SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: June 24, 2024

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Soham Phadke
Suvass Ravala
Joshua Jerome
Raghav Batra
Mubashir Hussain

ENTITLED

E-Scooter Black Box

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Thesis Advisor

Thesis Advisor

Department Chair

E-Scooter Black Box

by

Soham Phadke
Suvass Ravala
Joshua Jerome
Raghav Batra
Mubashir Hussain

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 24, 2024

E-Scooter Black Box

Soham Phadke
Suvass Ravala
Joshua Jerome
Raghav Batra
Mubashir Hussain

Department of Computer Science and Engineering
Santa Clara University
June 24, 2024

ABSTRACT

The burgeoning market for shared e-scooters is significantly hampered by the short lifespan of commercial e-scooters, which currently average just three months due to rough handling by users. To address this challenge, our project aims to extend the lifespan of shared e-scooters through an innovative onboard solution that discourages detrimental riding behaviors.

Our solution integrates a portable sensor hub from STMicroelectronics to capture ride data, which is then processed and sent via a user's iOS app to a Google Firebase backend. A machine learning model running in the cloud analyzes the data to extract valuable metrics. These metrics are displayed on a dedicated web application, enabling ride-sharing companies to monitor and influence user behavior effectively. By providing these insights, our solution not only promotes the longevity of the e-scooters but also enhances the operational feasibility for service providers, with the potential to transform the economic landscape of urban ride-sharing.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Current issues surrounding rental e-scooter longevity require a solution that can be implemented with minimal resource overhead. This necessitates modifications to existing rental e-scooters that can be maintained cost-effectively and encourage responsible use, subject to the following criteria: 1) A sensor box must be mounted on each scooter to record accelerometer and magnetometer data, 2) Data must be relayed via a mobile app to a central database, 3) A machine learning algorithm must analyze ride data to assess user treatment of the scooter, 4) The analyzed information must be accessible to the rental service provider to inform maintenance and usage policies.

1.2 Background

As cities become increasingly congested, ride sharing has become an appealing alternative to traditional modes of transportation. Advancements in technologies such as IoT (the internet of things) and advanced tracking have decreased the cost of ride sharing while supporting ease of access and feasibility, making ride sharing an attractive transportation alternative. For instance, only 15% of American adults had used Uber or Lyft in 2015, while in 2018 that figure had risen to 36%, even higher (45%) for urbanites [1].

Bicycle and scooter sharing are some of the most popular forms of ride sharing, especially in settings where most customers are looking to travel short distances for personal trips. In fact, 35% of all personal trips are less than 2 kilometers, while 75% are less than 10%, making both bicycle and scooter sharing highly attractive modes of transportation for city dwellers. Recent introductions of e-scooters and e-bicycles have made it progressively easier for customers to ride longer distances with greater speed and less exertion – factors highly desired for individuals living in high density regions [2]. The shared e-scooter market alone is estimated to be valued between $40 to $50 billion by 2025 [1].

Despite the rising demand for ride sharing services and the large-scale deployments of bicycle and scooter sharing fleets, there are numerous challenges that have hindered the adoption of these ride sharing options in some locations.

For example, there have been notable concerns regarding the socio-environmental dangers associated with bicycles left in improper locations and user mistreatment of the bikes – in some countries such as China, vandalism is a serious concern and has led to significant property damage and corporate losses for bicycle sharing providers [3]. Additionally, in Sweden, a study from 2020 showed that 33% of accidents with e-scooters are caused by traveling on hazardous surfaces, while 90% of e-bike traffic accidents are caused by cyclists' risky riding behaviors, including illegal occupation of motor vehicle lanes, over-speeding, running red lights, and going against the traffic flow [4]. Practical challenges have also been exacerbated by technical limitations, especially for e-scooters. Commercial e-scooters have an average lifespan of just three months, due to their heavy and rough rental usage and their purpose-built nature intended for private use – it has been estimated that e-scooters will likely generate a profit if they can last around six months [2].

Addressing the shortcomings of e-scooters and e-bicycles within the ride-sharing world is essential in order to boost the feasibility and cost-effectiveness of these solutions. The rough use and riding behaviors of users is the root of a number of problems: safety concerns, vehicle lifespan, and monetary corporate losses, to name a few. These are a lot more prevalent with e-scooters, which are significantly more susceptible to road imperfections and hazards due to the smaller wheel radius and lack of suspension components, making them significantly easier to detect via onboard sensors such as accelerometers. Other factors such as greater variability in riding dynamics across different kinds of scooters, and lower helmet usage make them a more significant safety concern in comparison to e-bikes [5].

1.3 Related Work

With this issue being as prevalent as it is, numerous existing e-scooter companies have attempted to tackle the problem in the hopes of boosting further e-scooter sales. From college professors to big companies like Bird, people have started to push out better rough riding detection for e-scooters. Generally these solutions can be placed into three categories: Privacy Based Solutions, Safety Based Solutions, and Maneuverability Based Solutions. Privacy based Solutions aim to detect rough riding while also ensuring the privacy of the rider and his surroundings. The Safety Based Solutions seek to prioritize safety over all of the other aspects in an e-scooter, leading it to usually be the most accurate in rough riding detection. Finally, the Maneuverability Based Solutions focus on accurately classifying the different types of movement e-scooters can make and then analyze rough-riding based on that.

1.3.1 Privacy Based Solutions

Related Work 1: Smart Scooter

The Smart Scooter is an example of the privacy based e-scooter rough riding detection technology as it discusses a novel, privacy-preserving, end-to-end sensor system [6] that employs lightweight machine learning models to provide

real-time feedback to users to prevent unsafe scooter operations. This study differs from other research in the vehicle safety market as most products use cameras to gather data. However, due to security concerns and poor performance during night times, this application attempts to use Radar Sensors and IMUs to try to gather data of the scooter's surroundings. The main issue with this is that RADAR is very low resolution and cannot make out anything compared to cameras.

They also use a Machine Learning Model on the back-end which is a 1D Neural Network to simply classify whether it is being ridden on a sidewalk or a road. It also tackles the issue of identifying safe parking spots. For example, lamp-posts and walls are safe, while roads and the middle of the sidewalk are unsafe. For this product, data is collected in real time through an app where users have to log in to the scooter and it gets fed through the app simultaneously. The data is not stored anywhere and the app just discards them immediately due to security and privacy concerns. The IMU data is used to classify the safety of the surrounding vehicles, terrain, and the type of transportation (example: city-bike vs mountain bike) while the RADAR sensors are used to identify stationary objects in the vehicle's path and then are used to identify the safety of the actual rider. In the future they want to use cameras to detect wrong way riding and parking management. Through the Random Forest Classifier which they used to classify between Sidewalks and Roads, they got 67% - 84% accuracy across their tests. One challenge they faced was that they had trouble collecting accurate speed data which limited their results to determine safety of the actual ride.

1.3.2 Safety Based Solutions

Related Work 2: Mobile Sensing Riding Analysis

The next product was a Safety Based Mobile Sensing System that leverages GPS, IMU and LiDAR scanners in order to perform a thorough analysis on the safety of the rider [7]. The GPS provides detailed logs on the direction, position and timestamp of the scooter at a particular time. The IMU measures the speed, acceleration, and general motion of the scooter. Finally the LiDAR scanner scans surrounding objects around it using a 360 degree laser. There is an additional camera installed for real time video footage of the scooter to review later if there are any issues with the scooter. In order to make the system more portable in field data collection, all sensors were connected with a Raspberry Pi platform for data acquisition, processing, and storing. They are powered by a portable low-voltage (5 V) battery pack. All sensor output was stored in a MicroSD card attached to the Raspberry Pi and can be downloaded wirelessly through a customized program developed by the research team. They also dive deep into the impact that vibration has on the small wheels and talks about the impact that rough terrain has on the small wheels of the e-scooter. They were able to detect rough riding and cracks for sidewalks much easier than on roads. Specifically they determined that Asphalt Roads were the safest to ride on while concrete roads were significantly more dangerous by a factor of 9.

Related Work 3: E-scooter Riding Law Adherence

This study focuses on how accurate e-scooters are into adhering to laws in Germany as the rise in e-scooters led to a lot of infractions and accidents in the past few years [8]. Moreover, this study wanted to access the factors that make an e-scooter safe and law adhering and the factors that do not so that rough riding detection systems can prioritize certain features over others. Through their study, they found out that after test 777 e-scooters, each with 12.5 hours of ride time, e-scooters posed a big threat to Germany's infrastructure, constantly causing harm to either the pedestrians or road traffic. Specifically, they states that one in ten e-scooters were observed riding against the flow of traffic without any warnings and one in four caused damage to the infrastructure currently in place. Furthermore, unsafe riding such as dual or triple riding was observed on roughly 5.1 percent of rides posing big threats to the rough riding metrics. Lastly, they emphasized that speed and weight were the two biggest factors when dealing with rough riding and to put more emphasis when dealing with riding data.

1.3.3 Maneuverability Based Solutions

Related Work 4: Rider Occlusion Detection

This research uses a novel, objective benchmark for partially occluded e-scooter riders to facilitate the characterization of vulnerable road user detection and classification models. This method provided a 15.93% improvement on the current state of the art e-scooter rider detection network, demonstrating that its model was superior [9]. They trained and tested the data on 1130 images including 543 e-scooter rider instances and 587 other vulnerable road user instances. This was created in order to characterize e-scooter rider detection and classification across a range of occlusion levels from 0 to 99% occluded. They were able to determine that the proposed methodology is more proficient at detecting partially occluded e-scooter riders with an overall accuracy of 0.599 or roughly 59% demonstrating that there is a lot of work to be done in this area. In their study, the number of false positives were the least for the proposed solution compared to the other reliable models, showing that the proposed model accurately can detect and avoid obstacles.

Related Work 5: Naturalistic e-scooter Maneuver Recognition

The Federated maneuver identification and Contrastive e-scooter Rider Learning system (FCRIL) aims to analyze e-scooter riding behaviors using inertial measurement unit (IMU) data for the purpose of classifying rider maneuver patterns such as turning, accelerating, among others. The FCRIL model consists of three core modules. This core includes - i) Rider Interaction Embedding Learning (RIEL) module for extracting features from time series statistical data using a long short-term memory mechanism (LSTM) and 2-dimensional convolutional neural networks (Conv2Ds) shown above. ii) Rider Interaction Learning and Maneuver Recognition (RILMR) module which is trained sequentially after the RIEL to avoid optimization conflicts. This module outputs the probability scores for the dif-

ferent riding behavior classes (shown above). iii) Asynchronous Federated Maneuver Learning (AFML) module to utilize edge device computing power to process collected IMU data on device, updating gradient parameters locally and uploading the updated parameters to the cloud instead of the entirety of the collected data.

Training each of the modules involved a total collection of 2,800 rider maneuver data records, 2,380 of which were taken from smartphone IMU sensors. The performance of the FCRIL model when compared to numerous baseline time series analysis models – including LSTM (Long Short-Term Memory), GRU (gated recurrent unit), RNN (recurrent neural network), DLSTM-Conv (LSTM-based denoising, followed by Conv2D layers) – demonstrates a significant improvement over existing baseline models, varying anywhere between a 5% to 28% improvement.

1.4 Contributions

Throughout each of the solutions investigated, no single approach is able to prioritize e-scooter safety, user data privacy and the increased longevity of rental e-scooters all at once. Since none of the solutions are targeted towards service providers of rental e-scooters, they are solely limited to the data analysis being collected by the e-scooter, or at best a live feedback mechanism to the scooter rider.

We believe a comprehensive, end-to-end solution pipeline is necessary in order to promote fair e-scooter usage and serve the best interests of the e-scooter service providers. Given the resource and time constraints of a final year, undergraduate senior design project, we resort to a preliminary iteration of such a pipeline (explained in the following chapter) in order to create a proof of concept, and build beyond existing solutions to monitor e-scooter usage behaviors while giving service providers access to fleet metrics and other related information. The complete source code can be found on GitHub [10].

Chapter 2

Proposed Solution

2.1 Overview

The proposed solution to the aforementioned concerns involves a 'Black Box' system that records e-scooter riding data for the purpose of detecting misuse and rough handling by the user in order to improve the cost-effectiveness and appeal of ride sharing services - we will refer to this generally as our 'Black Box' or 'e-scooter Black Box' [10]. The Black Box hardware consists of an onboard sensor board capable of storing incoming data from accelerometer and gyroscope sensors before relaying the results (post rental) to the ride-sharing corporations' central data system. Integral to the development process is a custom-built comprehensive data collection pipeline and analysis software that is flexible and scalable. To achieve this, we link together the ride sharing rental process with an iOS application for e-scooter users, a web application for service providers to monitor their assets, and a cloud database to store necessary riding data and related analysis outputs. The project necessitates the use of ride sharing assets, in addition to Black Box hardware such as a gyroscope, accelerometer and GPS sensors, control board, a portable power source, and GPU compute power in order to train our ML model.

Evident from the existing solutions is a lack of accuracy when it comes to spatial analysis, specifically through the LiDAR and Radar sensors. The ease of extracting riding events purely based on accelerometer and gyroscope data is significantly higher, making it an ideal focus for our solution. In our product, we focus on detecting riding events such as bumps, falls, and unstable riding behaviors across different terrains. We bypass significant resource costs associated with real time analysis of the data by implementing an offline, post-ride analysis system that serves the interests of the ride-sharing service providers.

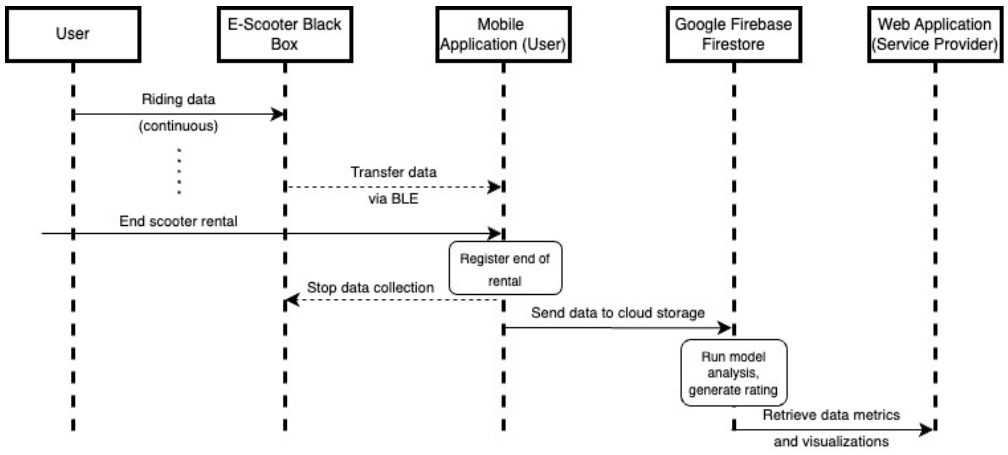

Figure 2.1: e-scooter Black Box Hardware Architecture Diagram

As shown in Figure 2.1 above, the following is a step-by-step process of how the proposed solution unfolds:

1. The user begins rental of an e-scooter equipped with our Black Box from a ride sharing service provider.

2. Prior to starting a riding session, the user pairs the Black Box with our iOS application on their smartphone and initiates data recording.

3. Throughout the riding session, a Bluetooth Low Energy connection is maintained to relay sensory data in real-time from the Black Box to the mobile device. This is necessary as limiting data transfer via BLE to post-ride would result in very long transfer times, as BLE data transfer rate is not sufficiently quick for transferring files as large as 100 megabytes.

4. Once the user logs the end of their riding session, the iOS application transfers the riding data collected from the Black Box to our cloud server.

5. Once the data transfer process is complete, the ride sharing service provider can access and view a detailed analysis of the user riding data using a dedicated web application designed for data processing and visualization.

2.2 Hardware Breakdown

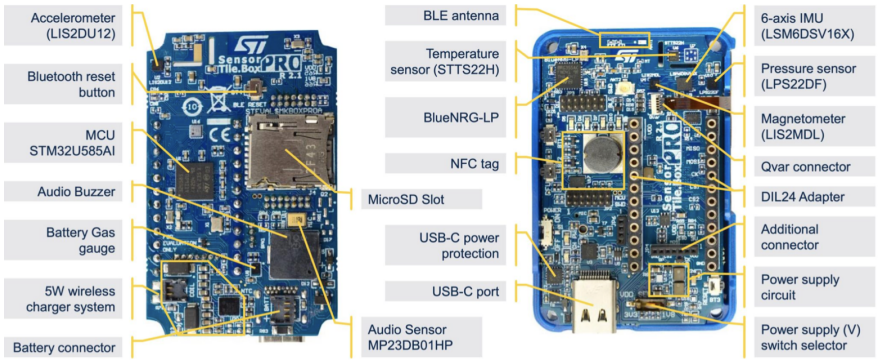

Figure 2.2: Internals of the STMicroelectronics SensorTile.Box Pro [11]

Our Black Box system is centered around a SensorTile.Box Pro [11] (as shown above in Figure 2.2) provided by STMicroelectronics, a multi-sensor hub responsible for taking the incoming data from the accelerometer and gyroscope sensors and writing the data stream to onboard storage (with the option to offload via Bluetooth Low Energy or BLE). The accelerometer will be responsible for measuring dynamic acceleration forces in three-dimensional space, while the gyroscope will measure the angular position of the scooter with respect to the ground. Combining the outputs of the two sensors, we can model the riding status of the e-scooter using different ML algorithms.

2.2.1 Ground Truth and Data Analysis

The central concept revolves around establishing a precise 'ground truth' for each classification category under consideration. This ground truth serves as a reference baseline, against which we measure and analyze the patterns and trends of the data collected in real-time from user activities or environmental interactions.

By meticulously comparing the real-time data against the established ground truth for each category, we aim to ascertain the most accurate trend representation or 'best fit line' for that specific set of conditions or activities. This comparative analysis allows us to discern subtle deviations or consistencies within the collected data, thereby enhancing the reliability and validity of our findings.

In the subsequent phase of our process, we apply statistical models and algorithms to interpret the compared data, enabling us to generate a 'likelihood probability score'. This score quantifies the probability of whether certain events or conditions, as predefined in our scenarios, have actually occurred. The higher the likelihood probability score, the greater the confidence we have in the accuracy of the data reflecting the real-world scenario being analyzed.

By adopting this refined approach, we aim to ensure a more accurate, reliable, and user-friendly experience in data collection and analysis, paving the way for more informed decision-making based on empirical evidence and precise measurements.

2.3 Software Breakdown

2.3.1 Blackbox Gateway iOS Application

Functionality

In addition to the Black Box hardware mounted on the e-scooter, we offer an end-to-end mobile iOS application, Blackbox Gateway, that facilitates the use of the e-scooter. Specifically, this app allows ride sharing users to log into an account, while easily and effectively logging their time riding the e-scooter and subsequently transferring the riding data to cloud servers for further analysis.

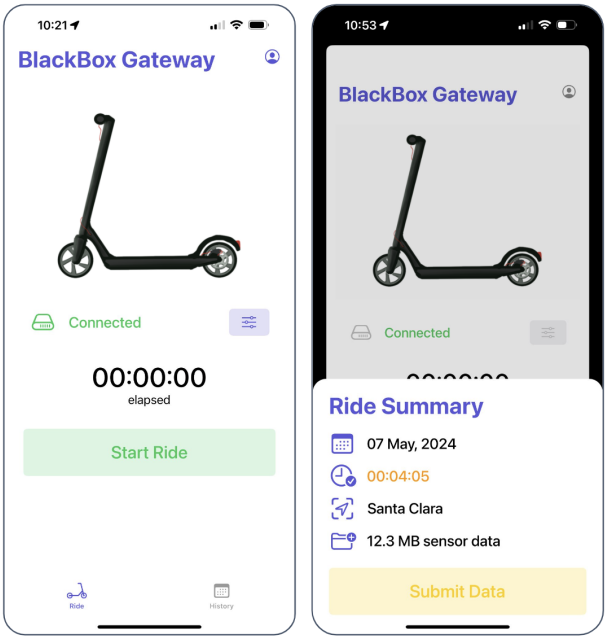

Figure 2.3: Blackbox Gateway iOS App for Data Offloading

The iOS application (as shown in Figure 2.4) provides an intuitive interface for users to sign into their account, pair their device with a Black Box mounted on an e-scooter, and collect their riding data needed for further analysis by the ride sharing service provider. The application stores and uploads data that the Black Box is able to collect through its magnetometer, accelerometer and gyroscope sensors in addition to location data. This data is necessary for the service provider to track their e-scooter fleet and monitor the riding and usage behaviors of their customers. Once a ride is logged, the Blackbox Gateway mobile application uploads the collected data to a cloud server hosted in Google's Firebase Firestore database [12], and triggers a subsequent Machine Learning-driven analysis of the riding data that is done through Google's Firebase Cloud Functions tool [13].

The Blackbox Gateway mobile application not only ensures that ride sharing service providers can monitor their

fleets remotely, but also has the potential for rapid user adoption – the use of the Blackbox will enforce better scooter treatment and safer riding behaviors, resulting in reduced service costs and lower rental fees. This benefits both the good-willed user and service provider business, creating a mutually beneficial relationship at the heart of which is the Blackbox Gateway application.

Dependencies

The STBlue SDK [14] is offered by STMicroelectronics to interact with BLE devices running STMicroelectronics' proprietary wireless protocols. This library is offered on both Android and iOS platforms, but due to the tight timeline of senior design and our team's prior experience with iOS development we chose to focus on developing an iOS application to facilitate data transfer between the physical e-scooter Black Box and the Google Firebase backend [12]. The most popular and respected industry standard for iOS development is the Swift programming language, which we chose to use along with Apple's native development tools [15] such as XCode and UIKit libraries that are essential for the development of all Swift iOS applications.

Integral to the development of the Blackbox Gateway mobile application specifically were the Google Firebase [12], STBlue [14], DGCharts [16], and JGProgressHUD [17] libraries. The Firebase [12] and STBlue [14] libraries are essential for enabling the core feature set of the Blackbox Gateway mobile application, and the remaining libraries were utilized to construct an appealing and intuitive user interface. Each of these libraries are explained below in order of decreasing importance to the core functionality of the Blackbox Gateway.

- **STBlue [14]** is an Android and iOS supported library that enables access to data transmitted from a Bluetooth Low Energy (BLE) device implementing the BlueST protocol. This library is used in multiple applications developed by STMicroElectronics including the ST BLE Sensor application. This library includes multiple nested dependencies including libraries such as STCore, YbridOpus [18], and others responsible for low-level hardware communication and data encoding from various types of on-device sensors.

- **Google's Firebase Library** provides tools to help develop application backends without managing the obstacles involved in managing servers. Not only does it offer effortless setup initially but also offers support for scaling up to large scale applications. Firebase has tools for server hosting, cloud storage solutions, and cloud compute resources, presenting a prime opportunity for us to prioritize efforts on getting our Blackbox pipeline up and running.

  We utilize the Authentication library [19] to ensure only registered users are able to access the application. Once a user has been authenticated, their data is retrieved from the Firestore database, which also houses user metadata such as ride duration, behavior analysis, and last registered scooter location. This allows us to keep the Blackbox Gateway mobile and web applications in sync with one another, while keeping the dedicated

interfaces for users and service providers separate from one another. Access to the database is restricted to verified applications, managed by the Firebase AppCheck library [20]. This enforces that only requests from authentic and untampered applications are processed.

- **DGCharts [16]** and **JGProgressHUD [17]** are frontend libraries aimed to simplify the necessary code to implement visual elements such as progress indicators and data graphs. JGProgressHUD [17] provides tools to help with communicating to the user during data transmission steps and other stages which require some time to execute.

Future Directions and Improvements

Ideally, we would have liked to design a system that would allow the monitoring of e-scooter behaviors while involving no additional commitment from the ride sharing customer, or a comprehensive mobile application that would involve the integration of rental services directly from the Black Box application, creating an intuitive, hassle-free experience for scooter ride sharing users. However, this is very difficult to accomplish without access to a given service provider's fleet management system, as different businesses employ numerous methods in order to recharge and maintain e-scooters, all while ensuring customers are able to rent scooters close to them.

The creation of the Blackbox Gateway iOS application was a preliminary approach to proving the concept of the end-to-end e-scooter monitoring system we aimed to create with the given constraints at hand – some of which include STMicroelectronic's hardware and software tools, a strict project timeline, and resource availability challenges.

2.3.2 Blackbox Gateway Web Application

Functionality

Additionally, we have a Blackbox Gateway web application that provides a secure interface for ride sharing providers to monitor their e-scooter fleet usage and riding safety of various ride sharing users.

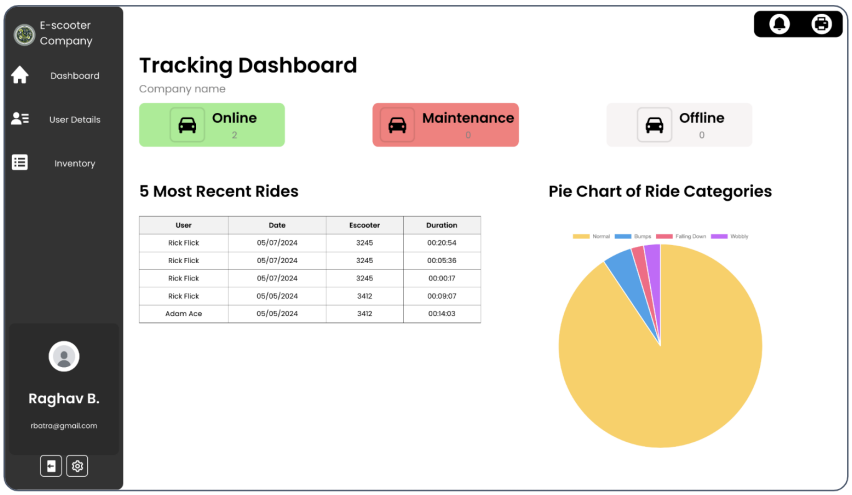

Figure 2.4: Blackbox Gateway iOS App for Data Offloading

The website (as shown in Figure 2.5) is designed to provide in-depth insights into each user's ride metrics and engagement with their e-scooter based upon user data from the Black Box. Upon authentication, the ride-sharing operator is introduced to an overview page, which systematically arranges all participants based on their accumulated performance metrics. These rankings derive from an array of criteria that assess users' efficiency and adherence to safety measures in recent e-scooter journeys. Participants achieving top scores due to exemplary riding habits are prominently displayed at the top of the list.

Beyond competitive features, the platform incorporates an alert system for critical updates or maintenance required to ensure the fleet's optimal performance. This mechanism is integral for communicating urgent fixes, necessary to uphold the fleet's reliability and safety. The interface also offers nuanced revenue analytics, detailing financial performance and highlighting times of peak profitability. This aids in identifying patterns and planning for enhanced revenue generation. Additionally, distinct tabs enable close monitoring of each e-scooter's condition, facilitating timely interventions for maintenance or repair.

To provide a detailed analysis of user conduct and efficiency, the site includes an intuitive sidebar with an extensive drop-down menu. This feature allows the provider to scrutinize the ride quality of individual users across different outings, fostering improved riding behaviors and operational transparency.

Dependencies

The dependencies used by the Blackbox Gateway Web Application can be broadly classified into two main categories: React Web Development and External Tools. Each is explained below in detail.

- **React Web Development [21]:**
  - **useState:** This Hook adds state management capabilities to functional components in React applications

[21], enabling dynamic and interactive user interfaces without the complexity of class components. Allows for dynamic and interactive user interfaces without requiring class components, making the code simpler and more readable.

- **useEffect:** Configurable to run under specific conditions using its dependency array, this Hook is a powerful tool for optimizing performance and behavior in React applications [21]. Specifically, it was utilized for fetching data from Google Firebase Firestore [12] in this application. Optimizes performance by running under specific conditions, reducing unnecessary re-renders, and enhancing the efficiency of data fetching operations. Enables dynamic, real-time updates to the user interface, improving the user experience with up-to-date information.
- **React Router [22]:**
  - **BrowserRouter (aliased as Router):** Facilitates seamless navigation between different components without reloading the page, crucial for enhancing user experience in single-page applications. Enhances user experience by allowing smooth, instant transitions between different parts of the application.
  - **Routes:** Acts as a container for Route components, determining which component should be rendered based on the current URL path. It was specifically used for navigating between different pages tied to various URLs within the application. Simplifies navigation by organizing and managing different routes, ensuring the correct component is displayed for each URL.
  - **useNavigate:** This navigation hook enables programmatic control over navigation within React components, such as handling redirects and other navigation events triggered by user interactions or application state changes. Provides flexibility and control over navigation, making it easier to handle complex routing logic and user interactions.

- **External Tools**
  - **Firebase [12]:** Provides real-time data fetching capabilities from Firebase's Firestore database, allowing the application to dynamically display updated user metrics and fleet conditions.
  - **Boxicons [23]:** Offers a comprehensive collection of vector icons that enhance visual consistency across web and mobile interfaces. In this application, these icons were specifically used for sidebar buttons. Enhances visual consistency and usability with a wide range of icons that are easy to implement and customize.
  - **Chart.js [24]:** A versatile charting library that uses HTML5 canvas to render responsive and customizable charts across various types, including pie, bar, and line graphs. These charts are used to visually represent

metrics related to user and e-scooter performance on different pages. Provides responsive, customizable charts that are easy to implement, enhancing data visualization and user engagement.

- **Chartjs-adapter-date-fns [25]:** Integrates Chart.js with date-fns to manage date and time data within charts, essential for precise time series data visualization. This integration is particularly used to enable tooltip hover functionality on all graphs, enhancing the interactive data exploration experience. Enhances time series data visualization with precise date and time management, improving the accuracy and interactivity of charts.

Future Directions and Improvements

To further enhance the Blackbox Gateway Web Application, several areas of development are envisioned:

- **Advanced Analytics Integration:** Implementing more sophisticated analytics algorithms to provide deeper insights into user behavior patterns and fleet efficiency, which could help in predictive maintenance and optimization of fleet deployment.
- **User Interface Improvements:** Continual updates to the user interface to improve usability and accessibility, potentially incorporating more intuitive navigation and real-time data visualizations that adjust based on user preferences or roles.
- **Enhanced Security Features:** Strengthening the application's security framework to ensure that user data and fleet information are protected against unauthorized access and cyber threats.
- **Scalability Solutions:** Developing solutions to enhance the scalability of the application to handle increasing amounts of data and user numbers without compromising performance.

By addressing these areas, the Blackbox Gateway Web Application can continue to evolve, meeting the growing demands of the ride-sharing industry and enhancing its service offerings.

2.3.3 Google Firebase

Functionality

Google Firebase serves as a comprehensive data bridge between the Blackbox Gateway iOS and web applications by providing consolidated central data storage and related tools for accessing and modifying it. The Firebase suite provides services such as a database (Cloud Firestore [12]), secure login tools (Authentication [19]), serverless, in-cloud file storage (Storage [26]), application verification tools (App Check [20]), and serverless, cloud execution of custom code (Functions [13]).

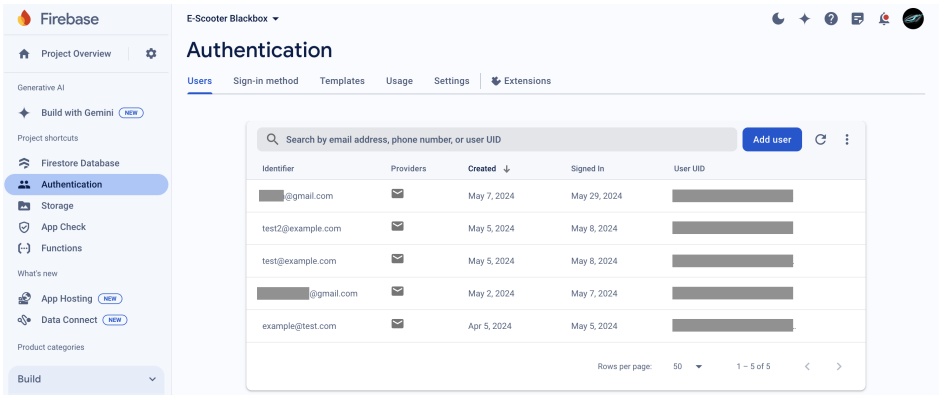

Figure 2.5: Firebase Authentication Console

Firebase Authentication [19] provides tools such as pre-fabricated UI libraries, backend infrastructure and software development kits (SDK's) to authenticate users within an application. It utilizes industry standards such as OpenID Connect and OAuth 2.0 for custom backend integration and has close connections with other Firebase services. This library provides key user authentication services for the Blackbox Gateway iOS and Web applications, and also restricts unauthorized access to user data stored in both Cloud Firestore [12] and Cloud Storage [26].

Firebase App Check [20] protects Firebase backend services from unwanted activity by preventing unauthorized client access to Firebase products such as Firestore [12] and Cloud Storage [26], as well as other non-Firebase backend resources. Enabling App Check means that devices running the Blackbox Gateway iOS or web applications need proof of an untampered, original application and device. This is key to ensuring backend server resources are accessed from an uncompromised device.

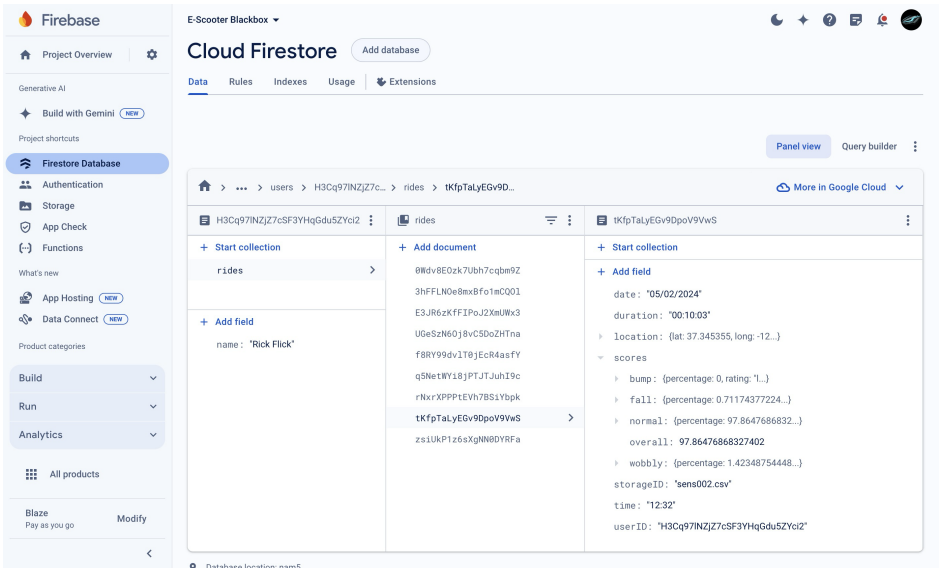

Figure 2.6: Cloud Firestore Database Console

The Cloud Firestore database [12] is a flexible yet scalable NoSQL database that allows data across applications to remain in sync with one another, while providing resources for tight-knit integrations with other Firebase and Google Cloud products. It supports flexible data structures in a document-collection hierarchy, advanced search capabilities, real time data synchronization along with offline cache support, and a scalable architecture powered by Google Cloud. For our e-scooter Black Box solution, we chose to store our data in two primary data collections, the first being for user related data such as name and other login information, and a second collection for all scooter data. The latter is organized by scooter ID, inside of which riding metadata for each user's rentals is stored, along with the storage file ID which is used to reference the rental's corresponding riding data within Cloud Storage. The structure for this database was heavily influenced by the data needed to be displayed in the Blackbox Gateway Web Application, which is used by service providers who are primarily concerned with viewing data organized either by individual scooters or users.

Firebase Cloud Storage [26] is a scalable object storage service that can be used to store user-generated content such as images, videos and other files. Its key features include robust, reliable upload and download operations, tight integration with Firebase Authentication providing secure data access, and a scalable architecture powered by Google Cloud. The e-scooter Black Box necessitates a way to store raw sensory user riding data in the cloud for analysis and visualization from the Blackbox Gateway Web Application. As a result, our Cloud Storage 'bucket' or storage container is organized by unique user identifiers, or UID's. These UID's are generated when a new user is registered via the Firebase Authentication library, and can only be written to or accessed from an application once the user is signed in, enabling secure access. Contained within each user's folder are their raw riding data files, which are analyzed by algorithms running in Google Cloud Functions [13].

Firebase Cloud Functions [13] is a serverless solution capable of executing custom programs triggered either manually via triggers such as HTTPS requests or background events. This allows the execution of custom code, written in languages such as JavaScript or Python, without the hassle of having to manage and scale physical servers and computational resources. Firebase's close integration with Google Cloud means that the vast computational resources available on Google Cloud can be utilized in tandem with the earlier mentioned Firebase resources for a powerful, end-to-end server solution addressing all the backend needs of the e-scooter Blackbox solution. Once the Blackbox Gateway iOS application records the end of a user ride, it uploads the raw user riding data to Cloud Storage, and ride details such as date, duration, storage ID and location to Cloud Firestore [12]. A trigger has been set up such that whenever a new document is created when a user rental ride is finished, a cloud function to analyze the new riding data is triggered. This function contains the end-to-end code to run our Machine Learning Ensemble model (explained in a later section), which automatically analyzes the newly added data record in Firestore [12], analyzes its corresponding ride data file from Cloud Storage [26], and writes the analysis scores and data to Cloud Firestore [12] without any needed intervention. This function takes less than three minutes to run on around half an hour of riding data.

Future Directions and Improvements

Moving forwards, possible improvements to our utilization of Firebase primarily involve adapting the backend architecture to address changes to the end-to-end Black Box solution flow. One such example is adapting the Firestore [12] database to store additional metrics and information to support novel visualization methods on the Blackbox Gateway Web application.

2.4 Data Offloading

Existing e-scooter service providers have employed numerous approaches to addressing the regular recharging and service needs of rental scooters, the latest of which involves a dockless e-scooter solution [27]. This offers users the flexibility to rent an e-scooter straight from their mobile phones, and end their rental wherever they please without having to worry about returning the scooter to a specified location. This is especially befitting for e-scooters due to the shorter ride duration, irregular rental patterns and dependence on factors such as environmental conditions that influence scooter rentals [27]. However, this dockless approach assumes that the primary reason e-scooters need to be taken out of service is for battery recharging purposes, and occasionally for other maintenance – tires, brakes, and the like. Most of the time, these rental respites are not necessary following a single customer scooter rental.

The e-scooter Black Box solution necessitates a higher level of connectivity with each individual rental scooter, as the service provider should be able to access each scooter's riding data that is being recorded without the user needing to physically return the scooter to a certain location, to ensure the user's convenience is not compromised. After spending a significant amount of effort on writing custom firmware for the SensorTile.Box Pro [11] for custom data logging to an onboard storage unit, we realized that compromising the ease of use for service providers to keep the user uninvolved was not the ideal course of action. Within the given time and resource constraints, we decided the most feasible solution to this issue was to utilize STMicroElelectronic's developer tools to create a mobile iOS application, Blackbox Gateway, that is responsible for the wireless integration between the sensor hardware and the service provider's web monitoring application via Bluetooth Low Energy (BLE).

Location-independent, wireless data transmission via BLE has the following advantages:

Low Energy Consumption: BLE technology is particularly advantageous as it consumes minimal power. This is crucial for a scooter device, where energy conservation is a priority to ensure the prolonged operation of the vehicle. By leveraging BLE, the Black Box can transmit data periodically without significantly impacting the scooter's battery life.

Cost-Effectiveness and Accessibility: Implementing Bluetooth technology for offline data transfer is cost-effective and widely accessible. Bluetooth-enabled devices are prevalent in the market, making it convenient for users to con-

nect their smartphones to the Black Box for data retrieval and analysis. Rental users can simply enable Bluetooth on their smartphones to connect to the e-scooter Blackbox device. This user-friendly approach simplifies the data transfer process, making it accessible to a wide range of users without complex setup requirements.

Data Security and Privacy: Offline transfer via BLE offers a level of data security as it operates within a short-range wireless network, reducing the risk of unauthorized access. Moreover, the data transfer can be encrypted to ensure the privacy and integrity of the information being transmitted from the Black Box to the paired device via modern technologies such as LE Security Mode 3 [28].

Once the data has been transferred from the Black Box to the device running our iOS application, another transfer is initiated from the iOS device to our cloud server, running on Google's Firebase Firestore [12]. This allows us to remotely access and analyze the data being collected from multiple ride sharing users without the need to wait for the restocking of e-scooters at the end of their rental period. Once the upload is successful, the Blackbox Gateway iOS application initiates a machine learning pipeline to analyze the riding data using Google's Firebase Cloud Functions API to begin the data analysis process without any additional action needed on behalf of the service provider or end rental user.

2.5 Data Processing

As a project that seeks to not only provide accurate analysis of data, but also fast and efficient collection of data, our team focused specifically on data collected exclusively through STMicroelectronics' SensorTile.Box Pro [11] which is able to collect sensor data quickly.

As a result, our dataset consists of data from nine distinct SensorTile.Box Pro's sensors [11], combining accelerometer, gyroscope, and magnetometer each with x, y, and z axis measurements to thoroughly analyze e-scooter rough riding. This varying sensor data allows for a detailed understanding of riding behavior, essential for distinguishing between different types of rough riding movements. The data is collected at a rate of 100Hz, which means that for every one minute of data each sensor collects 6000 data points and across the nine sensors, 54,000 data points are collected. In total, we ended up collecting 20 hours worth of supervised data which ended up leading to roughly 194 M data points to use in our data analysis portion. Although 194 M data points is significantly large, we still believe it is a small dataset compared to other e-scooter data sets that comprise more than 50 hours worth of data, however, as stated earlier, that data is out of scope for our project.

After the initial retrieval of data from the SensorTile.Box Pro [11], our team utilized an elaborate data analysis algorithm that involved classification, feature extraction, ensemble models, thorough storage methods, and cloud functions.

2.5.1 Classification

Before the data can go through the feature extraction process, due to the use of supervised machine learning models, our team went through and assigned one of four labels to every three seconds of data. We chose three seconds due to our extensive testing with different types of rough riding times. Most rough-riding methods last 1-2.5 seconds, so by observing every three seconds, it helps account for all different types of rough riding. Currently, our method only accounts for four different categories: Normal/Safe Riding, Bumpy Riding, Wobbly Riding, and Falling Down.

With the project's scope being focused on e-scooter durability, the following are the definitions of the different types of rough riding.

Normal/Safe Riding is riding in a safe manner, which means that the e-scooter is ridden in smooth terrain, operated in safe speeds and no sudden movement causing the e-scooter to get damaged.

Wobbly Riding is riding in a continuous and extremely side to side motion causing risk to both the rider as well as the e-scooter's tires.

Bumpy Riding is unsafe riding in rough terrain that has lots of cracks and holes, causing the e-scooter to constantly jump and land harshly. This leads to lots of unnecessary pressure being put on the tires of the e-scooter.

Falling Down is the riding in which the e-scooter either falls on its side or goes over a severe bump that causes the e-scooter to be falling down for a prolonged period of time.

2.5.2 Feature Extraction

The cornerstone of our data preparation phase is the extraction of sophisticated features from the accelerometer, gyroscope, and magnetometer data [29]. This meticulous process involves computing two types of features, statistical features and frequency features, which are:

Statistical Features:

- **Mean:** The mean of the sensor data provides the average value of the riding conditions, helping to identify the typical level of roughness experienced during e-scooter rides.
- **Variance:** Variance measures the variability in the riding conditions, indicating how much the roughness of the ride fluctuates from the average condition.
- **Standard Deviation:** Standard deviation quantifies the spread of the roughness levels, offering a clear measure of how much the riding conditions vary around the average roughness.
- **Peak-to-Peak Distance:** Peak-to-peak distance highlights the intensity of rough riding by measuring the range between the most extreme vibrations, capturing the severity of bumps and jolts.

- **RMS (Root Mean Square):** RMS captures the overall strength of the roughness, providing a measure of the energy in the vibrations and movements experienced during the ride.
- **RMS (Root Mean Square):** RMS captures the overall strength of the roughness, providing a measure of the energy in the vibrations and movements experienced during the ride.
- **Skewness:** Skewness measures the asymmetry of the roughness distribution, helping to identify whether the riding data is skewed towards more extreme rough conditions, such as frequent sharp turns.
- **Kurtosis:** Kurtosis indicates the tailedness of the roughness distribution, offering insights into the extremity of the riding conditions, such as the presence of sudden and severe jolts.
- **ZCR (Zero Crossing Rate):** ZCR detects vibratory movements by measuring how often the signal crosses the zero line, providing clues about the surface quality and the roughness encountered during the ride.

Frequency Features:

- **Spectral Centroid:** The spectral centroid indicates the dominant frequencies in the rough-riding data, highlighting whether high-frequency vibrations (indicating rough terrain) are prevalent.
- **Spectral Bandwidth:** Spectral bandwidth reveals the range of frequencies present in the riding data, distinguishing between smooth rides (narrow bandwidth) and rough rides (wide bandwidth with varied frequencies).
- **Spectral Entropy:** Spectral entropy measures the complexity or randomness of the frequency distribution, offering insights into the unpredictability and variability of the rough-riding conditions.

The feature extraction matrix is computed for every three second chunk of data to fully account for one of the four rough riding metrics described earlier. Within that three second data chunk, the same features were extracted every half-second to account for any actions that were between data chunks. For instance, wobbly riding occurred from 2 - 5 seconds, normally under the three second chunk, it would not identify wobbly riding from 0-3 seconds and only identify it happening from 4-6 seconds. So through calculating it every 0.5 seconds it accounts for actions happening in between.

2.5.3 Machine Learning

To analyze and classify e-scooter riding behavior accurately, we employ K-Nearest Neighbors (KNN) Decision Trees, Random Forest, and Gradient Boosting focusing on four categories of riding: Safe Riding, Falling Down, Wobbly Riding, and Bumpy Riding. These models were selected for their proven effectiveness in classification tasks.

In order to train our Machine Learning Model, we took the approach to split the 20 hour dataset 80% training and 20% testing, taking the standard approach to data splitting. Our model generated a high accuracy of 88%. Once this

model gets trained and tested, it is then used to predict the classifications of future rides and assigns classifications for each data chunk.

K-Nearest Neighbors (KNN)

The K-Nearest Neighbors (KNN) [30] algorithm is a type of instance-based or memory-based learning where the classification of a new sample is determined by the majority class among its 'k' nearest neighbors. It is inherently non-parametric, meaning it makes few assumptions about the form of the data distribution. This characteristic makes KNN particularly suitable for applications where the decision boundary is irregular.

- **Why We Used It:** In the context of e-scooter rough riding analysis, KNN can discern complex riding patterns by analyzing the proximity of different sensor readings in feature space. By evaluating the 'similarity' of riding sessions using the Euclidean distance between feature vectors, KNN can classify unfamiliar riding patterns based on known examples.

- **Advantages:** Highly flexible when adding more benchmarks: easily adapts to new data points without needing to retrain the model from scratch. Equal feature weights for all features: each feature contributes equally to the distance calculation, making it straightforward to implement and interpret.

- **Accuracy:** The KNN model achieved an accuracy of 68.5% on our dataset. The choice of 'k' — the number of neighbors to consider — was optimized to balance sensitivity with specificity, ensuring robust and reliable classification of riding behaviors. The model's performance reflects its ability to classify riding behaviors like bumps, falls, normal riding, and wobbly movements effectively.

Decision Trees

Decision Trees are a form of supervised learning algorithm used for both classification and regression tasks [31]. They work by splitting the dataset into two or more homogenous sets based on the most significant splitter/differentiator in input variables.

- **Why We Used It:** In the context of e-scooter rough riding analysis, decision trees can systematically break down our dataset into smaller subsets, thereby making the data easier to manage and interpret. Decision trees are particularly advantageous due to their simplicity and interpretability.

- **Advantages:** Easily handles missing values from sensors: Capable of managing incomplete data by considering the most significant available information. Deals with unbalanced data: Can prioritize features that lead to higher node purity, ensuring accurate classification even when some riding behaviors are less frequent.

- **Accuracy:** The Decision Tree model achieved an accuracy of 84% on our dataset. Each node in the tree represents a feature in the dataset, and each branch represents a decision rule, leading to a clear and straightforward representation of the decision-making process. This transparency allows for easy validation by domain experts, ensuring that the model's logic is sound and conforms to intuitive understandings of e-scooter dynamics.

2.5.4 Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and merges their results to improve accuracy and control overfitting. Each decision tree in the forest is built from a random subset of the training data, and the final output is determined by majority voting among the trees [32].

- **Why We Used It:** Random Forest is particularly effective for e-scooter rough riding analysis due to its robustness against noise and its ability to handle a large number of features without overfitting. This makes it ideal for analyzing sensor data with varied and complex patterns of e-scooter riding behavior.

- **Advantages:** Really accurate: Combines multiple trees to reduce variance and increase accuracy. Does not overfit with more features: Handles a large number of features well, maintaining performance even as complexity increases. Handles imprecise data: Robust to noisy data and missing values, making it suitable for real-world sensor data.

- **Accuracy:** The Random Forest model achieved an accuracy of 0.89 on our dataset, indicating a high level of precision in classifying different riding behaviors.

Gradient Boosting

Gradient Boosting is an ensemble method that builds models sequentially, where each new model corrects the errors of the previous ones. It focuses on learning from the mistakes made by prior models to gradually improve performance [33].

- **Why We Used It:** Gradient Boosting was chosen for its ability to handle complex data patterns and its flexibility in optimizing for various loss functions. It is particularly useful in fine-tuning the classification of e-scooter riding behaviors by focusing on the hardest-to-classify examples.

- **Advantages:** Handles missing and uneven data: Capable of dealing with incomplete datasets and varying data distributions. Very adaptable based on data: Adjusts to the specific characteristics of the data, improving overall performance through iterative learning.

- **Accuracy:** The Gradient Boosting model achieved an accuracy of 88% on our dataset, demonstrating its effectiveness in accurately classifying rough riding behaviors.

Ensemble Model

Our sophisticated model utilizes all classifications from four of the above models to generate its own prediction. We make sure to take the mode of all of the classifications and then create our own predictions [34]. For example, when classifying for seconds 0-3, assume that KNN classifies it as Wobbly, Decision Trees classify it as Safe, Random Forest classifies it as Wobbly, and Gradient Boosting classifies it as Wobbly. Our model would see that three out of the four models classified it as Wobbly, so it is most likely Wobbly Riding at that time period. This eliminates the discrepancy within a certain model and allows for a greater accuracy and reliability with our ensemble model overall. Our results back up this claim as we were able to achieve an accuracy of 88%, which falls in the upper quartile of model accuracies as 70-90% is considered accurate across models. This demonstrates that our feature extraction process and data collection did not lead to underfitting or overfitting our data.

Score Generation

After our model is able to generate a classification for a new e-scooter ride, the classifications are then passed in to our score generator function that generates two types of scores that companies can use in order to evaluate rider data. There are two main types of scores, an overall score associated with the ride as well as individual scores assigned to each classification.

- **Classification Score:** Classification score is a dictionary with the following two scores:

  Riding Percentage: Percentage of the total ride that was ridden in that classification. For example for Safe Riding a sample percentage might be 70%, suggesting that 70% of the entire ride was normal riding.

  Rating: A rating is assigned of either “low”, “medium”, or “high” signaling whether the Riding Percentage is higher or lower than an average ride.

- **Overall Score:** The Overall Score is the main score that is interpreted by most users to understand the rider’s degree of undesirable riding behaviors. It takes into account the rough-riding percentages and deducts points for extreme rough riding. This is a score from 0 -100, 0 being extremely poor riding and 100 being excellent riding.

The entire process of analyzing new rider data and running the ensemble machine learning model is run asynchronously through Firebase’s Cloud Functions, described in detail in section 2.3.3. This allows the model to generate real time scores for each ride, and takes roughly 2-3 minutes to run on a 20-30 minute long riding data set.

2.6 Solution Development

2.6.1 Agile Software Development

In this project, we adopted Agile software development principles, which emphasize flexibility, rapid iteration, and responsiveness to change. This methodology facilitated our ability to adapt to evolving project requirements and unexpected challenges, enabling efficient progress and timely deliveries.

2.6.2 Iterative Planning

Our approach involved detailed iterative planning where the project was broken down into manageable units. We scheduled regular planning sessions at the beginning of each sprint to reassess our goals and priorities based on the most recent results and stakeholder feedback. This ongoing planning process allowed us to refine our strategies and make informed decisions on the fly, ensuring alignment with our overall project objectives.

2.6.3 Dynamic Risk Management

We implemented dynamic risk management strategies to proactively identify potential issues and adjust our course of action accordingly. This included regular risk assessments during our sprint retrospectives, where we evaluated what risks materialized and how effectively we mitigated them. By continuously monitoring risk factors, we were able to implement preventive measures and respond swiftly to any issues, minimizing their impact on our project timeline.

2.6.4 Timeline and Project Risks

Initially organized into 2-week sprints, our timeline had to be adjusted as the project evolved. We faced several interruptions, particularly when data collection was incomplete, which stalled other aspects of the project such as iOS app development, machine learning implementation, and web application enhancements. These challenges necessitated a flexible approach to our sprint planning and delivery deadlines.

To address these complexities, we strategically divided responsibilities among four specialized groups: firmware, iOS app, machine learning, and web application. This division enabled us to tackle specific tasks more efficiently and manage interdependencies more effectively.

Overall, employing Agile methodologies provided the framework necessary for our team to navigate the multi-faceted nature of the project, ensuring adaptability and maintaining momentum towards successful completion.

Chapter 3

Constraints and Standards

3.1 Constraints

3.1.1 Senior Design Timeline

One of the primary constraints for this project was the strict timeline inherent to the senior design process. With a fixed academic schedule, our team had to continuously adapt tasks to fit the remaining time. This required a flexible approach to ensure the completion of a working Minimum Viable Product (MVP) within the available time frame. Each phase of the project, from initial design to final implementation, needed to be meticulously planned and executed to avoid delays and ensure timely completion.

3.1.2 Target E-Scooter Ride Sharing Service Providers

Our project aims to address the needs of e-scooter ride-sharing service providers, which introduced specific constraints. This necessitated equal emphasis on all parts of the end-to-end data pipeline, including data collection, transmission, analysis, and visualization. The success of our solution hinged on the seamless integration and performance of each component. Any blockage or inefficiency in one part of the pipeline, such as delays in data analysis or visualization due to data collection issues, could compromise the effectiveness of the entire system.

3.1.3 External Dependencies

Our project relies heavily on external hardware and software components, including several dependencies:

- **STMicroelectronics Hardware and Software:** We utilized STMicro hardware (SensorTile.Box Pro [11]) and software (STBlue SDK [14]) for data collection and processing. Any technical issues or communication overheads with these components needed to be resolved promptly to avoid delays.
- **GoTrax Scooter:** The physical e-scooters used for data collection, specifically GoTrax models, posed potential constraints. Maintenance issues such as flat tires or other mechanical problems could introduce time overheads, affecting our data collection schedule and project timeline.

3.1.4 Budget Constraints

Budgetary constraints also play a significant role in shaping our project. The cost of acquiring and maintaining hardware components, as well as potential software licensing fees, had to be carefully managed. We needed to ensure that our solution remained cost-effective while meeting all technical requirements. This necessitated a careful balance between quality and cost, often requiring creative solutions to maximize the utility of available resources.

3.1.5 Quality and Reliability

Ensuring high quality and reliability of the collected data and overall system performance was crucial. The data needed to be accurate and the system reliable enough to be trusted by ride-sharing service providers. This requirement influences our choice of hardware, software, and the overall design of the data pipeline.

3.1.6 Legal and Ethical Considerations

Legal and ethical constraints were also significant. The data collected by our system had to be handled in compliance with privacy regulations to protect user data. Ensuring that our system did not infringe on user privacy or violate data protection laws was a priority throughout the project.

3.1.7 Maintenance and Usability

The maintainability and usability of our system were essential considerations. Our solution needed to be easy to use for both end-users (ride-sharing customers) and service providers. Additionally, the system had to be easy to maintain and update, ensuring long-term usability and scalability.

3.1.8 Risk Management

Managing risks associated with technical failures, project delays, and data integrity was a continuous process. We implemented dynamic risk management strategies to identify potential issues early and adjust our plans accordingly to mitigate these risks effectively.

To summarize, our project faced multiple constraints related to timeline, external dependencies, budget, quality, legal considerations, maintenance, and risk management. Addressing these constraints effectively was key to the successful implementation of our innovative e-scooter monitoring solution.

3.2 Standards

Project Management Standards

Adopting the Agile software development methodology allowed us to manage the project's dynamic nature effectively. Key elements of our project management standards included:

- **Agile Software Development Cycle**
  - **Iterative Project Planning**: We employed iterative planning to continuously reassess and adjust our goals and priorities based on recent results and stakeholder feedback.
  - **Dynamic Risk Management**: Regular risk assessments were conducted to identify potential issues early and implement preventive measures.
  - **Scalability and Flexibility**: Agile principles ensured that our project could adapt to evolving requirements and unforeseen challenges, maintaining scalability and flexibility.
- **Project Timeline & Development**
  - **2-Week Sprints**: Our development process was structured into 2-week sprints, allowing for frequent pivoting and iterative improvements.
  - **Dynamic Scope Changes**: The project scope was adjusted dynamically throughout the year to address emerging needs and challenges.
  - **Delegation of Responsibilities**: Tasks were divided among specialized groups focusing on firmware, iOS app, machine learning, and web application development, ensuring efficient task management and expertise utilization.

Technology Standards

Our project employed a variety of technological standards to ensure high performance, compatibility, and ease of development. The selected technologies and their standards are as follows:

- **Swift**
  - **High Performance, Native IDE**: Swift [15] was chosen for its high performance and seamless integration with Apple's native development environment (Xcode). This provided a robust platform for developing the iOS application.
  - **STBlueSDK Familiarity**: Utilizing the STBlueSDK [14] in Swift allowed us to efficiently handle Bluetooth Low Energy (BLE) communication with the SensorTile.Box Pro [11].
- **React Component-Based, Declarative**: React [21] was selected for developing the web application due to its component-based architecture, which promotes reusability and maintainability. Its declarative nature simplifies the creation of interactive user interfaces.
- **C Programming Language [35] Efficiency and Performance**: The core firmware for the SensorTile.Box Pro [11] was developed in C [35], leveraging its efficiency and performance for low-level hardware interactions.

- **Python [36] + SciKit Learn [37]**
  - **Wide Range of ML Libraries**: Python [36], combined with SciKit Learn [37], provided a comprehensive set of machine learning libraries essential for our data analysis and modeling tasks.
  - **Versatile, Deployable on GCF**: Python's versatility and compatibility with Google Cloud Functions (GCF) [13] enabled us to deploy our machine learning models in a scalable and efficient manner.

Industry Standards

Throughout the project, we adhered to various industry standards to ensure the quality, interoperability, and sustainability of our solution:

- **Bluetooth Low Energy (BLE) Standards**: Implementing BLE standards ensured efficient and secure data transmission between the SensorTile.Box Pro and the mobile application.
- **IEEE Standards for Networking and Communication**: Adhering to IEEE standards facilitated reliable and standardized communication protocols.
- **ISO Standards for Software Development**: Compliance with ISO standards for programming languages and design methodologies helped maintain code quality and project management practices.
- **Accessibility and Usability Standards**: Ensuring our applications followed best practices for accessibility and usability made our solution more inclusive and user-friendly.

Incorporating these standards into our project not only enhanced the quality and performance of our solution but also ensured that it met industry benchmarks for reliability, security, and interoperability.

Chapter 4

Societal Issues

4.1 Ethical Impacts

The introduction of sensor-based monitoring on rental e-scooters raises significant ethical concerns regarding privacy and surveillance. While the primary intent is to ensure responsible usage and reduce wear and tear, it inherently involves the collection of detailed data on individual users' behavior and movements. This could lead to potential misuse of personal data, where users' riding habits might be tracked without their explicit consent or awareness. Furthermore, there is the risk of biased algorithms unfairly penalizing certain groups of users, particularly if the data used to train these algorithms lacks diversity or reflects existing societal biases. Ensuring transparency in data collection practices, user consent, and the ethical deployment of machine learning algorithms will be crucial to mitigate these concerns.

4.2 Social Impacts

Implementing a sensor-based monitoring system on e-scooters can have significant social implications. On one hand, it may encourage users to ride more responsibly, knowing that their behavior is being tracked and evaluated, potentially leading to safer streets and reduced vandalism. On the other hand, it could exacerbate social inequalities if penalties or rewards are not equitably distributed. For instance, individuals from lower socioeconomic backgrounds might disproportionately suffer from punitive measures if they cannot afford to pay for damages or penalties incurred. Moreover, this level of surveillance might deter individuals from using e-scooters altogether, thereby reducing accessibility to a form of affordable and environmentally friendly transportation.

4.3 Political Impacts

Politically, the deployment of advanced monitoring systems on rental e-scooters can influence regulatory frameworks and urban transportation policies. Governments may need to enact new regulations to oversee the ethical use of sensor data and ensure that rental companies comply with privacy standards. Additionally, there could be political

pressure to balance innovation with the protection of citizens' rights, leading to debates on the extent of acceptable surveillance in public transportation. Policymakers may also face lobbying from both the e-scooter industry and privacy advocates, impacting the speed and direction of legislative action. Moreover, successful implementation of these technologies could set a precedent for similar surveillance measures in other forms of shared transportation, prompting broader discussions on technology and civil liberties.

4.4 Economic Impacts

The integration of sensor boxes on e-scooters has significant economic implications. By reducing the frequency and severity of damages through better monitoring, rental companies can lower maintenance and replacement costs, thereby improving their profitability. Additionally, this could lead to more favorable insurance terms, as insurers might view the monitoring system as a risk mitigation tool. However, the initial investment in sensor technology and the ongoing costs of data analysis could be substantial, potentially leading to higher rental prices for consumers. Moreover, the enhanced accountability might shift some financial burden to users, especially if they are penalized for rough handling, which could deter usage and impact overall revenue for the rental services.

4.5 Health and Safety Impacts

From a health and safety perspective, the deployment of sensors on e-scooters can contribute to safer riding environments. By promoting responsible riding through monitoring, the likelihood of accidents and injuries could decrease. This could result in fewer emergency room visits and a lower strain on public health resources. Additionally, real-time data collection can help identify hazardous areas or common causes of accidents, enabling targeted interventions by city planners. However, there might be concerns about how these sensors are implemented, as overly intrusive monitoring could lead to user anxiety or distraction, potentially having an adverse effect on rider safety.

4.6 Manufacturability Impacts

In terms of manufacturability, integrating sensor boxes into e-scooters requires collaboration between hardware and software developers. Manufacturers will need to adapt their designs to accommodate the new technology, ensuring that sensors are robust and durable enough to withstand the stresses of regular use and environmental conditions. This could increase production complexity and costs, necessitating new manufacturing processes and quality control measures. However, the demand for such advanced e-scooters could spur innovation and lead to advancements in manufacturing techniques, potentially benefiting the broader tech and transportation industries.

4.7 Sustainability and Environmental Impacts

Sustainability is a key concern in the deployment of sensor-equipped e-scooters. On one hand, improved monitoring can extend the lifespan of e-scooters by ensuring better maintenance and reducing damage, thus decreasing the environmental impact associated with frequent replacements. Additionally, promoting responsible usage aligns with the goals of sustainable urban mobility by encouraging the use of shared electric vehicles over personal, fossil-fuel-powered alternatives. On the other hand, the production and disposal of additional electronic components, such as sensors and batteries, introduce new environmental challenges. Ensuring that these components are recyclable or have minimal environmental footprint will be essential to maintain the sustainability benefits of e-scooters.

4.8 Usability Impacts

The integration of sensor boxes on e-scooters can significantly affect usability. On the positive side, these sensors can provide valuable feedback to users, helping them to understand and improve their riding behavior. This could lead to a more intuitive and user-friendly experience, as riders become more aware of the impact of their actions on the scooter's performance and durability. However, the added complexity might also pose challenges, especially for less tech-savvy individuals who might find the technology intimidating or difficult to navigate. Ensuring that the system is user-friendly, with clear instructions and seamless integration with mobile apps, will be crucial to maximize the benefits without alienating any user groups.

4.9 Lifelong Learning Impacts

The use of sensor technology in e-scooters can promote lifelong learning by encouraging users to continuously improve their riding skills. Through the feedback provided by the sensors, riders can learn about safe and efficient riding practices, contributing to a culture of ongoing personal development. This aspect of continuous improvement can extend beyond e-scooters, fostering a mindset of self-awareness and responsibility in other areas of life. Additionally, as users interact with advanced technology, they gain digital literacy and technical skills that are increasingly valuable in today's world. This educational aspect not only benefits individual users but also contributes to a more informed and capable society.

4.10 Compassion Impacts

Introducing sensor boxes on e-scooters has the potential to foster a sense of compassion and community among users. By promoting responsible usage, individuals may become more considerate of others who use shared resources, recognizing the collective responsibility for maintaining communal assets. This could lead to a greater sense of

empathy and social cohesion, as riders become more aware of the impact of their actions on others. Additionally, by potentially reducing accidents and injuries through improved riding behavior, the community benefits from safer streets and a reduced burden on emergency services. However, care must be taken to ensure that the system does not unfairly penalize users, as this could lead to resentment and a lack of trust, undermining the potential for fostering a compassionate community.

Chapter 5

Conclusion

5.1 Takeaways

Our project aims to extend the lifespan of shared e-scooters through an innovative onboard solution that discourages detrimental riding behaviors. We develop a comprehensive system that includes a portable sensor hub (SensorTile.Box Pro [11]) for data collection, an iOS app for data transmission, a cloud-based backend [12] for data analysis, and a web application for data visualization. This end-to-end solution captures ride data, processes it using a machine learning model, and provides actionable insights to ride-sharing service providers.

Throughout this project's development, we learned the importance of an integrated and adaptable development approach. Implementing Agile methodologies allowed us to manage the project's complexity effectively, accommodating evolving requirements and unforeseen challenges. We gained valuable experience in hardware-software integration, data pipeline automation, and machine learning model deployment. This project underscored the necessity of balancing technical constraints with user experience and business needs.

Our solution offers several advantages, including enhanced scooter lifespan by identifying and mitigating rough-riding behaviors, a comprehensive data pipeline that efficiently collects, processes, and visualizes data, and scalability through the use of cloud-based infrastructure. However, there are also disadvantages, such as dependency on specific hardware components (SensorTile.Box Pro [11]), reliance on user compliance for data collection via the iOS app, and initial setup costs that might be a barrier for smaller service providers.

5.2 Future Work

Based on our findings and the project's outcomes, we identified several areas for future work. We aim to develop a unified app that integrates scooter rentals, ride data collection, and server uploads to ensure a seamless user experience. Additionally, enhancing the system to automatically process data upon server upload will enable near-instant riding status visibility for service providers. Exploring the capabilities of alternative hardware from STMicroelectronics for data offloading, including potential Wi-Fi support and on-device machine learning analysis, will further improve

our technology stack. Developing more intelligent firmware for data collection and filtering, designing a custom physical enclosure, and implementing a secure offloading service mode will refine the SensorTile.Box Pro [11]. Lastly, improving the functionality of the web application for fleet monitoring will make it more intuitive and feature-rich for service providers.

In conclusion, our project successfully developed a robust solution to extend the lifespan of shared e-scooters and enhance their operational feasibility. By addressing both technical and user experience challenges, we provided a scalable and valuable tool for ride-sharing service providers. Future work will focus on refining the system, improving automation, and expanding its capabilities to further enhance its effectiveness and user adoption.

Chapter 6

Acknowledgments

We extend our sincere gratitude to our advisors, Dr. Behnam Dezfouli and Dr. Yuhong Liu, for their support and guidance throughout the course of this project. Their expertise and insights were invaluable in navigating the challenges and ensuring the project's success. We also extend our thanks to Dr. Jacquelyn Hendricks for guiding us during the thesis writing process.

We are deeply grateful to STMicroelectronics for providing the necessary equipment and sensors, including the SensorTileBox.Pro and related equipment. Their generous support and state-of-the-art technology were critical components in the development of our solution. Without their contribution, this project would not have been possible.

Additionally, we would like to thank Santa Clara University for offering us the opportunity to undertake this project. The resources and facilities provided by the School of Engineering were instrumental in our research and development efforts. We also appreciate the support and encouragement from the university staff, faculty, and our fellow students who contributed to our work through their feedback and collaboration.

Finally, we acknowledge the contributions of everyone involved in this project, whose collective efforts and dedication were crucial in bringing our vision to fruition. Thank you to all who supported us along this journey.

Bibliography

[1] D. Schellong, P. Sadek, C. Schaetzberger, and T. Barrack. "The promise and pitfalls of e-scooter sharing." (Jul. 2022), [Online]. Available: https://www.bcg.com/publications/2019/promise-pitfalls-e-scooter-sharing.

[2] S. Guidon, H. Becker, H. Dediu, and K. W. Axhausen, "Electric bicycle-sharing: A new competitor in the urban transportation market? an empirical analysis of transaction data," *Transportation Research Record*, vol. 2673, no. 4, pp. 15–26, 2019. DOI: 10.1177/0361198119836762.

[3] Y. Yao, L. Liu, Z. Guo, Z. Liu, and H. Zhou, "Experimental study on shared bike use behavior under bounded rational theory and credit supervision mechanism," *Sustainability*, vol. 11, no. 1, 2019, ISSN: 2071-1050. DOI: 10.3390/su11010127.

[4] H. Stigson, I. Malakuti, and M. Klingegård, "Electric scooters accidents: Analyses of two swedish accident data sets," *Accident Analysis & Prevention*, vol. 163, p. 106 466, 2021, ISSN: 0001-4575. DOI: 10.1016/j.aap.2021.106466.

[5] Rider Guide. "5 surprising reasons you're more likely to be injured riding an electric scooter than a bicycle." (Jul. 2023), [Online]. Available: https://riderguide.com/safety/5-surprising-reasons-youre-more-likely-to-be-injured-riding-an-electric-scooter-than-a-bicycle/ (visited on 06/01/2023).

[6] B. Lovely. "Smart scooter: Solving e-scooter safety problems with multi-modal, privacy-preserving sensor technology and machine learning." (Jun. 2022), [Online]. Available: https://www.researchgate.net/publication/244205780_httpdiva-portalorgsmashgetdiva2304715FULLTEXT02.

[7] Q. Ma, H. Yang, A. Mayhue, Y. Sun, Z. Huang, and Y. Ma, "E-scooter safety: The riding risk analysis based on mobile sensing data," *Accident Analysis & Prevention*, vol. 151, p. 105 954, 2021, ISSN: 0001-4575. DOI: 10.1016/j.aap.2020.105954.

[8] F. W. Siebert, M. Hoffknecht, F. Englert, T. Edwards, S. A. Useche, and M. Rötting, "Safety related behaviors and law adherence of shared e-scooter riders in germany," Lecture Notes in Computer Science, vol. 12791, H. Krömker, Ed., 2021. DOI: 10.1007/978-3-030-78369-0_5.

[9] S. Gilroy, D. Mullins, E. Jones, A. Parsi, and M. Glavin, "E-scooter rider detection and classification in dense urban environments," *Results in Engineering*, vol. 16, p. 100 677, 2022, ISSN: 2590-1230. DOI: 10.1016/j.rineng.2022.100677.

[10] S. Phadke, S. Ravala, J. Jerome, R. Batra, and M. Hussain. "Siotlab/e-scooter-black-box." Accessed: 2024-06-06. (n.d.), [Online]. Available: https://github.com/SIOTLAB/E-Scooter-Black-Box.

[11] The ST Blog. "Sensortilebox pro." Accessed: 2024-06-04. (Dec. 2023), [Online]. Available: https://blog.st.com/sensortilebox-pro/.

[12] Google. "Google." Accessed: 2024-06-05. (n.d.), [Online]. Available: https://firebase.google.com/docs/firestore.

[13] Google. "Google." Accessed: 2024-06-05. (n.d.), [Online]. Available: https://firebase.google.com/docs/functions.

[14] STMicroElectronics. "Stmicroelectronics/bluestsdk—ios: Bluetooth low energy sensors technology software development kit (iosversion)." Accessed: 2024-06-05. (n.d.), [Online]. Available: https://github.com/STMicroelectronics/BlueSTSDK%E2%80%94iOS.

[15] Apple Inc. “Swift resources - apple developer.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://developer.apple.com/swift/resources/.

[16] D. C. Gindi. “Chartsorg/charts: Beautiful charts for ios/tvos/osx! the apple side of the crossplatform mpandroidchart.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://github.com/ChartsOrg/Charts.

[17] J. Gessner. “Jonasgessner/jgprogresshud: An elegant and simple progress hud for ios and tvos, compatible with swift and objc.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://github.com/JonasGessner/JGProgressHUD.

[18] F. Nowotny. “Ybrid/opus-swift.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://github.com/ybrid/opus-swift.

[19] Google. “Google.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://firebase.google.com/docs/auth.

[20] Google. “Google.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://firebase.google.com/docs/app-check.

[21] “React.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://react.dev/reference/react.

[22] Remix Software Inc. “Home v6.22.0.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://reactrouter.com/en/main.

[23] “Boxicons usage.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://boxicons.com/usage.

[24] “Api — chart.js.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://www.chartjs.org/docs/latest/developers/api.html.

[25] J. e. a. Kurkela. “Chartjs-adapter-date-fns.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://github.com/chartjs/chartjs-adapter-date-fns.

[26] Google. “Google.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://firebase.google.com/docs/storage.

[27] S. He and K. G. Shin, “Dynamic flow distribution prediction for urban dockless e-scooter sharing reconfiguration,” in *Proceedings of The Web Conference 2020*, ser. WWW ’20, Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 133–143, ISBN: 9781450370233. DOI: 10.1145/3366423.3380101.

[28] “Guide to bluetooth security.” (), [Online]. Available: https://www.govinfo.gov/content/pkg/GOVPUB-C13-PURL-LPS121099/pdf/GOVPUB-C13-PURL-LPS121099.pdf.

[29] F. S. Cabral, H. Fukai, and S. Tamura, “Feature extraction methods proposed for speech recognition are effective on road condition monitoring using smartphone inertial sensors,” *Sensors*, vol. 19, p. 3481, 2019. DOI: 10.3390/s19163481.

[30] L. E. Peterson, “K-nearest neighbor,” *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009, revision #137311. DOI: 10.4249/scholarpedia.1883.

[31] C. Kingsford and S. Salzberg, “What are decision trees?” *Nature Biotechnology*, vol. 26, pp. 1011–1013, 2008. DOI: 10.1038/nbt0908-1011.

[32] G. Biau and E. Scornet, “A random forest guided tour,” *TEST*, vol. 25, no. 2, pp. 197–227, Jun. 2016, ISSN: 1863-8260. DOI: 10.1007/s11749-016-0481-7.

[33] A. Natekin and A. Knoll, “Gradient boosting machines, a tutorial,” *Frontiers in Neurorobotics*, vol. 7, p. 21, 2013. DOI: 10.3389/fnbot.2013.00021.

[34] M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan, “Ensemble deep learning: A review,” *Engineering Applications of Artificial Intelligence*, vol. 115, p. 105 151, 2022, ISSN: 0952-1976. DOI: 10.1016/j.engappai.2022.105151.

[35] cppreference.com. “C language - cppreference.com.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://en.cppreference.com/w/c/language.

[36] Python Software Foundation. “Welcome to python.org.” Accessed: 2024-06-05. (n.d.), [Online]. Available: https://www.python.org/.

[37] Contributors. "Scikit-learn: Machine learning in python." Accessed: 2024-06-05. (n.d.), [Online]. Available: https://scikit-learn.org/stable/.

[38] M. Tabatabaie and S. He, "Naturalistic e-scooter maneuver recognition with federated contrastive rider interaction learning," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 6, no. 4, Jan. 2023. DOI: 10.1145/3570345.

[39] O. Ozturk, D. O. Kaya, and A. Onan, "Comparing the accuracy of machine learning models for accelerometer-based movement measurements," *Gait & Posture*, vol. 106, S157–S158, 2023, ESMAC 2023 Abstracts, ISSN: 0966-6362. DOI: 10.1016/j.gaitpost.2023.07.190.

[40] P. Zhao, A. Li, P. Pilesjö, and A. Mansourian, "A machine learning based approach for predicting usage efficiency of shared e-scooters using vehicle availability data," *AGILE: GIScience Series*, vol. 3, p. 20, 2022. DOI: 10.5194/agile-giss-3-20-2022.

Chapter 7

Appendices

This section includes detailed source code explanations and technical details regarding each aspect of the proposed e-scooter Black Box solution [10].

7.1 Black Box Hardware

7.1.1 Detailed Hardware Specifications and Capabilities

Microcontroller STM32U585xx:

- Core: ARM® Cortex®-M33 CPU with TrustZone® and Floating Point Unit (FPU).
- Clock Speed: Up to 160 MHz.
- Performance:
  - 1.5 DMPIS/MHz (Drystone 2.1).
  - 651 CoreMark® (4.07 CoreMark®/MHz).
- Ultra-Low-Power: 300 nA Standby mode, 19.5 µA/MHz run mode.
- Memory:
  - Flash: 2 MB with ECC, including 512 KB with 100k cycles.
  - SRAM: Up to 786 KB with ECC (722 KB with ECC ON).
  - External Memory Interface: Supports SRAM, PSRAM, NOR, NAND, and FRAM.
  - Octo-SPI Memory Interfaces: Two.
- Security:
  - TrustZone®, securable I/Os, and peripherals.

  - Secure Boot and Firmware: Unique boot entry, secure hide protection area (HDP), Secure Firmware Installation and Update (TF-M).
  - Encryption: Two AES coprocessors with DPA resistance, Public Key Accelerator, HASH hardware accelerator.
  - Random Number Generator: True Random Number Generator (NIST SP800-90B compliant).

Clock Management

- Crystal Oscillators: 4-50 MHz and 32 kHz for RTC.
- Internal Oscillators: 16 MHz, low-power 32 kHz, and multispeed 100 kHz to 48 MHz.
- PLLs: Three for system clock, USB, audio, and ADC.
- Power Management: Embedded regulator (LDO), SMPS step-down converter, RTC with hardware calendar and calibration.

I/Os and Timers

- Up to 136 Fast I/Os: Most 5 V-tolerant, up to 14 with independent supply.
- Timers: 17 (including motor-control, general-purpose, low-power, and watchdogs).

Analog Peripherals

- 14-bit ADC: 2.5 Msps, resolution up to 16-bit.
- 12-bit ADC: 2.5 Msps, autonomous in Stop 2 mode.
- 12-bit DAC: Low-power sample and hold.
- Operational Amplifiers: Two with built-in PGA.
- Ultra-Low-Power Comparators: Two.

Communication Interfaces

- USB: Type-C / USB power delivery controller, USB OTG 2.0 full-speed.
- Serial Audio Interface (SAI): Two.
- I2C: Four (FM+ 1 Mbit/s, SMBus/PMBus).
- USARTs: Six (ISO 7816, LIN, IrDA, modem).

- SPI: Three (five with dual OCTOSPI in SPI mode).
- FDCAN: One.
- SDMMC Interface: Two.

DMA Controllers

- 16- and 4-channel, functional in Stop mode.

Graphic Features

- Chrom-ART Accelerator (DMA2D): Enhanced graphic content creation.
- Digital Camera Interface: One.
- Mathematical Co-Processor: CORDIC for trigonometric functions, FMAC (filter mathematical accelerator).

Connections and IOs

- LEDs: Four user LEDs (Green, Red, Yellow, Blue).
- Buttons/Switches: Four (User BT1, User BT2/boot0, Reset, Power switch).

System Clock

The SensorTile.box Pro system clock is driven by an internal or external oscillator, typically set to 80 MHz by default using the PLL clock driven by a 16 MHz external oscillator. The system clock can be boosted to 120 MHz for higher performance.

Serial and USB Ports

- UARTs: Four (UART4 is connected to JTAG/SWD for the console).
- USB: USB-C connector for PC host connection, supporting various device classes.

Motion and Environmental Sensors

- **Magnetometer (LIS2MDL):** Measures the strength and direction of magnetic fields, expressed in microteslas (μT). This sensor aids in determining the device's orientation relative to Earth's magnetic North, essential for navigation and heading determination.
- **6-Axis Accelerometer (LSM6DSV16X):** Combines a 3-axis accelerometer and a 3-axis gyroscope, providing a comprehensive six-dimensional analysis of movement and orientation. It measures the rate of change in velocity along three axes and detects orientation and tilt by observing gravitational effects, delivering data in meters per second squared (m/s^2).

- **Gyroscope (LSM6DSV16X):** Captures rotational motion by measuring the rate of rotation around the device's three axes—pitch, yaw, and roll. This sensor is crucial for understanding angular velocity, usually measured in degrees or radians per second, enhancing the precision of motion detection.
- **Other Sensors:**
  - Altimeter/Pressure Sensor (LPS22DF): Measures altitude and atmospheric pressure.
  - Humidity Sensor (HTS221): Measures relative humidity.
  - Digital Temperature Sensor (STTS22H): Measures ambient temperature.
  - Microphone/Audio Sensor (MP23db01HP): Captures audio data.

Programming and Debugging

DFU Software Tools

To flash the SensorTile.box Pro, use the DFU (Device Firmware Upgrade) mode. Press and hold the BOOT0 button while connecting the USB-C cable to enter DFU mode.

SWD Hardware Tools

Alternatively, use SWD (Serial Wire Debug) hardware tools such as ST-LINK/V3 for flashing and debugging.

DFU Flashing

1. Install dfu-util: Recommended version v0.9 or higher.
2. Enter DFU Mode: Press and hold the BOOT0 button, then connect the USB-C cable to force the board into DFU mode.

7.2 Software

7.2.1 Blackbox Gateway iOS Application

The Blackbox Gateway iOS Application is responsible for providing e-scooter users with a way to interact with the physical Black Box mounted on the rental e-scooter. The application creates a data transfer bridge from the Black Box sensors to our Google Firebase backend solution [12][26].

Figure 7.1: Blackbox Gateway iOS App File Structure (Left), XCode Screen Previews (Right)

The project file structure revolves around two primary folders, one for the application's individual screens (labeled 'View Controllers' in Figure 7.1 above) and another for custom visual interface elements (labeled 'Cocoa Touch' in Figure 7.1 above). Each file within the 'View Controllers' folder consists of code and User Interface (UI) objects, followed by methods dictating how the application screen should react when it is first loaded or interacted with (as shown in Figure 7.2 below). On the other hand, every file within the 'Cocoa Touch' folder contains code modifying an existing UI object defined within Swift's UIKit Library [15] for custom behaviors and appearances.

```swift
import UIKit
import STCore
import STBlueSDK
import CoreLocation
import JGProgressHUD

class HomeVC: UIViewController, CLLocationManagerDelegate, RideSummaryDelegate {

    let locationManager = CLLocationManager()

    var curLoc: CLLocationCoordinate2D?
    var locAllowed = false
    var box: Node?
    var buttonSearch = true
    var riding = false
    var timer = Timer()
    var seconds = 0

    @IBOutlet weak var statusIcon: UIImageView!
    @IBOutlet weak var statusText: UILabel!
    @IBOutlet weak var searchButton: UIButton!
    @IBOutlet weak var startButton: UIButton!
    @IBOutlet weak var elapsedLabel: UILabel!

    override func viewDidLoad() { ••• }

    deinit { ••• }

    func updateView(_ connected: Bool) { ••• }

    @IBAction func searchForDevices(_ sender: UIButton) { ••• }

    @IBAction func startPressed(_ sender: UIButton) { ••• }

    @objc func updateTimer() { ••• }

    @IBAction func profileButton(_ sender: UIBarButtonItem) { ••• }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) { ••• }

    func timeString(time: TimeInterval) -> String { ••• }

    // MARK: CLLocationManager Methods

    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) { ••• }

    func showSuccess() { ••• }
}
```

Figure 7.2: An example of a Swift ViewController file

7.2.2 Blackbox Gateway Web Application

This React-based website is specifically designed for ride-sharing companies to efficiently monitor and track user behavior and the utilization of e-scooters. The platform provides comprehensive analytics on how riders use the service, enabling businesses to optimize operations, improve user experiences, and manage their fleet of e-scooters effectively.

Website Structure

```jsx
const NavigationBar = () => {
  const navigate = useNavigate();
  const [isPopupVisible, setIsPopupVisible] = useState(false);


  const togglePopup = () => {
    setIsPopupVisible(!isPopupVisible);
  };

  return (
    <nav className="navigation-bar">
      <div className="top-right-sidebar">
        <button id="notificationButton" className="notification-btn" onClick={togglePopup}>
          <i className="bx bxs-bell bx-md"></i>
        </button>
        <button id="printButton" className="settings-btn" onClick={handlePrint}>
          <i className='bx bxs-printer bx-md'></i>
        </button>
      </div>
      {isPopupVisible && (
        <div className="popup">
          <div className="popup-content">
            <h2>Alerts</h2>
            <div className="notifications-header">
              <span>Messages:</span>
            </div>
            <div className="notification-item">Scooter #3245 is low battery!</div>
            <div className="notification-item">4 New users added this week!</div>
            <div className="notification-item">Increase of scooter usage in Santa Clara!</div>
            <button id="closePopupButton" onClick={togglePopup} className="close">Close</button>
          </div>
        </div>
      )}
    </nav>
  );
};
```

Figure 7.3: Navigation Bar source code

The components folder houses all front-end visual code for the web page. A key part of this folder is the NavigationBar component (as shown above in Figure 7.3), which utilizes React's useState to manage states for opening a popup bar. Within the NavigationBar's return block, there is HTML code that structures the popup notification button alongside the print button, which is linked to the handlePrint function (Figure 7.4 below).

```
const handlePrint = () => {
  // Attempt to force background graphics in print
  const css = '@page { size: landscape; } body { -webkit-print-color-adjust: exact; }',
      head = document.head || document.getElementsByTagName('head')[0],
      style = document.createElement('style');

  style.type = 'text/css';
  style.media = 'print';

  if (style.styleSheet){
    style.styleSheet.cssText = css;
  } else {
    style.appendChild(document.createTextNode(css));
  }

  head.appendChild(style);

  window.onafterprint = () => head.removeChild(style); // Clean up after print
  window.print();
};
```

Figure 7.4: handlePrint function source code

handlePrint function:

- **CSS Setup:** The handlePrint function initializes by setting a CSS style for printing in landscape mode. It ensures that the page style remains unchanged and that the colors printed match exactly what is displayed on the screen. This helps maintaining visual integrity in printed materials.
- **Style Sheet Creation and Validation:** The function dynamically creates a ¡style¿ element specifically for printing. It checks if the styleSheet property is supported (for compatibility with older browsers) and uses it if available; otherwise, it falls back to appending a text node containing the CSS string.
- **DOM Manipulation:** The newly created style element is appended to the document's ¡head¿, ensuring that the print styles are temporarily part of the document during the print process.
- **Print Execution and Cleanup:** The window.print() method triggers the print dialog. Post-printing, an onafterprint event handler removes the style element from the ¡head¿. This cleanup prevents the print styles from affecting the webpage's normal viewing state.
- **Component File Structure:** This file's structure aligns with other component files in the directory, each employing similar but slightly varied hooks like useEffect and useState to handle different functionalities. This standardized but flexible structure allows for efficient management of UI components and their interactions within the application.

Data Retrieval

The primary function for fetching data, ‘fetchRides’ (Figure 7.5), is located in the utilities folder within the webpage directory on GitHub. The process of retrieving data from Firebase involves several steps due to its multi-level collection structure. Initially, the function attempts to access the ”Scooters” collection to fetch its documents. Subsequently, for each document in this collection, it accesses the corresponding ”Users” collection, navigates to each user’s ”Rides” collection, and retrieves each ride document. From these documents, data is extracted and formatted appropriately.

```
async function FetchRides() {
    // Create an object to store ride data grouped by user name
    const rideDataByUser = {};

    try {
        // Create a query to fetch all scooter IDs
        const scooterIdsCollection = query(
            collection(db, 'Scooters')
        );

        // Retrieve the documents from the collection
        const querySnapshot = await getDocs(scooterIdsCollection);

        // Loop through each scooter ID document
        for (const scooterDoc of querySnapshot.docs) {
            const scooterId = scooterDoc.id;

            // For each scooter ID, retrieve the users collection
            const usersCollectionPath = `Scooters/${scooterId}/users`;
            const usersCollection = query(
                collection(db, usersCollectionPath)
            );
            const usersSnapshot = await getDocs(usersCollection);

            // Loop through each user document
            for (const userDoc of usersSnapshot.docs) {
                const userId = userDoc.id;
                const userData = userDoc.data();
                const userName = userData.name;

                // For each user, retrieve the rides collection
                const ridesCollectionPath = `Scooters/${scooterId}/users/${userId}/rides`;
                const ridesCollection = query(
                    collection(db, ridesCollectionPath)
                );
                const ridesSnapshot = await getDocs(ridesCollection);

                // Loop through each ride document
                for (const rideDoc of ridesSnapshot.docs) {
                    // Here you can access each ride document and process it as needed
                    const rideData = rideDoc.data();

                    // Add the user's name and scooter ID to the ride data
                    rideData.escooter = scooterId;
                    if (isValidRideData(rideData)) {
                        if (!rideDataByUser[userName]) {
                            rideDataByUser[userName] = [];
                        }
                        rideDataByUser[userName].push(rideData);
                    } else {
                        console.log("Invalid ride data skipped", rideData);
                    }
                }
            }
        }
        // Now rideDataByUser contains ride data grouped by user name
        console.log(rideDataByUser); // Log the ride data grouped by user name
        return rideDataByUser;
    } catch (error) {
        console.error("Error fetching ride data:", error);
        return null; // Return null in case of error
    }
`
```

Figure 7.5: The ‘FetchRides’ function from the React source code

This method of data extraction is just one approach and can vary based on the specific data requirements. For instance, if the focus is solely on tracking e-scooter data rather than user data, the user-related information would be irrelevant, and a slightly different data retrieval strategy might be implemented.

7.2.3 Google Firebase

Google Firebase serves as a central data hub for the Blackbox Gateway iOS and web applications, providing essential tools for data storage and management.

Cloud Firestore

Cloud Firestore [12] organizes data hierarchically, using collections and documents. In this structure, a 'Users' collection contains individual user documents, and each user document can include a nested 'Rides' collection. Each ride in this collection holds documents with metadata that are pushed by the Blackbox Gateway iOS application upon ride completion, which are then fetched and displayed by the Blackbox Gateway Web Application. Each of these collections reside within a top level folder indicating the ID of the rental e-scooter in question, such as '3245'. A visual representation of the database structure can be seen below in Figure 7.6 below.

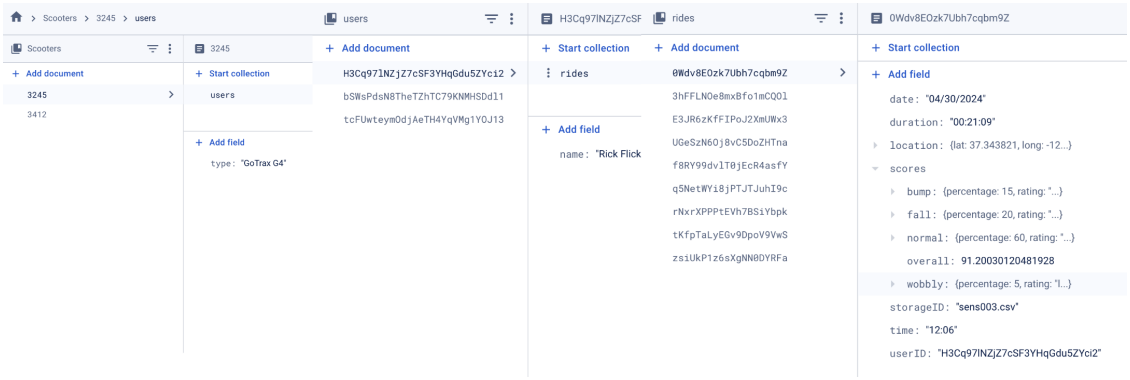

Figure 7.6: The database organization of Cloud Firestore

The information stored in Cloud Firestore [12] includes information regarding individual e-scooter rental rides. This consists of ride duration, start time, ending location, ride score information (produced by the Ensemble ML model) and a storageID corresponding to the sensory data file residing in Cloud Storage [26].

Once a new document is created within the Firestore Database by the Blackbox Gateway iOS App, a custom written Python [36] function is run to automatically analyze the raw ride sensory data and update the Firestore document with the assessed score categories.

Cloud Functions

Cloud Functions [13] enables the execution of custom written code without the need to handle servers or containers. For the e-scooter Black Box, this provided seamless integration within the Firebase tool suite, and enabling automatic ride analysis without the need for external intervention.

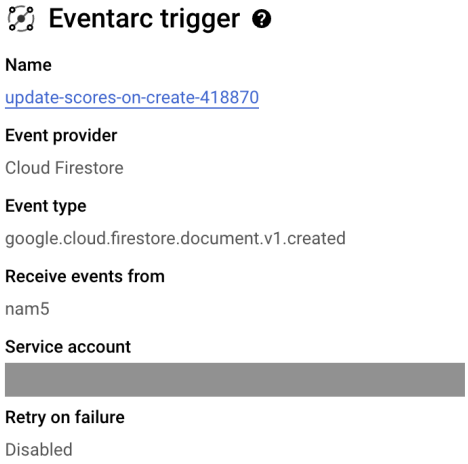

Figure 7.7: Trigger details for the Ensemble ML Model's automatic execution via Cloud Functions

As shown above in Figure 7.7, the 'google.cloud.firestore.document.v1.created' event type allows the automatic execution of custom written Python [36] code upon data upload by the Blackbox Gateway iOS application to Cloud Firestore [12]. Once the the ride document is created by the Blackbox Gateway iOS application, a storageID and rental ride metadata is uploaded, while a raw sensor data file with the name corresponding to storageID (highlighted in blue within Figure 7.8 below) is uploaded to Cloud Storage [26]. This update triggers the Ensemble ML model analysis - once this has completed, the 'scores' field is created with analysis regarding each of the riding benchmark categories, as highlighted in red within Figure 7.8 below.

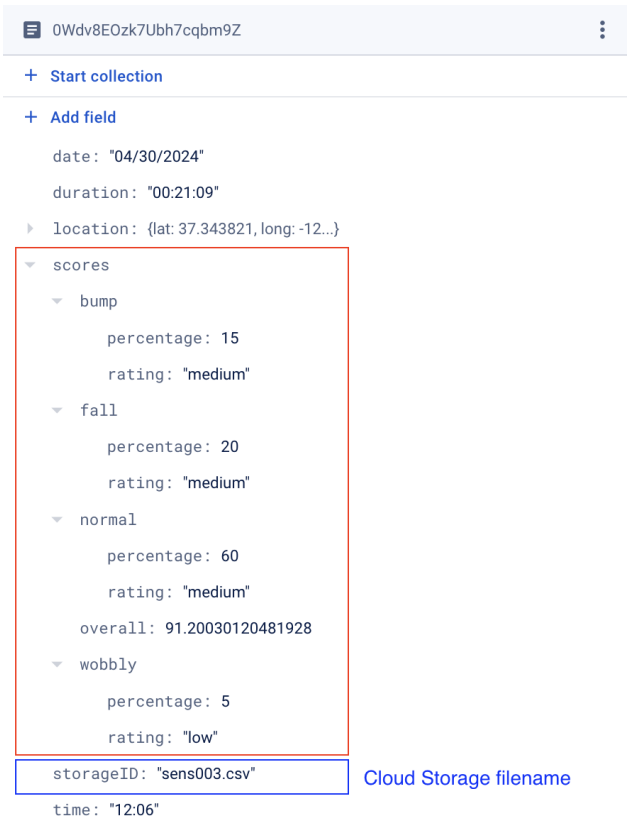

Figure 7.8: Cloud Firestore data as updated by Cloud Functions

Authentication

Firebase Authentication [19] provides backend services and tools to authenticate users to both mobile and web applications, with support for password, phone number authentication in addition to login via external identity providers.

The e-scooter Black Box solution utilizes Authentication [19] in a twofold manner: 1) to allow e-scooter rental service providers to log into the Blackbox Gateway web application to view their fleet usage behaviors and related metrics, and 2) to allow users to login to the Blackbox Gateway iOS application to log their rental rides and connect with the physical Black Box onboard the rental e-scooter in use.

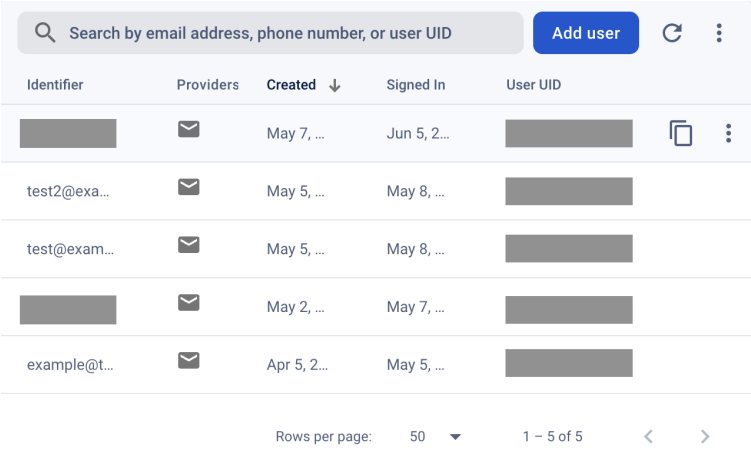

Figure 7.9: Firebase Authentication [19] Administrative Console

As shown in Figure 7.9 above, a unique User ID (UID) is created for each user registered with Firebase Authentication. This UID is used to navigate the file paths inside of Cloud Firestore [12] and Cloud Functions [13], ensuring only secure, authorized access to user data.

7.3 Data Processing

Splitting Up the Dataset

With the team having roughly 20 hours of data to work with, as described earlier in the paper the team split up the data into chunks of three seconds each to get feature extracted This allows accurate classification and idenrification of small time movements such as sudden shifts and turns. Below is the code in Python for those shifts. The code beloww takes in the entire csv file, uses the min and max time given in the file and accurately splits it into chunks of 300 data points(3 seconds) .

```
def feature_extraction_file(directory, filename):
    feature_extraction_matrix = []
    labels = []
    data = pd.read_csv(os.path.join(directory, filename))

    chunk_size_ms = 300
    min_time = data['Time [mS]'].min()
    max_time = data['Time [mS]'].max()
    num_chunks = ((max_time - min_time) // chunk_size_ms) + 1

    for i in range(int(num_chunks)):
        start_time = min_time + (i * chunk_size_ms)
        end_time = start_time + chunk_size_ms
        chunk_data = data[(data['Time [mS]'] >= start_time) & (data['Time [mS]'] < end_time)]

        features, label = feature_extraction(filename, chunk_data)
        feature_extraction_matrix.append(features)
        labels.append(label)
```

Figure 7.10: Chunking the Data into 3 second pieces

Feature Extraction

Once the data is split into three second chunks, the data is then sent to get feature extracted. A snipped of the feature extraction is shown below. It goes through each column, or sensor, and then calculates statistical features, spectral features and time features. Those features are then put into a matrix and sent over to the next part of the algorithm to get trained.

```
for column in columns:
    if 'Acc' in column:
        signal = data[column] / 1000.0
    elif 'Gyro' in column:
        signal = data[column] / 1000.0
    else:
        signal = data[column]

    f, Pxx = welch(signal, fs=fs, nperseg=300)
    centroid = np.sum(f * Pxx) / np.sum(Pxx)
    # entropy = -np.sum(Pxx * np.log2(Pxx))
    time_features = {
        'start_time': data['Time [mS]'].min(),
        'end_time': data['Time [mS]'].max()
    }
    spectral_features = {
        'centroid': centroid,
        'bandwidth': np.sqrt(np.sum((f - centroid)**2 * Pxx) / np.sum(Pxx)),
        'flatness': np.exp(sp_entropy(Pxx)) / np.mean(Pxx),
        'rolloff': f[np.cumsum(Pxx) >= 0.85 * np.sum(Pxx)][0]
    }

    statistical_features = {
        'mean': np.mean(signal),
        'variance': np.var(signal),
        'stddev': np.std(signal),
        'peak_to_peak': np.ptp(signal),
        'rms': np.sqrt(np.mean(np.square(signal))),
        'skewness': skew(signal),
        'kurtosis': kurtosis(signal),
        'zcr': ((signal[:-1] * signal[1:]) < 0).sum()
    }

    spectral_features_df = pd.DataFrame([spectral_features]).fillna(0)
    statistical_features_df = pd.DataFrame([statistical_features]).fillna(0)
    time_features_df = pd.DataFrame([time_features]).fillna(0)

    clean_spectral_features = spectral_features_df.iloc[0].to_dict()
    clean_statistical_features = statistical_features_df.iloc[0].to_dict()
    clean_time_features = time_features_df.iloc[0].to_dict()

    sensor_name = column.split(' ')[0][:-1]
    features[sensor_name] = {**clean_spectral_features, **clean_statistical_features, **clean_time_features}
```

Figure 7.11: Feature Extraction of the 3 second chunks

Prediction

Once the different models in our ensemble model are trained effectively using the 80-20 train-test split, we then attempt to predict the results to see the accuracy of the ensemble model. First, we store the model into its own joblib file using the joblib library. In order to predict, we utilize the joblib and tennsorflow's libraries where it gives us access to the function load() and the model.predict() in order to predict the outcome the code.

```
def load_and_predict(model_file, new_data):
    model = load(model_file)
    predictions = model.predict(new_data)
    return predictions
```

Figure 7.12: Load and Predict Code to Predict Outcome

Model Breakdown Analysis

Once the model predictions are all ran and created, it is then passed into the breakdown function in which it generates the predictions of all four of the models and takes the mode of the four for each time period to ensure the best accuracy. It also make sure to keep track of individual accuracies and percentages in order to be used later in the algorithm and displayed on the front end.

```
def rough_riding_breakdown(knn, dt, ann, lr, svm, nb, rf, gb):
    breakdown = []
    score_val = 0.0
    for i in range(len(knn)):
        predictions = [dt[i], rf[i], knn[i], gb[i]]
        breakdown.append(mode(predictions))
    print(breakdown)
    i = 0
    model_list = [knn, dt, ann, lr, svm, nb, rf, gb, breakdown]
    model_names = ['knn', 'dt', 'ann', 'lr', 'svm', 'nb', 'rf', 'gb','breakdown']

    for model in model_list:
        label_counts = Counter(model)
        total_items = len(model)

        print(f"Percentage breakdown of each label: {model_names[i]} \n")
        i+=1
        for label, count in label_counts.items():
            percentage = (count / total_items) * 100
            print(f"{label}: {percentage:.2f}%")
    label_counts = Counter(breakdown)
    total_items = len(breakdown)
    breakdown_percentages = {}
    print(f"Percentage breakdown of each label: breakdown\n")
    for label, count in label_counts.items():
        percentage = (count / total_items) * 100
        print(f"{label}: {percentage:.2f}%")
        breakdown_percentages[label] = percentage

    score_val = score_val/len(breakdown)
    return breakdown_percentages
```

Figure 7.13: Breakdown of each Model's Accuracy

Score Generation

After generating the breakdown for each of the classifications, the function goes in and generates both a rating(low, medium, high) of how risky the rough riding was alongside a percentage for how often the rough riding happened to display on the front-end.

```
def score_generator(breakdown):
    breakdown_percentages = post_processing(breakdown)
    benchmark_scores = {'normal': 79, 'fall': 7, 'wobbly': 3, 'bump': 11}
    deduction = 0
    scores = {}
    for label in breakdown_percentages:
        scores[label] = {}
        scores[label]['percentage'] = breakdown_percentages[label]

        if breakdown_percentages[label] < (0.7 * benchmark_scores[label]):
            scores[label]['rating'] = 'low'
            if label == 'normal':
                deduction+=2
            else:
                deduction-=2
        elif breakdown_percentages[label] > (1.3 * benchmark_scores[label]):
            scores[label]['rating'] = 'high'
            if label == 'normal':
                deduction-=2
            else:
                deduction+=2
        else:
            scores[label]['rating'] = 'medium'

    scores['overall'] = (breakdown_percentages['normal'] - deduction)
    return scores
```

Figure 7.14: Generating scores for the classifications