Final Report – Team The Fridge Bros

Team Members: Felix Ong Wei Cong, Max Nabokow, Sarthak Mani Pradhan, Seo Christopher

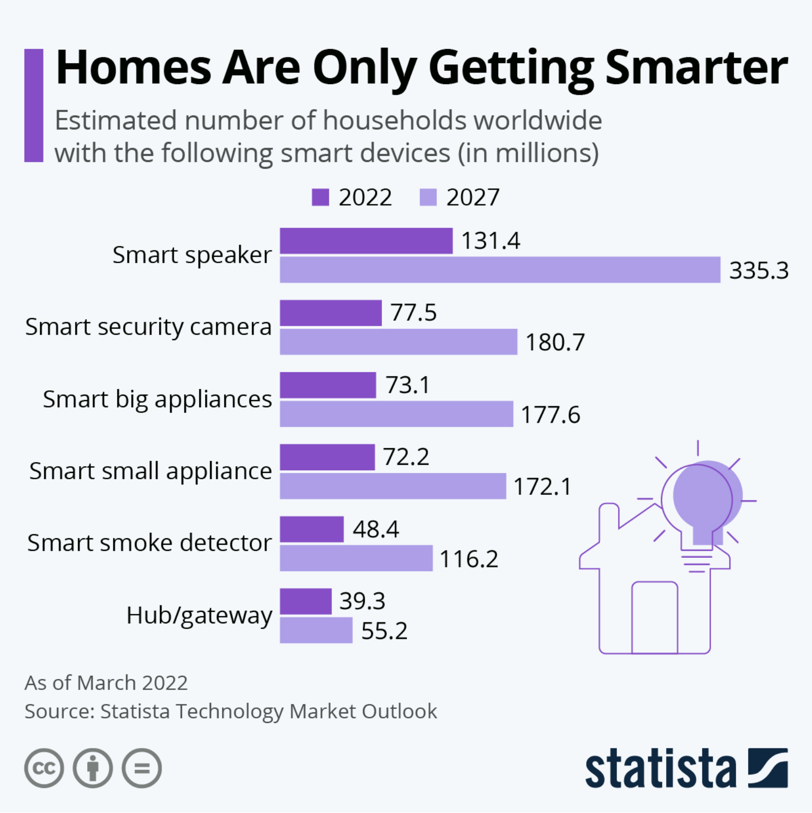
Project/Team Title: Smart Fridge/The Fridge Bros

Abstract:

This project explores the creation of a smart fridge IoT system which has essential features such as keeping track of fridge stock and prediction of the user's consumption patterns. The core of this system is a WeMos with a photoresistor sensor that triggers a camera to take an image of the fridge contents when the fridge door is open, sending the image and stock counts to a cloud-hosted database for long-term consumption pattern prediction. Current stock counts and consumption prediction information can then be viewed from an iOS application. By using a state-of-the-art object detection algorithm, YOLOv7, and the SARIMA time-series forecasting model, we were able to develop a relatively accurate and affordable smart fridge system that can be used to convert regular fridges into smart fridges.

Introduction:

Smart home devices are becoming increasingly popular in homes around the world. Data from the Statista Technology Market Outlook[1] estimates that over 73.1 million households are home to at least one smart big appliance such as refrigerators, for example, and this figure is expected to grow to as many as 177.6 million in the next 5 years.



Homes Are Only Getting Smarter

Estimated number of households worldwide with the following smart devices (in millions)

As of March 2022
Source: Statista Technology Market Outlook

It is easy to see why smart home devices are getting popular, as they offer many quality of life improvements to users. For example, smart fridges allow us to monitor stock levels in real-time from anywhere. This can be convenient when used remotely, for example, from a grocery store to figure out what we need to buy. This may also reduce wastage by reducing unnecessary trips to the grocery store and over-purchase of groceries.

However, while these benefits are appealing, having more features comes at a cost. Smart fridges like Samsung's offerings cost around 2000 dollars for the most basic model[2], which is much more than the cost of an average non-smart fridge.
Therefore, we plan to explore the creation of a smart fridge that only has the most essential features such as real-time stock level monitoring, making it more affordable. Furthermore, our smart fridge system will also be portable and can be placed in normal fridges to convert them into a smart fridge.

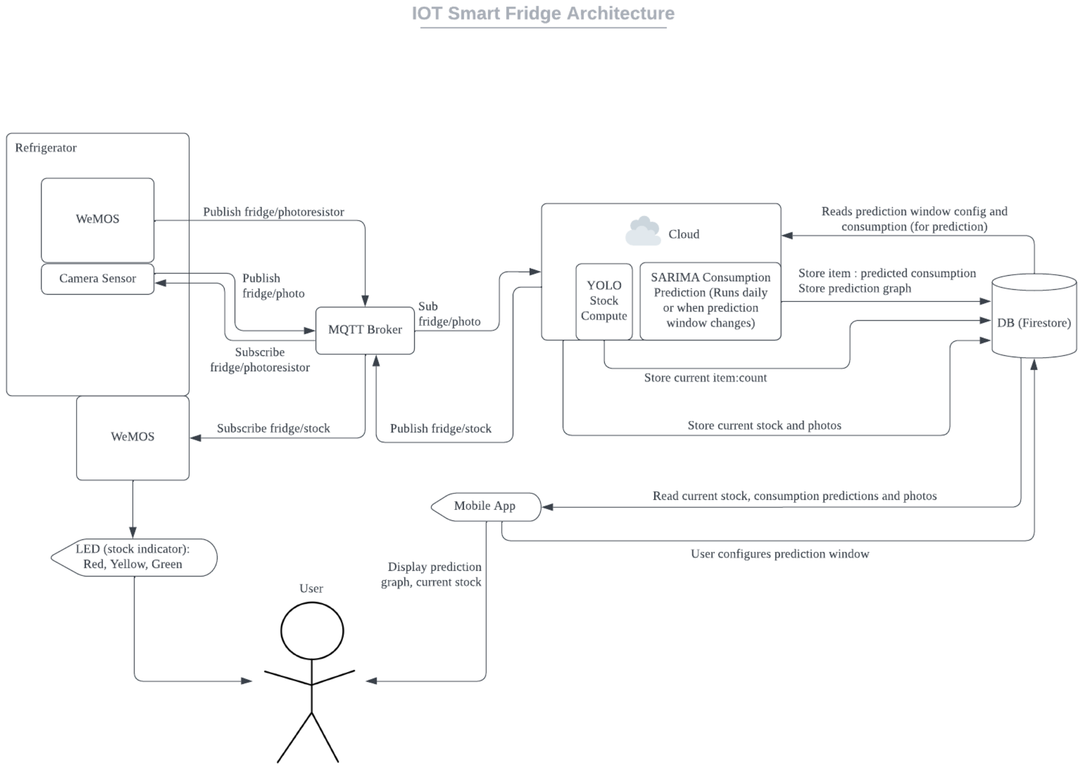
Our proposed IoT system is a smart fridge that is able to:

1. monitor food stock levels in real-time and
2. predict the demand for products based on consumption patterns of the user

This will allow users to efficiently grocery shop by only buying what they need and reduce food waste and energy waste. Better for the environment, and better for the wallet! There is also no need to purchase a brand new fridge just to use these features, since our system can be installed inside existing fridges.

Details of our approach:

The architecture diagram of our system is shown below.


IOT Smart Fridge Architecture
Refrigerator
WeMOS
Camera Sensor
Publish fridge/photoresistor
Publish fridge/photo
Subscribe fridge/photoresistor
MQTT Broker
Sub fridge/photo
WeMOS
Subscribe fridge/stock
Publish fridge/stock
Cloud
YOLO Stock Compute
SARIMA Consumption Prediction (Runs daily or when prediction window changes)
Reads prediction window config and consumption (for prediction)
Store item : predicted consumption
Store prediction graph
DB (Firestore)
Store current item:count
Store current stock and photos
Mobile App
Read current stock, consumption predictions and photos
User configures prediction window
LED (stock indicator): Red, Yellow, Green
Display prediction graph, current stock
User

Sensors used:

- photoresistor, to detect if the fridge door is open
- multi-color LED, to notify the user of stock count status
- webcam, to take pictures of the fridge contents

Number of IoT Devices: Four.

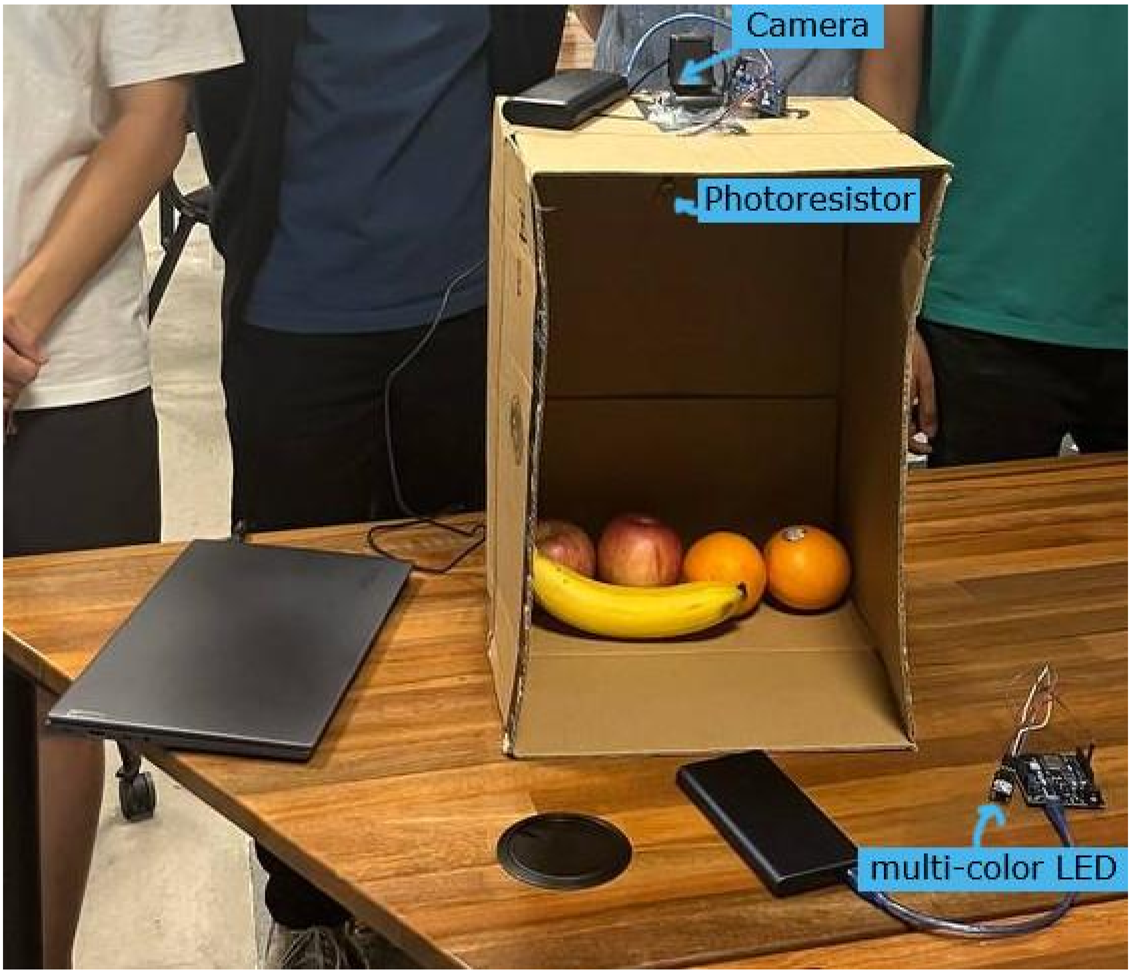
- 1 WeMos connected with the photoresistor inside the fridge
- 1 WeMos connected with the multi-color LED outside the fridge
- 1 Laptop connected with the webcam
- 1 Laptop serving as the MQTT broker and cloud server to run the machine learning algorithms

Other hardware used:

- iPhone for running the iOS application
- 2 5000mAh power banks for powering the WeMoses
- Cardboard box to simulate the fridge

The laptop with camera, WeMoses and cloud communicate via MQTT over the MQTT broker. The cloud and iOS application reads from and writes to a Firestore database.

The image below shows our system. The laptop running the local MQTT broker and serving as the cloud computation server is not pictured.



Machine Learning Models:
The data we are collecting is long-term food consumption patterns. By predicting the future consumption, the user is not only able to plan their grocery shopping better (for example, buying grocery only on the weekends based on the following week's consumption predictions), they are also able to use it to adjust their diets. We are using the SARIMA (seasonal autoregressive integrated moving average) time-series forecasting model as it is widely used for predicting future values based on previously observed values and has shown success in predicting demand and consumption patterns, especially with seasonal patterns, since food consumption may peak on weekends.
To monitor the stock levels, we run the You Only Look Once (YOLO)v7 [3] object detection algorithm on the image of the fridge contents. We are using the weights of the model which is pretrained on the MS COCO dataset[4], which consists of 80 categories of common objects, including food items such as the 3 fruits we are using in our demonstration, apple, banana and orange. YOLOv7 is used as it is state-of-the-art, being fast and accurate.

The flow is as follows:

1. The local MQTT broker on the laptop is started. This laptop also runs detect.py which carries out YOLOv7 and SARIMA.
   The laptop with camera runs camera.py to subscribe to "fridge/photoresistor" and prepare for taking pictures.
   The 2 WeMoses are powered by the power bank and connected to a mobile hotspot.
2. The photoresistor WeMos publishes a reading every 500ms to "fridge/photoresistor".
3. If the fridge door is closed, the camera laptop does not take pictures because the photoresistor reading received from "fridge/photoresistor" is too low.
4. If the fridge door is opened, a high photoresistor reading prompts the camera to start taking pictures every second, until a dark image is obtained. The last image before the dark image is published to "fridge/photo".
5. The cloud receives the image and runs YOLOv7. The stock status (LED color) is calculated based on predicted consumption from the database and is published to "fridge/stock". The image and current stock counts are also stored on the database.
6. The multi-color LED WeMos which is subscribed to "fridge/stock" changes color between red, yellow and green depending on how many of the 3 items are out of stock.
7. The user can check the iOS application at any time to look at the image of fridge contents, the current counts of each item, prediction counts and prediction graphs. The prediction window can be changed at any time from the app.

Implementation Details:

Communication Protocol

MQTT is chosen as the communication protocol as it is lightweight, efficient and easy to implement. On the WeMoses, the ArduinoMqttClient library is used. On the laptops running Python scripts, the Paho MQTT client library is used. Mosquitto MQTT broker is used on the local MQTT broker.
Communication between the application, laptops and the Firestore database takes place over the Firestore API.

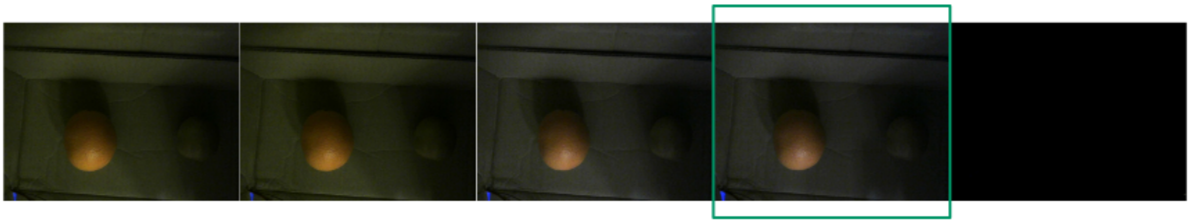
Photoresistor

The WeMos runs *fridge.ino*. This program connects the WeMos to a mobile hotspot and the local MQTT broker. In the main loop, the WeMos publishes the photoresistor reading every 500ms to "fridge/photoresistor". The photoresistor is an analog sensor which outputs a reading between 0 and 1024.

Taking Picture of Fridge Contents

We are getting around the limitation of not having a camera sensor that is able to plug into our WeMOS by using an external laptop camera and having our photoresistor+WeMOS communicate with it via MQTT. Image capturing in our project goes as follows:

The laptop connected with the webcam runs *camera.py*. This program connects to the local MQTT broker and subscribes to "fridge/photoresistor". Initially, when the doors are closed, the camera is switched off and no data is being published, The fridge door is only considered open when the reading received is more than 500. This threshold was determined by taking multiple readings of the photoresistor placed in our cardboard box with the flaps closed, representing the fridge. Even with the flaps closed, some light was still leaking in due to holes in our box, especially at the top where our photoresistor is placed. As a result, the doors are only considered open when a reading more than 500 is obtained.
The camera is accessed and controlled using opencv-python, a library for the OpenCV computer vision tools.
After the fridge door is open, the camera takes a picture every second, keeping the previously taken image in memory, until a dark image is obtained. A dark image is defined as an image that is right before the door is closed, such that the image before it should contain the most updated counts of fridge contents (nothing will be added or removed at this point). The image taken before the dark image is serialized into a json object (since it is an array of pixel intensities) and published to "fridge/photo".
The condition for a dark image is taken as an average pixel intensity value of less than 40. Since there is more noise at lower light levels which affects the ability of the YOLOv7 algorithm to detect shapes and patterns, we determined this threshold value of 40 by running multiple tests and taking the darkest image for which YOLOv7 can still detect all the objects in the fridge, and selecting the image after it to be the dark image.



The image in the green box above is the one that is published.

A note on multithreading in our project:

In our original, single-threaded program *camera.py*, we would begin continuously taking pictures as soon as the photoresistor sent a signal crossing the light threshold. Until a dark photo was detected, the process would block the receiving and handling of any new messages coming in via MQTT. Since MQTT preserves the ordering of messages, the messages we would receive after the camera loop ends were out of date since they were queued FIFO.

To fix this problem, we opted to handle the camera section of code in a worker thread so that the main thread could continue to receive and handle new messages. We ensured that only 1 worker thread could be created at a time and discarded incoming (outdated) messages in our main while that worker thread was busy taking photos.

Initially we wanted a “memoryless” broker, i.e. an MQTT broker with buffer size 0, but several of our implementations failed to work hence the multithreading solution. Python’s built-in thread pools suffer from a similar limitation, so in our workaround we kill threads to prevent those with outdated jobs from sitting in the waiting queue.

Object Detection and Stock Counts

The YOLOv7 algorithm is used for object detection, using the YOLOv7 model. We found that using the pretrained weights was enough to detect all the objects we had in the fridge correctly almost all the time, and there was no need for further training with images of fridge contents.

However, the original implementation draws bounding boxes around detected objects in the input image and outputs the drawn image, which is not useful for us. Therefore, we modified the detect function in the *detect.py* script to return a dictionary of each object and its counts instead.

detect.py is run on a laptop and connects to the local MQTT broker and subscribes to “fridge/photo”. Upon receiving the JSON serialized image, it is loaded and the object detection function is run to obtain the current stock counts. The predicted consumption is also read from the database. The stock status color is then calculated and published to “fridge/stock”. The color is “g” for green if all items are in stock (more than or equal to the predicted consumption count, “y” for yellow if at least 1 item is less than the predicted consumption count and “r” for red if all items are less than the predicted consumption count.

The previous stock count is kept on the FireStore, so that it can be accessed by the SARIMA algorithm in order to calculate consumption. This then allows the algorithm to find consumption patterns without the need for the low powered WeMos to do any more operations than necessary.

The image is encoded as base64 and stored on Firestore so that it can be viewed in the app.

Indicator LED

The WeMos runs *stocks_subscribe.ino.* This program connects the WeMos to a mobile hotspot and the local MQTT broker, and subscribes to “fridge/stock”. In the main loop, the WeMos receives the stock status color (“g”, “y”, “r”) and sets the color of the multi-color LED accordingly.

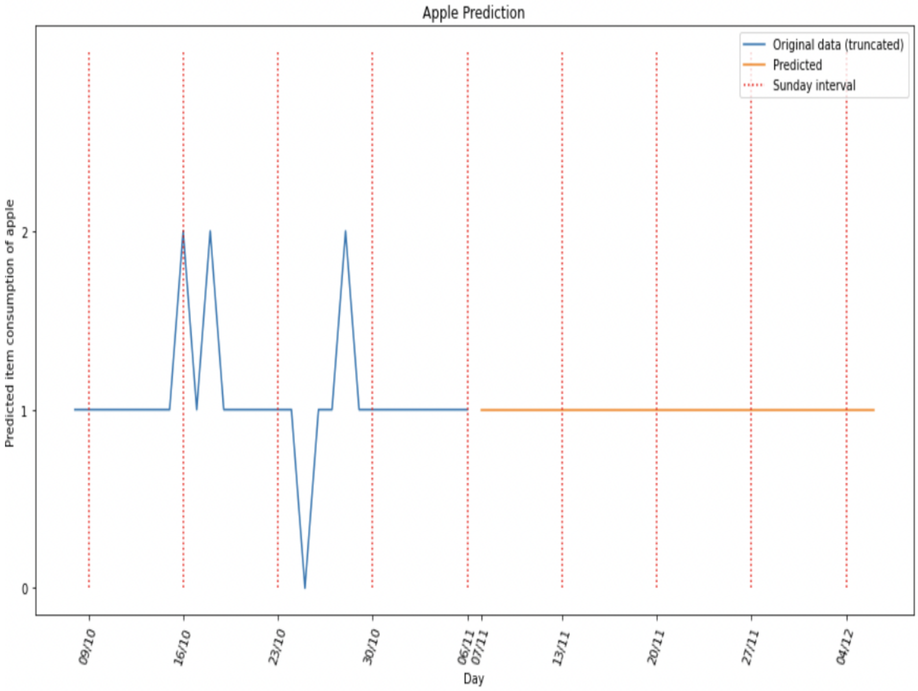
Consumption Pattern Prediction

The assumption behind our consumption prediction is that consumers have weekly periodic consumption patterns e.g. eating more bananas on the weekends vs mid week. For that reason we decided to use Seasonal Autoregressive Integrated Moving Average (SARIMA) as our model. SARIMA works well at producing short term forecasts on univariate, seasonal data, which is what we want.

For how we selected our SARIMA parameters, refer to *Experimental Evaluation*.

SARIMA will need existing data to predict on when the user first starts using the app, otherwise the user won't have predictions for the first month of use, which is not ideal. So we decided to generate 60 points of fake consumption data for each of our items to use as our initial dataset, following a noisy sinusoid to mimic actual usage. Each sinusoid used to generate the data was a little different, as well as the noise to mimic real life scenarios. It should be noted that none of the realism added into the initial datasets matters in the long term, as over time, predictions will be made on the user's actual consumption data - it could have very well been an initial dataset full of constants, although we felt that it would make a less exciting batch of initial predictions to the user.

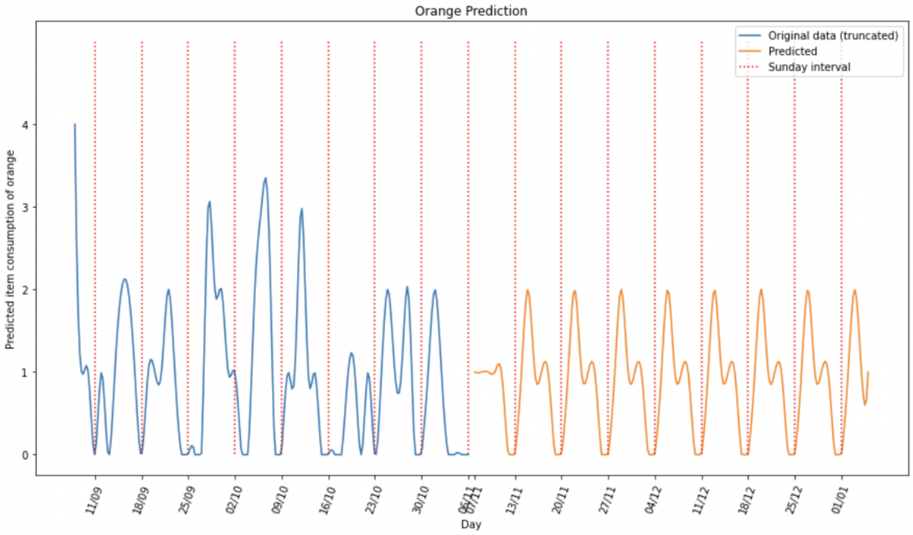
1 sample of our generated datasets with param descriptions + predictions:



Apples dataset (an apple a day keeps the doc away):

Generation params:
-Noise ~N(mu=1, sd=0.3)
-No periodicity, center around 1

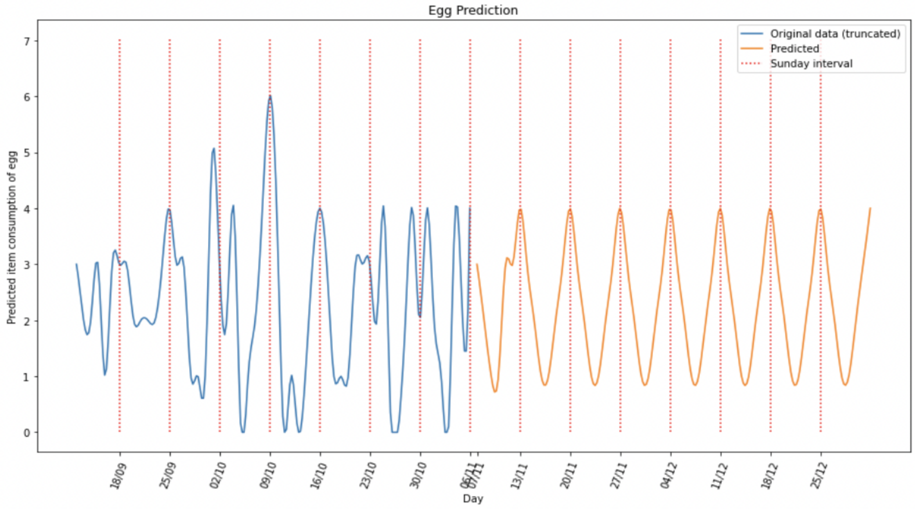
Notes: SARIMA predicts an apple a day, showing efficacy on some non-periodic data as well!



Oranges dataset:

Generation params:
-Noise ~N(mu=1, sd=1)
-Peaks mid-week around 2, minimum is 0

Notes: SARIMA predicts mid-week peaks



Eggs dataset:

Generation params:
-Noise ~N(mu=1, sd=1)
-Peaks on weekends (Sunday) around 3.5, minimum is 1

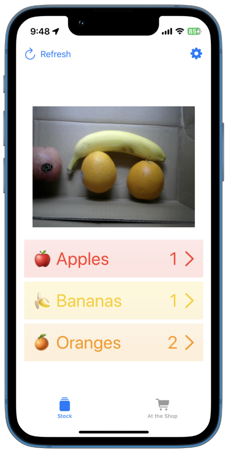
Notes: SARIMA predicts weekend peeks

We also allowed the user to change the length of time the pattern predicts for and much previous data will feed into the prediction. For example the user can change the history to only 7 days, ensuring that the SARIMA algorithm only uses the past 7 days of consumption to predict. This ensures that the system is flexible to drastic changes in consumption e.g. if family members moved out in the past week, and that the user can take advantage of this flexibility to adapt the fridge to their situations.

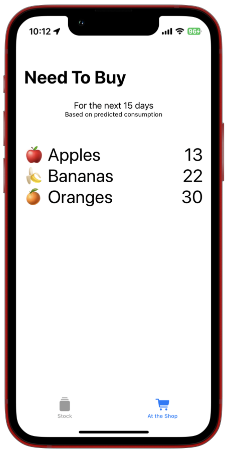
iOS Application (repo)

To create a convenient way for the user to interact with our Smart Fridge, we decided to build an iOS app.

The app is useful in 2 key scenarios: While on-the-go or at home around their living space, the user can get a real-time overview of the items in the fridge. We show the fridge stock in a colorful, easily glanceable interface. Every time we detect a stock update and finish the stock analysis, the app updates and shows the most recent stock with a "proof image". (If our stock recognition is ever inaccurate, this provides a foolproof way for the user to count the items themselves.) By tapping on any of the shown item counts, the user can see graphs of both historical and predicted consumption, all of which update automatically when the user changes our prediction parameters in the app settings.



🍎 Apples 1 >
🍌 Bananas 1 >
🍊 Oranges 2 >



Need To Buy

For the next 15 days
Based on predicted consumption

🍎 Apples	13
🍌 Bananas	22
🍊 Oranges	30

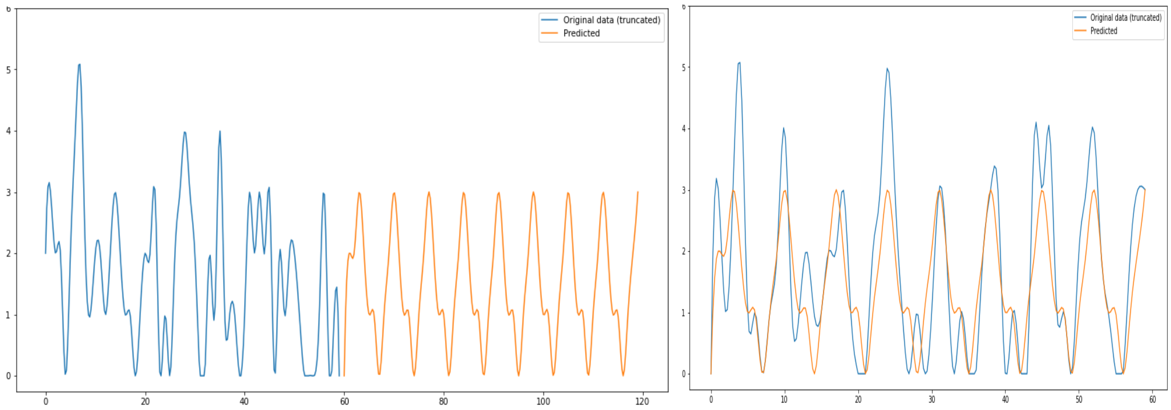
The other half of the app is the "At the Store" view. We all go to the store sometimes, not knowing our fridge stock *exactly*, and end up buying more than we need, spending money on food that doesn't even fit into our consumption habits. Especially for perishable food items, this can lead to food waste due to overstocking. The "At the Store" view aims to solve this exact issue. By calculating the predicted consumption over the user-set "prediction window" in the settings, the app gives the user an easy, glanceable view of their shopping list. Conveniently, rather than just showing a number greater than or equal to zero for each item,

we only show the items that actually need restocking. This not only reduces food waste and saves the user’s wallet, it also speeds up the shopping process!
Finally, the user can customize our dynamic consumption prediction algorithm from the iOS app’s settings. They can adjust the number of days into the future that we predict consumption for, and they can also change the number of historical days taken into account for our consumption prediction. This is useful in cases where consumption habits for one’s household changed recently, e.g. a family member moved out, someone passed away, or a new food-consuming member (or pet!) joined the family.
The iOS app interface was built using the SwiftUI framework and has a fully reactive information flow with a stripped-down MVVM architecture. It runs on iPhones of any size and follows Apple’s Human Interface Guidelines. The app interfaces with the Firestore database through the Firebase SDK.

Experimental Evaluation:

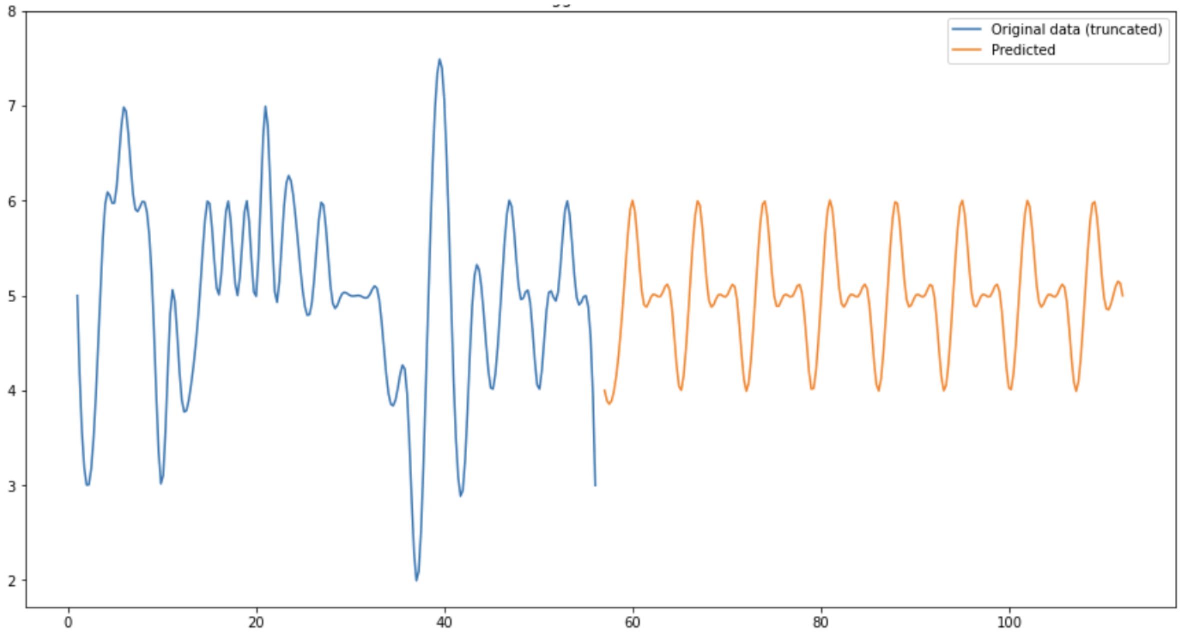
<u>Predictions:</u>
The configuration of SARIMA models is known to be a bit subjective, so we decided to choose our parameters by enumerating combinations of the SARIMA parameters over a large range, and choosing the parameters which yielded the lowest absolute error. We tested on a noisy sinusoid with a period of 7, peak/min of 3.5 and 0, and added noise ~Normal (mu=0, sd=1) - our best set of parameters yielded an average absolute error of 0.744 on our validation data over 500 iterations. This means that for this specific dataset, our predictions are on average 0.744 items off - and we are hoping that a good performance here will generalize to other weekly periodic datasets.

One iteration of testing with these parameters (also showing overlap with validation data).



Again, this assumes that the consumption patterns will actually look periodic, which they very well may not. We actually tested our model on Gaussian noise and found that SARIMA would sometimes find weekly periodicities in the noise. This is a weakness in our model, so for this project we are all in on assuming weekly patterned consumption. As a follow up, we could automate our testing so that our predictions can generalize to consumption patterns with any length period e.g. monthly, biweekly.

SARIMA making predictions on noise (chosen specifically to demonstrate, on average our predictions would predict flat on the center of noise):



Battery Life:

Power consumption for the WeMoses measured using the USB port were around 96mA on average. With our 5000mAh power banks, this gives a battery life estimate of around 5000mAh / 96 mA ~= 52h, which agrees with our testing that the WeMoses can be powered for slightly around 50 hours.

The camera was powered directly with the laptop and when measured it had a peak power usage of 500mA while taking pictures and around 200mA when idle. Our system conserves power by only taking images when the door is open, as this is the only time the contents of the fridge can be changed as well as be visible. For longer term usage, it is advisable to connect the camera to the main power source that powers the fridge. For a complete product we would need multiple cameras, perhaps on different sections of the fridge, so powering them off of the mains seems like the best choice to make. If we choose to power the single camera through a 5000mAh battery like the two WeMoses, it would only last around 20 hours which is not long enough of a time to reduce maintenance costs.

Contributions:

Christopher was responsible for implementing predictions + generating datasets and built in multi-threading for the photoresistor/camera.

Felix was mainly responsible for the implementation of the object detection algorithm and photoresistor-camera interaction.

Max built the iOS app and set up the Firebase backend.

Sarthak was mainly responsible for the usage of the MQTT broker, as well as with the interactions between the stocks and the LED lights.

Summary:

In conclusion, we have demonstrated a cheap smart fridge system that works well in real-time stock monitoring and consumption pattern prediction under certain assumptions. However, there are some limitations and challenges if it were to be applied in a more realistic scenario.

Firstly, we have fixed the detection and stock monitoring to 3 fixed items: apples, oranges and bananas. This was done for the ease of creation of historical consumption data and to allow us to bring and use real items for demonstration. The MS COCO dataset used to train YOLOv7 is also restricted to 80 items. To keep track of a larger variety of objects, the YOLOv7 model should be trained on a different dataset, perhaps one that specifically contains only items found in a fridge, although we were unable to find a good dataset.

Secondly, in our mockup, there is only a single shelf which has a height much greater than the shelves of a real fridge. This is because the webcam we used is designed for video calls and thus has a narrow field of view to capture mostly the face and less of the background. The webcam is also unable to focus well at closer distances to the objects, which is why we had to position it at the top of our mockup fridge, so that it can take a clear picture of the contents. In an actual smart fridge, better cameras with wide angle lenses are usually used.

Future work could include push notifications and online grocery shopping into the iOS application to place orders easily when stocks are running low.

References:

[1] https://www.weforum.org/agenda/2022/04/homes-smart-tech-market/
[2] https://www.samsung.com/sg/refrigerators/side-by-side/
[3] https://github.com/WongKinYiu/yolov7
[4] https://cocodataset.org/#home