IoT Machine Learning Based Parking Management System with Anticipated Prediction of Available Parking Spots

41

Grzegorz Chmaj and Michael Lazeroff

Abstract

Machine Learning based designs provide an extensive means to recognize patterns and do various kinds of predictions. In this paper we apply machine learning to the Internet of Things architecture to optimize access to smart parking. We assume that the system will detect which driver parked at which spot, and also will recognize driver's habit of returning to the car. This way we predict that the spot of the driver walking towards the parking lot or garage will soon be available for other drivers. Drivers looking for a parking spot will receive such information in advance, in a form of "Spot N will be available in X minutes". The design operates over the features offered by smartphone devices: to determine the parking spot, determine the walking driver's position and also serving as a base for a mobile application, so the system is convenient to use and doesn't require additional infrastructure.

Keywords

Machine learning · Internet of Things · Smart cities · Smart parking · GPS · Trajectory · Time estimation · ETA · REST API · Coordinates

G. Chmaj (✉)
Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, USA
e-mail: grzegorz.chmaj@unlv.edu

M. Lazeroff
Department of Computer Science, University of Nevada, Las Vegas, USA
e-mail: lazerm1@unlv.nevada.edu

41.1 Introduction

Smart cities implementing Internet of Things design include multiple components such as traffic light control, air quality monitoring, emergency response, smart buildings and smart parkings. Intense traffic and concentration of high number of cars in the centers of the cities led to development of various solution to minimize the time of finding an empty spot; this way also reducing the emission and in-parking traffic. Many solutions implemented already are based on sensors that are mounted at the parking spot (one sensor per one spot). Sensors are monitoring the presence of a vehicle at the parking spot and then either locally control the red/green indicator light, or are wired to a central control unit, that can count number of available spots and display this information to the drivers entering the parking garage. Smart parking approaches can also feature more advanced options like counting vehicles entering given floor, so number of available spots could be updated before these vehicles reached their spots. Other features are smart routing of exiting vehicles (for garages having multiple exits) and balancing the distribution of vehicles over the floors by the criterion of vicinity of free spots to the elevators (so the more occupancy, the farther available free parking spots are from elevators, regardless the floor).

We propose the live prediction of free parking spots ahead of time – for the optimization of traffic and parking time. In our approach, each driver participating in the system has a smartphone with our parking application installed (safe to assume it's obligatory for company parking garages, but not only). Manual spot entering is used to recognize that a particular driver has just parked in a spot of certain identifier (identifier describes the location on the floor but also the floor number). On the other hand, our design recognizes driver's habit of returning back to the car, so we can predict ahead of time, that the certain driver will be freeing the parking spot

S. Latifi (ed.), *ITNG 2022 19th International Conference on Information Technology-New Generations*, Advances in Intelligent Systems and Computing 1421,
https://doi.org/10.1007/978-3-030-97652-1_41

within some estimated time. This information is presented to drivers that are currently looking for a parking spot. Our idea is flexible and can have multiple factors included, such as different types of parking spots, different day times when a certain spot is available, parking fees and many other. We consider our system as easy to implement, as no other infrastructure like sensors, wiring etc. is required – smartphones, application and active users is sufficient.

41.2 Related Work

Internet of Things, specifically its implementation to cities – resulting in a concepts called Smart Cities faces multiple challenges, at the same time solving many old problems and providing numerous optimizations of city-related factors. The trends for IoT-enabled smart cities were surveyed in [1] – and included the confrontation of user approach and technological approach, along with the overview of components and their then presence on the market (in numbers) and projections for the following years. Authors also included the overview of IoT-Smart City applications, such as transportation and mobility, smart homes, smart infrastructure, IoT solutions for retail, healthcare, energy optimization and brief discussion of other areas related with smart cities. The technological look at the smart cities – in a perspective of services being delivered – were described in [2]. Authors specified the following services: building management, automation and home automation (for buildings), remote parking management, business fleet management and vehicle telematics (for transportation), home security and people protection (for security area), smart healthcare and hospitals (for healthcare) and optimization of distribution and usage of electrical energy (for electricity). The paper includes the results of survey done among IoT experts regarding specified IoT/Smart Cities aspects.

Aside of the research and engineering work being done for IoT Smart Cities in general, a lot of attention is brought to the transportation aspect of smart cities that is closely related to our work presented in this paper. The systematic analysis of transportation in smart cities was described in [3]. The work includes the taxonomy and results of review of 199 smart cities projects – geographically located around the entire globe. Authors of [4] analyze the IoT technologies selection to be applied into smart transportation. The analysis includes considering specific requirements of the application. Most of the technologies that are considered are IoT-specific or closely related. The aspect of data propagation for the smart city transportation was described in [5], where authors analyze the communications between multiple participants of Intelligent Transport System. Several factors are considered: latency, heterogeneous connectivity, mesh architecture, cloud-based approach and many others. Similarly, as in our work, authors included machine learning in the solutions – for the goal of optimization and independent decision making. Also the scenarios for smart cities and smart roads are included. The mobile object authentication, being important in the transportation area was de-scribed in [6]. Authors propose multilayer and hierarchical architecture for mobile objects, including: perception layer (mainly sensors, but also other typical IoT devices), network layer, mobility support layer and application layer. The mobility support layer provides the authentication of the mobile objects that are relocating between different environments and systems.

The convergence of Internet of Things and Machine Learning, being applied to transportation in smart cities was described in [7]. Authors describe multiple solutions in which Intelligent Transportation Systems operate using ML, also pointing out the trends and directions that possibly need more work (also mentioning smart parking). Extensive information is provided about ML techniques applied to the smart transportation. Applications considered in that work are: Route-Optimization-Navigation, Parking, Lights, Accident Detection, Road Anomalies and Infrastructure. Each application is analyzed against the ML solutions applied, including the ML-related details.

41.3 Proposed System

The proposed system predicts the time when a parking space is to become available. This system is most useful in places with high parking density, for example, a university parking lot or garage. The prototype of this system aims to leverage a user's smartphone as much as possible and limit the amount of physical hardware needed in the actual parking structures. Example execution:

(i) An individual is returning to the parking lot where their car is parked.
(ii) The machine learning model predicts that the person is indeed returning to the parking lot – and a time prediction to return is generated.
(iii) Patrons seeking a parking space can view the estimated time for a vacant spot through a mobile application.

Architecture consists of: Frontend Client (Mobile Application), Backend Server, REST API, Machine Learning Pipeline and Database. The Backend Server is the main focus of this paper. However, details about the role of the mobile application and some potential features it could include are discussed.

41.3.1 Mobile App

The main interface for users of the system would be through a mobile application. There are a few different purposes that the mobile application would serve:

- *Provide GPS coordinates to the Backend Server:* The Backend Server utilizes GPS coordinates in the Machine Learning Pipeline. Using the GPS API provided with many smartphone application frameworks, the individual's current location can be sent to the server and processed as needed
- *Allow users to pick their spot and checkout*: One barrier to using GPS coordinates to track a user's location is the loss of precision when inside buildings, including parking garages. Because of the limitation, precise GPS information could not determine which spot a user chooses. Since the system uses as minimal extra hardware as possible, users manually select the parking spot and "check out" when they leave. From a user experience standpoint, this area would benefit most from having an automatic system in place.
- *Display helpful information to those looking for a parking spot*: The mobile app can display beneficial information – such as a map of the parking lot, which spots are currently open, and the times for spots that are estimated to be available.

Specific implementation details and features to be included can be tweaked to suit individual needs. For example, in a university setting where there are many different parking spots for varying roles, it would be beneficial to have additional permissions for each user and potentially only display locations they have access to. Furthermore, existing systems such as parking passes and metered pay could be integrated into the mobile app for a comprehensive experience for the user. Depending on the parking lot provider's exact needs, a network of different parking lots could also be connected and integrated into the application. However, in this system prototype, the three features outlined above are essential to the application (Fig. 41.1).

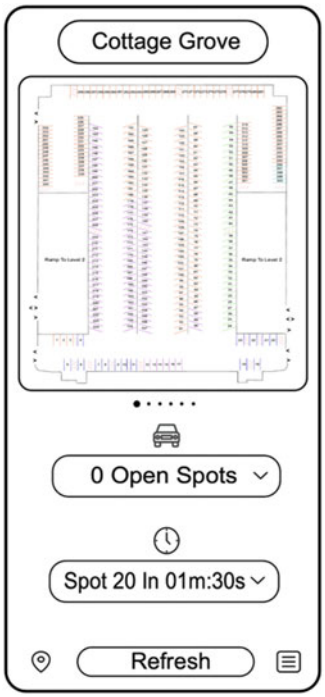


Fig. 41.1 Example mobile UI

41.3.2 Database

Google's Firebase databases were chosen for this system. This database service contains many benefits such as development speed, client syncing, and ease of use [8]. Furthermore, with Firebase's two different types of databases – the Real-Time Database and Firestore, and the tight integration with Google Cloud Platform services like data analytics, it can be quite valuable with IoT systems [9].

- *Real-Time Database*: The Real-Time Database contains the state of the parking lot. Individual spots are mapped in the database with information such as an identifier, type of spot, and relative location. The RTDB also contains the time predictions for spots to open – which can be queried.
- *Firestore*: The Cloud Firestore database contains all other data for the system. This includes information such as users and their permissions and useful information such as coordinate information for system users.

41.3.3 Machine Learning Model

41.3.3.1 Purpose

The goal of the machine learning model is to determine whether a set of coordinates is returning to the parking lot. The model receives a list of GPS coordinates as input and outputs a binary classification of 0 (does not return) or 1 (does return).

41.3.3.2 Dataset

The dataset used consists of GPS coordinates (latitude and longitude pairs) that simulate individuals' trajectories when traveling near the parking garage. The parking lot chosen for the system is a heavily trafficked garage at the University of Nevada, Las Vegas [10]. Entrances to the different locations were labeled and are shown in Fig. 41.2.

Fig. 41.2 Garage entrance labels

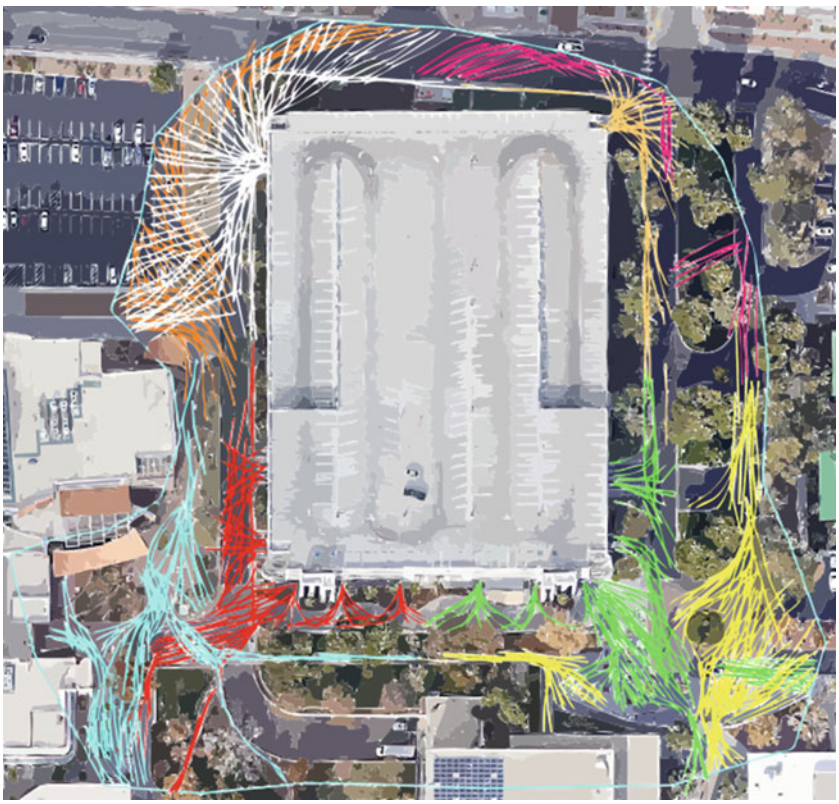


Fig. 41.3 Trajectories

The trajectories were created using Google Earth's path creation tool. The dataset consists of 776 simulated trajectories that are each four coordinates long. Each trajectory was labeled as entering the garage or missing the garage (Fig. 41.3).

The decision to use four long coordinate trajectories was not arbitrary. If the trajectories are too long – the predictions may not be made as frequently as needed. If the trajectories are too short – say, every one or two coordinates, it would be more challenging to train the model to make proper predictions. Four coordinates is a happy medium for achieving frequency in polling and accuracy.

The distributions of the different trajectories are shown in Table 41.1. A unique coloring is given to each category

Table 41.1 Trajectory distributions

Location	Entered garage	Missed garage
A	136 (Red)	131 (Blue)
B	149 (Green)	130 (Yellow)
C	82 (White)	74 (Dark Orange)
D	38 (Light Orange)	36 (Purple)
Total	405	371

of trajectories to distinguish them visually. It is essential to note the decision-making process behind the distribution of trajectories. For this specific garage, entrances A and B are more heavily trafficked than entrances C and D due to their location on campus. This bias towards locations A and B is reflected in the data – however, simulated data can never capture the actual traffic of the entrances to this garage. In a production environment, collecting actual traffic distributions to the garage entries would be crucial for accurate time prediction results.

41.3.3.3 Model Parameters

Each input to the model consists of four pairs of latitude and longitude coordinates (eight inputs total). The dataset is partitioned to a 70/30 training and testing split. Once partitioned, the data is scaled using the *MinMaxScaler* found in the Sklearn Python library. The *MinMaxScaler* is fit transformed to the training data, and then the testing data is subsequently transformed. Once the data is scaled, it is ready to be fed into the model for training.

41.3.3.4 Model Architecture and Training

The model is sequential with three dense layers: an input, a hidden, and an output layer. Eight neurons in the first layer correspond to the four pairs of latitude and longitude coordinates. The input layer uses the Rectified Linear Unit (ReLU) activation function [11]. The single hidden layer contains six neurons and as well uses the ReLU activation function. A single neuron and the sigmoid activation function are used for the output layer. The goal of the output layer is the binary classification of whether the coordinates enter the garage or miss the garage.

Loss for the model is calculated using the Binary Crossentropy function [12]. A learning rate of 0.01 is specified for the Adam optimizer, and the model is trained using 200 epochs as the limit. An early stopping callback function is declared, which monitors the validation loss for the model. It has a 15 epoch patience parameter and aims to prevent potential overfitting of the model.

41.3.3.5 Model Results

Valuable insights can be extracted from the results of the model training. The model accuracy steadily increases throughout 198 epochs, concluding at an average of about 95% accuracy on the training and validation data. A large gap between the training and validation data curves is absent, which suggests no significant amount of overfitting in the accuracy plot. For the model loss plot, the gradual decrease in the curve of both the training and validation data indicates that no considerable overfitting of the model occurs. The decline rate suggests that the model's learning rate is appropriate and does not learn too slow or too fast. The training and validation loss concludes at approximately a 15% loss. Although the accuracy and loss plots suggest that the model is not overfitting to the data during the training phase, testing on unseen data is essential for gauging the model's actual performance (Figs. 41.4, 41.5 and 41.6).

A separate testing dataset was created for testing the trained model. It contains 27 trajectories that miss the garage and 31 trajectories that enter the garage. In Fig. 41.7, the trajectories entering the garage are colored purple, and those missing are blue (Fig. 41.8).

We get precision results of 96% when the model predicts coordinates are not returning to the garage and precision of 97% when the model predicts they will. This is about on par with the results of the training data seen in the accuracy plot. These are promising results – the model is performing just as well with unseen data as it did with the training data. A relatively simple model (recall this model only has three layers) can accurately predict whether a set of coordinates is returning to a location. This finding could extend to other systems trying to determine some type of user intent where coordinates are the primary input. In this system, it is being used for predicting user intent to return to a location, however, this approach could be shaped for other types of intent depending on the specific goals of that system. Comparisons to similar models will be completed in future work.

41.3.4 Backend Server

The Backend Server is where the main work of the system takes place. The goal of the Backend Server is to have a scalable API that the front end (mobile application) can efficiently reference. The API follows the REST architecture and uses the FLASK web framework [13]. The benefits of using FLASK for the REST API were the development speeds, ease of use, and future scalability for features and traffic (Fig. 41.9).

41.3.5 Geofences

A geofence acts as a virtual perimeter around a physical location [14]. To create the geofences necessary for the system, polygons are made using Google Earth by defining

Fig. 41.4 Early stopping callback triggered

```
Epoch 194/200
17/17 [==============================] - 0s 4ms/step - loss: 0.1411
Epoch 195/200
17/17 [==============================] - 0s 3ms/step - loss: 0.1292
Epoch 196/200
17/17 [==============================] - 0s 4ms/step - loss: 0.1269
Epoch 197/200
17/17 [==============================] - 0s 3ms/step - loss: 0.1250
Epoch 198/200
17/17 [==============================] - 0s 3ms/step - loss: 0.1297
Epoch 00198: early stopping
```

Fig. 41.5 Model accuracy

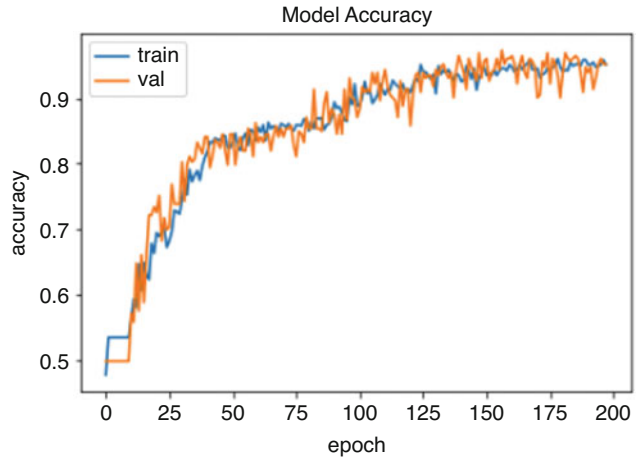
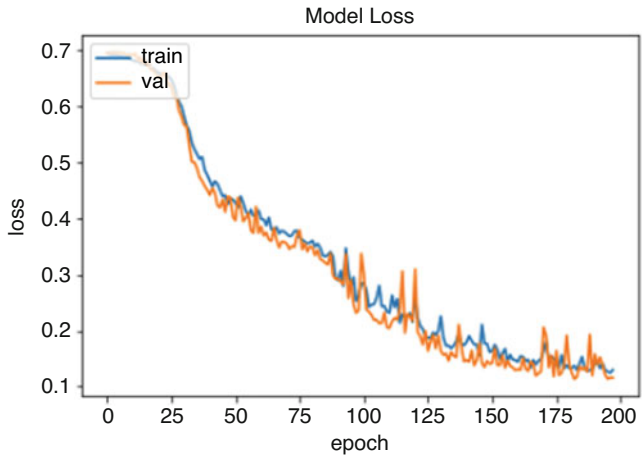


Fig. 41.6 Model loss



bounds around a region. The goal of the geofences is to limit unnecessary processing (Fig. 41.10).

Two geofences play critical roles in the pipeline.

- Inner Geofence – The inner geofence is a polygon defined by coordinates that encompass the garage. If coordinates are found within this geofence, we know that the person has returned to the garage.
- Outer Geofence – The outer geofence denotes the region around the garage where the model can reasonably predict. The perimeter defined by the outer geofence is based on several factors, such as the paths leading back to the lot, the traffic, and the buildings or structures surrounding the lot.

It is simple to determine whether a coordinate is inside the bounds of a geofence. Since a geofence is just a polygon defined by a set of coordinates (latitude and longitude pairs), a coordinate is inside the geofence if it exists inside the polygon. Using the Python math library Shapely, a polygon is defined using the coordinates from Google Earth [15]. Once the polygon object is created, coordinates can then be tested for if they are contained within the polygon.

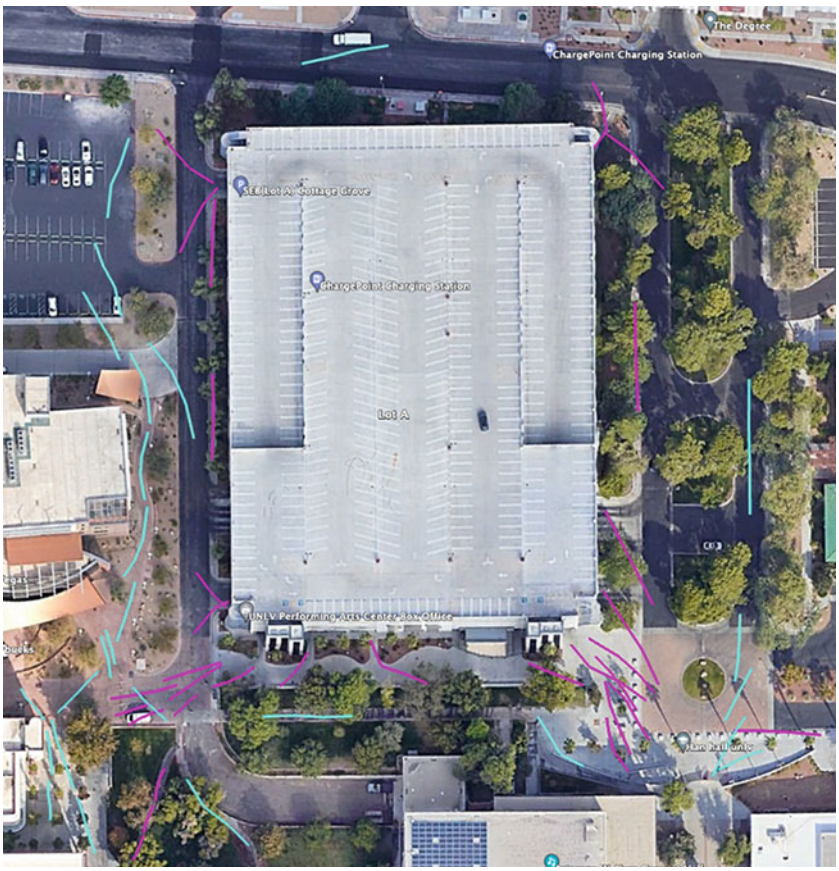


Fig. 41.7 Testing trajectories

Fig. 41.8 Test set results

	precision	recall	f1-score	support
0	0.96	0.96	0.96	27
1	0.97	0.97	0.97	31
accuracy			0.97	58
macro avg	0.97	0.97	0.97	58
weighted avg	0.97	0.97	0.97	58

With the pipeline, the last coordinate provided in the request is first used to determine whether it is inside the inner geofence. If it is, the user is in the garage, so quit. If the user is not in the inner geofence and is not in the outer geofence, quit. The machine learning model runs when the user is within the outer geofence but outside of the inner geofence. The purpose of this again is to maintain reasonable predictions by the model. It is only desirable to run the model when there is a likely chance they are returning to the garage instead of just polling continuously.

41.3.6 Model Prediction Consequences

As the model is a binary classification model, there are only two prediction possibilities – a prediction that the coordinates are going to the garage or not. If the model predicts the coordinates will not return to the garage, then there is no more processing that needs to be done, and the backend will wait for another POST request with more coordinates. If the model does predict that the coordinates will return to the garage, however, then the time prediction part of the pipeline can occur.

41.3.7 Time Prediction

Time predictions will only occur when the model predicts a return to the garage. The approach to predicting time is made as simple as possible in the first iteration of the system. Placemarks are created in Google Earth that define arbitrary "regions" in the garage. Spots are assigned to these different

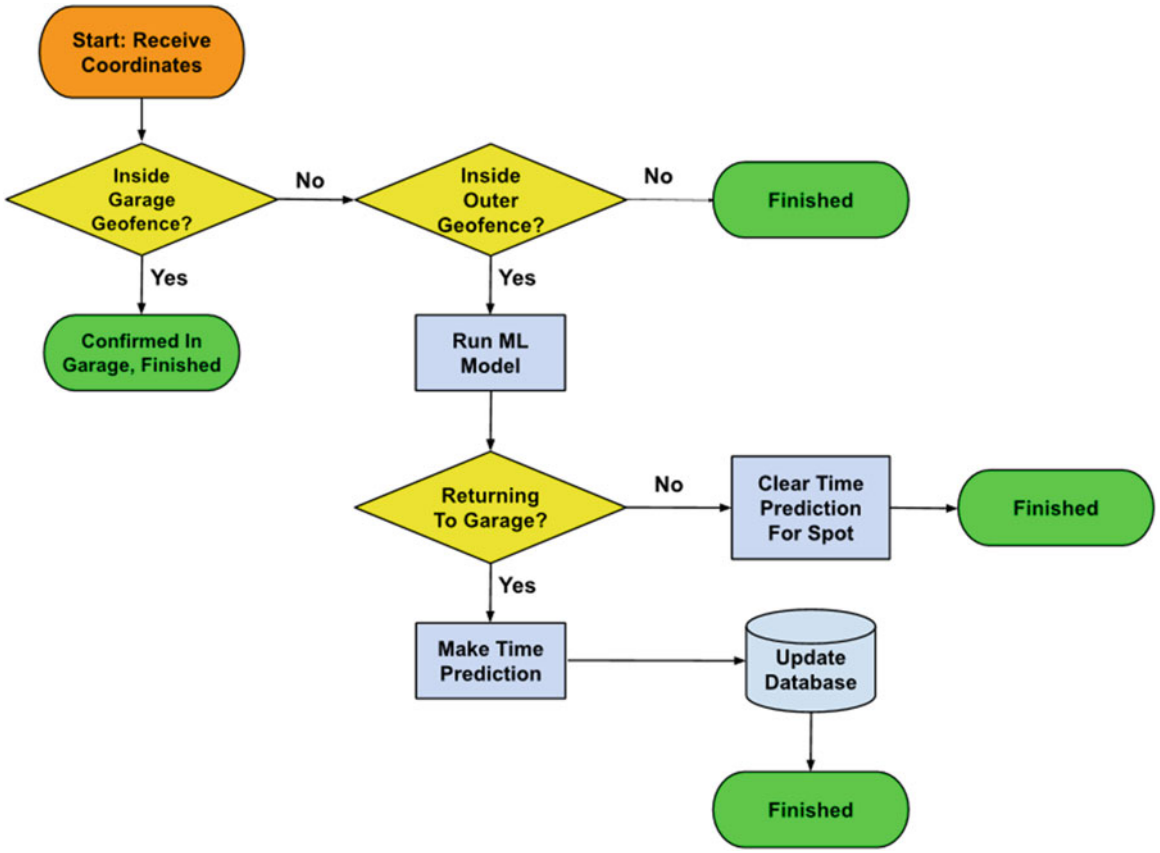


Fig. 41.9 Backend pipeline diagram

Fig. 41.10 Outer geofence

Fig. 41.11 Regional placemarks

regions based on their location. These placemarks are used to calculate the distance from a user's current location and their parking spot location.

$$Time = \frac{Distance}{Speed}$$

Description:

- Take the user's current location (provided in the POST request) and the regional location of their parking spot and calculate the distance.
- Set speed as a constant – to an average walking speed of 1.5 m/s
- Calculate the time using distance and speed

After calculating the time prediction, the database is then updated.

41.3.7.1 Multi-floor Garages

Although this system focused only on the first floor of the parking garage, other floors can be easily scaled using this approach by including a constant travel time to move between floors.

41.3.7.2 Dynamically Calculating Speed

It is not the case that everyone travels at the same speed – especially at a university setting where there are many modes of transportation used to get around. Instead of setting speed as a constant, an individual's speed could be calculated using the coordinate data and time stamps. With this information, we can calculate the movement speed unique to every individual to create more precise time predictions.

41.3.7.3 Accounting for Uncertainties

To account for uncertainties when patrons arrive back at their car and leave the parking lot – a buffer time can be added on top of the prediction time. A buffer time would help account for individuals' time to settle and leave their parking spot (Fig. 41.11).

41.4 Conclusions

The work presented in this paper addresses the problem of congestion in parking garages. It describes the machine learning based approach to predict the availability of parking spot ahead of time – using driver's geographical position, walking path, geofences and other aspects. The proposed system delivered high accuracy of predictions. Once the system detects that a driver walks towards the parked car, it notifies driver(s) currently looking for available spots – that the parking spot with a specified symbol will be available in a certain amount of time. In this preliminary work, the time at which the spot is going to be available is estimated independently of a certain driver, but in the future work the personal

data is going to be included to get more precise individual time predictions. Other elements planned are parking spot selection preferences, different categories of parking spots and consideration of reserved parking spots with flexible time schedule.

Acknowledgment This material is based upon work supported by the National Science Foundation under Grant No. 1950872

References

1. A. Alavi, P. Jiao, W. Buttlar, N. Lajnef, Internet of Things-enabled smart cities: State-of-the-art and future trends. Measurement **129**, 589–606 (2018). https://doi.org/10.1016/j.measurement.2018.07.067. ISSN 0263-2241
2. E. Park, A. del Pobil, S. Kwon, The role of Internet of Things (IoT) in Smart Cities: Technology roadmap-oriented approaches. Sustainability **10**(5), 1388 (2018)
3. T.G. Crainic, G. Perboli, M. Rosano, Q. Wei, Transportation for smart cities: A systematic review, in *The Eleventh International Conference on City Logistics*, June, 12–14, 2019
4. W. Ayoub, A. Ellatif Samhat, M. Mroue, H. Joumaa, F. Nouvel, J.C. Prévotet, Technology selection for IoT-based smart transportation systems, in *Vehicular Ad-Hoc Networks forSmart Cities*, Advances in Intelligent Systems and Computing, ed. by A. Laouiti, A. Qayyum, M. Mohamad Saad, vol. 1144, (Springer, Singapore). https://doi.org/10.1007/978-981-15-3750-9_2
5. A.A. Brincat, et al., The internet of things for intelligent transportation systems in real smart cities scenarios, in *IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019
6. M. Saadeh, A. Sleit, K.E. Sabri, W. Almobaideen, Hierarchical architecture and protocol for mobile object authentication in the context of iot smart cities. J. Netw. Comput. Appl. **121**, 1–19 (2018). https://doi.org/10.1016/j.jnca.2018.07.009
7. F. Zantalis, G. Koulouras, S. Karabetsos, D. Kandris, A review of machine learning and IoT in smart transportation. Future Internet **11**(4), 94 (2019)
8. https://firebase.google.com/docs/database
9. B. Padmaja, E. Patro, S. Mahurkar, G. Akhila, Google firebase based modern IoT system architecture. Int. J. Eng. Res. Technol. **9**(8), 107–110 (2021) Accessed 7 July 2021
10. https://www.unlv.edu/maps/pkg-1
11. A.F. Agarap, Deep learning using rectified linear units (ReLU), *arXiv:1803.08375*, 2018
12. https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy
13. https://flask-restful.readthedocs.io/en/latest/
14. https://www.verizonconnect.com/glossary/what-is-a-geofence/
15. https://shapely.readthedocs.io/en/stable/