QARTA: An ML-based System for Accurate Map Services

Mashaal Musleh[1], Sofiane Abbar[2], Rade Stanojevic[2], Mohamed Mokbel[1,2]
[1]Department of Computer Science and Engineering, University of Minnesota, USA
[2]Qatar Computing Research institute, Hamad Bin Khalifa University, Doha, Qatar
(musle005, mokbel)@umn.edu (sabbar, rstanojevic)@hbku.edu.qa

ABSTRACT

Maps services are ubiquitous in widely used applications including navigation systems, ride sharing, and items/food delivery. Though there are plenty of efforts to support such services through designing more efficient algorithms, we believe that efficiency is no longer a bottleneck to these services. Instead, it is the accuracy of the underlying road network and query result. This paper presents QARTA; an open-source full-fledged system for highly accurate and scalable map services. QARTA employs machine learning techniques to construct its own highly accurate map, not only in terms of map topology but more importantly, in terms of edge weights. QARTA also employs machine learning techniques to calibrate its query answers based on contextual information, including transportation modality, location, and time of day/week. QARTA is currently deployed in all Taxis and the third largest food delivery company in the State of Qatar, replacing the commercial map service that was in use, and responding in real-time to hundreds of thousands of daily API calls. Experimental evaluation of QARTA shows its comparable or higher accuracy than commercial services.

PVLDB Reference Format:
Mashaal Musleh, Sofiane Abbar, Rade Stanojevic, Mohamed Mokbel.
QARTA: An ML-based System for Accurate Map Services. PVLDB, 14(11): 2273-2282, 2021.
doi:10.14778/3476249.3476279

1 INTRODUCTION

The proliferation of GPS-enabled devices and the need for basic map services (e.g., routing, store finding, and traffic information) result in having a variety of commercial map services (e.g., Google Maps, Bing Maps, HERE, and Waze) that are ubiquitously used worldwide. On top of this, mapping services have become an integral component of other widely used services, including ride-sharing (e.g., Uber and Lyft), food delivery (e.g., Uber Eats and Doordash), and last-mile delivery (e.g., Amazon, UPS, and FedEx). All these services rely on the basic functionality of routing, which recommends the best route from one location to another. Hence, several research efforts were dedicated to come up with more efficient execution of shortest path queries [44, 46, 58, 75, 77, 81]. However, practically speaking, the execution time is no longer the bottleneck of these queries. The real challenge is accuracy, which heavily depends on the quality of the underlying map. If a company runs the most efficient algorithm on top of an inaccurate map, the result will not be acceptable. Another company with a more accurate map would offer better services even if it has a less efficient algorithm, yet still within an acceptable real-time response.

As a result, recent work has focused on building and updating the road network map, especially for regions in the world where maps are not easily available. Examples of such work include constructing the road network from drivers' GPS traces [7, 14, 65, 69] or satellite images [6, 15, 71]. Unfortunately, these attempts did not solve the routing problem mentioned earlier as the focus is mainly on having a more accurate *topology* of the map, while routing services rely more on the weights of the network edges (i.e., road segments). An accurate topological road network without accurate edge weights will be of no use to basic map services, routing, range queries, and others. Due to their direct dealing with customers, commercial map services have realized this fact, and have focused their recent efforts on building maps with accurate edge weights, which can be seen in the traffic layer in Google, Bing, or Apple maps. However, unfortunately, commercial map services suffer from two main issues: (a) the map accuracy is dramatically degraded in areas that either lack enough data or where maps are frequently updated, and (b) beyond a certain limit of API usage, customers have to pay a considerable cost for using map services, which is a major burden to small and medium enterprises.

This paper presents QARTA [1] ; an open-source full-fledged system that employs machine learning techniques to provide highly accurate map services. QARTA is currently responding to hundreds of thousands of daily API calls coming from its actual deployment in: (a) all Taxis in the State of Qatar (around 4K vehicles), and (b) a food delivery company (around 3K motorbikes). In both cases, QARTA has successfully replaced commercial map services that were in use for long. QARTA was triggered by a real need from the Taxi and delivery companies that not only the commercial service is expensive, but more importantly, it is outdated in both the topology and traffic metadata. The main reason for having such stale maps is that the underlying map is rapidly changing due to major country-wide constructions [4]. QARTA goes beyond supporting the basic routing service to accommodating other important queries such as range and k-nearest neighbor queries. All such queries would need to report an Estimated Time of Arrival (ETA) along with each returned result. The ETA accuracy highly depends not only on the accuracy of the underlying map but also on the understanding of the contextual error margins of the query result. QARTA takes such error margin into account before returning back the full answer.

QARTA is built with two principles in mind: (1) *Map-centric*. QARTA believes that a key point to the success of all map services

This work is partially supported by the National Science Foundation, USA, under Grant IIS-1907855.

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476279

[1]QARTA comes from both the Arabic word Kharta (map in Arabic) and the Latin word Cartography. We then replaced the first letter by Q as a reference for the State of Qatar.

is having an accurate map. Hence, a major part of the system is geared towards constructing an accurate road network in terms of both topology and edge weights. (2) *Query Calibration*. QARTA believes that the answer of any map service would still need to be calibrated based on contextual information (e.g., transportation modality, time of the day/week). QARTA employs machine learning techniques to calibrate its query answer. QARTA feeds its machine learning models with real data obtained from the running vehicles and motorbikes that use it.

QARTA completely separates map construction from query processing, where updating or constructing the map is a background process that does not affect the query response time. Map construction digests input live GPS traces through a light process that continuously and accurately updates the map. Meanwhile, query calibration relies on models built offline. Hence, it does not put any overhead on the underlying query processing, making QARTA response time as real time and scalable as the underlying query processor. A resilient feature of QARTA is that it does not come up with new query processing or indexing techniques. Instead, it significantly boosts the accuracy of these techniques by feeding them with an accurate map, and calibrating their answers. Experimental evaluation of QARTA, based on real data and actual deployment, shows that QARTA has significantly higher accuracy than currently available open-source solutions, and has a comparable to slightly better performance than commercial map services. All comes with a real-time response time.

Section 2 describes QARTA system architecture. The three layers of QARTA, *data layer*, *map making*, and *query calibration*, are described in Sections 3-5. Experimental evaluation is in Section 6. Related work and conclusion are in Sections 7 and 8.

2 QARTA ARCHITECTURE

Figure 1 depicts the map-centric QARTA system architecture, where the map lies in the center of the architecture, indicating that map construction is: (a) a major part of QARTA, and (b) isolated from the query processing. QARTA is composed of three layers, namely *Data layer*, *Map Making layer*, and *Query Calibration layer*, described briefly below. The gray modules in Figure 1 represent the new components designed only by QARTA, while other modules employ off-the-shelf solutions. QARTA visualization interface is described in details in its demo paper [3].

Data Layer. This layer is responsible for all data collection and preparation procedures as well as the infrastructure storage for all such data. The layer takes various forms of input data, including maps, Points of Interest (PoI), trips, trajectories, and the QARTA map itself. The input goes through data cleaning and sampling procedures, where cleaned and sampled data are all internally stored and indexed in typical spatial data warehouses. The *data layer* feeds the *map making* and *query calibration* layers by the data needed to construct the map and build the calibration models, respectively. Details are in Section 3.

Map Making Layer. This layer is responsible for building QARTA map, which is the main asset that QARTA has, and is the main reason behind its accuracy. To QARTA, building a map goes beyond finding the map topology, as it also includes finding the road edge

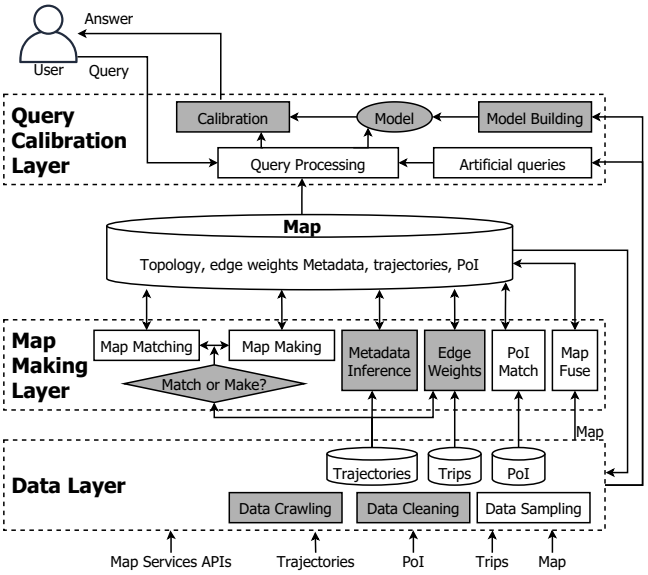


Figure 1: QARTA Architecture

weights and metadata. In addition, this layer makes a clear distinction between whether the trajectory data should be matched to the map or used to correct the map. Details are in Section 4.

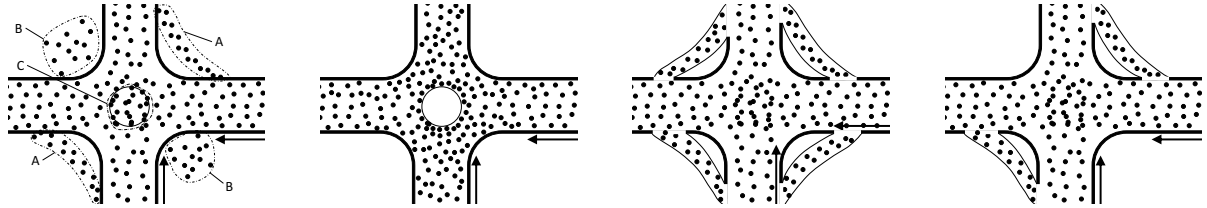
Query Calibration Layer. This layer supports user queries, including shortest-path, range, and k-nearest-neighbor queries. It significantly boosts the accuracy of existing spatial query algorithms by: (a) feeding the algorithms with a highly accurate map, and (b) calibrating answers by understanding the error margin of the deployed algorithms under various contexts, including transportation modality and time of the day/week. Details are in Section 5.

3 DATA LAYER

The *data layer* is responsible for data digestion and collection efforts while laying out the storage infrastructure. Some of QARTA input data are already rich and clean (e.g., official maps or sanitized list of Points of Interest (PoI)), hence we just store it in off-the-shelf spatial data warehouses. Meanwhile, a major part of QARTA input is noisy (e.g., inaccurate GPS readings, missing data, or misinterpreted data), hence QARTA develops its own *Data Cleaning* module that produces a cleaned version. For parts of the map that is short in data, QARTA employs its own *Data Crawling* module.

3.1 Data Cleaning

Though trajectory data is crucial for inferring road and traffic information, collecting them in the wild results in corrupted data that would harm the machine learning models. Although there is a plethora of systems and techniques for general data cleaning [38], they all fall short when dealing with spatial and trajectory data, mainly due to the spatial data distinguishing characteristics [29]. Recent attempts to trajectory data cleaning [22, 45] mainly focus on map matching trajectory data, which is another spectrum of problems that will be addressed in the *Map Making* layer. Hence, QARTA employs its own rule-based data cleaning module that addresses specific trajectory problems that came out from its actual deployment. Examples of such problems and rules include:



(a) Raw Data (b) Map Matching (c) Map Making (d) QARTA Match or Make

Figure 2: Match or Make

Rule 1: Trajectories with a stop. *If a trajectory T encounters significant speed reduction, while other trajectories on the same time and road reported normal speed, split T into T_1 and T_2 by removing the segment(s) form T that include the speed reduction. If the number of points in T_i is under a certain threshold, remove T_i.* The rationale is that we have observed a nontrivial fraction of raw trajectories collected by our fleet contain traffic-unrelated stops, e.g., a passenger asks the driver for a quick stop by a store. Including such trajectories in our dataset would give wrong traffic information, hence we had to either completely remove it, or just remove the erroneous segments. We go with the latter.

Rule 2: Unrealistic points. *For a point P_i in trajectory T, if the speed from the previous point P_{i-1} is above a certain threshold, remove P_i from T, and connect P_{i-1} to P_{i+1} directly.* The rational is that an unrealistically high speed between two consecutive points is an indication of a wrong GPS reading. So, we simply remove this point.

Rule 3: Missing points. *If there is a significant time difference (above a certain threshold) between two consecutive points P_i and P_{i+1} in the same trajectory T, split T into T_1 and T_2 by removing the segment (P_i,P_{i+1}). If the number of points in T_i is under a certain threshold, remove T_i.* The rationale is that a significant time difference between points is an indication of a missing point in between. Ignoring this would result in an inaccurate trajectory.

All rules rely on setting various threshold parameters. Higher thresholds get higher data quality but would miss real non-ideal scenarios. We currently set these thresholds manually. Finding the best tuning parameters is in the plan for next release of QARTA, and beyond the scope of this paper.

3.2 Data Crawling

QARTA crawls several governmental and open-access sites to enrich its repository of maps and PoIs, which is a straightforward development process. For trajectories and trip information, QARTA collects it continuously through its real deployment. In cases where QARTA is newly deployed or there is shortage of trip information, we may acquire traffic information either from UBER-movement-like platforms [73] that provide access to limited datasets or explicitly from commercial services [68] by sending API routing calls with (*origin*, *destination*, *timestamp*). In case of commercial map services, it is crucial to optimize the number of API calls to accommodate low-budget enterprises. Hence, QARTA employs its own smart crawler that utilizes the limited API budget as follows: (1) *Weekday/Weekend future days*. To ensure that crawled data is not affected by any transient congestion of road closures, we issue all our API calls for someday a few weeks ahead in the future. To ensure good coverage throughout the week, we assign a ratio of our calls to be during the weekend. (2) *Different times of the day*. To get good temporal coverage, we split API calls over different times of the day based on missing data. (3) *Short trips*. A large majority of our API calls are for short trips because long trips usually involve main roads, in which we usually have enough coverage. Short trips give more information about secondary and tertiary roads, which are most needed. (4) *POI trips*. A fraction of our API calls starts or ends at a PoI, as these are more likely to be trip destinations.

4 MAP MAKING LAYER

The *Map Making* layer is responsible for building QARTA map, including road network, edge weights, road metadata, and PoIs. The layer is also equipped with a *map fusion* module that merges map updates to an existing map [70]. Since there is already a plethora of techniques for building the road network and map matching (see Section 7), QARTA just employs state-of-the-art of these techniques. Meanwhile, QARTA identifies and addresses three main bottlenecks that have the most impact on accuracy, though largely overlooked by current research efforts,. These modules are *Match or Make* (Section 4.1), which smartly decides whether we need to deploy map-making or map-matching, *Edge Weight Inference* (Section 4.2), which finds road segments edge weights, and *Metadata inference* (Section 4.3), which finds road segments metadata .

4.1 Match or Make

Map making techniques [1] have an implicit assumption that GPS traces is ground truth, and use it to create/update the road network. Meanwhile, map matching techniques [12] have an implicit assumption that the underlying road network is ground truth, and match GPS traces over it. Both approaches may produce inaccurate results as both GPS traces and road network may suffer a high degree of uncertainty [11, 40]. Figure 2a gives a real example of a roundabout in Doha, Qatar, that was recently converted to a bridge, yet the map is still not updated, along with vehicle GPS traces. Figure 2b gives the result of applying a map matching technique, where point groups A and C are mistakenly classified as wrong points as they do not match the stale underlying map. Figure 2c gives the result of applying a map making technique, where point group B mistakenly produces non-existed road exits. QARTA avoids such problems through its *Match or Make* module. Given a road network R and GPS points P, this module decides on the part(s) of the map where R is more accurate than P and vice versa. For parts where R is more accurate, we call *map matching* to match P on R, otherwise, we call *map making* to update R based on P. Figure 2d gives the result of applying QARTA *Match or Make* module. It identifies that: (a) Points A and C are accurate and uses them to update the map, (b) Points B are inaccurate and matches them to the map. The module is composed of the following four steps:

Step 1: Finding accurate points and roads. We find $P_{Acc} \in P$ and $R_{Acc} \in R$ in three iterations: (1) Map match P and R and initialize P_{Acc} and R_{Acc} to points and roads that (almost) match perfectly. (2) Remove from R_{Acc} those roads with low number of matching points. (3) Remove from P_{Acc} those points that were matched on any of the removed roads. The final P_{Acc} and R_{Acc} present points and road segments that we are highly confident of their accuracy.

Step 2: Error injection. We aim to understand how does it look to have good points on bad roads and bad points on good roads. We select a representative set (with various road types and length) from R_{Acc} and inject various sorts of errors, including removing a road segment, shifting road coordinates, and reducing the road resolution, which will impact the accuracy of U-turns, highway exits, and sharp turn roads. We extract the set of good points on bad roads P_{Good} from P_{Acc} as the points that are in proximity of the roads with injected errors. Among the remaining P_{Acc} points, we extract P_{Bad}; a representative subset based on various factors (e.g., trajectory start/middle/end point, different matching scores, and type of matching road). We then introduce various kinds of inaccuracies (e.g., shifting point coordinates with a Gaussian error) into all points in P_{Bad}, making it a list of bad points.

Step 3: Feature extraction & model building. We match all points in P_{Acc}, P_{Good}, and P_{Bad} on R and record several kinds of features for each point, including number of good/bad points within a certain distance, distance from (and type of) previous and next points in the same trajectory, matching roads of the few previous/next points on the same trajectory, and matching scores with other road segments. We then run Random Forest Classifier [31] using the scikit-learn Python library [61] to build and train a model that maps the set of features for any point P to how good/bad it is.

Step 4: Match or Make. For each point P not in P_{Acc}, we know that P did not match well with any road segment, but we are not sure if this is because P is inaccurate or because the road it should match to is either missing or inaccurate. We go through two iterations: (1) we run all such points against the model built earlier to classify P as either bad or tentatively good point, (2) For tentatively good points, we check if they form some clusters, where we consider them definitely correct and use them to update the underlying map. For all other points, we add them to the list of bad points, for which we run a *map matching* algorithm.

4.2 Edge Weight Inference

Unlike edge length and maximum speed, that are static road edges attributes, and can be publicly available, accurate edge weights are usually inferred either through loop detectors [17], plate recognition [41], private GPS traces [36], or cell phone data [18], and are considered proprietary information. In addition, edge weights are usually presented as multiple values per edge (a.k.a time-dependent [20] or time-aggregated [24] graphs), where each value corresponds to a certain time interval. As existing research efforts for edge weight inference suffer from lack of scalability and overfitting (see Section 7), QARTA develops its own scalable and accurate *Edge Weight Inference* module. Given a road network topology, we find 168 weights for each edge, as one value for each hour of the week. Our module only needs very basic trip information, location and timestamp of origin and destination points of each trip τ, which is the 'lowest common denominator' of publicly available trajectory datasets. Then, we find the trip path $P_\tau = [e_0, \ldots, e_l]$ as a sequence of edges e_i by issuing a routing query to the off-the-shelf routing engine deployed in QARTA. For each edge e, with given length l_e (in meters) and *unknown* edge weight W_e (in sec/meter), the time to travel through e is $W_e \times l_e$. Hence, the time to travel through P_τ is $\sum_{e \in P_\tau} W_e \times l_e$.

Main idea. The main idea of our *Edge Weight Inference* module is to find the edge weights W per unit length that would make the time to go through each trip path P_τ as close as possible to the time difference between the origin and destination timestamps δ_τ [67]. Formally, given a set of trips Γ for a road network R, find the weights W_e of all edges in R that would minimize:

$$\sum_{\tau \in \Gamma} \Big(\sum_{e \in P_\tau} W_e l_e - \delta_\tau \Big)^2, \tag{1}$$

A direct solution to optimize this equation may result in zero or negative weights and/or suffer from over-fitting. Additionally, given hundreds of thousands of edges with unknown W's that need to be optimized for millions of trajectories, scalability is a major issue. Hence, we go through the following three tuning steps to come up with an alternative equation that we can solve using Constrained Ridge Regression analysis [32].

Tuning Step 1: Heavy edges inference. The main problem in equation 1 is that it allows each edge in the graph, regardless of its popularity, to act as a regression feature, which may lead to over-fitting as well as unnecessarily expensive computations. To avoid this, we distinguish between heavy (popular) edges, covered by a large number of trajectories and for which it is important to get highly accurate weights, and light edges covered by fewer trajectories, where we can afford having less accurate weights. We define the set of heavy edges H as the top k edges (default is 10K) in terms of the number of trips covering them. Our objective becomes finding the weights W of all edges in H. All other edges would have the same weight W_0 per unit length l. Hence, Equation 1 becomes:

$$\sum_{\tau \in \Gamma} \Big(\sum_{e \in P_\tau \cap H} W_e l_e + W_0 \sum_{e \in P_\tau \setminus H} l_e - \delta_\tau \Big)^2. \tag{2}$$

In addition to fixing the over fitting problem, this new formulation reduces the number of regression features by up to two orders of magnitude, which enables higher scalability.

Tuning Step 2: Heavy road detection. Many simplified map formats represent long (few hundred meters) roads by only their nodes that involve intersections with other roads. Then, one set of weights is needed for each edge between two intersection nodes. This is pretty inaccurate and does not fit our applications, where: (a) different parts of the same road could have different set of weights based on road curvature and width, (b) it is a common practice that passengers are dropped off in the middle of the road as many stores and houses lie on the road. Hence, QARTA uses very detailed map formats with large number of edges, which is also available in OpenStreetMap [55], where long roads are represented by a sequence of edges and nodes without intersections. Starting from that fine granularity, we find those subsequent edges that share road properties. To scale up Equation 2, we group each of such edges together as one *heavy road* with one weight W_g. Formally, we split the set of heavy edges H into r disjoint sets $H_1, \ldots H_r$, where

each H_g includes a set of connected edges with the same weight W_g. Hence, Equation 2 becomes:

$$\sum_{\tau \in \Gamma} \Big(\sum_{g: P_\tau \cap H_g \neq \emptyset} W_g L_g + W_0 \sum_{e \in P_\tau \setminus H} l_e - \delta_\tau \Big)^2. \quad (3)$$

where $L_g = \sum_{e \in H_g} l_e$ is the length of the heavy road H_g:

This reduces the number of unknowns to $r + 1$ (one weight for each of r *heavy roads* and one weight for all other *light* edges) and the number of model features by 75%, allowing higher scalability.

Tuning Step 3: Enforcing physical constraints. To avoid having zero or negative weights, we add a physical constraint that, for any heavy road g, $w_g \geq 1/maxspeed_g$, where $maxspeed_g$ is the publicly available maximum speed of g. To ensure that this constraint holds for all edges, we use Ridge regression regularization, where we tune Equation 3 by adding a regularization term that penalizes weights deviating from average speed:

$$\sum_{\tau \in \Gamma} \Big(\sum_{g: P_\tau \cap H_g \neq \emptyset} W_g L_g + W_0 \sum_{e \in P_\tau \setminus H} l_e - \delta_\tau \Big)^2 + \alpha \sum_{g} (W_g - \sigma)^2. \quad (4)$$

σ is the inverse of average speed of all trips. α is the regularization strength. A small α allows large weight variability and physical constraints violations. A very large α may put too much emphasis on the regularization term neglecting errors we strive to minimize.

Inferring edge weights. We divide all our trips based on their starting timestamp into a specified time granularity (default 168 hours/week). For each time granularity, we use scikit-learn Python library [61] for constrained ridge regression to find W that minimizes Equation 4. From our experiments, more than 99% of edge weights satisfy physical constraints. For the rest, we set edge weights to the minimum possible value $1/maxspeed$.

4.3 Metadata Inference

Several real-life applications, e.g., traffic modeling, driver behavior analysis, road safety, and telematics, heavily rely on map metadata such as the number of lanes, maximum speed, directions, and road types (e.g., highway, service road, bridge). Unfortunately, the availability of such metadata is very poor in most cities around the world [83]. One way to fill in missing data is to model the metadata inference as a Graph Convolutional Network [16, 34, 42], or any other ML technique. However, there are three challenges to address here: (1) feature engineering of metadata is crucial, regardless of the underlying ML model, (2) The variety of metadata types calls for going beyond a one-size-fits-all model, as each metadata type may need a different model/ML techniques, and (3) scalability is a major issue and may hinder the applicability of some models.

QARTA addresses these issues by framing metadata inference as a supervised learning problem, in which the task is to first find the *best* models that map road features to each metadata, then use these models to predict the metadata values for each road segment. To build such models, we go through two steps: (a) *Feature engineering*. For each road segment, we compute two sets of features: *structural* features that include road length, numbers of in/out junctions, and road curvatures, and *functional* features that include speed average and standard deviation, GPS points density, and distance to centerline. While *structural* features are computed as one value per edge, *functional* features are computed as one value per time granularity (e.g., hour) per edge. According to our comprehensive validation and testing, all these features affect metadata inference. (b) *Learning Kernel*. Given a target metadata L, for all road segments with known L, we form a set of annotated examples $\{(v_1, l_1) \dots (v_m, l_m)\}$ where v_i and l_i represent the feature vector and metadata value of road segment i. The annotated data is then partitioned into three separate datasets: *training*, *validation*, and *testing*. We experiment with different machine learning techniques, including logistic regression, support vector machines, random forests, boosting gradients, and deep neural networks, which are fine-tuned using training and validation datasets, to find the best performing model for each algorithm. The testing dataset is used to compare the inference accuracy of each algorithm and select the best one with its best parameters. Once we set on the best machine learning model (algorithm and parameters) to use for a given metadata, the model is stored in a database of models along with its key performance indicators, which can be used later for refinement or retraining purposes. The models can be deployed in batch or streaming modes. In the streaming mode, the machine learning model is wrapped in a RESTFUL API that users can query via different endpoints.

5 QUERY CALIBRATION LAYER

The query calibration layer is responsible for responding to query APIs from QARTA users. Our earlier version of QARTA only had the query part of this layer, which included off-the-shelf algorithms for shortest path, range, and k-NN queries. Yet, we had numerous users' complaints for inaccurate Estimated Time of Arrival (ETA), which is crucial for several applications, e.g., deciding on trip fares, scheduling multiple trips, dispatching a driver to a customer, finding k-NN or within a range PoIs. Hence, we equipped QARTA with the *calibration* process to fix the ETA issue, which is what sets QARTA apart from other map services.

The main idea is to continuously monitor, understand, and model the error margins of all deployed algorithms, and then use the model to calibrate the results [2]. Given a set of historical trips Γ, where each trip τ is (*origin*, *destination*, *start time*, *end time*), we send the first three attributes of each trip to our spatial query processor to estimate the arrival time (ETA). We then record the offset time ϵ, as the difference between ETA and ground truth end time. Then, we build a model $\mathcal{M}$ that maps each trip features to its ϵ. For a new trip T, we get its ETA from an off-the-shelf shortest path algorithm, then apply $\mathcal{M}$ to T features to predict ϵ and add it to ETA to produce a more accurate result.

It is important to note that $\mathcal{M}$ maps a trip to its ϵ not to its ETA. Having $\mathcal{M}$ for ETA would mean that for a network with N nodes, $\mathcal{M}$ would need to consider N^2 possible alternatives within its feature vector. This is not practical nor accurate for large networks (100+K edges), not only in terms of computations, but also in terms of datasets that cover every pair of nodes under various features. Hence, we opt to use *spatial zoning*, where the map is divided to Z zones (e.g., zip codes), where Z is usually 2-3 orders of magnitude less than N. Though this is manageable in terms of computations and datasets, reporting ETA between pair of zones would not be accurate. Hence we only report ϵ for the pair of zones. Finally, QARTA builds a separate model for each transportation modality and underlying algorithm, e.g., a model $\mathcal{M}_v$ for a fleet of vehicles, used by our Taxi company partner, and model $\mathcal{M}_m$ for a fleet of

motorbikes, used by our food delivery partner. It is important to have this distinction, as they encounter different error distributions. We also build a model for each shortest path algorithm we have. Our query calibration process is composed of the below phases:

Model Building: Feature Engineering. For each trip τ_i with offset ϵ_i, we build a feature vector v_i composed of several features including: (a) *spatial zoning*. The origin and destination locations are mapped to a set of non-overlapped spatial zones. We have observed that ϵ highly varies per source and destination zones. (b) *temporal zoning*. The start and end times of the trip are mapped to non-overlapped temporal zones across a whole week, as there is a significant change in ϵ based on the time of the day/week. (c) *Trip characteristics*. This includes the distance and duration of trips generated by the routing algorithm we want to calibrate.

Model Building: Training. The output of feature engineering is a feature vector v_i and offset ϵ_i for each trip $\tau_i \in |\Gamma|$. Our objective now is to find a model function M that maps v_i to ϵ_i while minimizing: $\frac{1}{|\Gamma|} \times \sum_{\tau_i \in \Gamma} L(\epsilon_i, M(v_i))$, where L is a loss function for the reported travel time. This ends up being a classical supervised regression framework, where we use the Gradient Boosting [23] tree-based method to solve it. We set L to be least-squares and fine-tune the Gradient Boosting Regressor to find the best combination of its hyperparameters, i.e., the number of trees and max depth of each tree. The output is a model $\mathcal{M}$ that is capable of predicting accurate traffic congestion offsets ϵ_i for any trip feature vector v_i.

Query Calibration. Whenever QARTA receives a shortest path query, we pass it to two simultaneously working modules: (1) *Query processing*, where an off-the-shelf shortest path algorithm is applied to get the query answer with ETA. (2) *Calibration*, where we extract the feature vector v from the query parameters, pass it to the model $\mathcal{M}$ that corresponds to the transportation modality and algorithm to get a calibration offset ϵ that we add to the query answer to report the final ETA. In this case, the calibration overhead is negligible. For range and K-NN queries, the calibration process would remain idle until the query processor computes the result items. We then create one feature vector for each result item and pass them in-bulk to our model $\mathcal{M}$ to adjust the ETA of each result item. The adjusted ETA may call for changing the ranking of the result items, or even calling the query algorithm again, if needed.

6 EXPERIMENTAL EVALUATION

All experiments are done using an actual deployment of QARTA server that daily receives 235K API calls and 977K GPS points, supporting all Taxis in Qatar (~4K vehicles) and a food delivery company (~3K drivers) [3]. For evaluation, we use the following three datasets: (1) Doha. 250K trips collected by us over a one month period through our taxi partner driving in the city of Doha (64K nodes and 148K edges), (2) Porto. 426K Taxi trips over three months in the city of Porto (35K nodes and 82K edges) [63], (3) NYC. 1.5M Taxi trips for a period of 6 months in New York City (250K nodes and 644K edges) [54]. For the three datasets, we only use the (*Origin*, *Destination*, *Start time*, *End time*) to represent each trip, which is the least common denominator for all publicly available datasets. We run all datasets through our data cleaning module (Section 3), which reduces the number of trips for Doha, Porto, NYC, to 195K, 360K, 1.2M, respectively, that we are highly confident about their accuracy. Since we have the ground truth travel time for each trip, we use the absolute/relative travel time errors as our accuracy measure. Unless mentioned otherwise, (a) we use the first 75% of our cleaned trajectories in chronological order to train QARTA for both edge weights and query calibration, and then test with the last 25% of the data, (b) edge weights and temporal zoning are built for 168 hours per week, while spatial zoning uses publicly available administrative zoning with 92, 79, and 2055 zones for Doha, Porto, and NYC, respectively [56], (c) QARTA uses the Open Source Routing Engine (OSRM) [57] for its off-the-shelf query library.

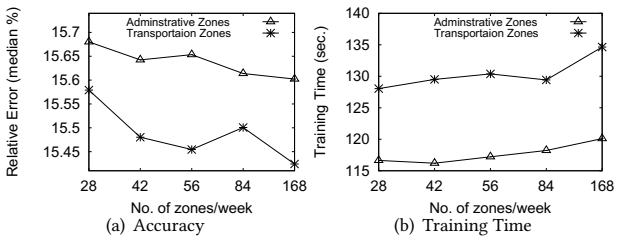


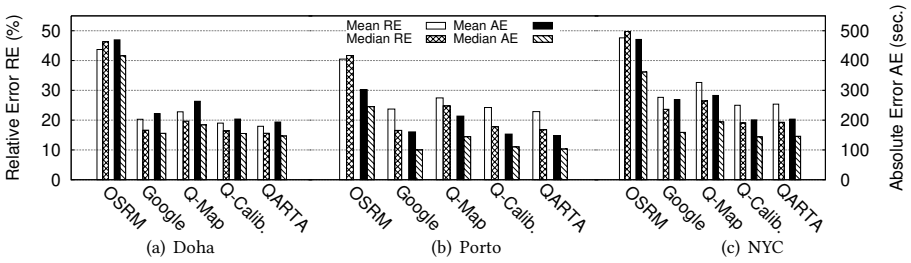
Figure 3: QARTA Spatial and Temporal Zoning

6.1 Setting QARTA parameters

This section aims to study and set the query calibration parameters in terms of the temporal and spatial zoning granularity. Figure 3 gives the effect of a finer granularity on the median relative travel time error (Figure 3(a)) and the model training time (Figure 3(b)) for Doha dataset. We make the granularity finer by increasing the number of temporal zones per week with values: 28, 42, 56, 84, and 168 (i.e., grouping the trips every 6, 4, 3, 2, 1 hour(s)). We also do the experiments for the two spatial zoning: (a) our default administrative zoning of 92 zones, and (b) a fine-grained transportation zoning of 1,839 zones defined by the Ministry of Transport and Communication in Qatar. It is clear from the figures that finer spatial and temporal zoning yield the highest accuracy, though suffer from higher training time. Since the training time is still manageable, we opt to use the finest temporal resolution (168 zones per week) as QARTA default value. For spatial zoning, though transportation zoning gives much higher accuracy, we decided to opt for using the administrative zoning in the rest of experiments, as it is more publicly available and accessible for other datasets.

6.2 Travel Time Accuracy

Figure 4 compares the overall accuracy of QARTA, Google Maps [25], and OSRM [57] in terms of the median/mean of relative/absolute error in predicting the trip travel time (with respect to the ground truth) for Doha, Porto, and NYC. For Google Maps, to ensure a fair comparison, we project each trip on the next month with a time that falls on the same hour/day/week as the actual trip. This is to make sure that Google Maps utilizes the right historical traffic information in its predictions. For OSRM, this does not matter as there is no traffic information. To test the impact of various QARTA components, we evaluate three versions of QARTA; Q-Map, which includes only the *Map Making* layer without any calibration (i.e., OSRM on QARTA map), Q-Calib, which includes only the *Query Calibration* layer with a map from OpenStreetMaps [55] (i.e., calibrating OSRM results), and the full QARTA. It is clear that



(a) Doha (b) Porto (c) NYC

Figure 4: QARTA vs Other Map Services

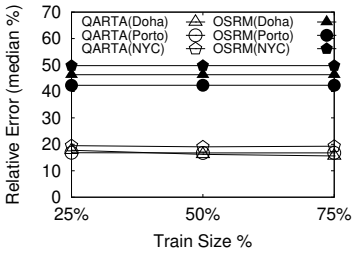
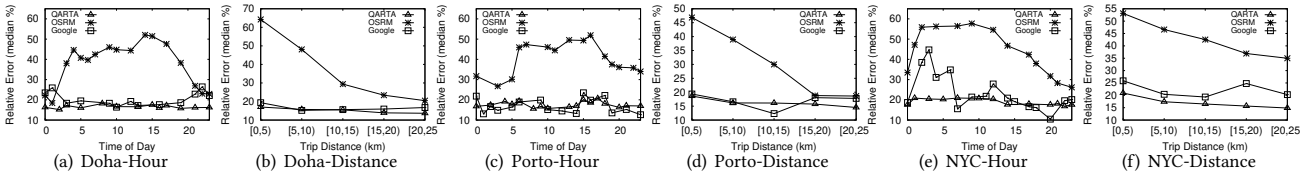
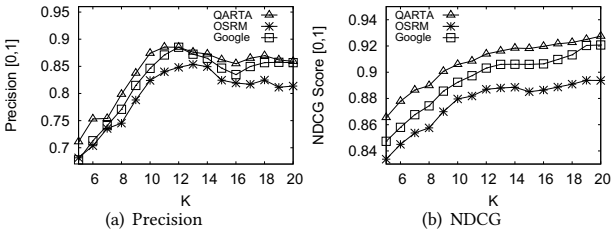


Figure 5: Accuracy vs Training Size



(a) Doha-Hour (b) Doha-Distance (c) Porto-Hour (d) Porto-Distance (e) NYC-Hour (f) NYC-Distance

Figure 6: Accuracy by Hour of Day and Trip Distance



(a) Precision (b) NDCG

Figure 7: K-NN Accuracy

OSRM has the worst performance in all datasets, mainly due to lack of traffic information, and hence its routing engine is based on road maximum speed. This shows how much the community needs open-source engines that are traffic-aware like QARTA, as routing with no traffic information encounters high errors that are close to 50% of the trip time. Meanwhile, Google Maps has way much better performance than OSRM with a median relative error of 17%, 17%, 24% for Doha, Porto, NYC. This is mainly due to the detailed traffic information that Google has access to. Q-Map, which basically injects QARTA traffic-aware map to OSRM engine results in a relative accuracy of 19%, 20%, 26% for Doha, Porto, NYC, which shows the great impact of having an accurate map in Open Source Routing Engines. Q-Calib, which just adds the calibration layer to OSRM engine achieves accuracy of 16%, 18%, 19% for Doha, Porto, NYC, making it almost as good as Google Maps. This also shows the great accuracy boost that QARTA can do to off-the-shelf routing engines. Finally, the overall QARTA, which includes both traffic-aware map and query calibration achieves an accuracy of 15%, 17%, 19% for Doha, Porto, NYC, making it even better than Google Maps. This positions QARTA as a strong candidate to replace commercial maps, which happened to our local Taxi and food delivery partners.

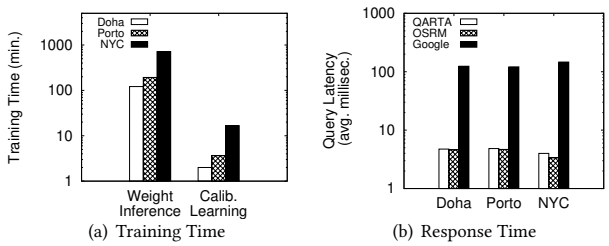
Figure 5 gives the effect of training data percentage of the whole dataset on the accuracy of QARTA for Doha, Porto, and NYC. OSRM is plotted as a straight line as it does not have training. QARTA has a very slight increase in accuracy, mainly because traffic is naturally periodic. So, training on one week is very similar to training over multiple weeks. Figure 6 gives the impact of the hour of the day and trip distance on the accuracy of OSRM, Google, and QARTA for our three datasets. For the hour of the day, OSRM dramatically fails during daytime, in which traffic has the most effect on trip duration. Meanwhile, QARTA consistently has a comparable or better performance to Google Maps throughout the day. This shows that QARTA was able to accurately infer traffic data in all traffic conditions. For trip distance up to 25km, QARTA and Google Maps have consistently comparable accuracy across all short and long distances. The shorter the trip the worse is OSRM, as short trips are more dependent on traffic information than long trips.

6.3 k-NN Accuracy

Predicting time of arrival does not only impact shortest path queries, but it also impacts K-NN and range queries looking for PoIs that are either k nearest to or within a range from a certain reference point. Figure 7(a) shows k-NN precision, i.e., number of items in the k-NN list that overlaps with the ground truth, of OSRM, Google, and QARTA using Doha dataset. Each point is computed as the average precision of 200 queries with diverse geographic locations, hours of the day/week. Unsurprisingly, they all achieve similar results. Given the PoI sparsity, we end up with very similar lists, even if PoI travel times are reported differently. However, the quality of k-NN answer is not only measured by what is in the list, but more importantly by the ranking of the results in the list. Hence, Figure 7(b) uses the Normalized Discounted Cumulative Gain (NDCG), as the quality measure of the K-NN list. NDCG [39] is a widely used ranking quality measure that takes into account not only the k results but also their relative positions within the list. For all values of k, QARTA consistently outperforms Google and OSRM. The accuracy of all methods increases with k, as the more PoIs we include, the farthest they are from the reference point, hence, their ranking is dominated by driving distance over driving duration.

6.4 Query Performance and Training time

Figure 8(a) gives the training time needed for weight inference and query calibration for 75% of Doha, Porto, and NYC datasets. As weight inference is mainly about solving Equation 4, which depends on both the number of edges and number of trips, NYC needs the most time. Overall, the time to solve Equation 4 is acceptable, given



(a) Training Time (b) Response Time

Figure 8: Performance of QARTA

that it is a one time process. One can think of running this procedure once a month or so to update the model. Meanwhile the time taken for query calibration is much less by two orders of magnitude. Query calibration only relies on the number of trips and zones we use to build our model. Clearly, NYC dataset suffer the most, but again, this is one time process.

Figure 8(b) gives the average end-to-end shortest path query latency over 25% of the trips for Doha, Porto, and NYC. Google has less performance (100+ ms), mainly due to the fact that it is an API call for a cloud service, which encounters network latency. Meanwhile, both QARTA and OSRM are locally installed in our taxis, so, calling them does not encounter any network delay, and hence has more than an order of magnitude better performance than Google APIs. OSRM has a slightly unnoticeable better latency due to QARTA calibration process.

7 RELATED WORK

This section positions QARTA contributions with respect to its related work in various system components.

Trajectory data cleaning. Research efforts in trajectory cleaning have focused either on trajectories map matching [22, 45, 82] or densifying trajectory points (a.k.a trajectory interpolation [51, 86], trajectory completion [48], trajectory restoration [43], or trajectory imputation [13]). QARTA can employ any of these techniques in its data layer. In addition, QARTA adds its own rule-based trajectory cleaning module that addresses specific cases came out of real deployment and is not addressed by existing techniques.

Map making/matching. Several research efforts have exploited two orthogonal approaches to increase the accuracy of available maps and GPS points: (1) *Map Making* [1], where the goal is to use GPS traces to either build or update current inaccurate maps. This implicitly assumes that GPS traces are truly accurate [7, 9, 14, 21, 30, 65, 69, 74]. (2) *Map Matching* [12], where the goal is to increase the accuracy of GPS points by matching them to the underlying road network. This implicitly assumes that the road network is truly accurate [8, 10, 33, 47, 64, 76]. Unlike map making/matching techniques, QARTA does not have any underlying implicit assumption for the accuracy of road network or GPS points. Hence, QARTA deploys its own *match or make* module that decides on which parts of the map or GPS points we should trust.

Edge weight inference. Research efforts on weight inference are either *edge-centric*, where the objective is to find static [37, 53, 84], time-dependent [79, 85], or stochastic [34] weights for each edge, or *path-centric* [19, 78], where the objective is to find weights for a set of paths, which is more accurate in modeling driving turn costs. QARTA opts to develop its own time-dependent *edge-centric* approach over a *path-centric* approach for three reasons: (1) path-centric approaches mainly support shortest path queries, while QARTA supports various sorts of map services, many of them require having edge costs (e.g., heat maps, traffic visualization, kNN queries), (2) As query processing in QARTA is off-the-shelf, we had to go with the more classical/common edge-based shortest path algorithms aiming for wider adoption of QARTA. (3) QARTA needs to be highly scalable. With detailed maps of 644K edges, the number of paths that one would need to compute cost for is prohibitive. Current path-centric [19, 78] approaches were evaluated on networks that are order of magnitude less than QARTA.

All current edge-centric approaches suffer from two main drawbacks: (1) *Scalability*. Existing technique were only applied to small road networks (few thousands edges [37, 84, 85] or tens of thousands edges [34, 53, 79]). Meanwhile, QARTA develops its own tuning steps to scale up Equation 1 to support hundreds of thousands edges. (2) *overfitting*.. Existing techniques treat all road segments similarly, which results in overfitting for very large networks. QARTA distinguishes popular road segments from less popular ones. Finally, while some of the existing techniques suffer from zero/negative edges weights [53, 84], static edge weights [37, 53, 84], and/or limited number of time-dependent weights [79, 85], QARTA ensures positive fine-granularity 168 weights per edge.

Spatial queries. Significant efforts were dedicated to various forms of classical shortest path-queries including static routing [44, 77], time-dependent routing [58, 75, 81], batch routing [46, 62], personalized routing [28, 49], and eco-routing [5, 26], where the input is (time-dependent) edge weights and the output is a recommended route with its total cost. Recent attempts that use machine learning for spatial queries [66] are mainly geared towards replacing existing algorithms with machine learning models (e.g., [27, 34, 50, 52, 72, 80]). All these algorithms would fit in the Query Processing module of the Query Calibration layer, marked as white box in Figure 1, indicating that any existing algorithm can be used there. This is orthogonal from our main contribution in this layer (*calibration*), marked as gray boxes in Figure 1. *Calibration* complements existing classical or ML-based algorithms by understanding and fixing their error margins, hence significantly boosting their accuracy. Our calibration currently supports queries that report ETA values. Stochastic queries [35, 59, 60] that return probabilistic result distribution is planned for next QARTA release.

8 CONCLUSION

This paper presented QARTA; an open-source full-fledged system that employs ML techniques to provide highly accurate map services. QARTA's success is due to two main features: (1) QARTA learns its own highly accurate map, with accurate edge weights that reflect detailed traffic throughout the week, (2) QARTA calibrates the result of map services through its built-in ML modules that continuously understand the error margin of various query processors, and use it to adjust the result. State-of-the-art spatial query and indexing techniques can be injected in QARTA, where their performance will be significantly boosted, taking advantage of QARTA highly accurate map and calibration process. Experimental results based on actual deployment of QARTA show that QARTA is significantly more accurate than open-source mapping services and as accurate as or better than commercial ones.

REFERENCES

[1] Sofiane Abbar, Mohammad Alizadeh, Favyen Bastani, Sanjay Chawla, Songtao He, Hari Balakrishnan, and Sam Madden. 2018. The Science of Algorithmic Map Inference (Tutorial). In *Proceedings of the International Conference on Knowledge Discovery and Data Mining, KDD* (London, UK). https://sites.google.com/view/algorithmic--map--making/home.

[2] Sofiane Abbar, Rade Stanojevic, and Mohamed Mokbel. 2020. STAD: Spatio-Temporal Adjustment of Traffic-Oblivious Travel-Time Estimation. In *Proceedings of the International Conference on Mobile Data Management, MDM* (Versailles, France). 79–88.

[3] Sofiane Abbar, Rade Stanojevic, Mashaal Musleh, Mohamed ElShrif, and Mohamed Mokbel. 2021. A Demonstration of QARTA: An ML-based System for Accurate Map Services. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 14, 12 (2021), 2723–2726.

[4] Sofiane Abbar, Rade Stanojevic, Shadab Mustafa, and Mohamed Mokbel. 2021. Traffic Routing in the Ever-Changing City of Doha. *Communications of the ACM, CACM* 64, 4 (2021), 67–68.

[5] Reem Y. Ali, Venkata M. V. Gunturi, and Shashi Shekhar. 2014. Spatial big data for eco-routing services: computational challenges and accomplishments. *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* 6, 2 (2014), 19–25.

[6] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David J. DeWitt. 2018. RoadTracer: Automatic Extraction of Road Networks From Aerial Images. In *IEEE International Conference on Computer Vision and Pattern Recognition, CVPR* (Salt Lake City, UT, USA). 4720–4728.

[7] James Biagioni and Jakob Eriksson. 2012. Inferring Road Maps from Global Positioning System Traces: Survey and Comparative Evaluation. *Transportation Research Record: Journal of the Transportation Research Board* 2291, 1 (2012), 61–71.

[8] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. 2005. On Map-Matching Vehicle Tracking Data. In *Proceedings of the International Conference on Very Large Data Bases, PVLDB* (Trondheim, Norway). 853–864.

[9] Lili Cao and John Krumm. 2009. From GPS traces to a routable road map. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (Seattle, WA, USA). 3–12.

[10] Erin W. Chambers, Brittany Terese Fasy, Yusu Wang, and Carola Wenk. 2020. Map-Matching Using Shortest Paths. *ACM Transactions on Spatial Algorithms and Systems, TSAS* 6, 1 (2020), 1–17.

[11] Pingfu Chao, Wen Hua, and Xiaofang Zhou. 2020. Trajectories Know Where Map is Wrong: An Iterative Framework for Map-trajectory Co-optimisation. *Proceedings of the International Conference on World Wide Web, WWW* 23, 1 (2020), 47–73.

[12] Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. 2020. A Survey on Map-Matching Algorithms. In *Australasian Database Conference, ADC* (Melbourne, Australia). 121–133.

[13] Chao Chen, Shuhai Jiao, Shu Zhang, Weichen Liu, Liang Feng, and Yasha Wang. 2018. TripImputor: Real-Time Imputing Taxi Trip Purpose Leveraging Multi-Sourced Urban Data. *IEEE Transactions on Intelligent Transportation Systems, TTIT* 19, 10 (2018), 3292–3304.

[14] Chen Chen, Cewu Lu, Qixing Huang, Qiang Yang, Dimitrios Gunopulos, and Leonidas J. Guibas. 2016. City-Scale Map Creation and Updating using GPS Collections. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining, KDD* (San Francisco, CA, USA). 1465–1474.

[15] Guangliang Cheng, Ying Wang, Shibiao Xu, Hongzhen Wang, Shiming Xiang, and Chunhong Pan. 2017. Automatic Road Detection and Centerline Extraction via Cascaded End-to-End Convolutional Neural Network. *IEEE Transactions on Geoscience and Remote Sensing* 55, 6 (2017), 3322 – 3337.

[16] Razvan-Gabriel Cirstea, Hilmar Gústafsson, Rasmus Riis Grønbæk Pedersen, Rolf Hakon Verder Sehested, Tamas Imre Winkler, and Bin Yang. 2020. A Road Segment Attribute Completion System. In *Proceedings of the International Conference on Mobile Data Management, MDM* (Versailles, France). 236–237.

[17] Benjamin Coifman. 2002. Estimating Travel Times and Vehicle Trajectories on Freeways using Dual Loop Detectors. *Transportation Research Part A: Policy and Practice* 36, 4 (2002), 351–364.

[18] Serdar Çolak, Antonio Lima, and Marta C González. 2016. Understanding congested travel in urban areas. *Nature communications* 7, 1 (2016), 1–8.

[19] Jian Dai, Bin Yang, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. 2016. Path Cost Distribution Estimation Using Trajectory Data. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 10, 3 (2016), 85–96.

[20] Ugur Demiryurek, Farnoush Banaei Kashani, and Cyrus Shahabi. 2010. A case for time-dependent shortest path computation in spatial networks. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (San Jose, CA, USA). 474–477.

[21] Stefan Edelkamp and Stefan Schrödl. 2003. *Route Planning and Map Inference with Global Positioning Traces.* Springer, 128–151. Rolf Klein and Hans-Werner Six and Lutz Wegner.

[22] Bettina Fazzinga, Sergio Flesca, Filippo Furfaro, and Francesco Parisi. 2016. Exploiting Integrity Constraints for Cleaning Trajectories of RFID-Monitored Objects. *ACM Transactions on Database Systems, TODS* 41, 4 (2016), 24:1–24:52.

[23] Jerome H. Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics* 29, 5 (2001), 1189 – 1232.

[24] Betsy George and Shashi Shekhar. 2006. Time-Aggregated Graphs for Modeling Spatio-temporal Networks. *Journal on Data Semantics* 11 (2006), 191–212.

[25] Goole [n.d.]. Google Maps API. https://developers.google.com/maps.

[26] Chenjuan Guo, Bin Yang, Ove Andersen, Christian S. Jensen, and Kristian Torp. 2015. EcoSky: Reducing vehicular environmental impact through eco-routing. In *Proceedings of the International Conference on Data Engineering, ICDE* (Seoul, South Korea). 1412–1415.

[27] Chenjuan Guo, Bin Yang, Jilin Hu, and Christian Jensen. 2018. Learning to Route with Sparse Trajectory Sets. In *Proceedings of the International Conference on Data Engineering, ICDE*. 1073–1084.

[28] Chenjuan Guo, Bin Yang, Jilin Hu, Christian S. Jensen, and Lu Chen. 2020. Context-aware, preference-based vehicle routing. *VLDB Journal* 29, 5 (2020), 1149–1170.

[29] Glen Hart and Catherine Dolbear. 2007. *What's So Special about Spatial?* Springer, 39–44. Arno Scharl and Klaus Tochtermann.

[30] Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. 2018. RoadRunner: Improving the Precision of Road Network Inference From GPS Trajectories. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (Seattle, WA, USA). 3–12.

[31] Tin Kam Ho. 1995. Random Decision Forests. In *Proceeding of the IEEE International Conference on Document Analysis and Recognition ICDAR* (Montreal, Canada). 278–282.

[32] Arthur E. Hoerl and Robert W. Kennard. 1970. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics* 12, 1 (1970), 55–67.

[33] Gang Hu, Jie Shao, Fenglin Liu, Yuan Wang, and Heng Tao Shen. 2017. IF-Matching: Towards Accurate Map-Matching with Information Fusion. In *Proceedings of the International Conference on Data Engineering, ICDE* (San Diego, CA, USA). 9–10.

[34] Jilin Hu, Chenjuan Guo, Bin Yang, and Christian S. Jensen. 2019. Stochastic Weight Completion for Road Networks Using Graph Convolutional Networks. In *Proceedings of the International Conference on Data Engineering, ICDE* (Macau, China). 1274–1285.

[35] Jilin Hu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2018. Risk-aware Path Selection with Time-varying, Uncertain travel costs: A Time Series Approach. *VLDB Journal* 27, 2 (2018), 179–200.

[36] Timothy Hunter, Ryan Herring, Pieter Abbeel, and Alexandre Bayen. 2009. Path and Travel Time Inference from GPS Probe Vehicle Data. In *Neural Information Processing Systems foundation, NIPS* (Vancouver, Canada).

[37] Tsuyoshi Idé and Masashi Sugiyama. 2011. Trajectory Regression on Road Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI* (San Francisco, CA, USA).

[38] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning.* ACM Books.

[39] Kalervo Järvelin and Jaana Kekäläinen. 2000. IR Evaluation Methods for Retrieving Highly Relevant Documents. In *Proceedings of the ACM International Conference on Research and Development in Information Retrieval, SIGIR* (Athens, Greece). 41–48.

[40] Hoyoung Jeung, Hua Lu, Saket Sathe, and Man Lung Yiu. 2014. Managing Evolving Uncertainty in Trajectory Databases. *IEEE Transactions on Knowledge and Data Engineering, TKDE* 26, 7 (2014), 1692–1705.

[41] Evanthia Kazagli and Haris N. Koutsopoulos. 2013. Estimation of Arterial Travel Time from Automatic Number Plate Recognition Data. *Transportation Research Record* 2391, 1 (2013), 22–31.

[42] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations, ICLR* (Toulon, France).

[43] Bozhao Li, Zhongliang Cai, Mengjun Kang, Shiliang Su, Shanshan Zhang, Lili Jiang, and Yong Ge. 2021. A Trajectory Restoration Algorithm for Low-sampling-rate Floating Car Data and Complex Urban Road Networks. *International Journal of Geographical Information Science* 35, 4 (2021), 717–740.

[44] Lingxiao Li, Muhammad Aamir Cheema, Mohammed Eunus Ali, Hua Lu, and David Taniar. 2020. Continuously Monitoring Alternative Shortest Paths on Road Networks. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 13, 11 (2020), 2243–2255.

[45] Lun Li, Xiaohang Chen, Qizhi Liu, and Zhifeng Bao. 2020. A Data-Driven Approach for GPS Trajectory Data Cleaning. In *Proceedings of the International Conference on Database Systems for Advanced Applications, DASFAA* (Jeju, South Korea). 3–19.

[46] Lei Li, Mengxuan Zhang, Wen Hua, and Xiaofang Zhou. 2020. Fast Query Decomposition for Batch Shortest Path Processing in Road Networks. In *Proceedings of the International Conference on Data Engineering, ICDE* (Dallas, TX, USA). 1189–1200.

[47] Yang Li, Qixing Huang, Michael Kerber, Lin Zhang, and Leonidas J. Guibas. 2013. Large-scale Joint Map Matching of GPS Traces. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information*

Systems, ACM SIGSPATIAL GIS (Orlando, FL, USA). 214–223.
[48] Yang Li, Yangyan Li, Dimitrios Gunopulos, and Leonidas J. Guibas. 2016. Knowledge-based Trajectory Completion from Sparse GPS Camples. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (Burlingame, CA, USA). 33:1–33:10.
[49] Yaguang Li, Han Su, Ugur Demiryurek, Bolong Zheng, Kai Zeng, and Cyrus Shahabi. 2016. PerNav: A Route Summarization Framework for Personalized Navigation. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (San Francisco, CA, USA). 2125–2128.
[50] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations, ICLR* (Vancouver, Canada).
[51] Jed A. Long. 2016. Kinematic Interpolation of Movement Data. *International Journal of Geographical Information Science* 30, 5 (2016), 854–868.
[52] Zhongjian Lv, Jiajie Xu, Kai Zheng, Hongzhi Yin, Pengpeng Zhao, and Xiaofang Zhou. 2018. LC-RNN: A Deep Learning Model for Traffic Speed Prediction. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI* (Stockholm, Sweden). 3470–3476.
[53] Sankarshan Mridha, Niloy Ganguly, and Sourangshu Bhattacharya. 2017. Link Travel Time Prediction from Large Scale Endpoint Data. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*. 71:1–71:4.
[54] NYC [n.d.]. Kaggle. New York City Taxi Trip Duration. https://www.kaggle.com/c/nyc-taxi-trip-duration/data.
[55] OSM [n.d.]. OpenStreetMap. http://www.openstreetmap.org/.
[56] OSM Boundaries [n.d.]. OSM Boundaries. https://osm-boundaries.com/.
[57] OSRM [n.d.]. Open Source Routing Machine (OSRM). http://project-osrm.org/.
[58] Dian Ouyang, Long Yuan, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2020. Efficient Shortest Path Index Maintenance on Dynamic Road Networks with Theoretical Guarantees. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 13, 5 (2020), 602–615.
[59] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. Anytime Stochastic Routing with Hybrid Learning. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 13, 9 (2020), 1555–1567.
[60] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. Fast stochastic routing under time-varying uncertainty. *VLDB Journal* 29, 4 (2020), 819–839.
[61] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of machine Learning research* 12, 85 (2011), 2825–2839.
[62] Shangfu Peng, Jagan Sankaranarayanan, and Hanan Samet. 2016. SPDO: High-throughput road distance computations on Spark using Distance Oracles. In *Proceedings of the International Conference on Data Engineering, ICDE* (Helsinki, Finland). 1239–1250.
[63] Porto [n.d.]. Taxi Service Trajectory. Prediction Challenge. ECML PKDD 2015. http://www.geolink.pt/ecmlpkdd2015-challenge/dataset.html.
[64] Efstratios Rappos, Stephan Robert, and Philippe Cudré-Mauroux. 2018. A Force-directed Approach for Offline GPS Trajectory Map Matching. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (Seattle, WA, USA). 319–328.
[65] Sijie Ruan, Cheng Long, Jie Bao, Chunyang Li, Zisheng Yu, Ruiyuan Li, Yuxuan Liang, Tianfu He, and Yu Zheng. 2020. Learning to Generate Maps from Trajectories. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI* (New York, NY, USA). 890–897.
[66] Ibrahim Sabek and Mohamed Mokbel. 2020. Machine Learning Meets Big Spatial Data (Tutorial). In *Proceedings of the International Conference on Data Engineering, ICDE* (Dallas, TX, USA). 1782–1785.
[67] Rade Stanojevic, Sofiane Abbar, and Mohamed Mokbel. 2018. W-edge: Weighing the Edges of the Road Network. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (Seattle, WA, USA). 424–427.
[68] Rade Stanojevic, Sofiane Abbar, and Mohamed Mokbel. 2019. MapReuse: Recycling Routing API Queries. In *Proceedings of the International Conference on Mobile Data Management, MDM* (Hong Kong, China). 279–287.
[69] Rade Stanojevic, Sofiane Abbar, Saravanan Thirumuruganathan, Sanjay Chawla, Fethi Filali, and Ahid Aleimat. 2018. Robust Road Map Inference through Network Alignment of Trajectories. In *Proceedings of the SIAM International Conference on Data Mining, SDM* (San Diego, CA, USA). 135–143.
[70] Rade Stanojevic, Sofiane Abbar, Saravanan Thirumuruganathan, Gianmarco De Francisci Morales, Sanjay Chawla, Fethi Filali, and Ahid Aleimat. 2018. Road Network Fusion for Incremental Map Updates. In *Processigns on the Intenranional Conference on Progress in Location Based Services, LBS* (Zurich, Switzerland). 91–109.
[71] Tao Sun, Zonglin Di, Pengyu Che, Chun Liu, and Yin Wang. 2019. Leveraging Crowdsourced GPS Data for Road Extraction From Aerial Imagery. In *IEEE International Conference on Computer Vision and Pattern Recognition, CVPR* (Long Beach, CA, USA). 7509–7518.
[72] Luan Tran, Minyoung Mun, Matthew Lim, Jonah Yamato, Nathan Huh, and Cyrus Shahabi. 2020. DeepTRANS: A Deep Learning System for Public Bus Travel Time Estimation using Traffic Forecasting. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 13, 12 (2020), 2957–2960.
[73] Uber [n.d.]. Uber Movement Data. https://movement.uber.com/.
[74] Suyi Wang, Yusu Wang, and Yanjie Li. 2015. Efficient Map Reconstruction and Augmentation via Topological Methods. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (Bellevue, WA, USA). 1–10.
[75] Yong Wang, Guoliang Li, and Nan Tang. 2019. Querying Shortest Paths on Time Dependent Road Networks. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 12, 11 (2019), 1249–1261.
[76] Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. 2013. Map Matching: Comparison of Approaches using Sparse and Noisy Data. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (Orlando, FL, USA). 434–437.
[77] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. 2012. Shortest Path and Distance Queries on Road Networks: An Experimental Evaluation. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 5, 5 (2012), 406–417.
[78] Bin Yang, Jian Dai, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. 2018. PACE: a PAth-CEntric paradigm for stochastic path finding. *VLDB Journal* 27, 2 (2018), 153–178.
[79] Bin Yang, Manohar Kaul, and Christian S. Jensen. 2014. Using Incomplete Information for Complete Weight Annotation of Road Networks. *IEEE Transactions on Knowledge and Data Engineering, TKDE* 26, 5 (2014), 1267–1279.
[80] Sean Bin Yang, Chenjuan Guo, and Bin Yang. 2020. Context-aware path ranking in road networks. *IEEE Transactions on Knowledge and Data Engineering, TKDE* (2020).
[81] Ziqiang Yu, Xiaohui Yu, Nick Koudas, Yang Liu, Yifan Li, Yueting Chen, and Dingyu Yang. 2020. Distributed Processing of k Shortest Path Queries over Dynamic Road Networks. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (Portland, OR, USA). 665–679.
[82] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S. Yu. 2017. Time Series Data Cleaning: From Anomaly Detection to Anomaly Repairing. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 10, 10 (2017), 1046–1057.
[83] Hongyu Zhang and Jacek Malczewski. 2019. *Quality Evaluation of Volunteered Geographic Information: The Case of OpenStreetMap.* IGI Global, Chapter 58, 1173–1201.
[84] Fangfang Zheng and Henk Van Zuylen. 2013. Urban Link Travel Time Estimation based on Sparse Probe Vehicle Data. *Transportation Research Part C: Emerging Technologies* 31 (2013), 145–157.
[85] Jiangchuan Zheng and Lionel M. Ni. 2013. Time-Dependent Trajectory Regression on Road Networks via Multi-Task Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI* (Bellevue, WA, USA).
[86] Kai Zheng, Yu Zheng, Xing Xie, and Xiaofang Zhou. 2012. Reducing Uncertainty of Low-Sampling-Rate Trajectories. In *Proceedings of the International Conference on Data Engineering, ICDE* (Washington, DC, USA). 1144–1155.