test is the full_test function which will show the entire flow of posting the weight outputs and then retrieving the weights and predictions for the front-end. The result would be similar to what is shown below.



```
Instructions for updating:
Use tf.cast instead.
 * Debugger is active!
 * Debugger PIN: 100-351-353
 * Running on http://10.27.186.8:5005/ (Press CTRL+C to qu
it)
10.27.186.8 - - [24/Apr/2019 15:36:46] "[37mPOST /test_po
st HTTP/1.1[0m" 200 -
10.27.186.8 - - [24/Apr/2019 15:36:46] "[37mGET /test_ret
rieve HTTP/1.1[0m" 200 -
10.27.186.8 - - [24/Apr/2019 15:36:51] "[37mPOST /test_po
st HTTP/1.1[0m" 200 -
10.27.186.8 - - [24/Apr/2019 15:36:51] "[37mGET /test_ret
rieve HTTP/1.1[0m" 200 -
10.27.186.8 - - [24/Apr/2019 15:36:56] "[37mPOST /test_po
st HTTP/1.1[0m" 200 -
10.27.186.8 - - [24/Apr/2019 15:36:56] "[37mGET /test_ret
rieve HTTP/1.1[0m" 200 -
10.27.186.8 - - [24/Apr/2019 15:37:01] "[37mPOST /test_po
st HTTP/1.1[0m" 200 -
10.27.186.8 - - [24/Apr/2019 15:37:01] "[37mGET /test_ret
rieve HTTP/1.1[0m" 200 -
10.27.186.8 - - [24/Apr/2019 15:37:06] "[37mPOST /test_po
st HTTP/1.1[0m" 200 -
10.27.186.8 - - [24/Apr/2019 15:37:06] "[37mGET /test_ret
rieve HTTP/1.1[0m" 200 -
```

```
96, '2019-04-24 15:36:51'], [9.92, '2019-04-
24 15:36:56']]}
{'test': [[10.0, '2019-04-24 15:36:46'], [9.
96, '2019-04-24 15:36:51'], [9.92, '2019-04-
24 15:36:56'], [9.88, '2019-04-24 15:37:01']
]}
{'test': [[10.0, '2019-04-24 15:36:46'], [9.
96, '2019-04-24 15:36:51'], [9.92, '2019-04-
24 15:36:56'], [9.88, '2019-04-24 15:37:01']
, [9.84, '2019-04-24 15:37:06']]}
```

Running of Frontend Tests

1) In the directory labelled React-Native, run the command “npm test” to begin the tests.
2) The name and respective components that are tested are listed in the terminal.

Machine Learning Description

Description of the ML model

The basic model is a Recurrent Neural Network (RNN) or a special variation of it, LSTM (Long Short-Term Network) to conduct time series forecasting using the past values of the weight of the food since the weight of the food at one point is dependent on its previous values. RNNs are neural networks that utilize sequential information. It is different from a normal neural network as it considers its previous step input and merges that information with the current step input. LSTMs are a variant of RNNs except that they consider long term dependencies, using gating mechanisms to store or release memory.

Our model is one that uses a combination of LSTM and GRU, which has been found to improve performance and accuracy based on research here. We also explored using Attention Neuron Networks but due to unforeseen complications, they could not be implemented. Nevertheless, the Python files and Jupyter Notebook are still in the directory. The build_attention_model function in the training.py file can be used to create the attention model and train it.

Attention Neural Networks are rather recent. Attention mechanisms in neural networks serve to orient perception as well as memory. Attention filters the perceptions that can be stored in memory and filters them again on a second pass when they are to be retrieved from memory. It has been found to solve long-term memory problems and could have better performance than LSTMs. The paper can be found here.

Raw data and transformations: “features” used to train the model

Since it is using simulated data as data in real life required collaboration with restaurant and networks of Food and Beverage providers, raw data was not used. Test size of 0.3 was used in Sklearn’s train_test_split function. We used call-backs such as EarlyStopping to stop training once validation loss has stagnated and ModelCheckpoint to save the best model which achieved the lowest validation loss. A validation dataset of 0.3 was used for obtaining the validation loss during training. The feature we used were primarily the weight of the food received from our weight sensor.

How it will continue to improve

Each time the food tray is refilled, a new csv file will be created to continue to process the input from the weight sensors while the old csv files containing the old weights will be used to further train the model to predict. By default, an increase in weight of 40% of the total food weight will trigger this algorithm.

Contributors

Chan Chia Ler (Business, Liaising and Hardware)

Chin Zhi Wei (Back-End Server and Machine Learning)

Ding Si Han (Front-end and Unit-Testing)

Batch of Nanyang Technological University's Renaissance Engineering Programme 2022