Optimizing Energy Consumption and Profit in Urban Robotaxi Fleets

1st Tarushi Mittal
Department of Computer Science
Rice University
Houston, United States
trm4@rice.edu

2nd Andrew Negrut
Department of Computer Science
Rice University
Houston, United States
agn5@rice.edu

3rd Jingwu Wang
Department of Computer Science
Rice University
Houston, United States
jw133@rice.edu

***Abstract*—Robotic taxis are becoming increasingly common as an autonomous option for transportation in urban areas across the world. This project explores the possibility of having robotaxis perform their daily operations with the aim of wasting less electricity and earning more profits, while additionally supporting community members lending EV chargers. A robust simulation framework was created that minimizes the vehicles' expenditures and redundant energy consumption while providing income for households leasing their home chargers for robotaxi use. An end-to-end visualization platform additionally gives users full agency over simulation parameters and showcases results graphically in real-time. At any point during the simulation, users may dynamically alter parameters to see the effects on their fleet of robotaxis. To enhance insights into simulation results, an AI-based robotaxi GPT can answer any questions users may have in real-time using natural language. The findings produced by this framework demonstrate that on average, robotaxi fleets adhering to the optimized decision-making algorithm achieve both lower electricity wastage and more profits compared to a control fleet of robotaxis.**

***Index Terms*—artificial intelligence, optimization, robot, robotaxi, simulation**

I. Introduction

Modern society is driven by a desire to constantly strive for innovations that improve the quality of everyday life. One such advancement that is rapidly becoming more popular is the autonomous electric car. These vehicles have recently been deployed as an alternative public transportation option in major metropolitan areas around the world. Technological developments often raise concerns, however, especially regarding their impacts both on their community and on the world at large. If steps are not taken to ensure the sustainability of these robotaxis, they are likely to generate a larger carbon footprint as the demand for this service continues to grow. With global warming becoming an urgent crisis, minimizing energy waste is critical. Finding ways to reduce their energy wastage is therefore crucial to ensuring that this emerging technology aligns with global efforts to combat climate change and promote a greener future.

Despite their popularity, it is challenging for robotaxis to compete with existing rideshare services due to their high investment and operating costs. The technology behind robotaxis requires complex systems, including advanced cameras, sensors, and software, to ensure safe driving. For robotaxis to become commercially viable and competitive, their operational costs—particularly the costs of the energy required to fuel them—must be minimized. Therefore, a key challenge is optimizing the charging strategies for robotaxis to reduce unnecessary, redundant driving. This includes determining which charging station a robotaxi should choose and the optimal time to charge. The availability of EV stations for robotaxis is often limited to public or commercial charging stations. However, many households have private access to chargers. If those stations could be made available to robotaxis, it could significantly reduce energy waste from driving to public charging stations or waiting for a charger.

This project explores how the environmental and monetary costs of a fleet of robotaxis operating in an urban neighborhood can be minimized to ensure sustainability and commercial success. It also investigates how these vehicles can help members of the community, and thus incentivize individuals with private EV stations to share their chargers with the robotaxis and help contribute to further environmental and societal benefits.

II. Related Work

The relationship between traffic patterns and resource consumption is a topic that has been heavily investigated by research teams, especially in recent decades. This section focuses on one example work that studies the possibility of optimizing the emissions of vehicles by using transportation simulation software to test the performance of different optimization techniques.

In 2015, researchers concerned about the greenhouse gas emissions of cars designed a traffic simulation to test existing algorithms that optimized fuel consumption. They evaluated how traffic signals could be optimally controlled to minimize the total fuel consumption as well as the driving times of the vehicles. They designed a better, hybrid optimization approach that combined the small-scale details of drivers in the simulation, such as the vehicle-specific features and lane-changing behavior of individual cars, with the larger-scale overall traffic trends of the simulation. To prove that their strategy, which combined microscopic and macroscopic analysis, produced the best results, they compared its performance against that of the microscopic approach alone and of the

macroscopic approach alone, and found that traffic signals that followed their proposed optimization significantly lowered the environmental footprint of the city [1].

Although numerous reports discuss strategies of minimizing the resource consumption of vehicles, it seems that analyzing the role that the prices of the resources plays traffic activity is less of a priority. Also, since robotaxi fleets have only just started to be deployed in global metropolitan communities, there seems to be few papers that investigate how their environmental and monetary costs impact those neighborhoods. The optimization technique implemented in this project is not quite as rigorous as ..., but it aims to help fill that gap by simulating the optimization of both energy wastage and total costs of a fleet of autonomous electric robotaxis.

III. Methodology

A. Front-End

The front end is developed in `TypeScript` using the `Next.js` and `Tailwind CSS` frameworks. It consists of two primary pages: a landing page and a data dashboard. The landing page is designed to engage potential users by featuring a Tesla robotaxi video and highlighting the key features of our prototype. From the landing page, users can proceed to initiate the simulation, which will direct them to an initialization form.

Within the initialization form, users have the flexibility to configure a range of simulation parameters:

1) **Mode:** Choose between a control baseline version or the optimized version.
2) **Map:** Select the city in which the simulation will run. Currently, only Houston is supported.
3) **Simulation Start Time:** The minimum start time is 0 seconds. Every 300 simulation seconds corresponds to one hour in real life. For example, if the start time is 300 seconds, that means the simulation starts at 1 AM in real life.
4) **Simulation End Time:** The maximum end time is 7,200 simulation seconds (24 hours in real life).
5) **Time Step:** The smallest discrete interval of simulated time. A smaller time step yields higher accuracy but requires more computational resources.
6) **Number of Robotaxis**
7) **Number of Charging Stations**
8) **Number of Passengers**

After submitting these parameters, the values are sent to the back end through a `POST` API. The website then waits approximately eight seconds to ensure the back-end services are fully initialized before redirecting users to the data dashboard page.

The data dashboard is divided into five tabs:

1) **Dashboard:** This tab displays real-time values of profit, revenue, and cost for all robotaxis, as well as real-time plots of cumulative electricity consumption, the number of chargers in use, the number of active passengers, and the number of occupied robotaxis. The plots compare the current values to their most recent previous values and show the percentage differences.
2) **Analytics:** This tab ranks all robotaxis by their cumulative electricity consumption in descending order and provides summary statistics such as maximum, minimum, mean, and standard deviation. Users can adjust the top X results they wish to see. The tab also provides an overview of battery levels for all robotaxis in ascending order, along with the same summary statistics, and again allows users to adjust the top X results. Battery levels are highlighted based on their values:
   - Less than 20%: highlighted in red
   - Between 20% and 50%: highlighted in yellow
   - Greater than 50%: highlighted in green

   Additionally, the tab shows the charger utilization rate (the percentage of chargers in use), the fleet utilization rate (the percentage of occupied robotaxis), the average passenger waiting time, and the passenger dissatisfaction rate (the percentage of passengers waiting more than 15 minutes). All these data points update in real-time.
3) **Map:** This tab displays the live positions of all robotaxis, passengers, and chargers on a Houston map, with each category represented by a distinct color (e.g., robotaxis in yellow, passengers in red, chargers in green).
4) **Parameters:** This tab allows users to dynamically adjust the number of robotaxis, passengers, and chargers during the simulation.
5) **Robotaxi GPT:** This tab enables users to ask questions about traffic information presented in the other tabs. The system can only retrieve information from the backend's GET APIs and will answer queries based on that data.

The Robotaxi GPT is built using OpenAI's `o1-mini` model and is developed through prompt engineering. This involves:

1) Defining the role of the model as a JSON data analyst.
2) Specifying the expected JSON data format, including an explanation of each field and how to interpret the corresponding datasets.
3) Outlining the types of questions the model should anticipate.
4) Providing guidelines on how the model should answer queries in a clear and concise manner.
5) Including examples of both exemplary and suboptimal responses to help shape its behavior.

B. Simulation Mechanics and Dynamic Functions

At a high level, the simulation framework is built upon two separate threads. One thread manages the SUMO software, and another thread runs a flask app which interacts with the simulation via HTTP methods. The simulation is started via a POST method which initializes an instance of the SimulationRunner class with the exact parameters of the user's choosing.

A series of POST requests defined in the flask application allows the users to dynamically change the simulation parameters in real-time. Such methods are available for addition and removal of taxis, people, and chargers in the simulation.

A series of `GET` APIs are also defined to retrieve various simulation metrics and related data. These include:

- Current simulation status
- GeoJSON data for the Houston map
- Cumulative electricity consumption for all vehicles
- Battery levels for the vehicles
- Number of active chargers in use
- Number of active passengers
- Number of robotaxis currently transporting passengers
- Average passenger waiting time
- Passenger dissatisfaction rate (percentage of passengers waiting more than 15 minutes)
- Positions of robotaxis, chargers, and passengers

A command queue serves as the interface between the simulation thread and the flask application. Any time a POST request is made to alter simulation parameters, the corresponding operation is added to the command queue. Every simulation step, the simulation thread checks the command queue and performs a removal. The queue removes commands in the same order that they came in, ensuring that the modifications to simulation parameters are performed in accordance with the order of the user's requests.

C. Implementing Optimization to Minimize Energy Wastage and Maximize Profits

One of the main goals of this project was to design an intelligent algorithm that could minimize the electricity wastage and cost of the robotaxi fleet. In order to prove that the algorithm actually did produce optimal results, the simulation was written so that it could either be run with or without the optimization. The performances of the base control, and the intelligent version were evaluated through metrics like completed reservations, total profits, energy usage, and distance traveled. This section of the report discusses the setup of the artificial environment and compares the decision-making process of robotaxis in both the control and the optimized scenarios.

1) Basic Framework: The simulated city supports the existence of three types of objects: charging stations, people, and robotaxis, all of which are initialized at random locations on the map. The EV chargers are the simplest of these because they can only be in one of two states, either active or inactive. Chargers in the simulation are considered to be residential stations owned by community members and leased for the vehicles to use. Operational stations can instantaneously bring any taxi that drives over it to full power, but the vehicle has to be intending to charge. In other words, if a car just happens to drive over a charger while in transit to some other location, its battery level will not increase. Robotaxis spend money to pay for the added power every time they charge. The cost for one vehicle to charge to full battery capacity one time is calculated through the expression

$$Base + (Energy\ Cost \times Charge\ Added) \tag{1}$$

where base is some fixed value in dollars, electricity cost is measured in dollars per kWh and fluctuates depending on the time of day, and charge added is the amount of power given to the vehicle in kWh. Within the simulation, the base value and electricity cost are determined according to real-world data. It is assumed that the owners of the robotaxi fleet pay this money to the household leasing the charger, and the household earns a profit equal to the base value each time a taxi charges.

Each person in the simulation represents one reservation request. These requests can be made at any time during the day, and the demand curve mimics real-world reservation trends. It is important to note that this means that if a user starts a simulation at the beginning of the day, such as at time 0, the number of people that will appear in the simulation will be significantly lower than the user-specified initial number of people, since the departure times of most of those passengers will still be in the future. If a user wants to confirm that the program did in fact process their specified number of people, it is recommended to start the simulation later in the day, such as at time 7100. Alternatively, if a user dynamically adds people, those requests will not be distributed, and instead take place immediately when the command is processed. Robotaxis earn money every time they successfully drop off passengers at destinations and complete reservations. The income from one reservation is calculated through the expression

$$Base + (Rate \times Distance \times Demand \times Time\text{-}of\text{-}Day) \tag{2}$$

where base is some fixed value in dollars, rate is some fixed distance rate in dollars per km, distance is the distance between the reservation's pickup and drop-off locations in km, demand is a multiplier that reflects the number of reservation requests that have been recently added, and time-of-day is another multiplier influenced by peak hours. Within the simulation, all five of these parameters are determined according to real-world data. The demand multiplier additionally accounts for live updates, since the simulation supports the dynamic input and removal of people. The simulation is scaled so that 300 seconds represents 1 hour in real time, or 7200 seconds is one day. This scaling is used to determine values like electricity prices and demand multipliers; however, the simulated time cannot always be scaled directly in this manner. For example, users may notice that one of the values outputted by the back-end of the program, average wait time per reservation, can be as high as several thousand. The purpose of this number is not to measure real time, but to instead serve as a comparison between the control and optimized simulations.

The robotaxis are constantly operational. Even when no reservation requests are made, unoccupied and unassigned taxis still explore the city, waiting for the next request. A vehicle that runs out of battery is removed from the simulation for 300 seconds (or one hour in real time). It is assumed that any roadside assistance, such as towing and recharging, takes place off-screen. When the car is returned, it is reinitialized at a random location with full charge. Robotaxis spend money every time this happens to pay for the roadside assistance, and this expenditure is calculated similarly to the cost of charging through equation (1), except that the base value now represents the price of a tow, which is considered to be $100 in the simulation. To avoid this situation, robotaxis try to make

reasonable decisions about when to charge, and the logic they use is discussed further in the next few sections.

2) Control Simulation: In the non-optimized version, the activity of autonomous vehicles mimics that of human drivers. One example is in the assignment of taxis to reservations. Real-world employees of rideshare services, such as those who work for Uber or Lyft, tend to choose reservation requests that are nearby. They also do not communicate with each other before selecting a request to determine which among them is closest to that request; instead, the request goes to whichever driver happens to see it first. The simulation copies this behavior by assigning each unassigned taxi to its nearest pending, unassigned reservation, but the order in which taxis are assigned is random.

Another similarity between the non-optimized robotaxis and real humans is the way they decide to recharge. The average driver tends to procrastinate charging, waiting until the battery level reaches some minimum amount, and this quantity varies from person to person. They usually do not consider the hourly price of electricity. As a result, vehicles in the control scenario will only go to an EV station if their charge level falls below a minimum value, which is randomly generated within a specific range, and they do not account for the cost.

3) Optimized Simulation: By contrast, taxis that follow the optimized algorithm effectively make decisions that lead to less electricity wastage and lower expenditures. The first of these is achieved through a heuristic strategy that drives how taxis are assigned to reservations. It is assumed that the cars communicate with each other and the locations of all vehicles in the fleet are always known. The robots take advantage of this information during assignment in order to minimize the distance each car needs to cover and eliminate redundant driving, thus reducing the amount of energy that each car uses per reservation.

The intelligent taxis maximize their profits by applying machine learning. They are provided with a database at the beginning of the simulation that records historical prices of charging vehicles at different times of day, and from this data they train a predictive model using Python's built-in Random Forest Regression. This formula creates multiple decision trees based on random subsets of the historical data and predicts that future values will be the average of the trees' results. Random Forest Regression was chosen as the best option from Python's machine learning library due to its accurate representation of non-linear relationships and its lower computational costs. When a robotaxi's battery level gets to a specific amount, the vehicle considers charging and applies the predictive model to guess what electricity prices will be in the near future. The taxi is more likely to charge now if these costs are generally expected to increase, otherwise it puts off recharging for a bit more time. By making decisions in this manner, the taxis tend to charge more often, but the amount of money they spend from charging is significantly reduced.

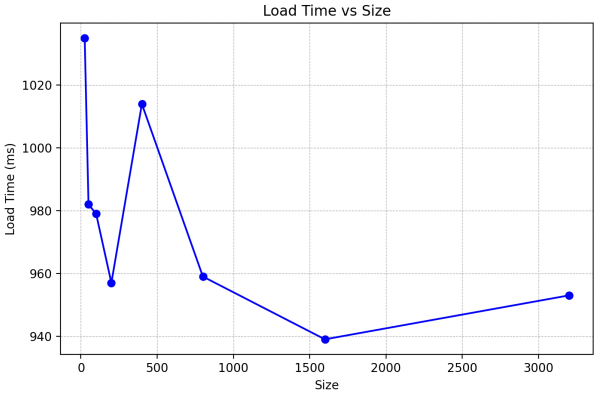

Fig. 1. Load time of data dashboard vs the size of data points

IV. Evaluation

A scalability test was conducted on the front-end application using dummy data and the Puppeteer framework. A series of tests were performed with datasets containing $[25, 50, 100, 200, 400, 800, 1600, 3200]$ dummy data points to measure the loading times of the data dashboard page. The results, summarized in Figure.1, show no noticeable increase in loading time as the number of data points doubles for each subsequent test. This consistency in performance can be attributed to the dashboard's design: it always slices out and displays only the most recent 50 data points. As a result, fetching larger amounts of data from the backend does not significantly impact the loading speed.

The performance of the optimized algorithm was evaluated by running it and the control ten times each over a period of 7200 seconds with the same numbers of chargers, people, and taxis and comparing the average final statistics from these executions. The two parameters that were weighed most heavily were the electricity wastage and the total profits. It is important to emphasize that the electricity wastage is not the same as the electricity consumption. The total electricity consumption is expected to be roughly the same between the control and the optimized simulations. This is because in both versions, unoccupied taxis that have not been assigned to a reservation are still circling the city until the next request is made, which means that all vehicles are always actively driving around and using energy. Electricity wastage on the other hand is an indication of how efficiently that energy is used, and is measured by analyzing how many reservations and charging trips were able to be completed with that energy usage.

With 100 chargers available, the 25 taxis in each base simulation managed to complete approximately 620 of the 1000 reservation requests and earned $10575 in total. Each car needed to charge roughly 1 time, and the total cost of these trips was $325. The resulting total profits was then $10250. On average, in order to successfully drop off 620 reservations and perform 25 charging trips, the taxis needed to consume a total of 177.61 kWh of electricity. The taxis in the optimized

scenario made more efficient decisions, completing over 650 reservations and earning a total of $10825. Although cars decided to charge a few more times than in the base control, the cost of these trips was significantly less, as the optimized taxis managed to complete 35 charging trips while spending only $280 in total. Thus, the net revenue was roughly $10540. These vehicles used a total of 175.77 kWh of electricity to fulfill 650 reservation requests and charge 35 times, clearly wasting much less energy than the control robotaxi fleet. It is evident from these values that even over the course of just one day, the robotaxis can significantly lower their redundant electricity consumption and maximize their profits through this optimization strategy. Table 1 displays the average results for these and other metrics.

TABLE I
AVERAGE RESULTS FROM 10 RUNS OF THE CONTROL AND OPTIMIZED SIMULATIONS WITH 1000 PEOPLE, 25 ROBOTAXIS, AND 100 CHARGERS

Performance Metric	Control Average	Optimized Average
Completed Reservations	619.60	653.00
Total Earnings	$10575.32	$10825.06
Charging Trips	25.80	34.90
Total Cost	$325.64	$283.26
Total Profits	$10249.68	$10541.80
Out of Charge Count	0	0
Distance Traveled	1248.68 km	1221.81 km
Total Energy Consumption	177.61 kWh	175.77 kWh

1) Limitations of the front-end: There are several counter-intuitive results displayed in the front-end that merit further clarification:

- **Negative cumulative electricity consumption:** Some robo-taxis may display negative cumulative electricity consumption. This anomaly is addressed in the following subsection and arises due to certain backend limitations.
- **Active passengers vs. input passengers:** The number of active passengers shown may not match the number of passengers originally input. This discrepancy occurs because the backend distributes passenger trips over the entire day. Thus, even if 100 passengers are specified at the start, only a small fraction might appear at the beginning of the simulation.
- **Chargers in use:** The number of chargers being utilized is frequently zero. This is a consequence of the backend's assumption that robo-taxis are instantaneously charged to full capacity. Consequently, no time is spent at charging stations.
- **Negative profits:** It is possible to see negative profits early in the simulation. Since only a limited number of passengers are traveling at the start of the day (i.e., at midnight), the initial earnings may be too low to offset the cost of electricity.

2) Limitations of the Simulation: It is worth noting some limitations of the simulation software that might possibly cause some anomalous results when users try executing this program. One significant flaw is the way that it records a vehicle's electricity consumption. Because energy usage is not a linear rate, since it depends on a variety of factors like speed, traffic light stops, and traffic congestion, it was essentially not feasible to come up with a formula to monitor this variable. The program instead calls on SUMO's built-in TraCI service to get each vehicle's consumption at each iteration of the simulation. However, it was observed that TraCI occasionally produced negative values. Although this obviously meant that the specific electricity consumption quantities might not always be accurate, it was decided that this did not pose too much of an obstacle to this project. This is because the goal was not specifically to calculate the energy usage, but to instead compare the energy wastage between the control and optimized algorithms. The negative quantities would occur randomly for both versions, and it was decided that comparing the averages across multiple runs would account for and cancel out these inconsistencies.

Another constraint concerns the visualization of charging stations that are dynamically added or removed during the simulation. This program represents EV chargers as induction loop detectors, and TraCI does not support changing the quantity of detectors after the simulation has started. As a result, although the program is able to keep track of active and inactive charging stations behind the scenes, only the initial number of chargers will appear in the simulation, regardless of the user's dynamic requests.

3) Limitations of the Program and Room for Improvement: These team members also want to acknowledge some limitations of their implementation of the simulation that make its execution less realistic and less ideal. The first of these is that taxis currently do not need to stop at a charger in order to gain electricity. Instead, vehicles go to full charge capacity instantaneously, regardless of their battery levels when they reach the charger. Requiring the cars to stop at chargers was initially intended to be part of the simulation, as this would make the optimization problem more interesting by having vehicles decide how much charge they wanted to add in order to minimize electricity expenditures while also ensuring that they did not miss peak demand times and lose potential earnings. However, significant issues were encountered when this was attempted, and these team members were not able to figure out how to solve those problems. One option that seemed like it might provide a solution was the possibility of replacing the induction loop detectors that represent charging stations with TraCI's charging station objects. Unfortunately, these team members did not discover this alternative until later in the project timeline, and did not have the opportunity to fully explore this idea.

Furthermore, it is not currently possible for the program to recreate certain initial conditions from one run of the simulation to another, since variables like the departure times of people, the hourly electricity costs and demand rates, and the initial locations of all objects are randomized. This affects user experiences because it is not sufficient to compare results from just one run of each of the control and the optimized algorithms; instead, users must consider the averages across multiple runs to accurately evaluate the performance.

Another constraint is related to the scalability of the program. As part of this project, the program was run on personal laptops with 100 chargers, 1000 passengers, and 25 taxis. It was found that with these parameters, executing the simulation over a time period of 7200 seconds required approximately ten minutes for the control version and fifteen minutes for the optimized algorithm. The time requirement increases as more objects are added, so it is recommended that users run this program on workstations with high-performance computing power for best results. Another option to bolster scalability includes integrating more urban areas in our framework, allowing users to select from a series of different real-life cities. This would allow for modeling in potentially larger areas, which could be particularly helpful for bigger robotaxi fleets.

In addition to addressing these limitations, future projects could also explore the possibility of using reservation demand trends to reduce the electricity consumption of the vehicles, rather than only minimizing energy wastage. The current implementation of the program has all robotaxis in the fleet circling the city when there are no pending reservation requests to be assigned, and this is true for both the control and the optimized simulations. As a result, the taxis are always constantly moving, and although electricity is consumed more intelligently in the optimized algorithm, the total energy usage is roughly the same between the two. One strategy to reduce the electricity consumption could be training a predictive model on historical data that recorded the number of reservation requests that were made at different times of day. This model could then allow robotaxis to predict the peak hours. From these guesses, the robotaxis could control how many vehicles should be circling the city at each hour, while all other taxis stay at some base location until they are needed. This would significantly reduce the total electricity consumption of the fleet.

V. Conclusion

This work presents an integrated simulation, optimization, and visualization framework aimed at enhancing the profitability of an urban robotaxi fleet while minimizing electricity wastage. By simulating robotaxis in a realistic urban environment and leveraging both heuristic and machine learning-based optimization strategies, the approach ensures that vehicles choose more cost-effective charging times and travel routes, ultimately translating into better resource utilization and improved financial outcomes.

The results demonstrate that fleets employing the optimized decision-making algorithm complete a greater number of reservations while reducing unnecessary energy consumption compared to a control scenario. Notably, the robotaxis guided by predictive models manage to strategically schedule charging based on anticipated electricity prices, driving down overall costs without compromising performance. Concurrently, the incorporation of community-based charging stations fosters a mutually beneficial ecosystem, incentivizing private EV charger owners to participate and reinforcing the economic stability of the robotictaxi service.

Although the simulation achieves meaningful improvements, several avenues remain for future exploration. More granular stop-and-charge logic, predictive modeling of reservation demand, and advanced scaling capabilities could yield additional efficiency gains. As urban centers continue to embrace autonomous electric fleets, these findings and methodologies serve as foundational steps toward more sustainable, cost-effective, and community-supportive mobility solutions.

VI. Contributors and Contributions

The project was divided into three main components and each was assigned to a team member based on the individual's personal experience and interests. Assignments were flexible, however, and although each contributor generally stayed with their initial tasks, one programmer's work often overlapped another's and there were frequent communications and status updates. Team members placed great importance on ensuring that the jobs were evenly divided so that no one person was left with the brunt of the project and that the members were responsible for making progress in a timely manner.

A. Jingwu Wang - Front-End and GET APIs

Jingwu independently developed the front-end of this project, completed most of the GET APIs and their corresponding functions in the backend, and integrated the frontend with the backend.

For the front-end, he implemented a landing page where users could learn what a robotaxi is through a short video, understand various features supported by our prototype, and initialize the parameters for the simulators. He closely worked with Andrew to ensure these parameters were correctly interpreted by the backend. For the dashboard, he created six major tabs: dashboard, analytics, map, parameters, Robotaxi GPT, and (optimization). For the dashboard, analytics, and map tabs, he retrieved data from the GET APIs he built in the backend and visualized it. For the dynamic parameters, he used POST APIs built by Andrew to ensure they were changed correctly. For the robotaxi GPT, he used OpenAI API and trained an o1-mini model by prompt engineering to interpret JSON data returned by various GET APIs from the backe-end and answer questions regarding traffic status. The optimization tab was deleted because the backend does not support simultaneous running of both the control and optimized versions. Originally, this tab allowed real-time comparison of metrics between these two versions.

For the back-end, based on Andrew and Tarushi's codebase, he implemented various GET APIs and their corresponding functions in the simulation runner to export useful data to the front-end through HTTP requests. Such data includes GeoJSON data for the Houston map, cumulative electricity consumption for all the vehicles, battery levels for the vehicles, the number of active chargers in use, the number of active passengers, the number of robotaxis with passengers, average

passenger waiting time, passenger dissatisfaction rate, and the positions of robotaxis, chargers, and passengers.

He was also responsible for integrating the front-end with the backend by sending HTTP requests to the correct APIs and retrieving, interpreting, and representing the data in the front-end.

B. Andrew Negrut - Simulation Base, Command-line Integration, Dynamically Changing Parameters

Andrew began his work by creating an initial simulation base upon which the optimization framework could be built. He integrated a network based on the actual structure of a downtown Houston area, and added electric vehicles to represent taxis and induction-loop detectors to represent people and chargers. After this base was established, he worked to tailor the simulation setup to facilitate more complex interactions, including replacing detectors with actual people and reservations, and using actual taxis instead of electric vehicles. Following the setup of the simulation framework, Andrew moved on to developing systems that would integrate the simulation to the visualization platform facing the user. The first of these systems was a set of command-line arguments that would interface with the early version of the visualization platform, allowing users to specify key simulation parameters. Some of the command-line arguments he developed included *–step-length*, *–sim-length*, *–num-people*, *–num-taxis*, and *–num-chargers*. These command-line arguments allowed the early visualization platform to run an end-to-end simulation based on the simulation base he had developed earlier, with the exact parameters the user passed in. Another system he built to facilitate the connection to Jingwu's visualization platform were methods that reported electricity usage for all simulation vehicles at regular timestamps. This functionality was used to send electricity information to the visualization platfrom in XML form for downstream reporting to the user. This functionality was later taken care of by a GET method written by Jingwu. After developing these integration systems, Andrew moved on to writing a series of 6 POST methods that enabled the user to dynamically change parameters while the simulation was running. These 6 POST methods included an add and remove method for each of taxis, people, and chargers. Lastly, he integrated a command queue as an interface between the thread that ran the SUMO simulation and the thread that ran the flask application. This allowed for requests to update the simulation in real-time to be handled in the order they came in.

C. Tarushi Mittal - Optimization

Tarushi took the lead in creating the optimization algorithm for the robotaxis. Using Andrew's early simulation as a starting point, she began by implementing taxi assignment and dispatch and wrote code to accurately monitor reservations as they were successfully picked up at their starting locations and dropped off at their destinations. She then accounted for vehicle battery levels by including trips to charging stations when taxis approached low power. Formulas were added based on real-world trends to calculate the earnings from completing reservations, as well as the costs of recharging. Once the control was complete, she added optimization strategies to minimize electricity wastage and maximize profits. The first of these techniques was a heuristic that saved energy by assigning taxis to reservations with the aim of reducing the distance traveled. She then implemented machine learning through a regression algorithm trained on historical prices of electricity for charging vehicles at different times of day. The robotaxis built a predictive model from this data in order to guess future energy costs, and they used these estimates to influence their decisions regarding charging. By this time, Andrew and Jingwu had integrated the front-end with the backend and added functions to allow users to dynamically adjust the simulation parameters, and Tarushi worked with them to combine the finalized control and optimized algorithms with their code. The last step was adding some extra features to make the program more realistic and interactive, such as varying the departure times of people to mimic real-world reservation trends and allowing users to start the simulation at a specific time of day.

REFERENCES

[1] C. Osorio, and K. Nanduri, "Energy-efficient urban traffic management: a microscopic simulation-based approach," Transportation Science, vol. 49, pp. 637–651, August 2015.