

# **AI Dining Suggestion App**

A Project Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment of

the Requirements for the Degree

Master of Science

By

Bao Pham

May 2019

© 2019

Bao Pham

ALL RIGHTS RESERVED

**SAN JOSÉ STATE UNIVERSITY**

The Designated Project Committee Approves the Master's Project Titled

**AI Dining Suggestion App**

By

Bao Pham

**APPROVED FOR THE DEPARTMENT OF COMPUTER  
SCIENCE**

Dr. Chris Pollett

Department of Computer Science

Dr. Mike Wu

Department of Computer Science

Dr. Kevin Montgomery

eHuman, Inc.

## **ABSTRACT**

Trying to decide what to eat can sometimes be challenging and time-consuming for people. Google and Yelp have large scale data sets of restaurant information as well as Application Program Interfaces (APIs) for using them. This restaurant data includes time, price range, traffic, temperature, etc. The goal of this project is to build an app that eases the process of finding a restaurant to eat. This app has a Tinder-like user friendly User Interface (UI) design to change the common way that lists of restaurants are presented to users on mobile apps. It also uses the help of Artificial Intelligence (AI) with neural networks to train both supervised and unsupervised learning models that can learn from one's dining pattern over time to make better suggestions at any time.

## **ACKNOWLEDGEMENT**

I would like to express my sincere gratitude and appreciation to Dr. Christopher Pollett for his guidance and support throughout the entire project. It was my honor to have him as my advisor with his in-depth knowledge and expertise in the field.

I would like to thank my committee members, Dr. Mike Wu and Dr. Kevin Montgomery for their valuable feedback as well as suggestions given towards the project. I also would like to extend my thankfulness to Allen McDivitt and Alexander Montgomery for being the great testers during this project.

Finally, I am grateful to my friends and family for always being a strong support for me throughout my career.

# TABLE OF CONTENTS

<b>LIST OF FIGURES .....</b>	<b>7</b>
<b>INTRODUCTION.....</b>	<b>8</b>
<b>PRELIMINARIES .....</b>	<b>11</b>
2.1 Front-end Technologies.....	11
2.2 Back-end Technologies .....	12
2.3 Middleware Technologies.....	13
2.4 AI Technologies .....	15
<b>APP DESIGN .....</b>	<b>18</b>
<b>IMPLEMENTATION.....</b>	<b>20</b>
4.1 API Implementation with Basic Features .....	20
4.2 External Services Integration for API.....	25
4.3 Front-end Implementation .....	27
4.4 Processing and Filtering Data for AI .....	29
4.5 Train Static AI Model .....	32
4.6 Train Reinforcement Learning Model .....	35
<b>EXPERIMENTS .....</b>	<b>38</b>
5.1 App Experiment .....	38
5.2 AI Model Experiment.....	41
<b>CONCLUSION .....</b>	<b>43</b>
<b>REFERENCES .....</b>	<b>44</b>

## LIST OF FIGURES

Figure 1. Technology stack assembly.....	14
Figure 2. A simplified structure of a neural network.....	16
Figure 3. Initial mockup for the UI.....	19
Figure 4. A sample result from a Google request.....	23
Figure 5. A sample result from a Yelp request.....	23
Figure 6. Samples of Restaurant object stored in database.....	24
Figure 7. A sample response from Open Weather Map.....	25
Figure 8. Image interactions within the app.....	28
Figure 9. A sample of a Restaurant object from the Yelp dataset.....	30
Figure 10. A snippet of the dataset after being processed.....	31
Figure 11. The structure of the baseline model.....	33
Figure 12. An example of Label Encoding and One Hot Encoding.....	36
Figure 13. The structure of the reinforcement learning training process.....	37
Figure 14. Variant A (left) and Variant B (right) of the app.....	39
Figure 15. Final design of the app.....	40
Figure 16. A sample of a restaurant after with model's prediction.....	42
Figure 17. Improvement of reinforcement learning model on more data.....	42

# INTRODUCTION

Currently, Google Maps and Yelp are two main applications of the web that provide lists of restaurants. Going through a list of possible options often takes time, and still does not provide a list of what most likely suits the user's taste. In addition, when a user starts browsing for options on Google Maps or Yelp, the reviews often play an important role in determining the final chosen. However, user's taste at a certain time does not always depend on the reviews of a restaurant. In other words, just because a restaurant has bad reviews does not mean the user would not go there as it might have some dishes that the user likes. Moreover, these applications do not really learn about a user's taste or observe a user's restaurant pattern to filter options from the list. They simply gather the restaurants around the user's location at a certain time and apply some pre-set filters on top of it, such as price range or restaurant category.

AI based recommendation systems are not new. In fact, there have been several papers discussing improvements to Yelp's recommendation systems using neural networks and machine learning algorithms. Sawant and Pai (2013) from Stanford University applied a few machine learning algorithms on the Yelp dataset to compare the results and performances<sup>[1]</sup>. These algorithms include single value decomposition, hybrid cascade of K-nearest neighbor, weighted bi-partite graph projection, and several others. Root Metrics Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are the values that were used in Sawant and Pai's paper to evaluate the outputs. This paper showed that machine learning algorithms can offer decent improvements to recommendation systems. Within the AI domain, Seo, Huang, Yang and Liu (2017) experimented using Convolutional Neural Network (CNN) – a neural network that is widely used for Image Classification in AI - for Yelp recommendation system<sup>[2]</sup>. After evaluating the



results with Mean Squared Error (MSE), Seo and others concluded that neural networks offer a new path for improving recommendation systems. They also mentioned that using a Long Short-Term Memory (LSTM) neural network, which are used in the present project, could be an area of future research.

As some can see from the above section of previous papers, AI models for restaurant recommendation have been trained based on the available restaurant data provided by Yelp, but not on dynamic or temporal data that is also relevant to the user's tastes on different situations. In other words, the models to date are static and would only provide the same output if given the same input of restaurants. Another thing to point out is that the models discussed above do not re-train themselves over time, so it is interesting to consider a reinforcement learning approach. One's preferences about food can change quite often. Thus, reinforcement learning, which is a technique used to train AI models in an interactive environment using its own actions and experiences, might be useful for recommendation system enhancements.

In terms of how to present restaurant data to the user, these Google Maps and Yelp apps still offer the traditional way of typing keywords, and the apps show the list of potential matches. Thus, these apps still require that the user somewhat knows what type of food they want to eat and the apps simply provide a list of options based on specific filters.

Therefore, for this project, the goal is to build an app which is inspired by another app called Tinder – a dating app that uses swipe action, and a new way of presenting data in which simplicity is the top priority for the users. Combining user friendly UI design and AI models, our app learns the user's dining pattern over time to help make restaurant suggestions. The app in this

project hence tries to address two main issues that are listed above which are saving time to find a restaurant and avoiding irrelevant options for the user's taste.

In regards to making better suggestions using AI, the models used in this project are inspired by a paper published by YouTube employees in 2016 <sup>[3]</sup>. Besides Yelp, YouTube is one of the big platforms that really takes advantage of recommendation systems using neural networks as revealed in detail in that paper. The paper explains that YouTube's recommendation system has two main layers. The first layer is called Candidate Generation. Basically, in this layer, neural networks were used to filter the options that are irrelevant for an average user. The second layer also used neural networks, but it focused on ranking the options that are filtered by the the first layer for a specific user. Since the YouTube recommendation system focuses on recommending videos to users, and the goal is to keep the users staying on its platform to continuously watch the videos as long as possible, the information that YouTube gathers from users such as video released date, video length, watching duration of a videos, etc. is different from restaurant related data needed for this project. However, the concept of using two layers of a neural network to sort the options in terms of different users' preferences is applied in this project and is discussed more later in this report.

The rest of the report is organized into five different chapters. The first chapter contains the technical background information that is needed to understand and justify the project. Chapter two discusses the design of the app and the explanation for it. The implementation for the app is discussed in detail in chapter three. Next, chapter four talks about testing and modifications for the app. Finally, the last chapter concludes the project with some ideas on future work.

# PRELIMINARIES

In this section, the technology requirements for this project will be discussed. This background information helps the reader understand the implementation that will be explained later on in the report. Overall, these technologies include the tools to build an end-to-end application with both back-end and front-end as well as the frameworks that handled the AI component of the project.

## 2.1 Front-end Technologies

After much research, consideration and experimentation, React Native was chosen to develop the front-end for this project. It is an open source mobile app framework developed by Facebook. There were a few reasons React Native was chosen for the front-end implementation. First and foremost, React Native supports multiplatform deployment. As the app was meant to be tested during AI implementation, testers might have different mobile devices with different operating systems, and limiting the app to only one environment is very inconvenient. React Native makes it easy to simultaneously develop and deploy mobile apps for both Android and iOS with most of the code being shared between these two platforms. Secondly, the main language for React Native is Javascript and this is a plus because Javascript is a common programming language. Thus, when choosing other technologies later on, there would be more options that support this language. Moreover, React Native has strong community support. There are lots of plug-ins that have been written by community members to ease the process of programming common components for mobile apps. In this project, some popular plug-ins were also utilized to speed up the development.

## 2.2 Back-end Technologies

For back-end implementation, MongoDB was chosen to store the data. MongoDB is an open source NoSQL database program. Unlike SQL database, MongoDB does not require fixed schemas for the data objects, and this is the main reason why it was chosen for this project. Because data that is gathered for the project comes from different sources and since the project is experimental, it was unpredictable for which kind of data the project would need at the beginning. Therefore, having MongoDB with dynamic schema for data objects would make it easier to store and update data. In addition, restaurant data would most likely be in JSON format, and because MongoDB is designed and developed to work with JSON objects, it can help reducing the overhead of handling JSON data or dealing with missing fields in JSON objects.

In addition to MongoDB, a server was needed to host the data as the data could not be hosted in a local machine. Once the app is deployed for testing purposes, the data needs to be available at all times, and mLab was used to achieve this. mLab is a fully managed cloud hosting service which is specialized for MongoDB. It also offers a free tier which allows up to one GigaByte (GB) of data which was enough for this project. The huge plus of mLab in this project was the web UI for developers. The data can be viewed and edited directly on its website with the developer's credentials. This made it extremely easy to see how data is stored, updated and formatted. The web UI also allows developers to perform basic queries such as: search, update, delete, etc. via buttons without having to write the queries. Another great feature that was helpful for the project from mLab is the ability to import and export data. During the project, these actions were performed for testing purposes.

## 2.3 Middleware Technologies

The goal of this project is to create a functional end-to-end application. After having the front-end and back-end technologies defined, a middleware component to connect both ends, was required.

At this point, the middleware is an application program interface (API). This API helps the front-end mobile app interact with data stored in it. Therefore, NodeJS was selected to handle this component. NodeJS is an open source and cross platform Javascript runtime environment which allows executing Javascript code outside of a browser. As mentioned above, React Native's main language is Javascript and NodeJS also uses Javascript. Thus, this was already a plus for technology compatibility. In addition, NodeJS is a very common platform for easily building fast and scalable network applications. The reason for this is because NodeJS is event-driven and non-blocking I/O. Event-driven can be understood as the server only reacts when an event happens, and non-blocking I/O means that server can respond to multiple clients' requests at the same time instead of handling them in order.

After selecting the framework, the next question was how the API would be implemented. The answer for that was creating a Representational State Transfer (REST) API. REST is an architecture for developing web services. In general, with RESTful API, a request is sent from the mobile app (client) and is processed by the API (server) to interact with the data. After the data is prepared for the request, the API would send back a response to the mobile app with this data. The API prevents clients from directly manipulate data in the database. By doing this, a separation between components of the whole system is created and it provides a lot of good properties including scalability, simplicity, reliability and portability. In the end, the simple

reason for choosing RESTful is because this architecture is simple, lightweight, fast and very common.

During the development process, the whole API was hosted on a local machine, but when it came to deployment and testing, the API was hosted on a machine that is always active. This is where Amazon Web Services (AWS) come into play. AWS is a well-known subsidiary of Amazon. It offers lots of different options for reliable, scalable, and pay-as-you-go cloud computing services to individuals and companies. It also provides a free, academic tier to students and institutions. In this project, a service called Elastic Compute Cloud (EC2) was used to host a server which is reliable in handling deployment and testing phases.



Figure 1. Technology stack assembly

## 2.4 AI Technologies

In order to fully understand the AI components which will be discussed later in this report, it is necessary to briefly explain the concept of a neural network. The neural network concept in technology is inspired by the biological neural network which exists in animal brains. It contains many different machine learning algorithms together to perform computations on data input to produce meaningful output that can be used to predict something on other related data.

The process of performing computations can be briefly described as follows: when a set of processed input data is fed into a network, this data is formed into a set of nodes. A set of nodes is called a layer in neural network. The data, then is processed with an activation function. A value of a node in a layer after going through an activation function, becomes the input for the next layer. Thus, the neural network at this point, produces another layer which can have a bigger or smaller number of nodes than the first input layer. These two layers are connected by different values, and each of these values is called a weight. The last layer in a neural network is an output layer. Typically, in this layer, each node would carry a meaningful value because this is the last layer of the training process. A neural network which has multiple layers to process data is also known as deep neural network. The reason why neural networks were considered for this project is because restaurant data contains categorical information such as restaurant categories, restaurant ratings, restaurant price range, etc., and there is a finite amount of factors that influence food decisions. Thus, if this categorical information can be categorized into numerical values then from there, with the help of neural networks, some patterns can be discovered to make some meaningful suggestions to the user.

Usually, the input data would be divided into two sets. A training dataset and a validation dataset. One neural network model can be trained via many iterations (epochs) with the training

datasets, and after each iteration, the results from the model would be validated by the validation dataset. At this point, there are two important factors to consider: accuracy value and loss value. The accuracy value represents the performance of the model after testing with the validation dataset and the loss value represents the errors that are made during training. Based on the loss value, the model would adjust the weights between the nodes to improve its performance, and from the accuracy value, the architecture of the neural network can be modified with different layer settings so that the model can produce better results.

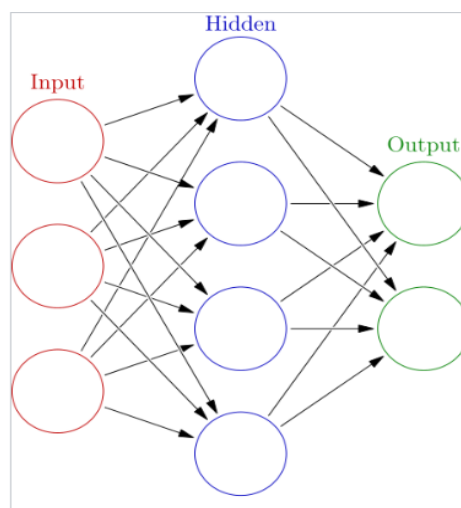


Figure 2. A simplified structure of a neural network

Regarding AI, Tensorflow was selected for the back-end because of its popularity and strong community support as it is an open source machine learning framework. Tensorflow was developed by the Google Brain team and has been widely used for AI applications since it was first released in 2015. Besides Tensorflow, Keras, which is an open source neural network library, was also used in this project to help speed up the neural network implementation. Keras



offers different implementations of commonly used neural networks on top of Tensorflow. Thus, it allows changing the architecture of the neural network with little effort.

Since the main programming language for Tensorflow and Keras is Python, it would create lots of incompatibilities if the AI model is directly implemented into the API written in Javascript. Hence, another service was required to achieve this feature and Flask technology was selected for this task. Similar to NodeJS, Flask is a web framework written in Python. Thus, it was used to run the service with the preloaded AI models so that the predictions could be made anytime.

As mentioned in the introduction about other related works, the AI models on those papers do not improve themselves over time. Therefore, reinforcement learning is implemented in this project to experiment if it can enhance the recommendation systems. Reinforcement learning concept can be explained as follows: in an environment, an agent makes some actions, and each of these actions puts the agent into different states. There are also different rewards depending on the actions. As a result, the goal is to train the agent so that it can learn as much as possible from the environment as well as the transitions between the states. From this knowledge, the agent then tries to take the actions result in the maximum rewards.

Reinforcement learning is an area that can be utilized with different algorithms, but in this project, Q-learning was the algorithm chosen to handle the reinforcement learning portion <sup>[4]</sup>. Q-learning in a nutshell is an algorithm which finds a policy that can tell the agent the actions to take to maximize the rewards. Because Q-learning is model-free, meaning that there is not a fixed goal for the model to try to achieve. Thus, in this specific case, the AI model would most likely need to learn from its own actions and make improvements from them.

## APP DESIGN

One of the issues with apps such as Google Maps or Yelp when it comes to their food suggestion feature is that they have a lot of information to present to the users and sometimes, it can be overwhelming. On the theme of dating apps, there have been many competitors such as eHarmony, okCupid, etc. However, Tinder, even though it came out later, has gained lots of traction and has become a success due to its simplicity. The interface of the app is designed as a deck of cards in which each card is displayed with big images of the object. Only the most important information is shown on top of the card as the intention is for the users to make quick decisions via images.

Inspired by Tinder, the project used this flow to create simplicity, hence encouraging the users to make quick decisions via images because at first glance, the presentation of a dish plays a really good role on whether or not one would want to try it. Besides images, information of the restaurant is also displayed. However, this information is limited to only a few main factors that possibly influence one's decision about the restaurant. It includes the restaurant's name, price range, busy hours (which infer waiting time) and driving time.

Swiping action is the main mechanism to use the app. When a swiping action is fired, a next card in the deck should show up. Images are set up in a carousel design. The main screen is divided into three main regions horizontally. Images are rotated to show when user taps on the left or right regions. If the user taps on the middle region, the image displayer goes to panning and zooming mode. This way, the user can interact with the images using fingers to pan or zoom. Within this mode, carousel design is maintained so that when the user slides left or right, the proper image is also displayed correctly.



Figure 3. Initial mockup for the UI

The design could change later on during the testing phase. However, there should be two main buttons on the first scene of the app to retain their functionalities. One is for the user to “like” the restaurant if they decide not to go to the restaurant at that moment. The other button indicates that the user wants to go to that restaurant, and the app at least should show the address of the restaurant. Besides that, the user would interact with the restaurant list via swiping actions. The concept of “disliking” a restaurant was not really clear, as sometimes the user’s preferences can depend on other factors as well. Thus, this feature may or may not change depending on feedback from testers.

# IMPLEMENTATION

## 4.1 API Implementation with Basic Features

After the technology stack was defined for both front-end and back-end, the implementation process started with building up the API.

Given its RESTful architecture, the API's structure follows Model-View-Controller (MVC) format. Designing a structure for the API requires high-level experience because similar to the foundation of a house, if it is not designed correctly, multiple issues might rise during implementation, which would take a tremendous amount of effort to fix once the house is built. That being said, the design was built around three core components of the API in this project which are restaurant, user, and restaurant-user-connection. Each of these components would contain five elements to make the API as loosely coupled as possible. The elements could be explained as follows:

- A router defines different URLs for different requests. An example for this router would be a URL of <http://localhost:3000/search> which would require the API to search for restaurants.
- A controller handles all requests to perform the correct actions. After getting the request, the router would call the controller to process the request. At this point, the controller breaks down the request information such as location information, user ID, restaurant ID, action taken, etc.
- A service incorporates the extra actions in case they are needed from other components while the controller is handling the request. For example, when searching for restaurants,

the service can perform a query to get the current timestamp to incorporate the returned restaurant data.

- An object creates a prototype of which properties a newly created object of that type should have. If an object of type Restaurant is created, it should have the following properties: place\_id, name, address, categories, views, etc.
- A model handles interactions for data of certain types with the database. A Restaurant model handles all the interactions with restaurant data including insert, update, delete, retrieve, etc.

Data collection was the next step needed during implementation. In order to do so, the API needed to generate queries to collect data from Google <sup>[5]</sup> and Yelp <sup>[6]</sup> which are the main sources for restaurant data given a specific location, then responds to the front-end application's request with a list of restaurants. At this stage, the data that is provided from the source should include name, location/address, price level, images and category. On a side note, as Google and Yelp data is proprietary, in this project, API keys to query data from them were obtained for only academic purposes (free tier).

After the data was obtained, the next question was how it should be presented to the user. From the design, we could see that images are required, not just information about the restaurant. However, only one image reference is returned from Google and Yelp results. Thus, other queries to get extra images from Google and Yelp were required. The task of how to display the image would be handled by the front-end mobile app.

Making sample requests for data from Yelp came with a couple issues: rate limit and image list limit. This was when the things have become very challenging. Yelp only allows ten queries per second and three images per restaurant. That being said, Yelp data is now mainly

used to cross-check with Google data and catch the restaurant that Google may not have data for, as Yelp is more well known for its review data. On the other hand, Google does not have a query rate limit and image list limit. Thus, the application runs all results from Yelp through Google for extra information.

One major issue arose at this point. As the restaurant data was pulled from Google and Yelp, there were duplicates. Thus, this restaurant data should be handled properly by using a mapping system that would not take in data which had already been in the database. Google and Yelp might have similar data, but they organize their data in different ways. For example, the address information of a Restaurant object coming from Google is stored under vicinity, and this field contains the full address of the restaurant including street name, city, and zip code. On the other hand, Yelp breaks down the address information into different fields such as address, city, zip code, state, and country. String type data was also not the reliable fields that could be used to eliminate duplicates as there could be special characters in this type of data, and Google or Yelp handle them differently. Eventually, the information chosen for mapping system to get rid of duplicate restaurants was the location information with latitude and longitude. Even though this information was float type and was not super accurate, after rounding the location to the fifth decimal point. It was much more reliable to use it with the mapping system to avoid duplicates than other data.

Overall, during API implementation, there was a significant amount of work put in with lots of trial-and-error attempts to make sure all of the data was handled and processed correctly. As this data would be used to train the AI models in the next steps, if a mistake happened along the way, it could have caused incorrect results for the experiments.

```

{
  "geometry": {
    "location": {
      "lat": 37.344459,
      "lng": -121.838002
    },
    "viewport": {}
  },
  "icon": "https://maps.gstatic.com/mapfiles/place_api/icons/restaurant-71.png",
  "id": "022beae10dff8f3de57a303f0e0605479c758cef",
  "name": "La Mejor Taqueria",
  "opening_hours": {
    "open_now": true
  },
  "photos": [],
  "place_id": "ChIJG798FjfNj4AR7Th0RBwZWU8",
  "plus_code": {
    "compound_code": "85V6+QQ San Jose, California",
    "global_code": "849W85V6+QQ"
  },
  "price_level": 2,
  "rating": 4.3,
  "reference": "ChIJG798FjfNj4AR7Th0RBwZWU8",
  "scope": "GOOGLE",
  "types": [
    "restaurant",
    "point_of_interest",
    "food",
    "establishment"
  ],
  "vicinity": "2003 Story Rd # 975, San Jose"
},

```

Figure 4. A sample result from a Google request

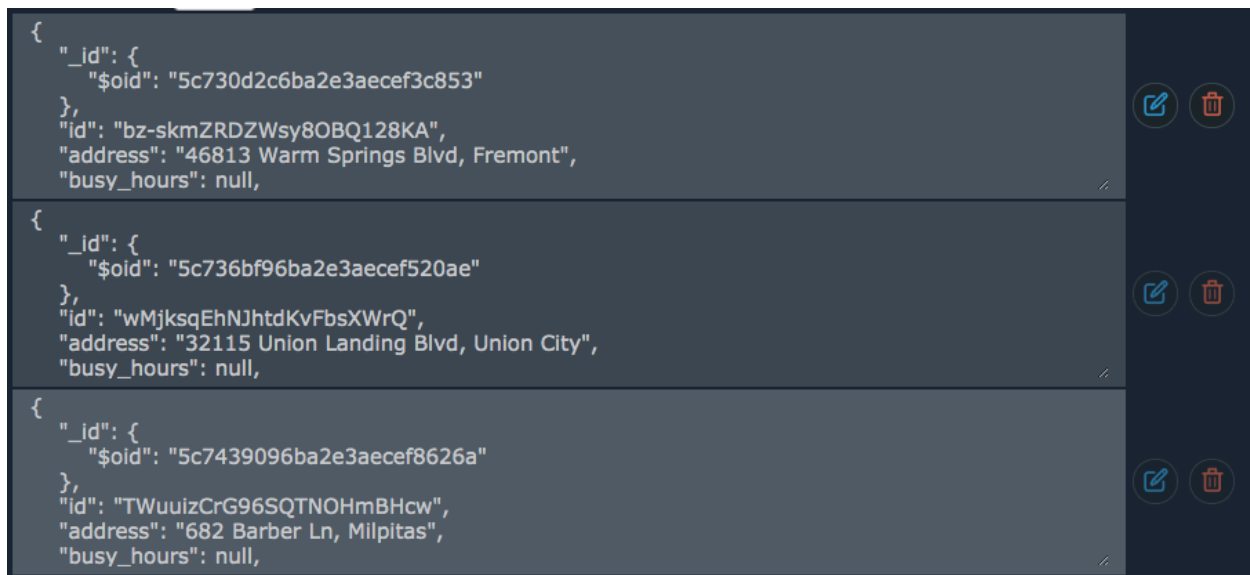
```

{
  "id": "DFz_ahwurm2iI-diUE0iLw",
  "alias": "mexican-hot-dogs-san-jose",
  "name": "Mexican Hot Dogs",
  "image_url": "https://s3-media4.fl.yelpcdn.com/bphoto/HdjIieAZdScDLtRs-xIsaq/o.jpg",
  "is_closed": false,
  "url": "https://www.yelp.com/biz/mexican-hot-dogs-san-jose?adjust_creative=WcXgJyb8BoQN0PJeH47AMQ&utm_campaign=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=WcXgJyb8BoQN0PJeH47AMQ",
  "review_count": 36,
  "categories": [],
  "rating": 5,
  "coordinates": {
    "latitude": 37.351923,
    "longitude": -121.824805
  },
  "transactions": [],
  "price": "$",
  "location": {
    "address1": "2770 Story Rd",
    "address2": "",
    "address3": "",
    "city": "San Jose",
    "zip_code": "95127",
    "country": "US",
    "state": "CA",
    "display_address": [
      "2770 Story Rd",
      "San Jose, CA 95127"
    ]
  },
  "phone": "",
  "display_phone": "",
  "distance": 606.3315765431433
},

```

Figure 5. A sample result from a Yelp request

After a few API testing iterations, one important thing was noticed: a large amount of queries was being sent to Google. This was because when performing a query to search for restaurants at certain locations, the results only show one image reference. To get more images, an additional request needed to be made to the Google API. Therefore, extra queries were needed to get more images (at least ten), and when each image is shown, another query is also made. This created another challenge on how to limit the queries to Google, and the solution was caching data to reduce processing time and to prevent the application from surpassing the free tier. However, how the caching system should be inserted into the API was not as simple as it seemed. After weighing the pros and cons of different possible options, the caching system was implemented to store the restaurant data before it was sent back to the client. As a result, when the application requests for restaurant data and the restaurants are in the database, the API can just return them instead of making repetitive queries to Google.



```
{
  "_id": {
    "$oid": "5c730d2c6ba2e3aecef3c853"
  },
  "id": "bz-skmZRDZWsy8OBQ128KA",
  "address": "46813 Warm Springs Blvd, Fremont",
  "busy_hours": null,
}

{
  "_id": {
    "$oid": "5c736bf96ba2e3aecef520ae"
  },
  "id": "wMjksqEhNJhtdKvFbsXWrQ",
  "address": "32115 Union Landing Blvd, Union City",
  "busy_hours": null,
}

{
  "_id": {
    "$oid": "5c7439096ba2e3aecef8626a"
  },
  "id": "TWuulzCrG96SQTNOHmBHcw",
  "address": "682 Barber Ln, Milpitas",
  "busy_hours": null,
}
```

Figure 6. Samples of Restaurant object stored in database



## 4.2 External Services Integration for API

Oftentimes, food decisions rely on some other factors, not the restaurants themselves<sup>[7]</sup>. For example, during cold weather, people tend to prefer warm food like soup or pasta rather than sushi. In addition, other factors such as traffic or waiting time also influence one's decision on what to eat. Because of these reasons, extra temporal information also needs to be collected to help the AI model make better suggestions given a certain time of day. Having said that, only Google and Yelp APIs are not enough to gather information for all of the factors. Thus, the next goal was to collect data from other external sources besides Google and Yelp to enhance the suggestions made by the AI model later on.

As mentioned above, three extra types of information were added to the API: temperature at a given location, time to commute to the restaurant from that location and the busy hours of the restaurant. For temperature, the Open Weather Map<sup>[8]</sup> service was used. The reason it was chosen was because its free tier is sufficient enough for testing purposes. The Google Distance Matrix and Google Place APIs served as the main source for traffic information and busy hours.

```
"coord": {
  "lon": -121.83,
  "lat": 37.35
},
"weather": [
  {
    "id": 502,
    "main": "Rain",
    "description": "heavy intensity rain",
    "icon": "10n"
  },
  {
    "id": 701,
    "main": "Mist",
    "description": "mist",
    "icon": "50n"
  }
],
"base": "stations",
"main": {
  "temp": 287.67,
  "pressure": 1002,
  "humidity": 89,
  "temp_min": 287.05,
  "temp_max": 288.75
},
```

Figure 7. A sample response from Open Weather Map

Since this data is external, it could be gathered at the client side or the server side. However, to maintain consistency that the front-end app should only handle the task of displaying data and to keep the number of queries as low as possible, this service is implemented within the API. Once the client makes a request to the API to pull the restaurant list, the API would then collect data from Google and Yelp APIs. At this point, after getting the restaurant results, the API also makes requests to external services with the current location of the client and the restaurant data to collect extra information that does not come from Google and Yelp. Restaurant data, after being incorporated with the external information, would be sent down to the client as well as written to the database.

One thing that needs to be mentioned here is that with this external data, it usually does not stay fixed like the other information about restaurants. Temperature can change depending on the day, travel time can change based on the time of the day and busy hours can be different at different timestamps. Therefore, this data is not permanently saved onto the database. Instead, it would be pulled from the external services every time the client makes a request. The good thing about this is more data would be collected when a user takes actions on any restaurants. This, in turn, would be very useful later on when it comes to training the AI models. However, the trade-off for extra information is it would take longer for the server to process this data and combine it into the results sent back to the client.

In the big picture, we can see that, with AI, data plays a very important role, yet collecting data would always come with trade-offs. Thus, finding a good balance between the two is something one needs to keep in mind when it comes to mining big data.

## 4.3 Front-end Implementation

React Native was chosen to develop the front-end mobile app and the first step was to set up the environment for it. A machine that runs OSX was required because in order to build the app for iOS, it can only be developed on OSX and there is no strict requirement to build an app for Android devices in regards to operating system.

The first thing the app should do after it is installed into a device is getting a unique ID for that user. At this point, authentication was not a really concern to get the ID, but it was for the AI implementation later to learn about each specific user. Thus, the app makes a request to the API and a unique ID is returned if the app is installed for the first time.

The main feature of the app was the ability to allow the user to swipe on the deck of cards. After much testing and consideration, a plug-in called React Native Deck Swiper was selected for this task. There were a few plug-ins that support the same functionality. However, React Native Deck Swiper was the easiest and most supported one. Basically, after getting the restaurant data from the API, the app would then get one of the image references from the return list and display it on the card. Besides the image, there is other information that should be displayed as well, and to keep the symmetrical design of the app, four types of information were selected: restaurant name, distance to the restaurant, price level and busyness level. The plug-in supports four different actions, which are swiping in four directions: up, down, left and right, but as of this point, only two directions were enabled. Swiping left indicates the user does not have any interest for that restaurant and swiping right indicates that the user likes that restaurant, but does not want to go there at the moment and the next restaurant should show up.

In each restaurant card, it is also a great feature if the user can interact with the images, not just only one per restaurant. Thus, the image references were set up to become a carousel which holds no more than ten images. The app also allows the user to pan or zoom the image using fingers, and for this specific feature, a plug-in called React Native Image Viewer was utilized. Essentially, when the user taps on the middle portion of the screen divided into three equal parts, the app would enable a new view which contains the image in the black background so that the user can pan and zoom with their fingers. When the app is in this mode, the sliding action is also enabled and the user can slide left or right to navigate to next or previous image accordingly. In order to get out of that mode, the user can swipe down and the app brings back the initial mode which show the deck of cards.

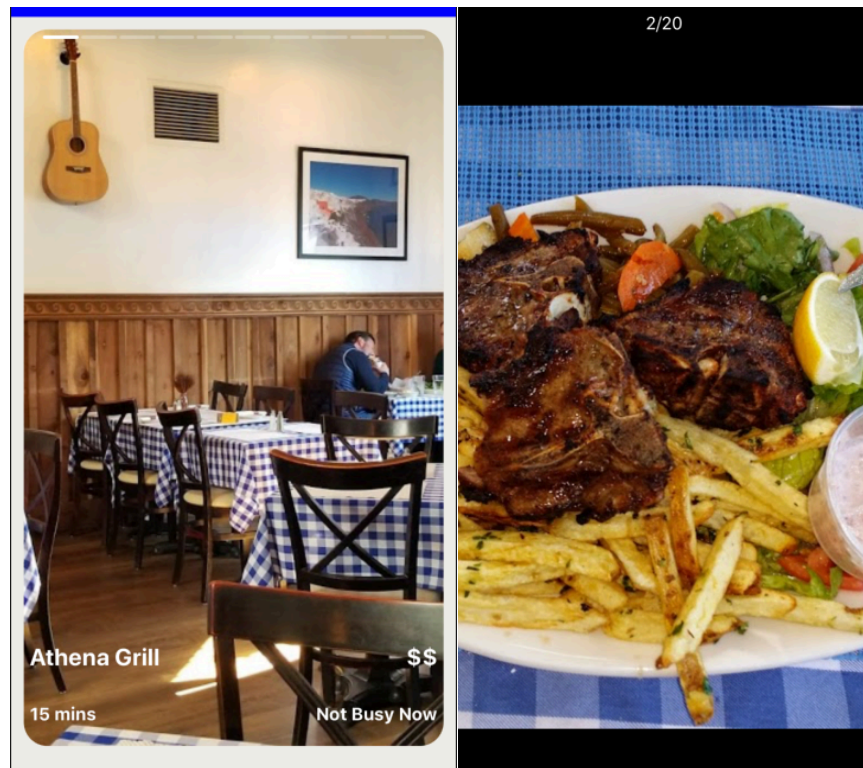


Figure 8. Image interactions within the app

## 4.4 Processing and Filtering Data for AI

When it comes to restaurant data, one would think of Yelp as the first important source that comes to mind. It is easy to understand because Yelp is one of the largest review systems for restaurants with lots of images, ratings and reviews. Taking advantage of the huge amount of data coming in every day, Yelp has hosted a challenge<sup>[9]</sup> with its anonymous available data for people who want to help improve its recommendation system and because the core component of this project is using AI to improve the suggestions, Yelp's dataset was chosen to train the models.

The current dataset of Yelp's challenge is about 12 Gigabytes and it includes business data, user data, review data and photo data. In this project, due to time and resource constraints, business data and review data were mainly utilized. With business data, the main information needed was business identification and food category. As Yelp primarily focuses on reviews of businesses, there were lots of different types of businesses in this data, not just restaurants. Therefore, a couple filters were created to make sure the leftover training data for business only contains restaurant information. There were two ways to filter these businesses. One way is filtering business whose categories are not relevant to food. All of the categories could be obtained by scanning through 188,594 business objects with some of the irrelevant categories being: "salons", "insurance", "fitness", etc. The other way is filtering the businesses that have relevant attributes to restaurant such as: "GoodForMeal", "RestaurantsDelivery", "RestaurantsReservations", etc. As some businesses do not have both categories and attribute information, two ways, mentioned above, were used to get only businesses that are restaurants.

```
{
  "business_id": "Apn5Q_b6Nz61Tq4XzPdf9A",
  "name": "Minhas Micro Brewery",
  "neighborhood": "",
  "address": "1314 44 Avenue NE",
  "city": "Calgary",
  "state": "AB",
  "postal_code": "T2E 6L6",
  "latitude": 51.0918130155,
  "longitude": -114.031674872,
  "stars": 4.0,
  "review_count": 24,
  "is_open": 1,
  "attributes": {
    "BikeParking": "False",
    "BusinessAcceptsCreditCards": "True",
    "BusinessParking": { "garage": False, "street": True, "validated": False, "lot": False, "valet": False },
    "GoodForKids": "True",
    "HasTV": "True",
    "NoiseLevel": "average",
    "OutdoorSeating": "False",
    "RestaurantsAttire": "casual",
    "RestaurantsDelivery": "False",
    "RestaurantsGoodForGroups": "True",
    "RestaurantsPriceRange2": "2",
    "RestaurantsReservations": "True",
    "RestaurantsTakeOut": "True"
  },
  "categories": "Tours, Breweries, Pizza, Restaurants, Food, Hotels & Travel",
  "hours": {
    "Monday": "8:30-17:0",
    "Tuesday": "11:0-21:0",
    "Wednesday": "11:0-21:0",
    "Thursday": "11:0-21:0",
    "Friday": "11:0-21:0",
    "Saturday": "11:0-21:0"
  }
},
```

Figure 9. A sample of a Restaurant object from the Yelp dataset

Since businesses are of all types, review data contains reviews for all of these businesses as well. Thus, extra filters were created to get only reviews for the filtered businesses. Because business data was resolved at first, the rest of the data at this point would be easier to filter since it just needs to be related to the businesses. Thus, the filter for this review data would be any review that contains restaurant ID. In the review data, the main information which is rating is also included. Subsequently, following review data, user data was filtered based on user ID in each review itself. In addition, user data did not need to contain unnecessary information at this point such as age, gender, number of reviews, etc. as these users would not use the app from this project, but the data was used to train a baseline model.

As the Yelp dataset contained the string type, the entire dataset after being filtered needed to be converted into integer or float types. The reason for this was because Tensorflow, at the time this project was developed, could only support these types of data. Therefore, a mapping system was created to assign each user ID and restaurant ID a unique number. Since restaurant categories had string type, they were also converted into integer.

The mapping system would start with the business, user, and category data to export three files containing the indexes of the restaurant, user, and category appearances in the data accordingly. After that, the mapping system continued by going through every review in the review data to map each restaurant ID, user ID, and category with their proper indexes. This process took quite some time due to the large amount of data, and for each review, there were a few checks to make sure no duplicates were generated as well as no reviews were skipped.

The total number of all different data collections are as follows: 1,418,418 reviews with ratings, 32,516 restaurants, 191,904 users and 69 restaurant categories.

```
user_id · business_id · category · rating
0 · 0 · 6 · 5
0 · 1 · 0 · 1
0 · 2 · 2 · 2
0 · 3 · 6 · 5
0 · 4 · 2 · 5
0 · 5 · 6 · 5
0 · 6 · 0 · 4
1 · 15 · 0 · 5
1 · 16 · 5 · 2
1 · 17 · 0 · 5
1 · 18 · 14 · 3
1 · 19 · 0 · 2
1 · 20 · 0 · 5
1 · 21 · 9 · 5
```

Figure 10. A snippet of the dataset after being processed

## 4.5 Train Static AI Model

In order to evaluate and compare the results for different neural network architectures, a matrix factorization with neural networks would be the first option to set up the baseline with the given dataset. The goal of this static model was to serve as a first layer in the two-layer recommendation system to filter the irrelevant options from the list of random selected restaurants. At this point, the data that could be utilized included business IDs, user IDs and ratings. The structure of this baseline model can be described as follows: business IDs and user IDs were initially fed into an input layer. After that, this input layer would run the data through an Embedding Layer. An Embedding Layer is a neural network layer which is provided by Keras. It is often used when training AI models with text data. A requirement for the Embedding Layer is that each word needs to be represented by a unique integer. Thereby, the Embedding Layer can translate these word values into a smaller vector space and thus making the data computationally efficient. However, in this case, after being translated to a smaller output vector dimension, the data would be in a matrix format. Therefore, a Flatten Layer is added after the Embedding Layer to flatten the data into one-dimension format because business IDs and user IDs are just different unique values that have no correlations. Lastly, the two output vectors gotten from the Flatten Layer would be combined using a Dot Layer. This Dot Layer simply computes a dot product for the data from the two input vectors. The model was then compiled with data from business IDs, user IDs as two input vectors and ratings as the output vector.

The baseline model was set up to train for ten epochs and MSE was chosen to be the loss function. The embedding output dimension value for this baseline model was 30. For each epoch, the model took roughly 90 minutes to complete. The MSE value went down from 15.7 to 0.7 and accuracy went up from 0.33 to 0.81. The results were a little surprising, but could be



explained as the input data was quite large and a little sparse. This was because some users only have one rating, and some restaurants are only rated once.

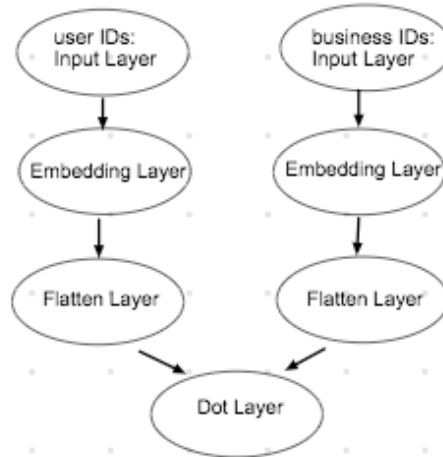


Figure 11. The structure of the baseline model

After having the results from the baseline model, the next step was to see if it could be improved. Thus, a few fully connected layers were introduced into the architecture of the model. A fully connected layer in a neural network is a linear operation which maps every input from one layer to every output of another layer by a weight. However, a dimension of the output layer can decrease throughout the neural network as data is condensed to provide the last meaningful output values. As a result, the Dot Layer in the baseline model was replaced by a Concatenate Layer just to combine the business ID and user ID vectors without producing any product. Following the Concatenate Layer, there were a total of three Dense Layers implemented into the neural network with three different output dimensions: 128, 64 and 1. In addition, to avoid over-fitting problem, two Dropout Layers were inserted between the three Dense Layers with the dropout rate of 0.2, randomly setting 20% of inputs to zero.

The new model was trained for five epochs and each epoch took around 23 minutes to complete. This was clearly a significant reduction in training time. The MSE loss value rapidly decreased from 2.4 to 0.3 after the training was finished. The accuracy also increased from 0.31 to 0.82. The model was then tested with other slightly modified architectures, adding additional dense layers and changing some settings such as output dimensions, embedding value, dropout rate, etc., but the results did not seem to improve much with the increase in training time trade-off. Therefore, at that point, the optimal architecture was three Dense Layers with two Dropout Layers.

Having the model set up was a good start, but how to utilize it with the API was a big question. The reason for this is because the model was trained on the Yelp dataset, but the app would not be used by the users from the Yelp dataset. Similarly, the restaurants from the Yelp dataset are not the same as the ones that the app would receive from the API. Thus, it required some correlations between the data that the model was trained on and the data that was gathered by the API to incorporate them. The only answer for this question was the restaurant category. As mentioned earlier in the data processing section, there were a total of 69 restaurant categories in the Yelp dataset, and most of them appear in the data from the API. Therefore, another vector of restaurant category was implemented into the neural network. The architecture remained the same, but now the model would be compiled with one more input vector, which is restaurant category.

As the training process was handled, the next step was making predictions using the model. Because there were no correlations between users in two sources of data like those mentioned above, an average user who had a decent amount of ratings (15) in the Yelp dataset, was selected to be the input for the model in this project.

## 4.6 Train Reinforcement Learning Model

Since the above static AI model served as the first layer for the recommendation system, the reinforcement learning model would serve as the second layer to learn and make suggestions for a specific user. How to train the model and where the training data was from, were the two challenging questions at this point.

From the front-end app design, there is some data which could be utilized. This data was collected from the activities that the user could perform using swiping actions to indicate his/her interest to a specific restaurant. This data was stored in the restaurant-user-connection table in the database. However, it was organized in an object type format in which each field of the objects contains different types of information. These types of information were date, timestamp, restaurant rating, restaurant price, restaurant category, temperature, distance, busyness, and the action that the user took. Since the goal of this reinforcement learning was to learn about the user's actions and explore some patterns deriving from those actions, a LSTM neural network was the chosen option to train the model. A LSTM is a common neural network that is used to train on sequence data. One example for its usage is a text generator. In any language, sentences are formed following certain structures. Here, the job of a LSTM is to train a model which can learn those structures from a large amount of text data so that when an incomplete sentence is fed to the model, it is able to generate the next words from the patterns that it has learned.

As the restaurant-user-connection data is not a sequence data type, the question now was how it could be processed to become sequence data. This step required a certain level of knowledge about neural networks and careful consideration, because if the data was processed improperly, the training process as well as the model could have become meaningless.

One possible solution for the above question was unfolding all of the Restaurant objects and treating them as a small portion of a text dataset. In order to do so, each Restaurant-User-Connection object was unfolded to become an array of values. Each of these values came from the data in the Restaurant-User-Connection object after being converted all to numerical values. Most of the data had already been in integer or float types except for date and restaurant category. These two fields needed to be converted differently. A date value was converted to a range from 0 to 6 with 0 being Monday and 6 being Sunday. There were 69 restaurant categories and thus those were mapped to values from 0 to 68. However, there was one important fact about neural networks: if categorical data is not handled in the right way, the model might treat some with higher values better than the others. To remedy this problem, Label Encoder and One Hot Encoder were applied. A Label Encoder converts labels for categorical data into numbers, and One Hot Encoder is used to convert decimal data into binary format.

Label Encoding			One Hot Encoding			
Food Name	Categorical #	Calories				
Apple	1	95				
Chicken	2	231				
Broccoli	3	50				

→

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Figure 12. An example of Label Encoding and One Hot Encoding

In this model, the anticipated output should be which action the model thinks the user would most likely perform for a specific restaurant. Types of user action were still undetermined at this point as the app needed to be tested first to get some feedback from the testers. Nevertheless, the summary of the training process can be described as follows: a set of Restaurant-User-Connection objects were unfolded into an array of values. In this case the set

had 20 restaurants. A random number of sets would be selected from the database to be fed into the model as the training data in which each set uses 19 restaurants including user actions and the 20<sup>th</sup> restaurant's information as the input. The output, in this case, would be the action that the user has performed on the 20<sup>th</sup> restaurant. After training the model, the model can be used by the following procedure: when the most recent 19 restaurants that a specific user performs the actions on would be selected to unfold, the new restaurant information that the model has to make the prediction on is appended to the unfolded restaurant data. This data would then be given to the model and the model would make the prediction for what action it thinks the user would most likely perform on the new restaurant.

One of the popular applications for the reinforcement learning is training a model to play games. However, unlike games in which the model would immediately get back some rewards after making a certain action, with this application, the swiping actions for the restaurants are discrete and have no effects on each other. Therefore, in order to train the model to learn about the user's food pattern, a service was scheduled on the server to retrain the model every 24 hours with the most recent user actions.

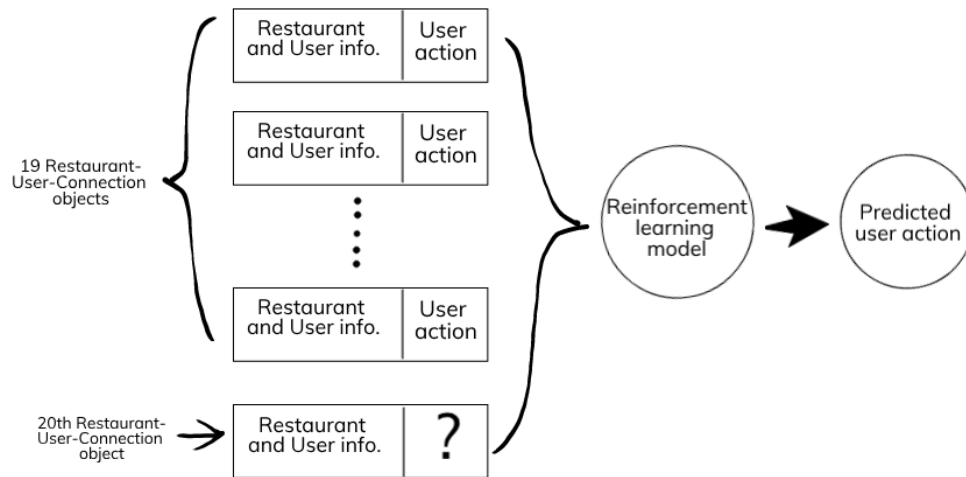


Figure 13. The structure of the reinforcement learning training process

# EXPERIMENTS

There were a total of five testers for this app. According to Nielsen Norman Group<sup>[10]</sup>, this number of testers is sufficient to find almost as many usability problems as you would find using more testers. Lots of informative and valuable feedback was provided from these testers during the whole project. As this project was experimental, A/B testing was performed to decide what would be the best app UI. Due to time constraints, the reinforcement learning portion was trained on two testers and they provided unbiased feedback for evaluation and improvement.

## 5.1 App Experiment

When the first build of the app was released to the testers, there were two variants for the app in terms of button functionalities. In the Variant A, swiping actions allowed the user to go back and forth between the restaurant cards in the deck. On the other hand, Variant B treated swiping actions as “liking” or “disliking” a restaurant.

Because Tinder had become quite popular during the testing phase, the concept of swiping to indicate whether or not one would like the object displayed on the card, which was inspired by Tinder, was the first thing that came to the testers’ mind when they saw the app. Therefore, the most common feedback for Variant A of the app was that the app was not very intuitive in terms of button functionalities. Variant A also created confusion. When the user wanted to like a restaurant, the tester swiped right, and the app displayed the next restaurant in the deck. With the new restaurant, the user, at this point, decided to dislike it, that is swipe left on it. Because of Tinder’s influence, the tester thought the next restaurant would show up, but instead, because the swiping left in Variant A was to go back to the previous restaurant, the

restaurant that they swiped right on, now showed up again. Hence, a loop was generated. Variant B of the app made more sense to the testers. However, a majority of the testers agreed that the concept of “disliking” did not apply sometimes as they simply had not gone to some of the restaurants before, and thus the testers could not say that they did not like these restaurants.

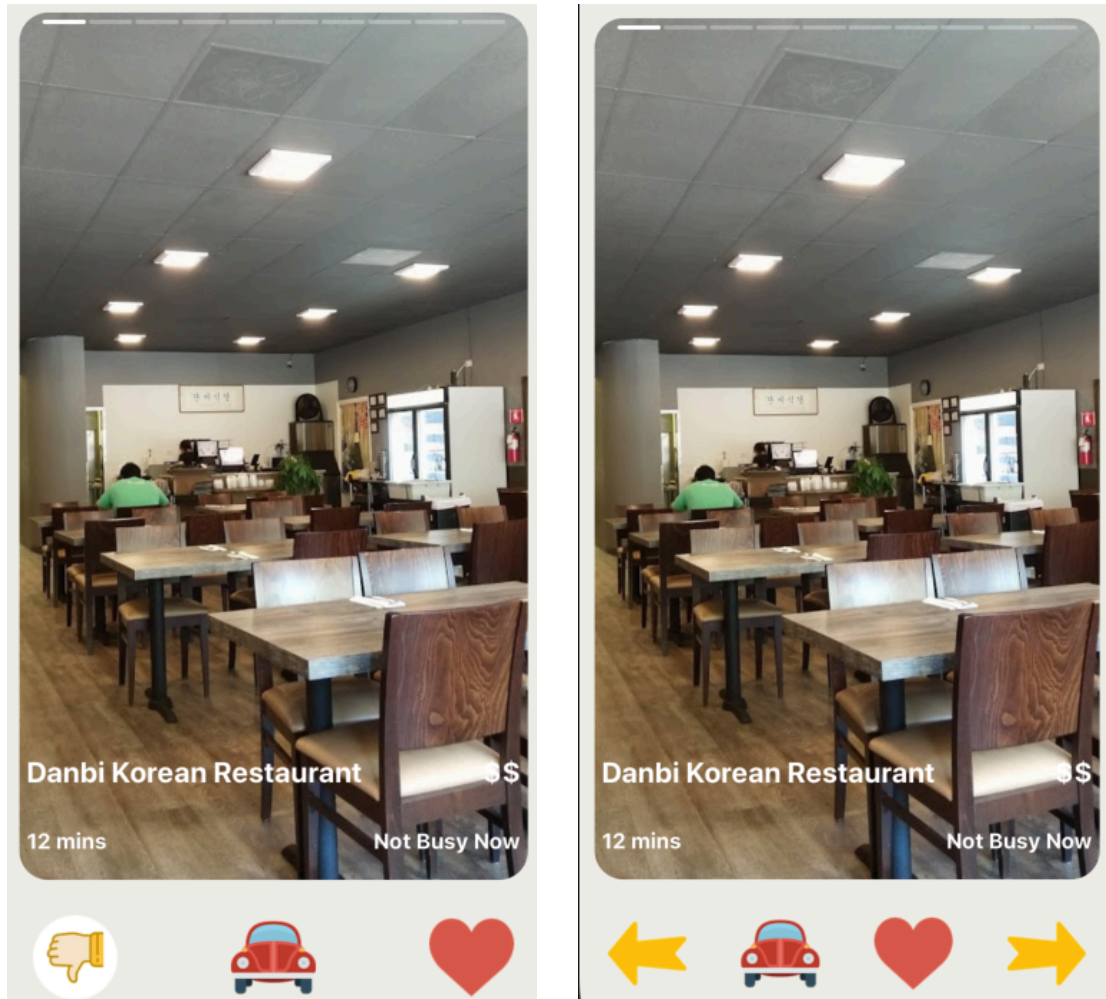


Figure 14. Variant A (left) and Variant B (right) of the app

Influenced by the testers, a new state which is called “neutral” was created for the app. This mode was used for the case in which the user neither likes nor dislikes a restaurant. The

testers also stated that sometimes they swiped too fast and skipped some restaurants. Therefore, a feature which allows the user to go back to the previous restaurant would be nice to have. After incorporating all of the testers' feedback with justifications, a final design had five buttons in total. Three buttons to indicate the user's actions: like, dislike, and neutral. Each of these buttons would give the action a score ranging from 0 to 2 accordingly. One button was designated to indicate that the user wanted to navigate to restaurant at that moment. This button, besides showing the name and address of the restaurant, also ranks the user action as 3 points. The points from the last four buttons were used to train the reinforcement learning model. The last button, as mentioned above, was used to go back to the previous restaurant in case the user swipes too quickly.

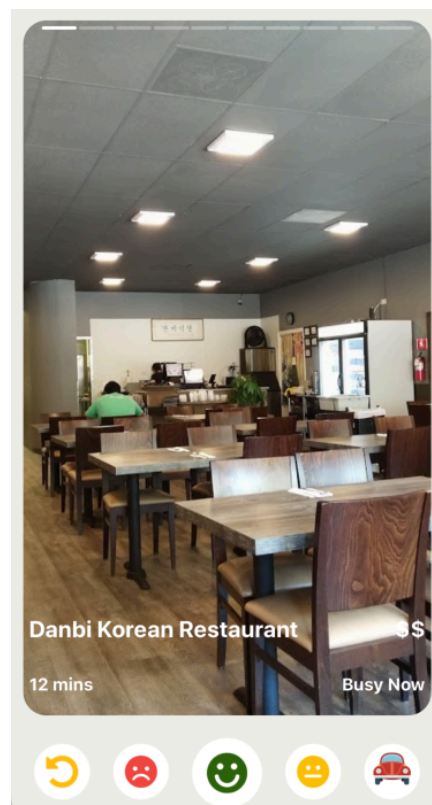


Figure 15. Final design of the app



## 5.2 AI Model Experiment

For the first release of the app, only the static model, which was the first layer in the recommendation system, was incorporated into the API. This was because the reinforcement learning model needed some data from the testers to train on.

For the first layer of the recommendation system, the static model predicted the ratings that the user would give to the restaurants. By doing this, the goal was to sort all of the restaurants by the predicted ratings so that the restaurants which the model thought the user would rate high were pushed to the top of the list, thus speeding up the food decision making process. After running the list of the restaurants through the model, surprisingly, the model performed quite well. The prediction ratings were very close to their actual ratings of the restaurants which were obtained from Google and Yelp.

After getting some data from the testers, the second build of the app was released, and this time, the reinforcement learning model was combined with the static model. The results from the static model were run through the reinforcement learning model to get the predicted actions of the user. At first, the testers did not notice the difference, as the predicted actions were not too accurate. A majority of the predicted action values were the same since the model was trained on very little data. The model accuracy only went up to 70%. Nevertheless, as the testers used the app more, the accuracy started to improve. Even though it was not much, the predicted action values became more diverse at this point. One important point was observed from the experiment: after the user had swiped left on a few restaurants that have the same category, when a new restaurant at a different location was gathered, even though its rating was not so low because it has the same category that the user had swiped left on before, the model tended to push it down in the list of restaurants.

```

    address: "408 Barber Ln, Milpitas"
  ▶ busy_hours: (7) [{...}, {...}, {...}, {...}, {...}, {...}, {...}]
    dislikes: null
  ▶ distance: {distance: {...}, duration: {...}, durationInTraffic: {...}}
    favored: null
    id: "8KayUocIVMfgfEuqP1Q-rw"
    in_db: true
    likes: null
  ▶ location: {lat: 37.4218596148519, lng: -121.916425418683}
    name: "Pepper Lunch USA"
    open_now: true
  ▶ photos: (10) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}]
    place_id: "ChIJQfdhQCbJj4ARE9zXHEhMh-Q"
    predicted_action: 2
    predicted_rating: 3.4051642417907715
    price: 2
    rating: 3.5
  ▶ temperature: {temp: 292.59, pressure: 1015, humidity: 72, temp_min: 289.15, temp_max: 295.93}

```

Figure 16. A sample of a restaurant after with model’s prediction

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 20)	1760
dense_1 (Dense)	(None, 4)	84

Total params: 1,844  
 Trainable params: 1,844  
 Non-trainable params: 0

```

None
Epoch 1/5
2019-04-26 15:03:01.026312: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your
CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
FMA
300/300 [=====] - 3s 10ms/step - loss: 0.8267 - acc: 0.3083
Epoch 2/5
300/300 [=====] - 2s 8ms/step - loss: 0.6900 - acc: 0.4533
Epoch 3/5
300/300 [=====] - 2s 8ms/step - loss: 0.6356 - acc: 0.6742
Epoch 4/5
300/300 [=====] - 2s 8ms/step - loss: 0.5989 - acc: 0.7017
Epoch 5/5
300/300 [=====] - 2s 8ms/step - loss: 0.5750 - acc: 0.7017

```

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 20)	1760
dense_1 (Dense)	(None, 20)	420
dense_2 (Dense)	(None, 15)	315
dense_3 (Dense)	(None, 10)	160
dense_4 (Dense)	(None, 4)	44

Total params: 2,699  
 Trainable params: 2,699  
 Non-trainable params: 0

```

None
Epoch 1/5
2019-04-27 14:56:05.425934: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
600/600 [=====] - 5s 9ms/step - loss: 0.7159 - acc: 0.4567
Epoch 2/5
600/600 [=====] - 5s 8ms/step - loss: 0.6325 - acc: 0.5512
Epoch 3/5
600/600 [=====] - 5s 8ms/step - loss: 0.5808 - acc: 0.6988
Epoch 4/5
600/600 [=====] - 5s 8ms/step - loss: 0.5491 - acc: 0.7532
Epoch 5/5
600/600 [=====] - 5s 8ms/step - loss: 0.5287 - acc: 0.8034

```

Figure 17. Improvement of reinforcement learning model on more data

## CONCLUSION

From our experimental results, we see that how the data is presented to the user plays a very important role in recommendation systems. In this project, using the Tinder presentation format obviously eased the food decision making process. In regards to future improvements, an activity log or favorites list could be implemented to retain extra important data from the user. This data not only helps the user learn more about the user's food habits, but also could be very useful for AI model training later. In addition, extra specific information about the restaurants the user would like to know can be gathered and provided such as ambiance, parking availability, etc.

Due to the constraints of the project such as time, limited data for the free tier of Google API and limited testers, the app was not tested intensively enough to observe significant results. Nonetheless, from the experiments, we can see that the concept of using AI and especially reinforcement learning can be very useful for food recommendation systems. By nature, food decision making can be a time-consuming process because there are multiple factors involved in it. With the human brain, it is hard to keep track of all the information to decide what would suit one's tastes best at a given time. However, from this project, it is possible to know that, given enough data from different factors that possibly influence food decisions, using AI with a neural network can help make the best suggestions. Future work for the AI component in this project could be to use it to make suggestions for eating with a group. After the model is trained for individual users, it could be used to make suggestions for a group with extra constraints such as meeting halfway, reasonable price for different members of the group or finding the most satisfied restaurants given some food allergy types.

## REFERENCES

- [1] Sawant, Sumedh, and Gina Pai. *Yelp Food Recommendation System - Machine Learning*. Stanford University, 2013, [cs229.stanford.edu/proj2013/SawantPai-YelpFoodRecommendationSystem.pdf](https://cs229.stanford.edu/proj2013/SawantPai-YelpFoodRecommendationSystem.pdf).
- [2] Seo, Sungyong, et al. *Representation Learning of Users and Items for Review ...* University of Southern California, Visa Inc., 2017, [pdfs.semanticscholar.org/4946/89f4522619b887e515aea2b205490b0eb5cd.pdf](https://pdfs.semanticscholar.org/4946/89f4522619b887e515aea2b205490b0eb5cd.pdf).
- [3] Covington, Paul, et al. "Deep Neural Networks for YouTube Recommendations – Google AI." *Google AI*, Google Inc., 2016, [ai.google/research/pubs/pub45530](https://ai.google/research/pubs/pub45530).
- [4] Hasselt, Hado van, et al. "Deep Reinforcement Learning with Double Q-Learning - ArXiv." *Google DeepMind*, Google Inc., 2016, [arxiv.org/pdf/1509.06461.pdf](https://arxiv.org/pdf/1509.06461.pdf).
- [5] "Google APIs." *Google Developers*, Google, [developers.google.com/products/](https://developers.google.com/products/).
- [6] "Yelp API." *Yelp Developers*, [www.yelp.com/developers/](https://www.yelp.com/developers/).
- [7] "Supplemental Nutrition Assistance Program (SNAP)." *Food and Nutrition Service, Committee on Examination of the Adequacy of Food Resources and SNAP Allotments; Food and Nutrition Board; Committee on National Statistics; Institute of Medicine; National Research Council; Caswell JA, Yaktine AL, Editors. Supplemental Nutrition Assistance Program: Examining the Evidence to Define Benefit Adequacy. Washington (DC): National Academies Press (US); 2013, [www.fns.usda.gov/snap/supplemental-nutrition-assistance-program-snap](https://www.fns.usda.gov/snap/supplemental-nutrition-assistance-program-snap).*
- [8] OpenWeatherMap.org. "Current Weather and Forecast." *Open Weather Map*, [openweathermap.org/](https://openweathermap.org/).
- [9] "Yelp Dataset Challenge." *Yelp Dataset*, [www.yelp.com/dataset/challenge](https://www.yelp.com/dataset/challenge).
- [10] How Many Test Users in a Usability Study? (n.d.). Retrieved from <https://www.nngroup.com/articles/how-many-test-users/>