INTELLIGENT URBAN TRANSPORT TRACKING AND MANAGEMENT SYSTEM

Associate Prof. Samer I. Mohamed, October University for Modern Sciences and Arts MSA, saibrahim@msa.eun.eg

Abstract — Egypt's current bus system is large enough to satisfy a significant portion of the populations demand yet fails to do so due to mismanaged resources. Static lines and unclear schedules create a confusing and unappealing user experience which pushes more of the population to cars for their transportation needs. This clearly leads to more congested streets which result in a net loss of productivity as well as an increase in stress, unnecessary fuel consumption, and harmful emissions. An intelligent bus solution is multi-faceted. It consists of (1) connected buses which are capable of providing their geo-location data, feedback about driving behavior, and health data to detect failures before they occur; (2) cashless payment through RFID cards to ensure much tighter control over pricing; (3) a processing server or cloud, in which all of the incoming data would be handled; (4) knowledge systems which dynamically optimize bus schedules and routes through learning algorithms; (5) and a mobile application to capture demand and inform passengers of bus arrival times. The main functions and algorithms of the proposed system are achieved based on machine learning algorithms and web technologies, whilst the hardware component is implemented based on System-on-Chip technology with custom hardware to interface with the vehicle. This paper will focus on the software component of the proposed solution. It is shown that by applying the proposed system to a previously static bus system that fuel consumption, maintenance costs, and carbon emissions can be reduced by 10-20% while overall passenger satisfaction is increased.

Index Terms— Neural networks, particle swarm optimization, Urban transportation, vehicle routing problem, web application

1. INTRODUCTION

Transportation systems typically involve two main parties: the operator, and the passenger; each with a different set of priorities and objectives. The operator’s objective is to provide a profitable transportation service to as many passengers as possible, while the passenger’s objective is to get from source to destination as quickly and comfortably as possible. This conflict of objectives is the root cause of two of the biggest issues facing public bus transportation systems: optimization of resources, and transparency of information.

A. Mismanaged Assets

Egypt’s Greater Cairo Metropolitan Area is host to one-fifth of Egypt’s population 19 million inhabitants. This population causes an incredible amount of stress on existing transport infrastructure and results in a net loss of productivity as well as an increase in stress, unnecessary fuel consumption, and harmful emissions. The traditional solution is to heavily invest in mass-transit infrastructure; however, Egypt already licenses. Enough buses to handle a significant portion of this demand. As of 2017, Egypt has licensed close to 156,000 buses [1]at an estimated average capacity of 35 seats per bus. This amounts to over 5.4 million seats, certainly enough to alleviate the stress of the 4.28 million licensed private cars and taxis[1] since a majority of vehicles tend to be low or even single-occupancy. The real problem lies in the operational inefficiency found inherently within the static bus system. Routes are often defined based on outdated feasibility analyses and resource distribution is pre-allocated to specific routes based on historical demand. This often creates incoherence between the real-world supply and demand where a lack of the former on highly demanded lines means a significant portion of buses will stay idle for long periods of time until they are needed. The inability to find transport to or near passenger destinations greatly detracts from the user experience and forces more of the population to adopt more cars which contribute to the aforementioned problems.

B. Lack of a Unified Digital Infrastructure

The current state of the bus transport system in Egypt can only be described as chaotic. As it stands, to get from point A to point B using the formal or informal bus system, users must be aware of the relevant hand signals - as well the bus” hubs” used for transit - in order to avoid running late or even getting lost. Users are left guessing where, when, and if they might even find a ride to their destinations; there is neither clear system nor a database of the buses and their routes. Therefore, a supply optimization problem cannot be tackled without the supporting digital infrastructure that is not currently available. Emerging startups have attempted to tackle this issue indirectly by layering gathered data on top of this outdated infrastructure. One such example is Transport for Cairo (TFC) who has recently published the first ever General Transit Feed Specification (GTFS) formatted feed for buses in Egypt, although most of their data is based on estimates and observations. Another application-based startup, Flare, attempts to build on the data made available by TFC, by crowd sourcing GPS data, to more accurately represent the real-time movements of each bus within the system. While these approaches attempt to make sense of a broken system, only one application-based startup, SWVL, has attempted to directly challenge the service operators directly by providing a reliable technology-first solution, however, have missed the mark by catering to middle and upper classes.

C. The Transportation Landscape

Access to transportation is the single most important factor in an individual's ability to escape poverty. Egypt’s current strides in building new cities and a new capital have opened up opportunities in which these systems could flourish. Newer cities have a unique chance to demonstrate what is possible when long-term urban planning is conducted with a technology-first mindset. By providing a more demand responsive user-experience, we can expect more of the population to leave their cars behind in favor of the buses as...
they have shown to do with ride-sharing applications [2]. This reduction in volume of vehicles should lead to a dramatic decrease in congestion and therefore drastic reductions in carbon emissions. Intelligent Transport Systems (ITS) for public transportation are all built on a similar foundation of objectives. Each system then identifies itself with extra features to enhance the overall experience for all parties involved. Figure 1 describes KPIs solution for an intelligent transport system. Generally, there is a system on-board the bus capable of many functions used to enhance the passengers experience such as the Automatic Vehicle Location (AVL) for accurate tracking of the bus, surveillance cameras for added safety, the audio passenger information system (PIS) to announce upcoming stops, electronic ticketing machines (ETM) for cashless payment, and vehicle health monitoring and diagnostics for quicker incident response [3]. This system leverages cellular and satellite networks to transmit and receive data to-and-from the bus command center which is capable of monitoring the entire fleet remotely, as well as to the PIS application on smart phones. While this system is designed from the ground-up on a solid digital infrastructure, the system is still inherently static both in terms of the routes and the scheduling of the buses.

II. LITERATURE REVIEW

Aided Information and Routing System (DAIR) [4]. A car equipped with DAIR could send emergency messages and road condition information to a service center. The 1970s witnessed an early generation of bus automatic vehicle location (AVL) mapping technology. During the 1980s through to the 1990s, as technology became increasingly cheaper and more capable, technologies supporting improved traffic management emerged. The transportation industry recognized new highway infrastructure-based technologies as a competitive business opportunity that would add value to their products. New technological developments such as microprocessors, computers, sensors, communications technologies, and GPS became prominent in the transportation industry around the 2000s enabling automatic fare collection, passenger counting, and improved vehicle diagnostics [4].

A. Transport for London

The bus system in London is mainly used by tourists; most locals prefer the subway as it is much faster. However, to encourage users to interact more with the bus system, Transport for London has implemented some useful features such as a messenger bot and an SMS service to aid the main website in providing the most accurate information about their fleet [5]. These efforts to make information more transparent and accessible to the passengers has resulted in a better experience, yet despite that, bus patronage has been on a downwards trend since 2008.

B. Seoul Transport Operation & Information Services

After the 2004 reforms, the government greatly increased its control over bus routes, schedules, fares, and the overall public transport system design. This enabled the implementation of many technologies that enhance the user experience and encourage most travelers to opt for public transit options over driving, even for long distance journeys as shown in Figure 2 for the TOPIS system [6].

C. SWVL

Swvl is an Egyptian app-based startup through which users can book fixed-route bus trips at prices 60-80 percent lower than competing ride-hailing services and without surge or peak pricing. This application tackles the same issues as this project; however, it approaches the problem through the use of out of service tourism buses. It provides similar features in terms of vehicle tracking, ease-of-use, and a mobile-first approach.
III. PROPOSED SYSTEM DESIGN

A representation of the Intelligent Urban Transit System (IUTS) can be seen in Figure 3 whereby the design is built on four main pillars: On-board Bus System, Bus Stop System, Central Management System, and a collection of Web Interfaces. This is a holistic approach to addressing the issue of public bus transportation in Egypt as it allows both the operator and the user to contribute to the regulation of supply and demand [7]. The efficiency of such a system can be evaluated by measuring the passenger demand and the number of vehicles supplied to meet that demand; these indicators can be combined into one metric known as the passenger load factor. The passenger load factor is a measure of capacity utilization and is expressed as the ratio of passenger-kilometers traveled to total seat-kilometers traveled. The IUTS provides a real-time schedule, location, arrival time, and route to the travelers via the cellular network. It also provides a web-based platform to operators for administration and information analytics tasks. The data generated by the on-board tracking system, along with the requests received from travelers, is fed into a set of algorithms capable of predicting upcoming demand and optimizing routes and fleet distribution for the reduction of costs as well as negative environmental impacts.

![Figure 3: High Level System Block Diagram](image)

Due to the nature of this application, every sub-system communicates wirelessly over a cellular network. The bus stops are responsible for acquiring the demand from legacy users (who may not have access to a smartphone), whilst the mobile interface is responsible for acquiring the remaining demand. This demand at each source node (bus stop) in the network is sent to the Central Management System (CMS) that consumes all the data input from the different sources towards the objective functions of the overall proposed system for dynamic routing and satisfying the raised demand. For the proposed system to function optimally, it must be allowed to explore the data space for a long period of time to train the predictive model across a wide spectrum of scenarios. After exploration, the system can be tuned to then exploit the accumulated data set and make predictions about future data points. These predictions can be utilized to generate optimized routes utilizing the optimum number of buses to maximize the passenger load factor [8]. Once routes are generated, the fleet’s state is considered, and the most suitable bus is matched to the route. Once a bus route is determined, users may track their bus in real-time through the mobile interface.

![Figure 4: Sub-system Transactional Flow Diagram](image)

The aim of these interfaces is to democratize public transport by allowing the supply-demand chain to be equally controlled by the passengers and the operators. As Figure 5 demonstrates, the proposed solution allows passengers to reserve their seats ahead of their rides and track their bus on the live map view. Furthermore, the push notification engine can be leveraged to provide helpful timed information to the passenger. The operator dashboard provides a complete overview of the entire fleet’s day-to-day operation including crucial data about their current location, route, speed, and cost. This analytics data enables better decision making for the operator. The dashboard goes one step further to lay out alerts received by the on-board diagnostics (OBD) unit to notify operators of possible upcoming maintenance.

![Figure 5: User Interfaces Sub-block Diagram](image)
A. Detailed Block Diagrams

For an application of this scale, a modern and highly supported software stack [10] was chosen, the details of which are highlighted inside of each sub-block in Figure 12. The choice of these web technologies is crucial as it determines this solutions ability to penetrate emerging markets with their often low-storage capacity and low mobile data usage requirements. The Typescript, React Native, Apollo, and PostgreSQL (TRAP) stack is built on JavaScript and enables simultaneous handling of thousands of requests with ease as shown in Figure 6.

![Figure 6: Detailed CMS Sub-block Diagram](image)

Typescript is a superset of the JavaScript language; built by Microsoft and with a focus on large applications, this language transpires into JavaScript at no added cost and with the benefit of syntactically superior code which reduces the likelihood of bugs during development due to invalid or improper type declarations. React Native is a framework for building native mobile applications using the web-languages JavaScript and React. This technology facilitates a reduction in development overheads as the application would work cross-platform across iOS and Android automatically. Apollo is an implementation of GraphQL designed for the needs of data-driven applications such as the proposed solution. Apollo resides in the layer between application server and the database; it is essentially the application programming interface (API) that allows us to only query as much information as we need from the database. This creates a more secure database that can only be accessed with a set of pre-defined functions, rather than the usual REST approach of returning all rows in a query. We chose to use a structured database, PostgreSQL, due to the relational nature of our data; furthermore, PostgreSQL is widely supported and is very reliable. The final sub-component is the Machine Learning algorithms, the pipeline of which will be explored in the following sections.

B. Data Pipeline

Figure 7 highlights the designed layout of the CMS database to accommodate the demand responsive system. The database is divided into 8 tables: 2 of which are passenger specific and define their profile as well as trip history; the other 6 are operator specific and define an operators fleet in terms of the type of buses, trip history, drivers information, licensed bus stops, and finally a Current Bus table that keeps a log of real-time information used to train the ML algorithms.

![Figure 7: DB schema](image)

To create a truly demand-responsive system, the algorithm stack must operate in the fuzzy domain whereby a combination of historical and real-time demand is utilized alongside the other input features to predict upcoming demand as well as optimize the fleet for this demand. The proposed system accomplishes this operation, periodically, using two algorithms. In order to understand the choices illustrated by Figure 8, we must accurately define the problem in data science terms.

![Figure 8: Machine Learning Algorithm Pipeline](image)

The first problem solved by demand forecast from passengers either from bus stop of mobile app is known as a “Time-Series Prediction”. In our literature review, we encountered plenty of systems that solve time-series prediction problems [11], [12]. KLM Air France, for example, uses a Long-Short Term Memory Recurrent Neural Network (LSTMRNN) to flatten a time scale and use it as an input vector to the RNN. This, alongside other features such as weather, date, time, and season all affected
the final prediction. This is similar to the approach in the proposed system. The input data includes: day of the week, timestamp of historical demand, historical source and destination coordinates, historical polylines, weather, and calendar events (national holidays, major sporting events). As illustrated in Figure 9, the demand data takes on two forms: (1) a single value total representative of the total demand in the network in that period; and (2) an Origin-Destination (O-D) matrix detailing the fraction of the total demand present at each node in the network. These inputs are processed and a prediction is made for the total number of requests across the network as well as their distribution O-D matrix. The forecasted passenger demand OD(DmanOD[ Gn]) matrix is then combined with a stored, node-to-node distance O-D matrix (DistOD[Net]) to calculate a passenger load factor (plf) function.

Figure 9: Transactional Flow Diagram of Data Pipeline

This plf is parsed into the optimizer module where the algorithm will attempt to find a minimum (optimum) number of resources (supply) represented by set of buses, and the respective node -sequences, required to satisfy this demand. Figure 10 illustrates a generic O-D matrix that will be used in the proposed solution to encode the distances and the passenger demands for every node-to-node interaction in the network.

The second problem is known as a Vehicle Routing Problem, (VRP) whereby a given commodity must be delivered from a depot to a set of locations along an optimal route with the optimum number of vehicles as constrained by their capacity. There are plenty of variations on the traditional VRP; ones including multiple commodities with different sources and destinations such as the Multi-Compartment VRP(MCVRP), the Multi-Depot VRP with Pickup and Delivery requests (MDPDR) [18], and Bus Transit Route Network Design Problem (BTRNDP) [19]. These problems have typically been solved using three methods: Linear Programming, Heuristics, and Meta-heuristics. Historically, the Genetic Algorithm (GA) method was favored due to the NP-Hardness of the problem and GAs ability to find a suitable approximate solution with in a reasonable amount of time. However, a recently developed algorithm called Particle Swarm Optimization (PSO) has proven to be much faster at converging with a strong resistance to getting stuck in local minima [15]. While PSO is a similarly iterative algorithm, it utilizes intelligence from a swarm of particles to determine the best direction for convergence in a search space. PSOs operating principle can be described as follows:

1. Each particle in the swarm is initialized at a random point in the search space;
2. Each particle is assigned 3 vectors, one initialization velocity in some random direction, the particles own best known position from the previous iteration, and the best position found globally by the best particle;
3. These position and velocity vectors then move each particle by the resultant vectors magnitude and direction;
4. Step 2 and 3 are repeated until convergence. This method is different to GA in that it can cover quite large search spaces very quickly and does not require the problem to be differentiable (does not use Gradient Descent). Each particle is representative of a candidate solution; once a feasible solution has been discovered, a particle can be decoded to a set of node sequences that can then be assigned and mapped. Once the problem is formulated as a VRP, with the objective function and the set of constraints, PSO can be deployed as a solver to determine the optimum routes and the minimum number of vehicles required as demonstrated in Figure 11.

Figure 9: Transactional Flow Diagram of Data Pipeline

Figure 10. Example O-D Matrix

Figure 11: Example Solution for Multi-Depot Capacitated VRP
The VRP formulation chosen in this project is the MDPPVRP referenced in [18] as it allows for multiple pickup and drop-off locations as would be observed in a transit network. Similarly, the particle decoding method used in [18] will be utilized.

IV. SOFTWARE IMPLEMENTATION

A. Objective Mapping

The software specifications mentioned in section III described the delivery of a mobile application experience for passengers as well as a central management system (CMS) for operators to view and manage the logistics of their fleet. There are multiple approaches to this design problem, a web-application approach was chosen for the proposed system as it allows the development of a single codebase that can be deployed across both mobile and desktop platforms simultaneously using Progressive Web App (PWA) technology. The following stack of web technologies was selected to rapidly develop and deploy the application, as seen in Figure 12.

![Figure 12: Web Application Stack](image)

The web application is built on Typescript; which is as upper set of JavaScript primarily used to provide static typing, classes, and interfaces. The benefits of this is that data types are checked during compile time instead of runtime which reduces the number of runtimes errors and greatly increases the speed at which the application can be developed. The supporting tooling used around Typescript is also vastly superior with better code autocomplete, type info, automated documentation, and error detection. Ergo, with real-time feedback from the IDE, less time is spent debugging and refactoring code. Not only does the language have a massive support community, but it’s also used by top industry leaders such as Google and Reddit which makes it an excellent choice as the foundation of our proposed system development. The second key foundation to this application is the server architecture. Since the application needs to serve information dynamically tailored to each user, the use of a traditional REST architecture and developing every single possible API endpoint would be incredibly cumbersome and complex in comparison to the chosen technology: GraphQL. GraphQL is a query-driven API endpoint used to package only the necessary data into one request to the server. This avoids technical debt in the long-run as it reduces the possibility for under- as well as over-fetching. The best implementation of this technology comes packaged in a library called Apollo GraphQL. Apollo manifests itself on the NodeJS server as the hosting entity and acts as a liaison between the front-end (with the Apollo Client) and the PostgreSQL database. PostgreSQL was used for its excellent support for geographic objects such as LatLng (Latitude/Longitude) coordinates and is incredibly powerful for GIS applications when teamed with libraries such as PostGIS. Our main choices of tooling were:

1. Type graphQL, a framework used to make database schema definitions, user permissions, and authentication as easy as defining classes; and
2. Type ORM, an Object-Relational Mapper which abstracts the underlying SQL commands into a set of logical commands used to speed up development.

Finally, React.js was the obvious choice for our frontend framework due its flexibility, ability to serve dynamic HTML components, its out-of-the-box support for a mobile experience with PWA service workers, its ability to transform in to a native mobile application using React-Native, as well as the ability to deploy natively on desktop environments with Electron. Furthermore, React enables a declarative approach to design with emphasis on re-usable, modular, styled functional components.

As aforementioned, the above design decisions were made to not only meet the system design objectives, but also to streamline and optimize the developer’s experience. This was necessary to ensure rapid prototyping of features, test-driven development, and a quick delivery time. Therefore, to sum up, the TRAP stack operates as follows:

1. ReactJS frontend provides all user-facing UI elements.
2. Actions & form inputs are parsed into the Apollo Client which communicates with the server through a set of Mutations & Queries
3. NodeJS server implements the Resolvers necessary to fulfill these queries & mutations
4. PostgreSQL is used for the database due to its reliability and excellent support for Geo-spatial objects
5. Typescript is used as the language of choice to ensure type-safety and catch errors at compile-time, therefore, TypegraphQL is leveraged to define the database schema and Type ORM is used to provide the necessary CRUD APIs for the database.

B. Use case scenarios

The below user journey diagrams shown in Figures 13, 14 describe the exact functions needed by each user and their respective implementation on the CMS.

![Figure 13: Mobile application user story](image)
The steps taken by a mobile user (passenger) as illustrated in Figure 13:

1. A typical mobile user will login by passing in their credentials; this sends a login mutation to the server which resolves to verify if the credentials match any available in the database. If this is a new user, the register page takes the required fields and parses them to the backend using a register mutation.

2. The next step for the user is to find a ride by searching for the nearest upcoming trip that passes through their pick up and drop off locations. This data is captured through a mutation and a basic kNN search algorithm (k-Nearest-Neighbor) is used to determine the nearest available stops to the user’s chosen locations.

3. The search returns a set of all available upcoming trips, ordered chronologically. The user can then select the trip that best fits their needs. If no such trips exist, the user is presented with a “request trip” button to capture this unfulfilled desire – this is the key indicator for demand that will be leveraged in the Machine Learning (ML) stack.

4. Depending on which trip a user selects, they are presented with one of two options: (1) If the trip is “smart” i.e. contains the hardware implementation (bus onboard kit for location tracking), the user is asked to reserve a seat and can then track the bus’s location on the map. (2) If the trip does not contain the onboard hardware kit, the trip details are simply displayed to the user with no further possible input (static display). Note: This detail is specific to the prototype implementation.

5. Once the user boards the bus and swipe their RFID card, other users will exhibit an update to the capacity counter to represent the current number of passengers on-board this trip.

The user journey for the operator is quite different and is illustrated in Figure 14 as follows:

1. Since operators are major entities, they require manual registration and therefore the only possible entry to the dashboard is by providing the correct credentials.

2. Once entered, the operator is presented with a multitude of possibilities. Every entity in the database can be controlled through the dashboard and Create/Read/Update/Delete (CRUD) functionality is available using the GUI. An operator will typically setup their digital fleet structure through a series of create operations (to create Bus, all available stops, register drivers, and create static lines (if any exist)).

3. Upon completion of this setup process, the operator may begin to create and schedule trips as necessary. For this operation, the operator may choose to define the exact stops covered by a trip (black arrows), or, if it is a recurring trip, they may choose to create a trip and assign the set of stops predefined in a line entity (red arrows).

4. Once a trip has been created, the operator can schedule the start time of the trip and track the exact location of the vehicle (as made possible with the Update Location mutation).

C. Adding CRUD operations

As illustrated in Figure 15 all the mutation and queries necessary for the successful implementation of these user journeys and is formatted as follows:

< query/mutation name >(…arguments) : < return type >

A subscription is GraphQLs version of a WebSocket, i.e. a full-duplex communication channel over a single TCP connection. This is used to create connections to each smart bus such that the vehicle can constantly update its location. One query of note is of course the “tripsByLocation” query which takes a search radius in kilometers; pickup and drop off locations as Latitude/Longitude (LatLng) objects; and is capable of returning a list of upcoming trips that intersect with both the requested pickup & drop off locations. This search algorithm leverages KDBush, a library for spatial point indexing built on k-Nearest-Neighbor. It operates as follows:

Figure 14: Operator user story

Figure 15: Database Interactions
1) Define a time window between the current moment and 2 days from now
2) Query DB for a list of all trips scheduled in the upcoming time window
3) Parse the received list of stops into GeoKDBush instance
4) Run geokdbush.around(index, longitude, latitude, [maxResults, maxDistance, filterFn]) which returns an array of the closest points from the given location ordered by increasing distance
5) Store list of stops into Near Pickup Location array
6) Filter list of trips to ones with stops in Near Pickup Location
7) Query DB for list of stops for this filtered list of trips
8) Parse received list into second GeoKDBush instance
9) Run geokdbush.around() with drop off location as argument
10) Store list of stops into Near Drop Off Location array
11) Query DB for trips By Ids with filtered list of stops in Near Drop Off array.

This entire workflow has been tested and will return a list of up to 1000 closest points within 8ms, and can index over 130,000 points in less than 70ms.

V. DATA PIPELINE IMPLEMENTATION

Tensor flow [20] was chosen due to its large support community and its ability to provide a production-ready deployment of the algorithm built on the platform. Since Tensor flow supports Ker as [21], the platform makes it easy to build the required neural networks for demand prediction. Furthermore, the implementation can then be ported to Tensor flow JS for easy integration with the CMS. The predictions made by the RNN can then be passed into the optimization algorithm written using NumPy [24], as that library allows the direct translation of the mathematical formulation (MDPDR-GVRP) to python code. The output of this optimizer can be passed back into the CMS in the form of a Trip Creation HTTP request made to the graphQLend point as shown in Figure 16. The following methodology was followed in order to successfully implement the data pipeline from start to finish:
1) Planning and Setup,
2) Data Collection and Labeling,
3) Model Training and Debugging

A. Planning and Setup

This foundational phase ensures everything is prepared to begin implementation of our models. First, the data pipelines goals must be defined, a set of assessment metrics must be chosen, a baseline for comparison must be set, and all there sources necessary for the code base must be gathered. The pipeline hopes to, based on historical and real-time data, predict upcoming demand in both magnitude and trajectory, and then create efficient routes to satisfy this predicted demand. With these goals in mind, the main problem can be broken down to a set of sequential predictions, and an optimization problem. The prediction models will be assessed based on their precision with respect to actual demand, and the optimizer will be compared to the baseline static route metrics. The specific metric which aims to be maximized is the Passenger Load Factor (PLF), which encompasses the ratio of cost per passenger km to cost per vehicle km. A maximum value here ensures that vehicles are not underutilized (i.e. driving routes with empty seats). Choosing a baseline is difficult due to the lack of publicly available metrics on public transportation here in Egypt, however, some local references for current static system operates at a peak efficiency of 87.2% if the total number of enrolled passengers were to utilize the service very single week-day. Real day-to-day operating efficiency is realistically much lower. Finally, the prerequisites for the Implementation of this solution will include the use of: the publicly available PySwarms and Geohash libraries; the Keras API through Tensorflow for neural network modelling; the excellent Pandas library for data handling and manipulation; and of course MatPlotLib for data visualization.

B. Data Collection

Since the proposed system includes the underlying infrastructure, the datasets architecture is defined by the database schema highlighted in section 3. However, due to unforeseen complications, we were unable to collect enough data points to demonstrate the effectiveness of the chosen pipeline design without effectively overfitting. Therefore, in order to combat his issue, trip data from New York City Taxi & Limousine Commission (NYC-TLC) will be used instead. TLC provides data points for every yellow & green cabs, as well as for hire vehicles. Due to the sheer amount of data present, the yellow and green cab data was decidedly sufficient. These records include relevant columns such as: pickup and drop off date-time stamps, pickup and drop off coordinates, total trip distance, and passenger count. The proposed approach requires the data be broken down into two, similarly time-binned, versions of the dataset. One version of the dataset includes the total number of passengers across the entire network, and the second contains geohashed pickup and drop off coordinates. Geohashing is a form of encoding for geospatial coordinates(lat/lng) in the form of English characters into what are called geohashes. These geohashes can be anywhere from 1 to 12characters long with finer granularity and precision achieved with longer strings of characters. The precision of geohashes is illustrated in table 1. Although finer precision can be achieved, we generally found that 6 characters represent an adequately precise geohash for this application.
The purpose of geohashing is two-fold; on the one hand it creates natural regions of divide which serve as clusters for incoming demand. Secondly, they allow the use of a novel approach to this problem, which is the use of a character-prediction recurrent neural network to predict upcoming source and destination regions.

C. Model Training and Debugging

For this application, we are leveraging two fairly well-known LSTM-RNN implementations: Single-variable time series forecasting LSTM, and Tensorflows char-RNN inspired by Andrej Karpathy [20]. For the former, we begin by visualizing the dataset to gain some information about the data trend and seasonality this is demonstrated in Figure 17. After loading the data into a data frame, it is normalized using a Min-Max Scaler preprocessing class from the SciKit Learn library and split into 67-to-33% for training and testing respectively.

Figure 17: Visualized Passenger Data Results

VI. DATA PIPELINE IMPLEMENTATION

From there, the dataset is split again from its current NumPy array form into a sequential dataset with a number of previous time steps to be used as input variables to predict the next time period. The default configuration of this function creates a dataset where x is the number of passengers at time t, and y is the number of passengers at t+1. The network is then comprised of a single-input layer, single-output layer, and a single hidden layer with four LSTM blocks. After running the model for 100 epochs, we achieve a training score of 22.93 RMSE and a test score of 47.53 RMSE. The results are visualized in Figure 18.

Figure 18: Visualized Training-Testing Data Results

Figure 19: Next Character Prediction LSTM Architecture

Tensorflows Character RNN takes the geohashed pickup and drop off locations as inputs and creates a dictionary of all the unique characters found in the first 250 lines of the dataset. Once each unique character is mapped to an index, a maximum length sequence is determined and the data is split into a training and test dataset. For this application, the dataset is iterated over character-by-character and the target output character is a function of the previous steps, as context, in addition to the current input character. From there, the model is built as a function which takes the total length of unique characters (vocab size), the input layer size dimensions (embedding dim), number of hidden-layer RNN units, and the batch size for quicker processing. Since the output layer is the dimensionally identical to the input layer, its represented as with tf.keras.layers.Dense(vocab size). Finally, a softmax loss function is used to train the model, along with the Adam Optimizer which is preferable here to classical stochastic gradient descent.

VII. PROPOSED SYSTEM VALIDATION AND TESTING

After testing each function individually on the GraphQL playground, it was time to connect the backend server to the
front-end and test the returned data for proper parsing to the end user. The passenger will typically start by logging into their account. During this process, the user types in their credentials into HTML form inputs and hits the login button which invokes the login mutation. This mutation receives the input, hashes the password, and checks the user’s input against the entries in the database. If a match is found, the mutation returns the users ID, first name, and last name. The returned data is stored in local context on the client side such that the user’s experience is tailored to them. The user is then redirected to the home page; a map view with a marker at their captured location, along with a search bar. The search bar used is part of the GeoSuggest library and will provide accurate autocomplete suggestions for a user search query. Once a user searched for a destination, they are redirected to the search view. Here, their current location coordinates along with the geocoded coordinates of their search location are passed into a search query. The search query carries out the algorithm outlined in section 3 and returns a trip JSON object with the following parameters: trip ID, scheduled start time, a list of the stop addresses associated with the trip, driver details, and a tracking identifier which denotes whether a trip can be tracked or not (this specific parameter is only relevant to the proof of concept deployment on the university bus fleet, ideally all trips would contain the hardware necessary to be tracked). The search view contains two geo suggest search bars for a user to change their desired pickup and drop off locations; this automatically re-fetches the query from the server. Since in this application prospective demand needs to be captured as well, a second function on this page exists denoted by the button “Request a Trip”, which invokes a mutation that captures the current timestamp and users desired pickup and drop off location. This demand is compiled into the dataset used by the prediction models in order to enhance future prediction sand create better suited routes. This is the main integration point between the mobile application front-end and the data pipeline, whereby these captured requests can be preemptively accounted-for in future predictions.

Upon selecting a suitable trip, the user is redirected to one of two very similar pages. For static trips (smart bus is not enabled), the user is only shown the trips route on a static map as well as some auxiliary trip details; such as driver name, vehicle license plate, and scheduled time. Tracked trips show the exact same details, except they also add a button to reserve a spot on the bus. This button invokes a reservation mutation which adds this user to the trips reservation list. The reservation status is set to Planned by default, and this is one of the main integration points with the OBU (Smart Bus Onboard Unit).

The user is then redirected to a tracking page where they can view the vehicles live position on a map. This is the second main integration point with the OBU; whereby the GPS modules coordinates are displayed on every reserved users tracking view. Once a bus has reached the user, the user boards the vehicle and swipes their RFID card, at which point their status is changed to board. The driver’s view contains a sidebar with a list of the reserved passengers that is automatically updated as soon as a passenger reserves a seat. Once a passenger boards, an icon is made available to the driver which invokes an exit mutation to de-board users upon arrival at their destinations. Boarding and exiting also resolve to capture the exact moment at which both of those operations were invoked, this allows us to capture more accurate pickup and drop-off times which are vitally useful for the prediction algorithms.

Figure 20: Driver frontend interface

VIII. CONCLUSION

To conclude, the need for demand-responsive mass transit is apparent. Buses are an excellent candidate for this, especially in Egypt, where the majority of the population is made up of tech-savvy youth. The main challenges faced by the traditional bus system include lack of digital infrastructure, outdated distributions and schedules, as well as outdated or inefficient routes. The proposed solution attempts to tackle these problems by operating in the fuzzy domain, making predictions about passenger demand and preemptively customizing and optimizing routes and schedules to as many passengers as possible. This is in addition to the light-weight mobile application and the RFID payment system which will enhance the overall user experience. The benefits of such a system would extend across social, economic, and Environmental improvements. While there have been similar attempts at solving the various sub-problems present within the traditional bus system, very few have attempted to re-imagine the overall experience and challenge the culturally established norms about the bus system.

IX. ACKNOWLEDGMENT

This paper and the research behind it would not have been possible without the engagement and support of MSA student Bishoy Maher whom I would like to thank and appreciate all his efforts, insights and expertise that greatly assisted through the course of this research. Finally, it is with true pleasure that I pay my sincere gratitude and acknowledgement for Bishoy’ contributions to achieve the research objectives as planned.

REFERENCES


[27] T. Schmidt, R. Philipsen, and M. Ziefle, "From v2x to control2trust - Why trust and control are major attributes in vehicle2x technologies," International Conference on Human Computer Interaction, in press.


[33] Nor Haízan, W., Mohamed, W., Salleh, M.N.M., Omar,


