Research Practicum Project Report

Predicting Dublin Bus Journey Times

Paul Reidy, Andrew Hughes, Alvaro Garcia

A thesis submitted in part fulfilment of the degree of

MSc. in Computer Science (Conversion)

Group Number: 13

COMP 47360

UCD School of Computer Science

University College Dublin

August 23, 2021

Project Specification

Bus companies produce schedules which contain generic travel times. For example, in the Dublin Bus Schedule, the estimated travel time from Dun Laoghaire to the Phoenix Park is 61 minutes (http://dublinbus.ie/Your-Journey1/Timetables/All-Timetables/46a-1/). Of course, there are many variables which determine how long the actual journey will take. Traffic conditions which are affected by the time of day, the day of the week, the month of the year and the weather play an important role in determining how long the journey will take. These factors along with the dynamic nature of the events on the road network make it difficult to efficiently plan trips on public transport modes which interact with other traffic.

This project involves analysing historic Dublin Bus data and weather data in order to create dynamic travel time estimates. Based on data analysis of historic Dublin Bus data, a system which when presented with any bus route, departure time, the day of the week, current weather condition, produces an accurate estimate of travel time for the complete route and sections of the route.

Users should be able to interact with the system via a web-based interface which is optimised for mobile devices. When presented with any bus route, an origin stop and a destination stop, a time, a day of the week, current weather, the system should produce and display via the interface an accurate estimate of travel time for the selected journey.

Abstract

This report describes a new application for Dublin Bus that aims to provide more accurate travel time estimates than those provided by static timetables. We describe how we innovate beyond the project specification by also estimating and presenting the uncertainty around these time estimates in a novel way and by providing extensive user authentication and customisation options. The uncertainty estimates are based on a Neural Network model but we find that this model does not outperform simpler benchmark models on average, although there is significant variation in model accuracy across the bus network. We also conduct a small user survey and find that while users usually prefer time ranges to single point estimates, they struggle to interpret and find value in more complex uncertainty visualisations. The report details the application, development and testing process and results, major contributions of each team member, weaknesses of the application, and future work.

Application Link: https://mystifying-swirles-3ffb35.netlify.app
GitHub: https://github.com/Botazio/UCD-DublinBus

Acknowledgments

We would like to thank our mentors, Gavin McArdle, Madhusanka Liyanage, Félix Balado and Pádraig Cunningham and demonstrator Laura Dunne for their support and assistance throughout the implementation and writing of this Research Practicum. We would also like to thank our family and friends who took part in our survey as well as for their patience and sympathetic ear.

Table of Contents

Chapter 1: **Introduction**

As cities and populations expand across the world, public transport plays an increasingly important role in offering accessible, convenient, affordable, and environmentally-friendly transit options. Providing reliable and accurate travel time estimates is a major challenge, however. Static timetables cannot account for changes in travel time caused by factors such as weather or traffic and improved accuracy is a frequently requested improvement in these services. In a survey of Dutch travellers, Grotenhuis et al. [8], for example, find that more accurate travel time estimates are among the most desired improvements across all age groups, particularly in the planning stage before a journey has begun. Lam and Small [13] show that commuters in California place significant monetary value on more reliable travel time estimates.

Dublin Bus is the largest and most popular public transport service in Dublin with over 148 million passengers using bus services in Dublin in 2019, an increase of 7.5% from 2018 [14]. There appears to be relatively high user dissatisfaction with the current web and mobile applications for the service. The iOS app, for example, has an average rating of 2 out of 5 stars [1]. Although many of these reviews are several years old and dissatisfied users may be more likely to leave a review, many of the complaints relate to the inaccurate bus arrival times or unexpected delays in travel times.

This project aims to develop a user-friendly web and mobile application for Dublin Bus that addresses these unmet needs. In addition to the core functionality that a user expects in a bus transport application, we aim to provide more accurate travel time estimates that can account for weather, traffic, and temporal factors. We also innovate beyond the project specification by calculating and presenting the uncertainty for our travel time estimates in a novel way. In addition, we provided extensive user authentication and account customisation options.

The remainder of this report is structured as follows. Chapter 2 provides detailed descriptions and screenshots of our main features and innovations. Chapter 3 describes how we worked as team in the project and the changes we made to the proposed team structure. We detail our technical architecture and the rationale for our choices in Chapter 4. In Chapter 5 we discuss the testing we conducted on different aspects of our project and the results. Chapters 6-8 are individual chapters written by each team member separately. Chapter 6 describes the main contribution of each team member and Chapter 7 is a more detailed literature review. We each conclude with a discussion of limitations and future work in Chapter 8.

Chapter 2: **Description of Final Product**

In this chapter we outline our application and how it meets and innovates on the problem specification. Instead of implementing every possible feature we could think of, we tried to restrict ourselves to what we consider the core functionality of the application plus two key innovations.

2.1 Journey Planner

This feature allows the user to plan a trip on a particular line within the next 7 days and to get accurate travel time estimates for the trip given the time of day and weather conditions. The predictions are based on machine learning models we trained a dataset of 2018 Dublin Bus data instead of relying on an external API such as those provided by Google Maps.

A screenshot of this feature is shown in Figure 2.1. When the user clicks on the Planner section a search bar is displayed for the lines in the network. All search bars in this application give suggestions while typing. After choosing a line, the user can search for a departure and destination stop on this line and finally choose whether the to journey will take place now or some time in the next 7 days. A search bar at the bottom of the screen is activated once these fields are filled out.

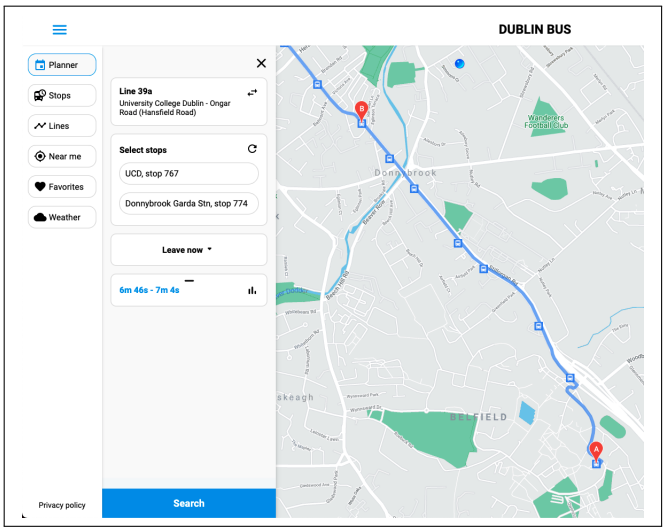

Figure 2.1: Journey Planner

The search results display an estimated range of travel times for this journey rather than a single estimate. The lower and upper bounds are the 20th and the 80th percentile of travel time estimates for that journey, respectively. The user can expand these results to get a breakdown for all of the stops on the journey. Finally, if the user clicks on the graph icon next to the travel time estimates a more detailed graph will be displayed to show the uncertainty estimates for the travel time. This will be discussed in more detail in Section 2.7.1.

If the user is logged in, a popup may appear after a prediction has been presented asking for a rating out of 5 for this feature. The intention of this feature is to continuously gather anonymous

feedback on the accuracy of the predictions and to allow the developers to investigate routes with unusually low ratings.

2.2 Stops and Lines Search

The Stops section allows the user to search for any stop in the network by name and/or number. To avoid a cluttered user interface on the map, we implemented a dynamic clustering approach to gather bus stops into groups depending on the zoom level as shown in Figure 2.2. The real-time bus arrivals for the chosen stop are displayed and map view is centered on it. The user can also click on one of the stops displayed on the map without searching to get the same information. An option icon is displayed beside the search bar which allows the user to hide the clusters and bus stops icons.

There is a similar search feature for Lines. After the user searches for a line, a box with information about that line is displayed and the line is displayed on the map. These lines are drawn based on the geolocation data from Dublin Bus. Clicking on the clock icon will open the static timetable for that line on the Dublin Bus website in a new tab.

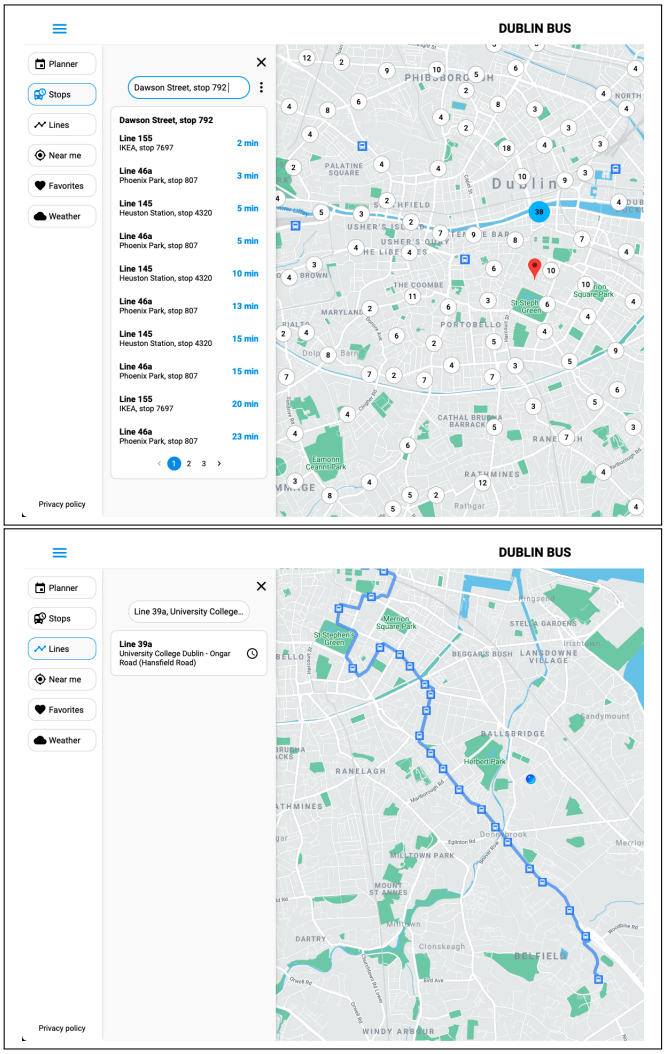

Figure 2.2: Stops Search (top) and Lines Search (bottom)

2.3 Near Me

When a user opens the application for the first time, they are prompted to share their location. If they agree to this a marker will be displayed at their location and when this user clicks on the Near Me section a search for nearby stops will be performed and these stops will be shown on the map as in Figure 2.3. The user can customise both the number of stops to display and the maximum distance to search for nearby stops using the options icon next to the search bar. This section will display an appropriate message if there are no stops close to the user or if they have not agreed to share their location.

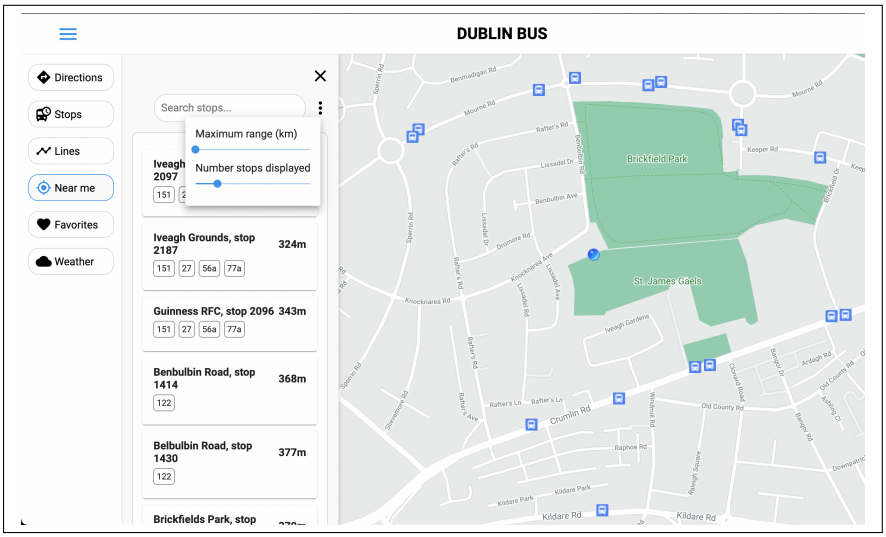

Figure 2.3: Near Me

2.4 Favorites

Users who have signed up for an account can mark stops and lines as favorites. This section allows these users to search through these stops and lines and plot them on the map as shown in 2.4. Users who are not signed in or have not saved any favorites are shown an error message.

2.5 Weather

This section displays the weather in Dublin. The user can browse through a general weather forecast for the next 7 days, a more detailed weather forecast for the next 24 hours, a chart of the next 7 days humidity and temperature, and, finally, today's UV index, sunrise, sunset and humidity.

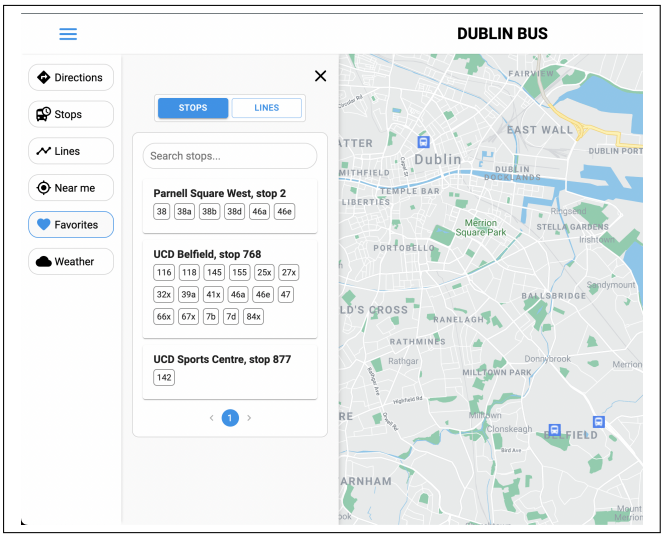

Figure 2.4: Favorites

2.6 Mobile Friendly

The problem specification requires that the web application is optimised for mobile devices. Based on previous experience we knew that it was important to consider the mobile design from the beginning rather than trying to adjust the styles later on in the project which would likely be much more error-prone. All of our new frontend features were tested and demonstrated on both desktop and mobile devices before they were considered completed. Figure 2.5 show the mobile layout of menu bar and the Weather page as an example. The active section is shown at the top of the screen on mobile and the user can switch between the menu and the map using the toggle below this title.

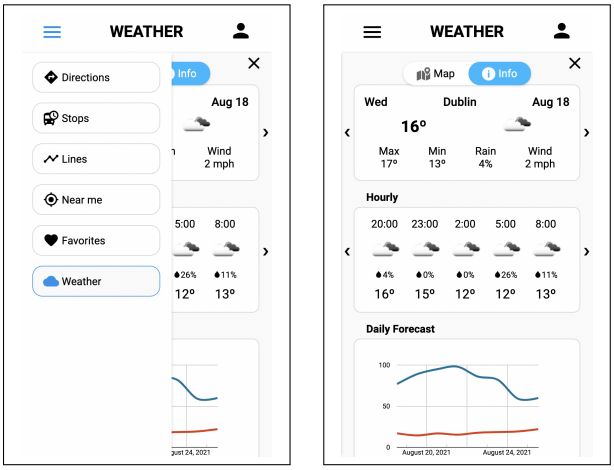

Figure 2.5: Mobile Views (iPhone 6)

2.7 Main Innovations

We consider the above as core features in our application to meet the project specification. In this section we highlight our two main innovations that go beyond the problem statement.

2.7.1 Travel Time Uncertainty and Quantile Dotplots

Our main area of innovation is to estimate and intuitively present the uncertainty around our travel time estimates rather than providing a single estimate for a journey. We chose this for several reasons. First, we were very impressed by the visualisations and results in Kay et al. [10] which was introduced to us at the beginning of the research practicum. Second, we had never seen a travel application that attempted to do this and one of the project mentors confirmed that he had never seen a team successfully implement this. Third, the feedback and reviews on the current Dublin Bus app highlight that the false precision of estimates is a major source of dissatisfaction [1]. Finally, we thought that estimating uncertainty is both an interesting technical challenge and a highly-relevant policy issue as we continue to deploy increasingly powerful machine learning systems in our society and try to understand their limitations and reliability ([4], [12]).

In standard machine learning (ML) models, the trained model will give the same output for a given set of inputs. Our first challenge was therefore to come up with a way to get a range of estimates from the same model given the same input. We intended to closely follow the methodology in Kay et al. [10] but we found it much more difficult than expected to understand exactly how they came up with their uncertainty estimates, even after reviewing the code on their GitHub repository [11]. We realised that the focus of that paper was on the presentation and user interface of the uncertainty estimates rather than on the technical details of the machine learning models.

We therefore began to look for some alternatives and found an influential paper by Gal and Ghahramani [7] which describes a relatively straightforward way of deriving uncertainty estimates from a Neural Network by using a technique called 'Dropout'. This involves randomly turning off some of the nodes in the network to produce slightly different results for the same inputs. We use this to produce a range of estimates for the same inputs for a given journey. One downside of this approach was that it meant that we were required to use a Neural Network even if we felt that other model types might be more suited to the travel time prediction problem overall. However, we were encouraged by the finding from Reich et al. [16] that simple neural networks have generally shown the best performance on prediction tasks in this area.

Once we had this range of estimates for a given journey, our next task was to visualise this data in an intuitive way. Kay et al. [10] show that most users are more successful at interpreting discrete outcomes as probabilities instead of density curves. We therefore follow their approach of transforming the density curves we estimated into "Quantile Dotplots" as shown in Figure 2.6. We also show a detailed explanation of how to interpret the graph which users can toggle by clicking the question mark icon.

To access the Quantile Dotplot users must click on the graph icon next to the journey time estimate circled in red in Figure 2.7. We chose not the present the graph in the main results to avoid cluttering the UI and because Kay et al. [10] find that many users are not interested in this information.

When designing the Quantile Dotplots, we also faced the same trade-off between precision and interpretability or 'glanceability' as in Kay et al. [10]. Increasing the number of dots in the graph to 100, for example, would allow it to more accurately mirror the underlying probability distribution. However, it would also make it harder for users to count the number of dots before a particular point. In the end, we decided to follow the finding in Kay et al. [10] that 20 dots provides a good balance between precision and interpretability. A second important parameter was the number of predictions requested from the trained Neural Network. For example, asking for 100 different predictions for each adjacent stop pair in a journey might allow us to more accurately represent the probability density curve for that journey but at the cost of longer computation times. After some testing we settled on requesting 10 predictions for each adjacent stop pair for short journeys and 5 predictions for longer journeys.

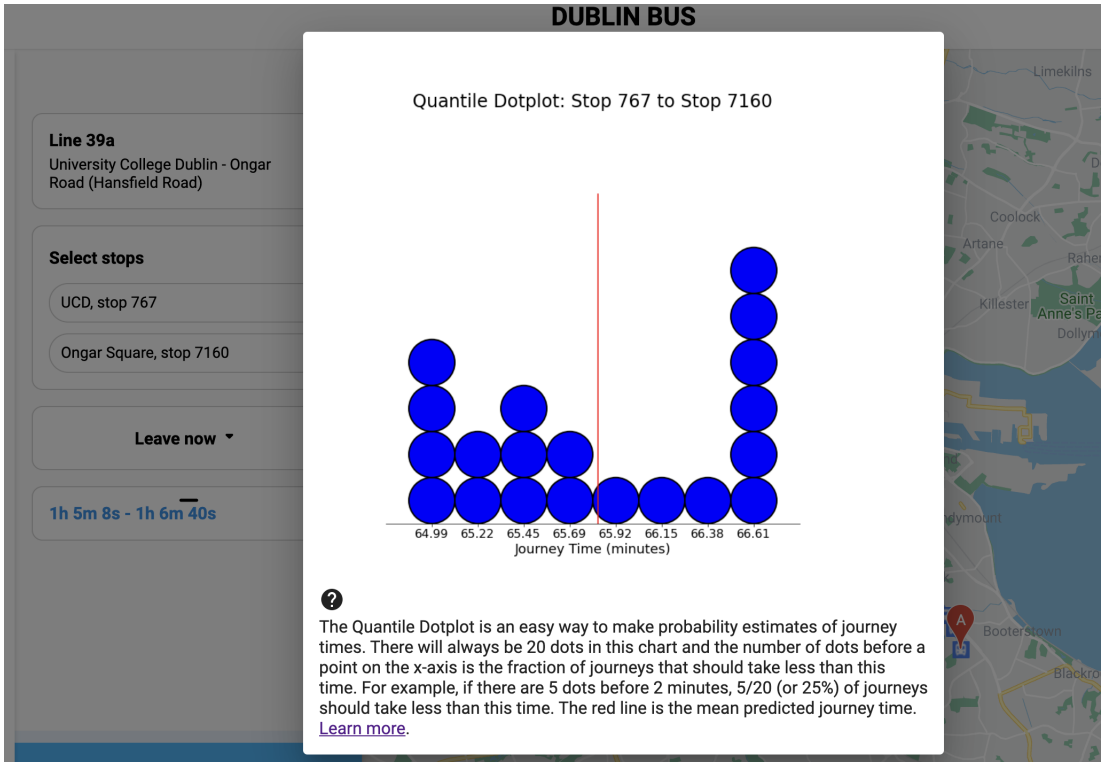

Figure 2.6: Quantile Dotplot

Overall, we believe that this approach of combining Neural Network Dropout with Quantile Dotplots is a novel and interesting contribution to the travel time estimation literature. In addition, Kay et al. [10] focused on uncertainty estimates for when a bus would arrive at a stop whereas we have concentrated on uncertainty around travel times between stops.

2.7.2 Authentication and User Customisation

Our second area of innovation is to provide extensive user authentication and customisation features. Clicking on the user icon in the top right-hand corner of the screen brings the user to the sign-in page. Users can sign up using an e-mail address or through their Google or Facebook accounts. The user can stay logged in for up to 24 hours on the same browser. The top bar icon changes to a custom photo or to a circle with the user initial letter in it when the user is logged in. When the user clicks this active icon a popup appears allowing the user to log out or to navigate to the user settings page. The user can customise several aspects of the application using the tabs in the settings page as shown in Figure 2.8:

- Profile: change profile details such as photo, username, email or delete the account
- Appearance: customise the application or map theme using a predefined or a custom theme
- Favourite: add or remove favourite stops or lines as shown in Section 2.4
- Markers: customise which useful markers are shown on the map (e.g., parks, hospitals)
- Feedback: provide feedback in a free-text form and toggle feedback alerts which occasionally pop up asking for a rating for a feature.

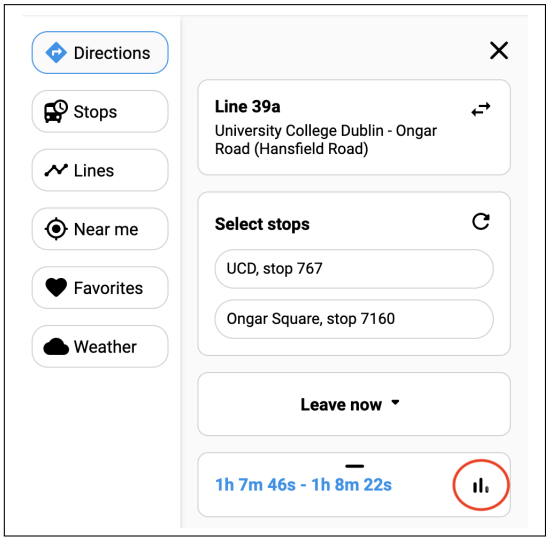

Figure 2.7: Icon to Access the Graph (bottom, circled in red)

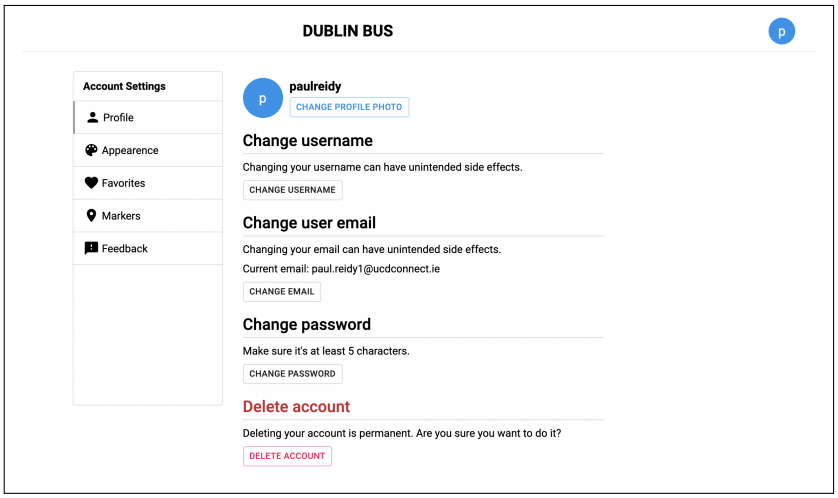

Figure 2.8: User Settings Page

Chapter 3: **Development Approach**

3.1 Agile

We had all worked under an Agile methodology in a previous group project for COMP30830 and agreed to follow this process again because we found the structure it provides very helpful and did not want to risk trying an entirely new or unstructured approach. We used many of the main aspects of the Agile methodology such as breaking up the project into a series of 2 week sprints, creating an MVP, holding daily stand-up meetings, and allocating work as tickets on a JIRA board. We also took quite detailed notes from our stand-up meetings each day so that we would have a log of issues, discussions, and decisions which we expected to be useful for completing the reports at the end.

However, we did depart from the standard Agile approach in some areas based on lessons we had learned from our previous work. For example, we did not try to assign 'story points' to the tasks to estimate how long each would take because we had previously found that this often became guess work and was not worth the time and effort it required to maintain. We also did not hold any regular retrospective, sprint planning, or backlog refinement meetings. Instead, we discussed any planning or other issues on the daily stand-ups, at the beginning of the sprint, or in ad-hoc meetings. Our goal with this was to reduce the number of meetings and allow us to spend more time developing features. We also felt that a team of only 3 members would not require as many meetings as larger teams to communicate effectively. This modified approach seemed to work in general but we may have underestimated the risk of skipping these meetings and more regular formal planning and prioritisation meetings in particular would likely have been very useful to adjust timelines and decide which proposed features to drop.

3.2 Team Roles

There were four proposed team roles at the beginning of the project but since we only had three members in our team we had to discuss how to divide these roles in our initial meetings. All of our team members had chosen Code Lead as their first preference. We therefore agreed that it seemed more natural to us to split up into backend, frontend and data-analytics roles like in industry teams so that we would all get a chance to contribute significant amounts of code. However, we knew that there was a risk that this would lead us to neglect the important tasks that are covered by the Customer and Coordination Lead roles, for example. To avoid this, we agreed to rotate the Coordination Lead every sprint and that all team members should contribute to the Customer Lead role. Overall, we feel that these changes suited us better than the proposed team roles. However, we did not actually rotate the Coordination Lead as planned and we feel that this role may have been neglected by the team and we could have benefited from being more deliberate about this.

To ensure all team members kept up to date with other parts of the project, we also required that everyone read and approve code before it was committed to our main branch. This worked well initially but it became increasingly difficult and time-consuming for team members to review

other parts of the project as the scale and complexity of the code grew. When we were under time pressure at the end of the project these approvals often became a formality instead of serving their intended purpose of improving code quality and team understanding. We may have reduced this problem by making smaller changes which were easier to review but it is difficult to know how we could have avoided this problem entirely given the tight deadlines for the project.

3.3 Conflict Resolution

As outlined in our Team Agreement, our first approach to resolving conflict was to try to discuss it as a team with the person who felt most strongly about the issue making their case for it and the other two members responding in turn. We all tried to raise any issues to the team for discussion as quickly as possible rather than leaving them unaddressed for long periods.

If it was not possible to come to an agreement in this way we resorted to a vote. It was useful in this case to have only three members on the team because it ruled out tied votes. The main area of conflict in our project where this voting process was used was in the decision on what features to implement and how to prioritise them. We each researched some innovative feature ideas and presented their advantages and disadvantages to the team before voting on which seemed most exciting and realistic to us and used these results to rank the features in order of priority. We found this was more useful than trying to imagine specific customers or use cases and building feature requirements around this perspective. We used the same voting procedure towards the end of the project when deciding which proposed features to drop.

Overall, then, we seemed to manage conflict reasonably well and we did not have to resort to the more serious conflict resolution strategies in our Team Agreement. We all tried to be flexible if other team members had other commitments outside of the project or if particular tasks were taking longer than expected by adjusting the work allocation among team members rather than complaining.

Chapter 4: **Technical Approach**

In this section we discuss the most important features in our technical architecture and approach. Figure 4.1 shows an overview of our technical architecture. We decided to keep our frontend and backend separate so they could easily run on different services and communicate with each other over HTTPS. Our Neural Network architecture and uncertainty estimates required more computational power than was available on the server provided by UCD so we hosted this backend service on a separate, high-performance machine provided by Google Cloud Platform.

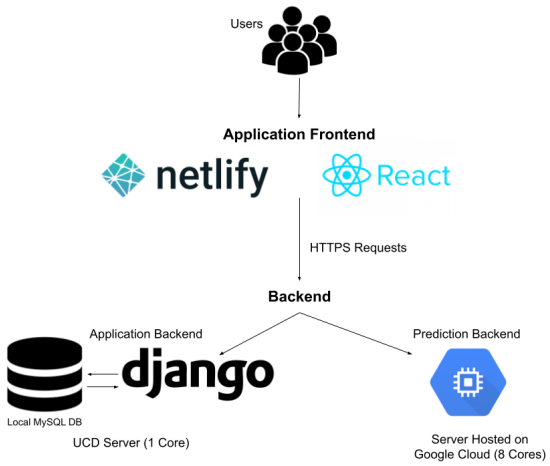

Figure 4.1: Technical Architecture Overview

4.1 Tech Stack

We considered a number of factors when deciding which tools and technologies to use in our project. We wanted to use this project as a chance to try new tools that are widely used in industry and compare them to what we have already used. However, we tried to be careful not to overwhelm ourselves with too much to learn at once and to pick technologies that would be well-suited to our task and also that are stable and well-tested with good documentation and tutorials. We tried to critically analyse a number of sources when making our decisions.

4.1.1 Frontend

We decided to use React as our frontend framework because one member of the team had used it in prior projects and showed us how it can be used to make the frontend codebase more modular and to easily manage state. In addition, it has an extensive community of pre-built components, styles, and icons. We all had experience of how large and unwieldy a plain JavaScript codebase can become without a framework like this. The StackOverflow 2021 Developer Survey [17] also shows that React is one of the most widely used and popular frameworks among developers. One downside of using React was that it requires time to adjust to a different way of reasoning about program state and it was hard for the team members who were not working on the frontend to

dedicate the time to learn it as we became increasingly specialised. We still feel that React was a good choice overall though because it allowed the codebase to be kept clear and modular and for our frontend developer to be very productive.

4.1.2 Backend

From the start of the project we agreed to use Python for our backend services because we did not want to learn an entirely new language. We then compared two of the most popular web frameworks for Python: Django and Flask. We had all used Flask in a COMP30830 project and found it intuitive. However, our research suggested that Django might have better support for user authentication and accounts by default and this made us favour it over Flask ([5], [6]). We also felt that learning another web framework would be useful and not too risky compared with trying to learn both a new language and a framework. After adjusting to the differences from Flask, we found it quite easy to work with Django and that the two frameworks are very similar. Django seemed to require more code to set up initially and we prefer the 'lightweight' style of Flask overall but in the end we do not feel it had much impact on our project.

4.1.3 Data Analytics

Jupyter Notebooks are a widely used tool in data analytics. However, our experience in previous projects made us wary of some of their limitations. In particular, they can make results hard to reproduce because they hold hidden state which can change between runs of the same code [9]. They can also be difficult to use with version control tools like GitHub because they mix markdown with code and the kernel occasionally crashes when training models for long periods. For all of these reasons we decided to write our data-analytics code in standard Python scripts from the beginning and to commit this code to our repository for review by other team members. We only used Jupyter for debugging or for creating graphs from output. This seemed to work well overall and we would make the same decision if we were to begin the project again.

As discussed in Chapter 2, we need to use Neural Networks to generate uncertainty for our travel time estimates. This was not possible with the scikit-learn library that we were most familiar with and it was difficult to compare libraries for Neural Networks since we did not have much experience in this area. We settled on using Keras since it seemed well-documented and aims to provide a simple interface to the Tensorflow platform. The learning curve for this library was steep initially but we were fortunate that the popularity of Neural Networks in the last few years means that there are many useful tutorials available and we would choose this library again.

4.1.4 Database

We considered several factors when deciding which database technology to use. First, we decided to use a SQL database rather than NoSQL because we are all more familiar with SQL and we did not expect to need the schema flexibility or JSON-style structures that NoSQL databases offer.

Second, we compared three of the most popular SQL databases: MySQL, PostgreSQL, and SQLite. Our research suggested that SQLite would be the simplest and quickest to set up since the data is stored in a file rather than hosted on a server. However, this also meant that it could not be easily accessed over a network and that it might have limited concurrency support compared with other databases [18]. We expected concurrency to be an important issue for a transport application that is intended to be used by thousands of people every day so we ruled out SQLite.

Turning to MySQL and PostgreSQL, it seemed that they are very familiar overall, particularly in more recent versions. We opted for MySQL in the end, however, because some research, such as a detailed analysis by the Uber Engineering Team [20], suggested that it was better for high-concurrency, read-heavy applications and we did not expect to need the additional features that PostgreSQL could offer such as JSON records. However, with the benefit of hindsight, it seems like the choice between MySQL and PostgreSQL did not matter much in the end at the scale of our application.

Finally, we also had to consider where to host our database. We considered hosting the database and the Django backend on the same server to avoid any network latency but were concerned that this would cause performance problems on the small server that UCD provided. An alternative option was to host the database on a cloud platform which would simplify deployment and performance and would make the database accessible from anywhere. However, our experience in a previous project made us concerned about the network latency of this option.

In the end, we started with a MySQL database hosted in AWS but ended up switching to a local MySQL database because the latency was too high. We also used SQLite for testing and quickly prototyping changes at various stages in the project. It was interesting and helpful to compare these different database options but we probably waited too long before committing to our final choice because it caused considerable confusion at times when it was not clear which database was currently in use by the team.

4.1.5 Deployment and Code Quality Tools

From the start of the project we wanted to follow a "Continuous Deployment" philosophy where any changes that were committed to our frontend application would be made publicly available in an app we could immediately test. We did not want to delay deployment until the end of the project when we would be busy with other features and bug fixes. We used Netlify as a tool for this and found it intuitive and straightforward. Our Django backend, by contrast, was not automatically deployed in this way because we wanted to be able to easily customize the backend and database settings as discussed above. Instead, we had to manually deploy it on the UCD VM. This may have been a mistake because we spent considerable time and effort managing this deployment and should have done more research on a tool that could handle this for us.

We also wanted to serve our application over HTTPS to protect user data and avoid any security warnings. Netlify automatically setup HTTPS for our frontend application. We used NGINX combined with the certbot utility to set up and manage the certificates required for HTTPS for our Django backend. We were pleased with how quick it was and how few configuration changes were required for this.

Our initial plan also included using Docker for deployment because it is a widely used technology in industry that we wanted to learn and we thought it might help us to easily move the application to a new server at the end of the project. However, we decided to postpone the introduction of this tool until we had a basic version of the application running. In one of the weekly presentations for the project, many of the teams also faced questions on whether Docker would introduce unnecessary complications. Reflecting on these questions and answers led us to abandon our plan to use Docker and we think this was a wise decision overall because we were already under significant time pressure at the end of the project.

Finally, we knew that maintaining code quality and clarity would be a major challenge as the project developed and deadlines approached. To counteract this we set up code linting tools from the start to catch basic errors. We chose pylint for Python because it is integrated by default with the IDE we were using. While it did find many simple errors, we were surprised and

disappointed with how many unhelpful warnings it provided and how much configuration effort it took to suppress these and would likely choose a different tool for a future project. ESLint, the tool we used for linting our JavaScript code, by contrast, worked much better with minimal configuration.

4.2 Prediction Modelling Architecture

We compared two main ways of modelling the problem of predicting bus journey times. The first approach we considered was to train a separate model for each adjacent pair of stops in the network as described in Pandurangi et al. [15] and shown in Method 1 of Figure 4.2. An advantage of this approach is that we can combine data from many different routes that travel between the two stops. If we want to predict the travel time for longer journeys then we can simply add up the predictions for each adjacent pair of stops along the way. A disadvantage of this approach is that there are several thousand adjacent pairs of stops which means that we would have to train many models. Some stop pairs also have very few observations either due to lower popularity of the route or data quality issues which may make their results unreliable.

An alternative and simpler approach that was suggested to us in a mentor meeting was to make a prediction for the entire end-to-end journey time for a particular route in one direction and then proportionally break this prediction up as required as shown in Method 2 in Figure 4.2. The percentage of total travel time used to break up the predicted end-to-end journey time would be calculated as simple averages from the historical data. An advantage of this approach is that there are significantly fewer models to train (one for each route in each direction only) and potentially more data and fewer data quality issues for each model. However, we were unsure how well splitting up the full route prediction would work so we decided to start with the previous approach and planned to use this full-route prediction as a comparison in our testing.

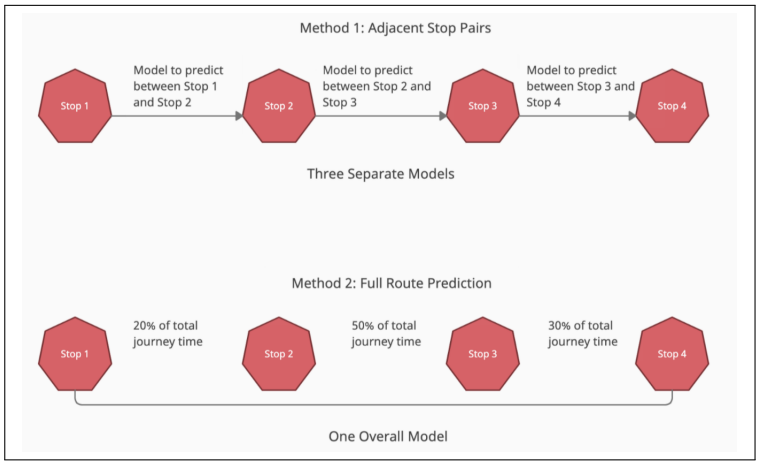

Figure 4.2: Prediction Modelling Architecture

For our prediction model we chose Neural Networks because our travel time uncertainty estimates method required us to use this algorithm as described in Section 2.7.1. Reich et al. [16] also find that simple Neural Networks have been the most popular and successful in the bus travel time estimates literature and this finding combined with our limited time and computational resources led us to keep our architecture relatively simple with 2 hidden layers. We provide more details on the final model architecture and features we settled on after testing in Section 5.3.

Chapter 5: **Testing and Evaluation**

5.1 Frontend Testing and User Survey

We conducted a small user survey of friends and family members (12 respondents) to get feedback on the usability of our frontend application at the end of the project. In general, users reported that they found the app intuitive and easy-to-use with 75% of respondents rating the overall design as at least 4 out of 5, for example. The feedback on the travel time uncertainty was more mixed. 83.3% of respondents preferred the range of journey time predictions that we presented (e.g., 50-55 minutes) over the single estimates seen in other applications. However, a majority of respondents (58.3%) either did not understand the Quantile Dotplots or did not find them valuable. These results did not completely surprise us because we anticipate that the Quantile Dotplots would only be of interest to a small subset of users and we agree that further work is needed to make them easier to read as discussed in Kay et al. [10]. The survey also helped us to identify and resolve a number of bugs in our application such as layout problems on other devices and browsers and we feel that it would have been valuable to conduct more surveys earlier on in the project to help us guide our design decisions. A breakdown of the survey responses is shown in Appendix A.

- Selenium - low coverage, ran out of time
- Lighthouse - what were our scores and did we make any changes to raise them?

5.2 Backend Testing

To test the performance and scalability of our backend, we conducted some basic load testing using locust. With 5 concurrent users making approximately 1 request per second, we found that most of the endpoints in our backend application had a mean and median response time of 1 second or less after we implemented suitable caching. Endpoints that required significant computation that could not be cached were the main exceptions to this. Most importantly, the endpoint that handled our travel time predictions, had a median response time of 4.3 seconds with significant variance depending on the journey length. For example, the minimum response time was 1.35 seconds while the maximum was almost 24 seconds. We began to see many failed requests to this endpoint due to timeouts once we increased the number of concurrent users to 15, even while the response times for the other endpoints remained low. We feel that these results are reasonable given the time and budget constraints we faced in the project.

- Write-up results from Django unit tests (Andrew)
- Coverage (Andrew)

At the outset of the project, we had also planned to write extensive unit tests for all of our backend code. We did write some unit tests early on in the project and these tests are run in

an automated pipeline on GitHub every time a pull request is opened or code is merged into the main branch. However, the test coverage was very low by the end of the project and our test suite was not extensive enough to be useful in catching most errors. There are a few reasons why this happened. First, we found it difficult to write tests for many of the backend features which involved interactions with APIs or databases because it involves setting up mock versions of these. Second, we were concerned that making unit tests compulsory would slow our development process down too much. Our plan was to come back to writing these tests in the final sprint but we found that we had too much other work that seemed higher priority at that time. This is a significant limitation of our testing process and we now feel that writing tests from the start could have saved time in the end and given us more confidence about making changes towards the end of the project.

5.3 Data Analytics Testing

Testing and evaluating the data analytics part of the project presented a number of challenges. First, our chosen approach of training a model for each adjacent stop pair resulted in over 4,000 models to evaluate. In previous projects we had only ever trained a handful of models at most and it was relatively straightforward to know whether changes were improving the outcomes. With so many individual models to evaluate we had to rely on summary statistics and scatter plots of the training and test scores across the models to know whether a change to a particular feature or the model architecture was improving the overall results. As expected, such changes would usually improve the scores for some models while lowering them for others.

Second, the volume of the data for each stop pair meant that testing changes would often take many hours which made it difficult to track and make changes quickly unlike in other areas of the project where the results of code changes were immediate and clear. While we wanted to test our changes on all of the available data, it seems likely that choosing a representative sample or focusing on the main routes could have saved us considerable time.

Finally, the temporal aspect of the dataset made some aspects of testing more challenging. For example, when splitting our data into train and test sets, we had to be careful to avoid data leakages that could occur if we trained our models on data that occurred at a later time than the test set. To avoid this bias we used rolling time periods instead of random shuffling for cross-validation as recommended in Bergmeir et al. [3]. In a small number of cases this caused problems because it resulted in large differences in the distribution of features between the training and test datasets which in turn caused poor results on the test set.

5.3.1 Main Results

The most common evaluation metrics in the papers surveyed in Reich et al. [16] were RMSE and MAPE. We decided to use RMSE exclusively after discovering some of the limitations of MAPE (e.g., bias towards lower predictions [19]). We used Grid Search to optimise parameters such as the number of neurons in each layer, the dropout rate, and the number of training epochs. However, the number of parameters to tune and the size of the data made this more challenging than expected and we were forced to confine our search to a small range of parameters that we expected to be most important. The best performing model had 10 neurons in the first hidden layer, 15 neurons in the second hidden layer, a dropout rate of 0.2, and 15 training epochs.

We also spent considerable time on feature engineering and testing. At the start of the project

our expectation was that the time of day and the day of the week would be among the most important features in the model. We compared several approaches to modelling these features. For example, we tried separate models with dummy variables for each day and hour or a binary variable to indicate the weekend only. To our surprise, there was very little difference in the RMSE on the training or test set for any of these approaches. In the end, we chose to use cosine and sine transformation of the datetime features as is commonly recommended by machine learning practitioners (e.g., Audevart et al. [2] pp. 209). We also had features for rain in the current hour, rain in the previous hour, and a dummy variable for a bank holiday.

The final model we settled on after all of this tuning achieved an mean RMSE of 28.37 seconds on the training and 29.40 seconds on the test data across all adjacent stop pairs. However, the distribution of RMSE scores was quite wide with a standard deviation of over 60 seconds. To get a sense of how well this model was performing, we created a very simple "benchmark" model that would predict the historical average travel time between two stops in all cases (i.e., no features) as suggested in Reich et al. [16]. To our surprise and disappointment, this "model" proved quite hard to beat and outperformed our baseline Neural Network with a mean RMSE of 18.27 seconds on the training data and 19.04 on the test set. We also trained a Linear Regression model for comparison purposes and found that it also performed better than our Neural Network with a mean RMSE of 17.84 and 18.21 seconds on the training and test sets, respectively.

These results suggested that our Neural Network model may have been too complicated, particularly for stop pairs with few observations. Plotting the RMSE against the number of observations for each stop pair showed that there were many RMSE outliers and that these were all for stop pairs with relatively few observations. We discussed this extensively in our mentor meetings and were advised to plot learning curves for our models to see if we could find a minimum number of observations before the models stopped improving. We found that in general the training and test RMSE stopped improving after between 8,000 and 12,000 observations. Approximately 75% of the stop pairs had at least 8,000 rows and almost all of the very large RMSE outliers had much fewer rows than this. Once we excluded any stop pairs with fewer than 8,000 rows we found that the scores for our Neural Network model were similar to the above benchmarks. Appendix B shows more details on all of these results.

Finally, we discussed whether we should even deploy models that did not have many rows and had high RMSE scores. As an alternative we considered falling back to either the historical averages or the current timetable for these stop pairs. However, we found it difficult to come up with a reliable criterion for deciding whether a model was good enough to deploy and we were advised in a mentor meeting to deploy all of the models we had trained but to highlight that the most accurate predictions are likely to occur on popular routes with more data.

5.3.2 Other Results and Comparisons

In addition to looking at the training and test RMSE of our models for each adjacent stop pair, we also sought to evaluate and compare our models in other ways. For example, as well as evaluating how well each model performed on adjacent stop pairs, we also examined how well it performed on predicting the time for a full route from start to finish. We followed the approach of Pandurangi et al. [15] of choosing a small number of routes and we also found that our adjacent stop pair model was often less accurate than the 2018 timetable for the full route even though our model was different to that paper. On the 46A route, for example, we found our adjacent stop pair model had an RMSE of 557 seconds compared with 479 seconds for the static timetable.

We also did some brief comparisons with the full-route modelling approach discussed in Section 4.2 (Method 2 in Figure 4.2). For example, taking a small sample of end-to-end trips on the 39A route, we found that the full-route model outperformed our adjacent stop pairs approach with a

RMSE of 938 and 1920 seconds, respectively. The full-route model also performed better on a selection of shorter journeys on this route (RMSE of 858 vs. 1,885 seconds). Our original plan was to do a comprehensive comparison of both modelling approaches before deciding which one to deploy but we did not have enough time to complete this and ended up deploying the adjacent stop pairs model since that was our first approach. But these preliminary results indicate that the full-route model may have been a better choice.

Finally, in one of our mentor meetings we were advised to consider testing how well-calibrated the probabilities from our travel time estimates are. For example, if we predict that a journey between two stops should take less than 10 minutes in 80% of cases, what fraction of journeys actually take less than this? We agree that this would be a useful exercise but unfortunately we did not have enough time to implement this kind of testing but we consider it an important next step in our work.

Bibliography

[1] Apple App Store. 2021. Dublin Bus Ratings and Reviews. Retrieved August 20, 2021 from https://apps.apple.com/ie/app/dublin-bus/id450455266#see-all/reviews

[2] Alexia Audevart, Konrad Banachewicz, and Luca Massaron. 2021. *Machine Learning Using TensorFlow Cookbook* (1st Edition). Packt Publishing.

[3] Christopher Bergmeir, Rob J. Hyndman, and Bonsoo Koo. 2018. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics and Data Analysis*, 120 (April 2018), 70-83. DOI: https://doi.org/10.1016/j.csda.2017.11.003

[4] Umang Bhatt, Javier Antorán, Yunfeng Zhang, Q. Vera Liao, Prasanna Sattigeri, Riccardo Fogliato, Gabrielle Gauthier Melançon, Ranganath Krishnan, Jason Stanley, Omesh Tickoo, Lama Nachman, Rumi Chunara, Madhulika Srikumar, Adrian Weller, Alice Xiang. 2021. Uncertainty as a Form of Transparency: Measuring, Communicating, and Using Uncertainty. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, (AIES '21), May 19–21, 2021, Virtual Event, USA. ACM, New York, NY, USA, 20 pages. DOI: https://doi.org/10.1145/3461702.3462571

[5] Django Software Foundation. 2021. User Authentication in Django. Retrieved 20 August, 2021 from https://docs.djangoproject.com/en/3.2/topics/auth

[6] Kite. 2019. Flask vs. Django: Choose Your Python Web Framework. Retrieved 20 August, 2021 from https://www.kite.com/blog/python/flask-vs-django-python

[7] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning* - Volume 48 (ICML'16). JMLR.org, 1050–1059.

[8] Jan-Willem Grotenhuis, Bart W. Wiegmans, and Piet Rietveld. 2007. The desired quality of integrated multimodal travel information in public transport: Customer needs for time and effort savings. *Transport Policy*. 14(1), 27-38. DOI: https://doi.org/10.1016/j.tranpol.2006.07.001

[9] Joel Grus. 2018. I don't like notebooks.- Joel Grus (Allen Institute for Artificial Intelligence). Retrieved 20 August, 2021 from https://www.youtube.com/watch?v=7jiPeIFXb6U

[10] Matthew Kay, Tara Kola, Jessica R. Hullman, and Sean A. Munson. 2016. When (ish) is My Bus? User-centered Visualizations of Uncertainty in Everyday, Mobile Predictive Systems. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 5092–5103. DOI: https://doi.org/10.1145/2858036.2858558

[11] Matthew Kay, Tara Kola, Jessica R. Hullman, and Sean A. Munson. 2016. Data and code for "When (ish) is My Bus? User-centered Visualizations of Uncertainty in Everyday, Mobile Predictive Systems" (CHI 2016). Retrieved August 20, 2021 from https://github.com/mjskay/when-ish-is-my-bus

[12] Benjamin Kompa, Jasper Snoek, and Andrew L. Beam. 2021. Second opinion needed: communicating uncertainty in medical machine learning. *npj Digital Medicine* 4,4. DOI: https://doi.org/10.1038/s41746-020-00367-3

[13] Terence C. Lam and Kenneth A. Small. 2001. The value of time and reliability: measurement from a value pricing experiment. *Transportation Research Part E: Logistics and Transportation Review*. 37(2–3), 231-251. DOI: https://doi.org/10.1016/S1366-5545(00)00016-8

[14] National Transport Authority. 2020. Massive Jump in Passenger Journey Number as Commuters Flock to Public Transport. (January 2020). Retrieved August 20, 2021 from https://www.transportforireland.ie/news/massive-jump-in-passenger-journey-number-as-commuters-flock-to-public-transport/

[15] Ankhit Pandurangi, Clare Byrne, Candis Anderson, Enxi Cui, and Gavin McArdle. 2020. Design and Development of an Application for Predicting Bus Travel Times using a Segmentation Approach. In *Proceedings of the 6th International Conference on Geographical Information Systems Theory, Applications and Management*, (GISTAM 2020), 72-80. DOI: 10.5220/0009393800720080

[16] Thilo Reich, Marcin Budka, Derek Robbins, and David Hulbert. 2019. Survey of ETA prediction methods in public transport networks. arXiv:1904.05037. Retrived from https://arxiv.org/abs/1904.05037

[17] StackOverflow. 2021. Developer Survey Results 2021. Retrieved 20 August, 2021 from https://insights.stackoverflow.com/survey/2021

[18] SQLite. 2021. Appropriate Uses For SQLite. Retrieved 20 August, 2021 from https://www.sqlite.org/whentouse.html

[19] Chris Tofallis. 2015. A Better Measure of Relative Prediction Accuracy for Model Selection and Model Estimation. *Journal of the Operational Research Society*. 66(8), 1352–1362. DOI: https://doi.org/10.1057/jors.2014.103

[20] Uber Engineering. 2016. Why Uber Engineering Switched from Postgres to MySQL. Retrieved 20 August, 2021 from https://eng.uber.com/postgres-to-mysql-migration/

Appendix A: **Survey Results**

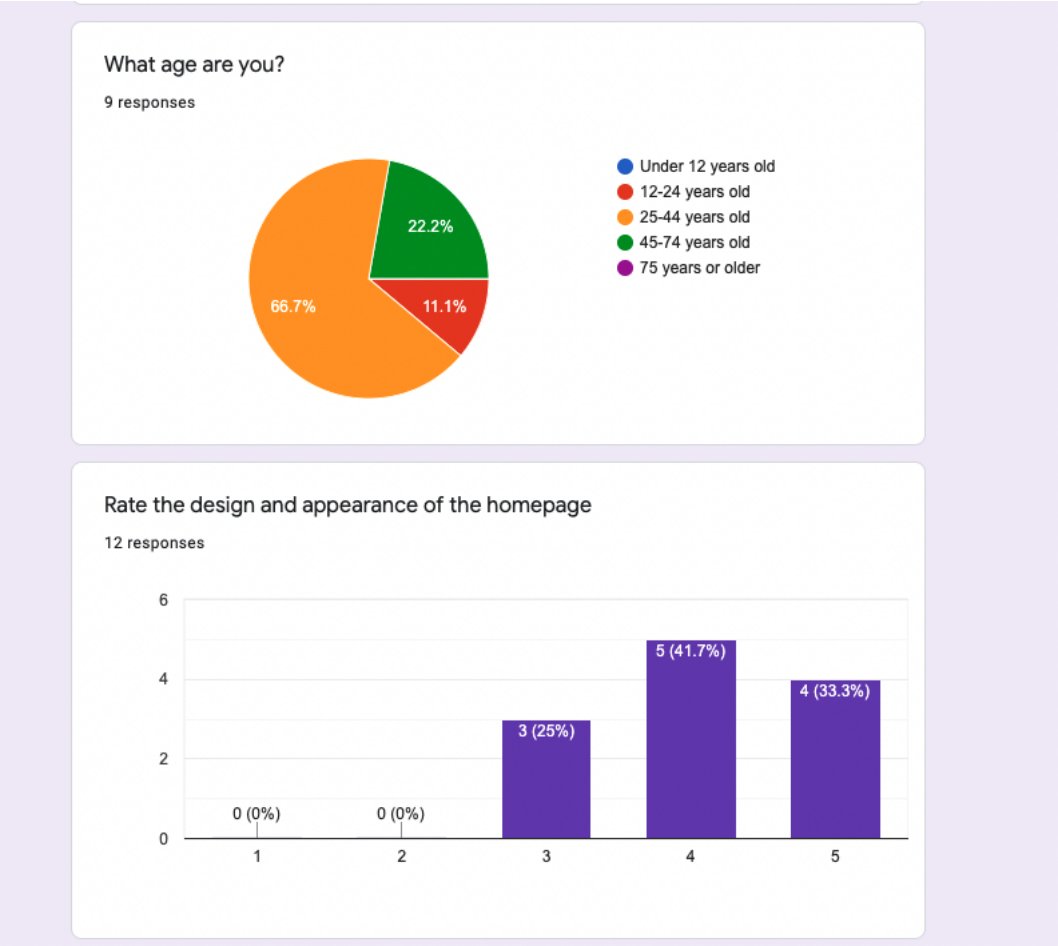

Figure A.1: Survey Results Part 1

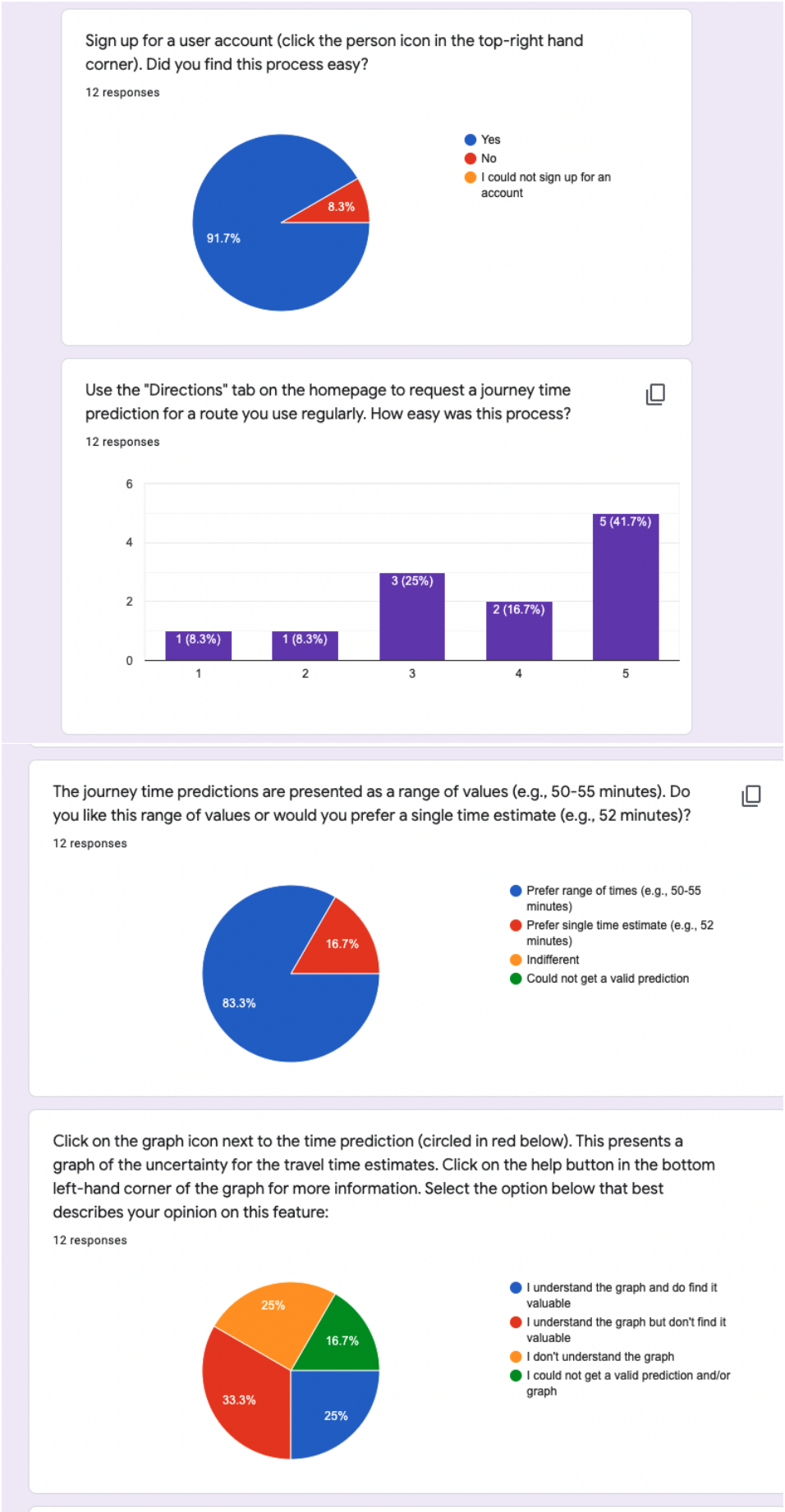

Figure A.2: Survey Results Part 2

Appendix B: **Model Results**

Table B.1 shows summary results from 4 models:

- Our main Neural Network model on all adjacent stop pairs
- The Historical Benchmark model on all adjacent stop pairs. This "model" involves calculating the average journey time between two stops in the historical data and using that as the prediction
- Linear Regression model on all adjacent stop pairs
- Our main Neural Network model only for those adjacent stop pairs with more than 8,000 rows

Figure B.1 shows a sample of scatter plots of training and test RMSE against the number of rows for each stop pair. This highlights that there are significant outliers but they are all clustered in stop pairs with a relatively small number of rows.

Table B.1: Model Results Comparison

	Neural Network		Historical Averages Benchmark		Linear Regression		Neural Network ($n > 8,000$)	
	Train RMSE	Test RMSE	Train RMSE	Test RMSE	Train RMSE	Test RMSE	Train RMSE	Test RMSE
n	4531	4531	4542	4542	4531	4531	3363	3363
mean	28.37	29.40	18.28	19.04	17.84	18.21	18.44	18.98
std	62.27	64.55	26.21	26.91	28.43	25.89	25.06	24.90
min	1.12	1.13	1.04	1.09	1.04	1.06	1.12	1.13
25%	7.41	7.56	5.45	5.58	5.23	5.34	6.94	7.04
50%	13.75	14.08	10.03	10.48	9.68	10.06	11.99	12.26
75%	28.85	29.43	24.53	25.22	23.26	24.24	24.48	24.87
max	1342.92	1477.79	993.79	881.26	1001.59	890.33	1017.53	895.62

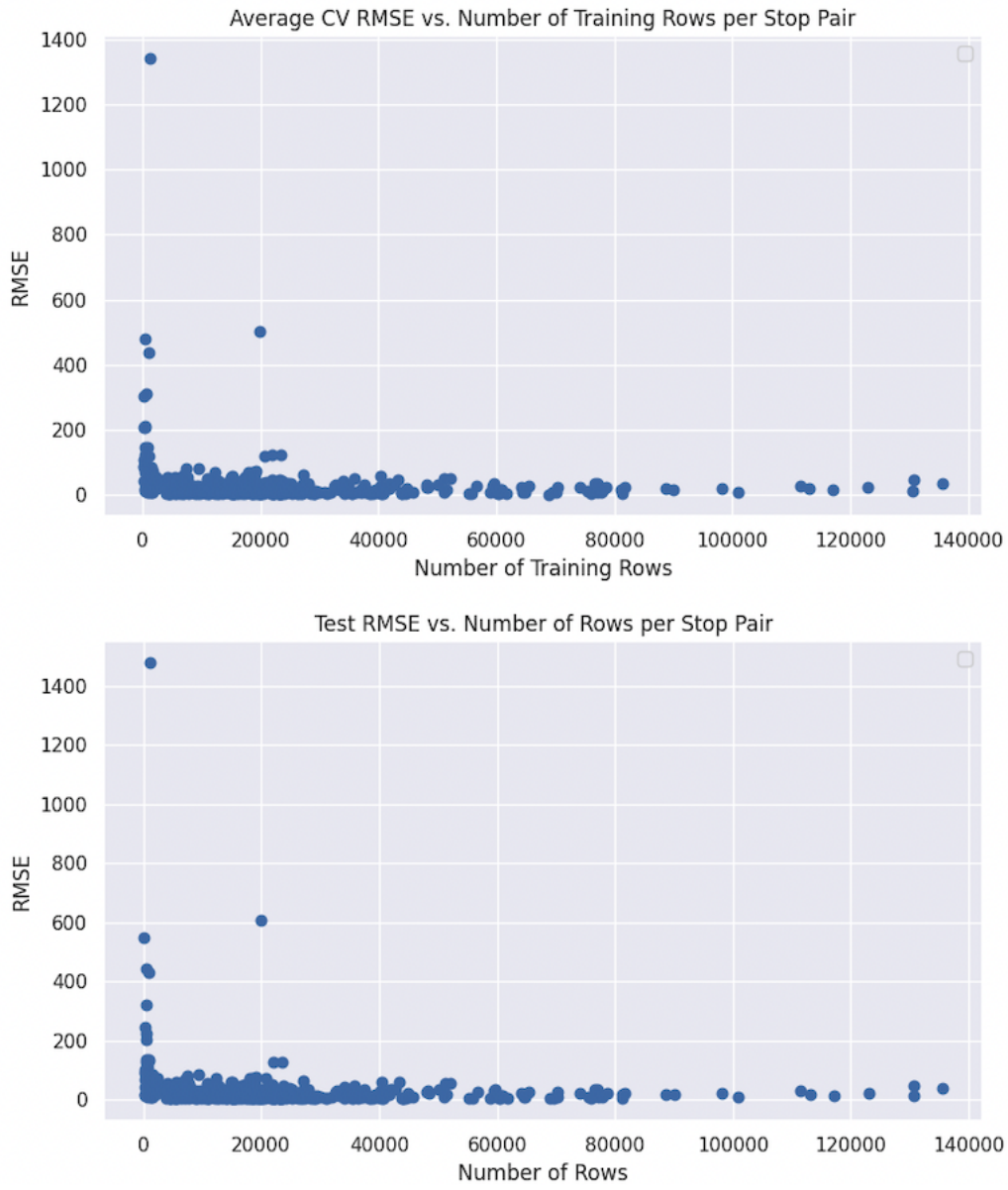

Figure B.1: Scatter Plots of RMSE against number of rows per stop pair