

# Parking Availability System

## Preliminary Report

Erik Meurrens, Benjamin Simonson, Ryan Jalloul, Evan Tobon, Samer Khatib

## Abstract

The Parking Availability System is a low-cost, portable edge device designed to operate within parking facilities to monitor their occupancy. The system is designed to address difficulties parking at the University of Florida campus and offers a solution by implementing a smart parking system within parking facilities on campus. It tracks traffic entering and exiting parking facilities using an Ultralytics YOLOv11n lightweight computer vision model deployed on a Raspberry Pi 5 edge device to detect cars transiting into and out of these facilities. A remote automated license plate recognition pipeline is used to read license plates for cars transiting facilities in order to verify, identify, and account for cars entering and exiting. The produced real-time data, as well as general parking facility information, is stored in the cloud using an AWS RDS SQL database. A mobile application can access this data so that users can access real-time parking availability information for parking facilities where this system is configured on the University of Florida campus. By deploying this system, the project aims to reduce the stress and frustration experienced by commuters and visitors to campus by offering real-time information in advance so that they can make informed campus parking decisions early.

## Introduction

The limitations of parking at the University of Florida (UF) have a negative effect on campus life for students, faculty, and staff, acting as a source of stress, wasted time, additional traffic congestion, and increased vehicle emissions. Live parking data for the campus is only available at select locations, making it difficult for those who must drive to campus to find parking during busy periods, as it is impossible to check real-time availability prior to arriving. That lack of live parking data is what this project aims to address. The goal of the Parking Availability System (PAS) is to provide real-time, accessible parking capacity tracking information for individuals who wish to park at the parking facilities on campus. This information would be beneficial for off-campus commuters and visitors to campus, reducing the stress, frustration, and confusion drivers can feel trying to find parking, as well as the congestion that can occur at parking facilities, during busy traffic periods on campus. Our project, the PAS, was inspired to help the UF community, alleviating the symptoms of the lack of easily available parking resources for the campus and promoting quicker and more reliable transportation to and from campus, by offering a way to access parking availability information in real time via PAS.

## Background

UF deploys a similar occupancy tracking system, the Genetec AutoVu system, that uses an automated license plate recognition service to identify and read license plates and count vehicles that are currently in the facility. While effective in license plate identification, this solution lacks availability across campus facilities, only being available in Parking Garage 14. This lack inconveniences drivers searching for available parking when none exists. Additionally, [1] argues in favor of a smart parking system with real-time availability accessible via a mobile application, from the perspective of two civil engineering researchers at UF. However, the proposed system in [1] as well as the current system comes with a significant cost burden and is facilitated via subscription services that may go underutilized. Specifically, this system maintains a “\$6000 servicing cost, \$11,000-\$8,000 per lane, and \$2,495 per camera” [1]. In contrast, our project will provide a more affordable solution to the AutoVu at less than \$200 per camera, as a one-time equipment cost, excluding installation fees. Our project

will also provide virtual observability into parking facility current availability to streamline an individual's search into finding a parking spot.

For the mobile application component, our project is built from the prior art of UF Computer Science alumnus Drew Gill and his EZ Park UF codebase [2], with his permission. This app provided a map for users to view details about parking locations across campus, but did not offer a feature for viewing real-time availability at garages. To speed up development of PAS and apply our system to a user-facing component, we built upon his codebase to implement the mobile application component for our system.

## Needs and Impact Analysis

The need for PAS on campus is exemplified by existing negative opinions of the current parking situation. In a survey conducted by our team, we found that 89% of respondents expressed negative attitudes towards the existing campus parking situation. Anecdotes from respondents included having to spend upwards of 10 minutes to find parking, being unable to find parking near a desired location, and giving up after circling a parking facility without finding an available spot. These results conclude that parking difficulties cause stress, frustration, congestion, and waste time. There is a need to make parking availability information more accessible for the campus community, and PAS would be an effective solution.

TABLE I. RESULTS FOR ATTITUDES RELATING TO CURRENT PARKING SITUATION

Question	Negative Response Rate Towards Current System
"Overall, I am satisfied with the current parking situation on-campus."	89%
"It is easy to find a parking spot within my designated parking permit color."	80%
"Parking availability impacts my campus experience (e.g., attending classes, participating in events, etc.)"	91%
"I avoid driving on campus due to the current parking situation."	90%

This project will benefit the campus community by providing drivers with a mobile application to reference parking resources on campus and view real-time parking availability. This will benefit campus by alleviating the lack of easily accessible parking resources, thus reducing the stress, frustration, and confusion drivers feel trying to find parking on campus. Additionally, this would make it easier for drivers to plan their routes, promoting better traffic flow on campus as drivers will know where they will be parking ahead of time. Thus, PAS will also enable safer driving around campus, improving the safety of pedestrians and drivers alike. However, one potential drawback is privacy concerns, given that this system requires a live image feed to operate. While UF parking facilities are public spaces, we guarantee that PAS will only collect video data for identifying cars transiting parking facilities and no more than what is necessary and for no other purpose. With this in mind, we believe that the benefit to the community outweighs the privacy considerations.

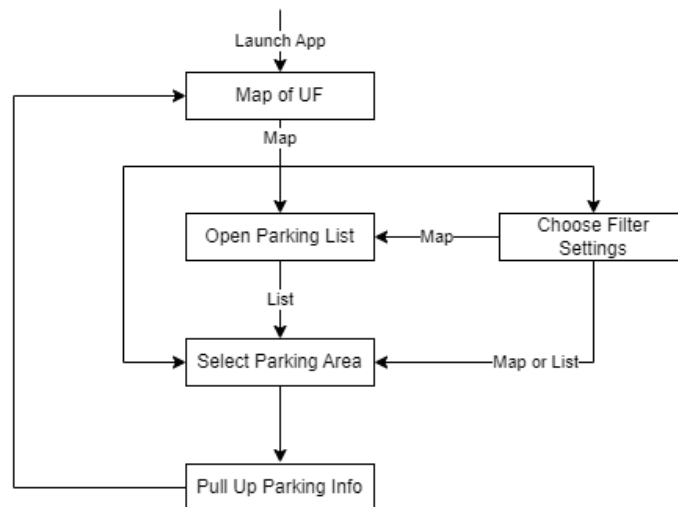
## Design

To achieve the goal of PAS, providing real-time parking information easily accessible to users, our team developed four critical components: a mobile application, an edge AI device to be deployed in parking facilities, the car-detection pipeline, and a backend database to store lot and car information.

### *External Interface*

#### *Mobile Application*

Users can interact with PAS and retrieve real-time parking facility information using a mobile application. Users can navigate the app according to the user flow diagram in Figure 1. Users will navigate a map of the campus with parking facilities plotted on the map. Users can either select a location on the map page or on the list view page. Additionally, users can choose to filter locations based on certain characteristics which affect the locations available on the map and list view. With a selected parking location, users can view the facility's information on the lot info screen. This page's details are updated periodically by polling the lot table in a remote PostgreSQL database. The application communicates with the PostgreSQL database API endpoints via HTTP requests by making 'GET' requests to get information on either a specific lot or all lots.



**Figure 1.** Mobile Application User Flow Diagram

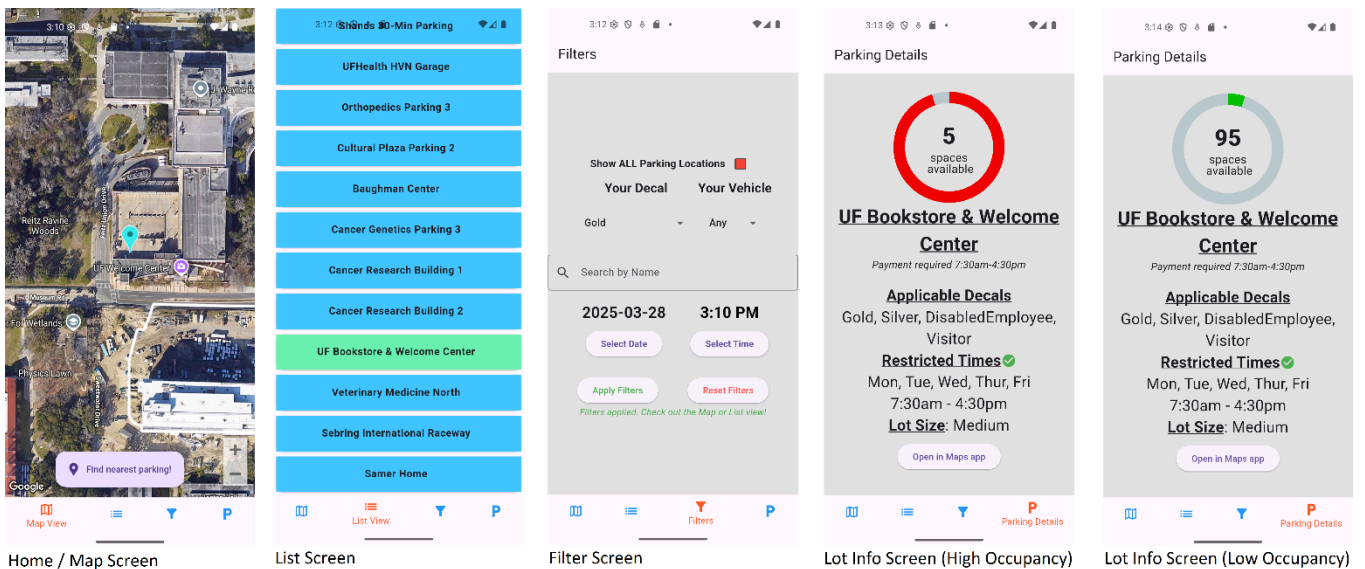


Figure 2. Parking App Screen Frames

### Raspberry Pi Edge AI Device

The edge AI device is implemented using a Raspberry Pi 5 (RPi5) with 8 GB of DRAM. To measure parking facility throughput, two devices must be used, one configured for inflow and the other for outflow. A camera on the device captures an image feed of the environment, fast enough to provide an accurate and timely representation of the surroundings. The captured frames are fed into a YOLOv11n object detection model to capture a license plate and track the plate's movement to prevent recounting it. Upon identifying a license plate, a request is sent via the database API to update the occupancy of the device's corresponding lot in the PostgreSQL database according to the direction of traffic flow measured. Additionally, the frame with highest confidence of a license plate is sent to the remote backend of the car detection pipeline to undergo license plate number recognition to log the car entering the facility. Figure 3 depicts this sequence of actions.

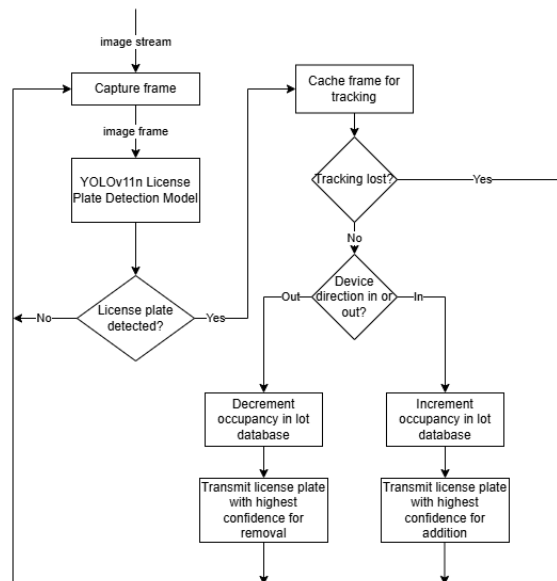
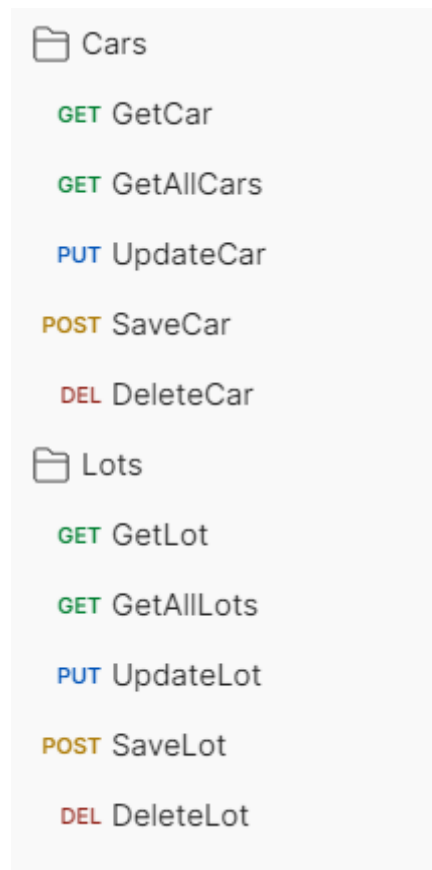


Figure 3. RPi Flow Diagram

### Persistent State

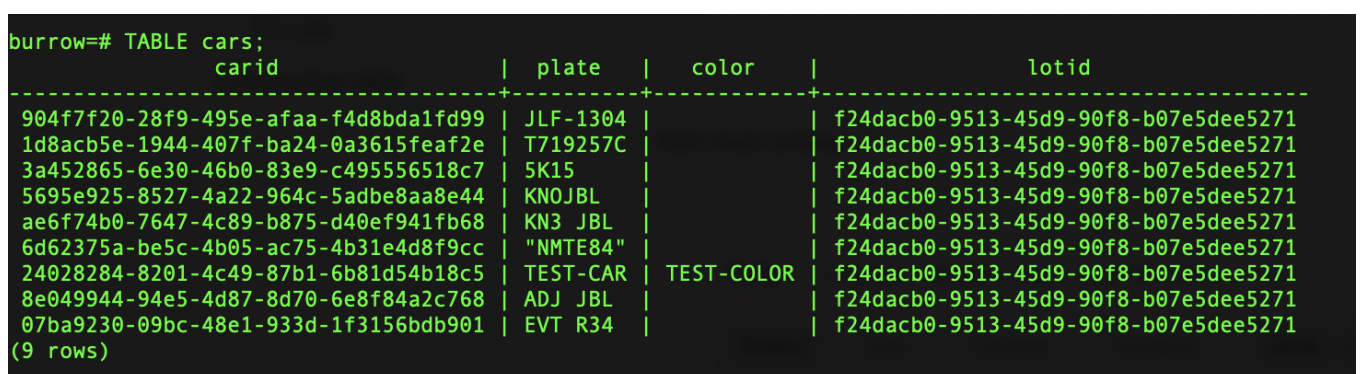
## Backend Database

The database is hosted using AWS RDS as a PostgreSQL database and is used to store information about all the lots on UF campus and the cars within lots configured with PAS. It employs a RESTful API utilizing the CRUD functions in Figure 4 for accessing data from the lot and cars tables seen in Figures 5 and 6. It will receive updated information, such as the new occupancy counts as vehicles enter or exit facilities and maintain this information for the mobile application to reference.



📁 Cars
GET GetCar
GET GetAllCars
PUT UpdateCar
POST SaveCar
DEL DeleteCar
📁 Lots
GET GetLot
GET GetAllLots
PUT UpdateLot
POST SaveLot
DEL DeleteLot

Figure 4. Lot and Cars API CRUD Functions



```
burrow=# TABLE cars;
```

carid	plate	color	lotid
904f7f20-28f9-495e-afaa-f4d8bda1fd99	JLF-1304		f24dacb0-9513-45d9-90f8-b07e5dee5271
1d8acb5e-1944-407f-ba24-0a3615feaf2e	T719257C		f24dacb0-9513-45d9-90f8-b07e5dee5271
3a452865-6e30-46b0-83e9-c495556518c7	5K15		f24dacb0-9513-45d9-90f8-b07e5dee5271
5695e925-8527-4a22-964c-5adbe8aa8e44	KNOJBL		f24dacb0-9513-45d9-90f8-b07e5dee5271
ae6f74b0-7647-4c89-b875-d40ef941fb68	KN3 JBL		f24dacb0-9513-45d9-90f8-b07e5dee5271
6d62375a-be5c-4b05-ac75-4b31e4d8f9cc	"NMTE84"		f24dacb0-9513-45d9-90f8-b07e5dee5271
24028284-8201-4c49-87b1-6b81d54b18c5	TEST-CAR	TEST-COLOR	f24dacb0-9513-45d9-90f8-b07e5dee5271
8e049944-94e5-4d87-8d70-6e8f84a2c768	ADJ JBL		f24dacb0-9513-45d9-90f8-b07e5dee5271
07ba9230-09bc-48e1-933d-1f3156bdb901	EVT R34		f24dacb0-9513-45d9-90f8-b07e5dee5271

(9 rows)

Figure 5. Visualization of Cars Table in Database

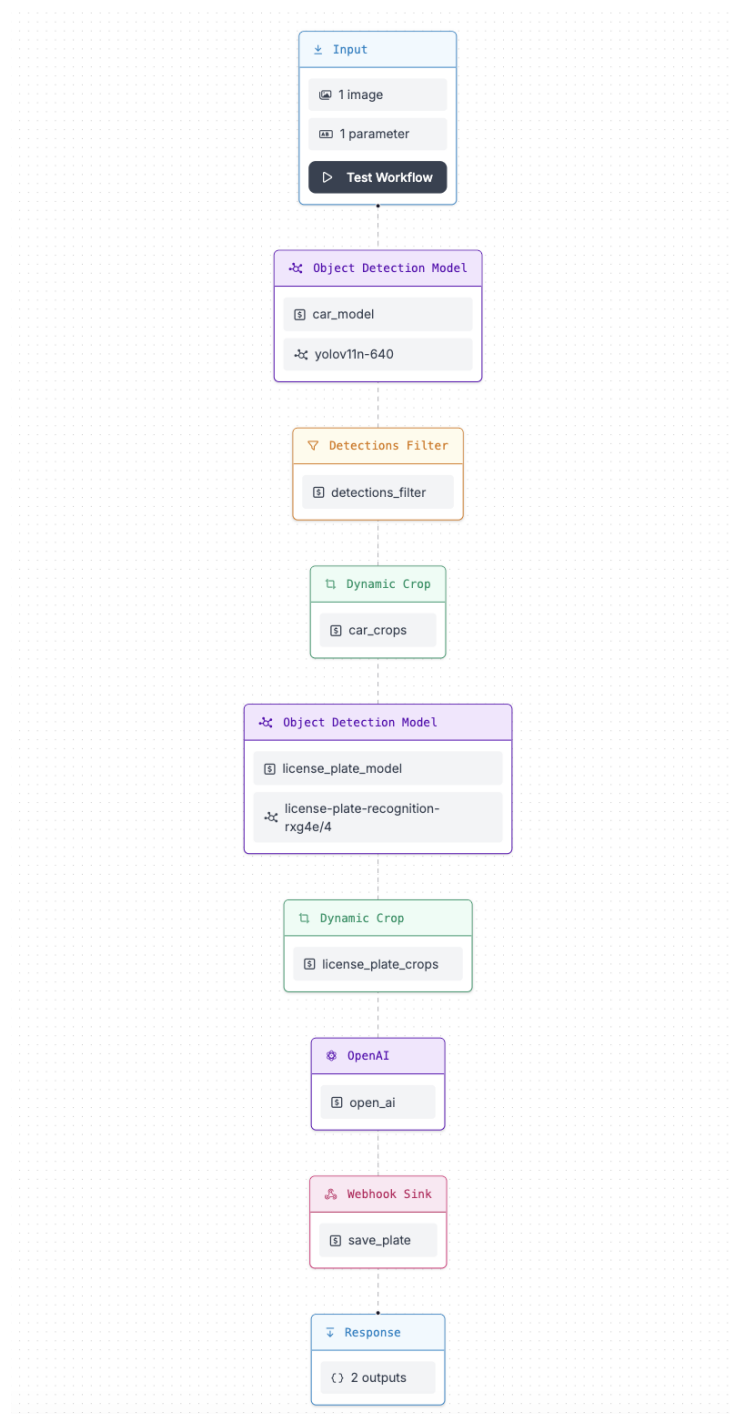
lotid [UUID]	latitude [NUMERIC]	longitude [NUMERIC]	name [VARCHAR]	address [VARCHAR]	open [TIME]	close [TIME]	days [VARCHAR]	decals [VARCHAR]
0210feeb-cf1d-42a7-95f1-000000000000	29.647502	-82.359249	Fraternity Row 2		08:30:00	15:30:00	{M,T,W,R,F}	{parkAndRide,red1,disabledSt}
0226350f-5dc0-433e-af11-000000000000	0.407479	0.345672	test name	5678 Elm St	12:35:22	12:35:22	{M,T}	{Red,Green}
02bf2c6a-1810-4fe2-90f1-000000000000	29.645655	-82.337330	Parking Garage 8		08:30:00	15:30:00	{M,T,W,R,F}	{orange,red1,red3,disabledSt}
0405bdde-778a-4de7-90f1-000000000000	29.639044	-82.346679	Parking Garage 2		07:30:00	17:30:00	{M,T,W,R,F}	{gold,silver,visitor,disabledSt}
060dc221-bd70-408a-af11-000000000000	29.649438	-82.339431	Tigert Hall/13th Street Market		07:30:00	16:30:00	{M,T,W,R,F}	{motorcycleScooter}
06fa4bf0-3ecb-4ef7-a0f1-000000000000	29.645198	-82.348377	UF Bookstore & Welcome Center		07:30:00	16:30:00	{M,T,W,R,F}	{gold,silver,disabledEmployee}
09549251-14f3-44ab-90f1-000000000000	29.645754	-82.352848	Flavet Field		08:30:00	15:30:00	{M,T,W,R,F}	{green}
1114af86-d158-4b14-b4f1-000000000000	29.640779	-82.341599	Parking Garage 10		08:30:00	15:30:00	{M,T,W,R,F}	{visitor,motorcycleScooter}
11945155-ac9e-45c7-86f1-000000000000	29.647512	-82.342973	Inner Road Motorcycle		07:30:00	16:30:00	{M,T,W,R,F}	{motorcycleScooter}
1284e0bd-fe3f-4156-84f1-000000000000	0.083723	0.871200	test name	1234 Main St	12:38:42	12:38:42	{M,T}	{Red,Green}
128ce783-ff24-4a47-be11-000000000000	29.639699	-82.356552	IFAS Parking 3		08:30:00	15:30:00	{M,T,W,R,F}	{orange,motorcycleScooter}
1429020c-9ba3-4662-a0f1-000000000000	29.646837	-82.350160	East Hall		08:30:00	15:30:00	{M,T,W,R,F}	{orange,motorcycleScooter}
18a4bda1-b294-47a9-af11-000000000000	29.634154	-82.351701	Veterinary Medicine West		08:30:00	15:30:00	{M,T,W,R,F}	{parkAndRide,green}
18a9b8f3-7f9c-41bc-af11-000000000000	29.637907	-82.367623	Southwest Rec Center		08:30:00	15:30:00	{M,T,W,R,F}	{parkAndRide,disabledSt}

**Figure 6.** Visualization of Lot Table in Database

## Internal Systems

### Car Detection Pipeline

The car detection pipeline employs Ultralytics' YOLOv11n object detection model running on the edge device as well as a Roboflow OCR pipeline operating in the cloud. The edge device has models configured to detect cars and license plates. Images frames are fed into the car detection model. If a car is detected, the frame is fed into the license plate detection model, and the result is cached while the car is being tracked so that multiple counts do not occur. When the car is gone, the license plate frame with the highest confidence value is transmitted to the backend Roboflow OCR pipeline. Figure 7 visualizes this pipeline.



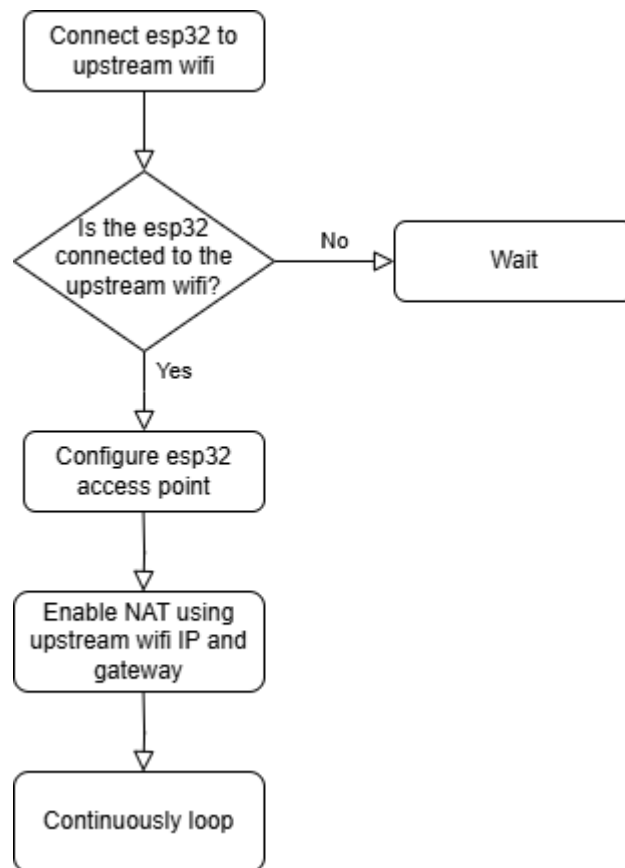
**Figure 7.** Car Detection Pipeline

### ESP32 Wi-Fi Router

The ESP32 utilizes the built in Wi-Fi capabilities of the ESP32 to create a NAT router. The functionality of the NAT router is to forward the requests made to the ESP32 access point network to the upstream Wi-Fi network, allowing the ESP32 access point network to access the internet. This is done by first connecting the ESP32 to the upstream Wi-Fi, and then pulling the IP and gateway to send the client requests to the upstream Wi-Fi. The ESP32 was configured with a password to ensure that only authorized users can access the network. The ESP32 can perform at



a max rate of around 15 Mbps download and upload speeds while the ESP32 is within close proximity to the client device.



**Figure 8.** ESP32 Execution Flow Diagram

## Tools, Design Constraints, and Engineering Soundness

The mobile application was developed within the Android Studio IDE, providing IntelliSense error checking and linting capabilities as well as an emulator allowing for building the project on an emulated device to test and verify functionality from a user's perspective. The Flutter framework was used to build a functional, cross-platform application using the Dart programming language, implementing in-depth debug tools to view perform breakpoint traversal, memory viewing, and application performance metrics to further verify functionality. The Google Maps Platform SDK was used to offer a map API to implement navigation and location functions familiar to users so that they can seamlessly and intuitively orient themselves on campus and navigate parking locations. The application employed HTTP requests to communicate with the backend database through a RESTful API that employed CRUD operations.

The edge device was developed through various software and physical means. To develop physical housing, SOLIDWORKS CAD software was used to develop physical models to be 3D printed. The physical board our team programmed to implement the device was a Raspberry Pi 5 (RPi5), offering a 4-core ARM A76 processor, operating at a maximum frequency of 2.4 GHz, and 8 GB of RAM. Software on the device was developed within a 64-bit Linux environment. The SSH protocol was used to remotely access and program the device. The APT and PIP package managers were used to download dependencies needed to implement the car-detection pipeline. Software on the board was

programmed using bash and Python. An ESP32 is alongside the edge device as a router to extend Wi-Fi range. The firmware was developed within the Arduino IDE, allowing compilation and uploading firmware to the board. Espressif device drivers were used to interface with the ESP32's Wi-Fi functions.

The car detection pipeline was developed using a YOLOv11n trained object detection model provided by Ultralytics to identify cars and license plates. This model was chosen as it was the most efficient to operate on IoT devices. A technique of "fine-tuning" was used to improve the model and is an industry standard of taking models that have been trained in a similar vein of some machine learning application, and then further training it/providing data that suits respective needs. Roboflow OCR and OpenAI GPT-4 Vision are deployed via AWS for enhanced feature extraction.

The remote database hosted on AWS RDS as a PostgreSQL database adhered to key schema design standards such as applying constraints, using primary and foreign keys, integrating modular design for simple growth or alteration. The database is communicated with using a RESTful API on an AWS EC2 instance written in Golang that implements CRUD functions to access database data through HTTP requests from the mobile application and RPi. Some things that could be applied or changed to improve the usability are consistent datatype standards to ensure data exchange is consistent and limited preprocessing is required.

## Results

### *Alpha Testing Results*

#### *Raspberry Pi License Plate Detection Pipeline*

A goal of the project is to get as close as possible to an inference speed of 10 frames per second without significantly sacrificing model performance. Utilizing processed image inputs into the model at a size of 640x384 pixels, we were able to achieve the resulting statistics in Table 2 on the RPi, without any deliberate tuning or changes of the model to improve inference speed.

TABLE 2. License Plate Detection Inference Speed

Input Size	MinTime	MeanTime	MaxTime	MaxFPS	MeanFPS	MinFPS
640x384	203.4	271.14	373.4	4.916	3.688	2.678

Even under ideal conditions, the YOLOv11n model does not meet our performance criterion. To remedy this, we will investigate the following options: overclocking the processor, maximize hardware utilization using multithreading, changing the model format, dimensionality reduction of input, or use of a different model.

#### *Raspberry Pi Wi-Fi Antenna Testing*

Wireless connection performance metrics were measured for both the on-board Wi-Fi modem of the RPi, as well as a USB Wi-Fi modem with an antenna, at various locations using the following Linux command sequence,

```
iwconfig wlan0; iw dev wlan0 link; iw dev wlan0 info
```

and the resulting statistics were summarized in Table 3.

TABLE 3. Modem Performance Statistics

Modem Type	AvgBitRate (Mb/s)	AvgLinkQuality	AvgReceivedSignalStrength (dBm)
On-board (5 GHz)	54.53333	79%	-55
Antenna (5 GHz)	115.175	91%	-46.3333

The results demonstrate a better signal via a higher average link quality and higher received signal strength using the USB Wi-Fi modem, which is what we were hoping to achieve. The results indicate that USB Wi-Fi antenna could help maintain a stable connection with the ESP32 repeater.

#### *Database API*

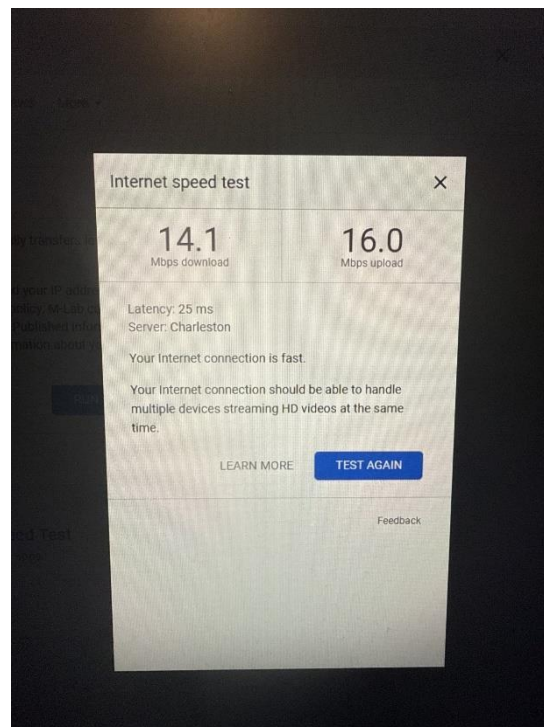
The code for the backend database API had unit tests written for each core function of the lot and car tables (e.g. Get, GetAll, Save, Update, and Delete), ensuring consistent and verified functionality of the API's code by processing input and comparing the result with the expected output. These unit tests are automatically triggered via GitHub Actions upon code commits or pull

requests to ensure the correctness of our code changes to enable continuous integration and prevent regressions.

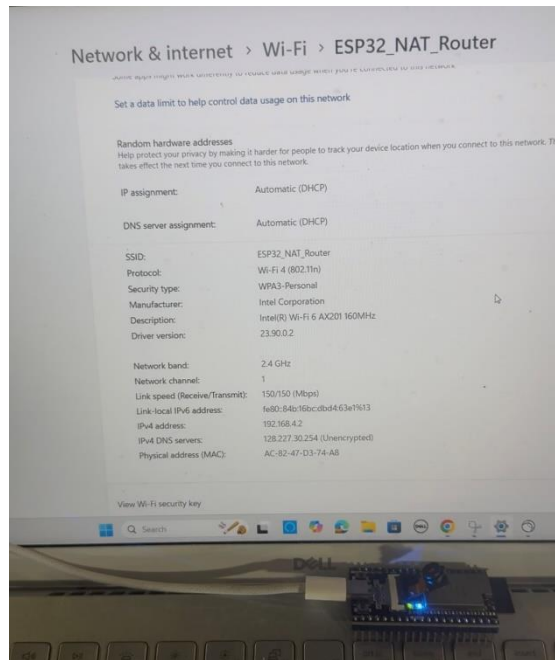
## *Beta Testing Results*

### *ESP32 and Connection Metrics*

A test of the ESP32 router was conducted between the ESP32 and our computer within Parking Garage 14 which emulates the deployment environment. As seen in the figure below, the connection established between the laptop and the ESP32 proves more than sufficient in providing the desired stable source, as our minimum target data rate is 300 kbps, which we achieve at all distances between 0-150 feet using the router. Further connection testing needs to be done with the RPi for verification.



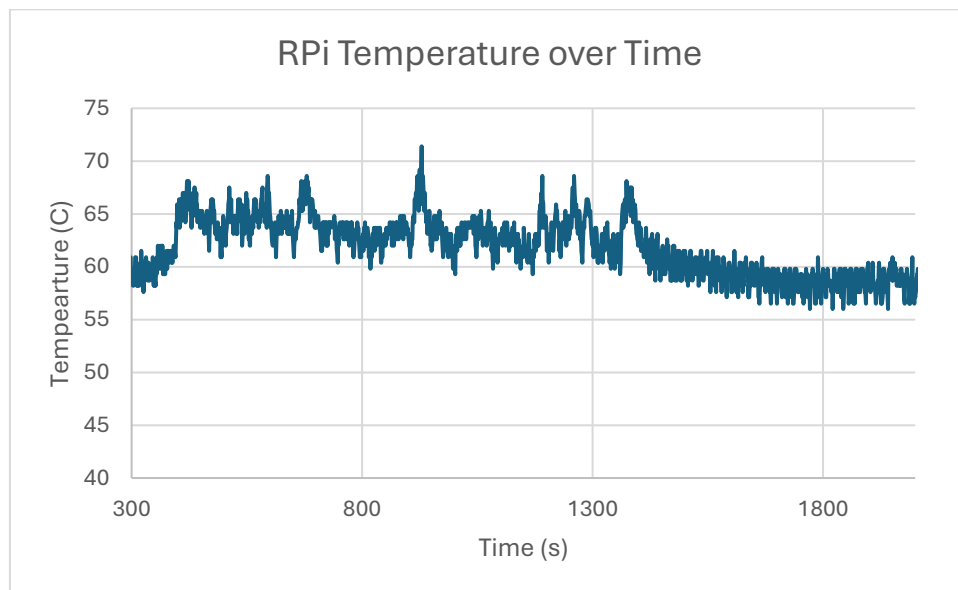
**Figure 9.** Wi-Fi speed performance from ESP32 host



**Figure 10.** ESP32 connected to computer

### *Raspberry Pi Edge Device Hardware Benchmarking*

During testing within Parking Garage 14, a hardware benchmark was run, periodically polling the device for core temperature, frequency, voltage, and throttle state. This was done to investigate the strain of the car detection pipeline on the hardware of the edge device as if it were deployed in the field normally. As seen in Figure 11, temperatures remained around 60 °C at idle, and during model testing peaked at about 71.4 °C. Given the reported throttle point of the RPi's CPU is 80 °C, this shows that there is available room for overclocking the processor to maximize hardware performance to improve model inference speed.



**Figure 11.** RPi Temperature Benchmark During Test

### *Raspberry Pi Housing/3D printed Parts*

Designing a custom 3D-printed enclosure for the Raspberry Pi was a crucial step in creating a stable, protected, and functional setup for photography and mounting applications. The enclosure was tailored specifically to accommodate key hardware like the camera module and add-ons. As the project developed, practical challenges during assembly led to several design refinements. Internal wiring required rerouting, prompting layout adjustments to maintain structural integrity while improving cable access. Heat management became another priority, leading to the addition of a fan slot to improve airflow and prevent overheating during prolonged use. A redesigned camera holder is also underway to enhance stability and alignment for better image capture. While the idea of creating a completely new enclosure was considered, iterative improvements to the existing model proved to be the most efficient path forward. This process highlighted the value of building upon a solid foundation through targeted enhancements rather than unnecessary redesigns.



**Figure 12.** Cuts were made to make wiring easier



**Figure 13.** New additions – Added Ventilation, Camera, and Fan stand offs

## Status

### Completed Tasks

Mobile Application	
Task	Completion Date
Wrapper functions for backend API completed to allow app to read from database lot table	12/05/2024
App ready for use: Refactored database access functions. Fixed database read errors. App updates real-time parking count periodically when lot information page is accessed.	03/23/2025
Edge AI Device	
Task	Completion Date
First iteration of edge device housing	11/22/2024
Edge license plate detection model deployed on edge device	01/20/2025
Second iteration of edge device housing	02/25/2025
ESP32 network extender functional	03/27/2025
Updates to second iteration of edge device housing	03/28/2025
Backend Database	
Task	Completion Date
Backend Database and API functions: All Lot and Car API CRUD functions and final iterations of Lot and Car table layouts completed	12/05/2024
Car Detection Pipeline	
Task	Completion Date
First iteration of edge license plate detection model completed	11/22/2024
Roboflow OCR demo pipeline completed	11/22/2024
Second iteration of edge license plate detection model tested	03/25/2025

### In Progress

#### Raspberry Pi License Plate Inference Speedup (Est. Date: 04/06/2025)

The current edge device inference speed still averages about 3 frames per second. We are looking into solutions to speed this up to a target of 10 frames per second. Current ideas include model input dimensionality reduction, overclocking the device, changing the model to a more compressed format, or using a different model entirely.

#### Raspberry Pi Image Transfer and Image Fidelity (Est. Date: 04/06/2025)

The RPi5 is struggling to clearly capture images of license plates of vehicles. If a vehicle moves too fast, or the RPi5 is not fixed at an appropriate angle, images become blurry and license plates difficult to discern. Currently, the RPi5 is being tested with higher shutter speeds, alongside ISO adjustment to compensate for reduced light, to reduce the probability of blurred images.

#### Edge Device Housing Improvements (Est. Date: 04/06/2025)



The edge device housing is still undergoing changes to allow for more ventilation to accommodate likely increases in temperature due to overclocking and maximizing model performance.

### *Planned Tasks*

Implement Edge Device Lost Connection Functionality (Est. Date: 04/14/2025)

The edge device needs to be able to cache processed frames to be sent to the database when connection is lost. This will involve creating a temporary store for processed frames upon failure to connect to the backend. When the connection is reestablished, the device will transmit frames from the temporary cache according to the order they were sent.

Implement Remote Device View (Est. Date: 04/14/2025)

For final demonstration purposes, the edge device should broadcast the processed image stream to be reviewed remotely. The stream will be broadcast on the device's IP address on an available port and can be accessed by any device on the UF network.

### **Conclusion**

In conclusion, this project is aimed at improving parking resources at UF by making real-time parking availability accessible. PAS is meant to be a simple, affordable, and effective solution to implement to alleviate the university's parking issues. It incorporates industry standards in training an object detection model to recognize license plates, cutting edge 3D printing techniques to provide a durable and robust housing, an enjoyable user interface that provides additional features beyond parking facility occupancy insights, and is built to last for many years to come in any weather condition.

### **References**

[1] L. Du and S. Washburn, "SMART PARKING SYSTEM ON UF CAMPUS," Apr. 2019. Available: <https://fora.aa.ufl.edu/docs/38/2018-2019/SmartParkingProposalLiliDuScottWashburn.pdf>

[2] D. Gill, "ez-park," GitHub. Available: <https://github.com/drew-gill/ez-park>. Accessed: October 26, 2024.