Towards Highly Accurate Map Services

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Mashaal Abdo Yahya Musleh

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prof. Mohamed F. Mokbel

May, 2025

Acknowledgements

This dissertation would not have been possible without the unwavering support of many people around me, whose encouragement and belief in me sustained me through every step of this journey.

First and foremost, I extend my deepest gratitude to my advisor, Prof. Mohamed Mokbel. His guidance and steadfast mentorship have shaped both my academic path and my personal growth. From our earliest discussions to the final stages of this work, his support has been a constant source of inspiration and strength.

My heartfelt thanks go to my mother, Derhemah, whose love and patience have carried me through countless challenges. Her unconditional faith has been a silent yet powerful force behind this achievement. Her deepest wish has always been for us to pursue the best education, and she has made countless sacrifices to make that possible. Despite needing me close by, she never hesitated to say, "Go and study," always placing our future ahead of her own needs. I also thank my sister, Mashael, and my brother, Raed, whose support and encouragement have never faltered, even across the thousands of miles that separated us.

To my wife, my closest friend, anchor, and tireless cheerleader, Dr. Ghadeer: no words can fully express my gratitude. Your presence, patience, and unwavering support throughout this PhD journey, especially during its most difficult moments, are gifts I will always be in your debt for. You stood by me with unshakable faith and strength, often believing in me more than I believed in myself. This journey is as much yours as it is mine, and I will always be grateful to have you by my side.

I am deeply grateful to the former and current members of DMLab at UMN. It has been a privilege to learn from and work alongside such an inspiring group. In particular, I would like to thank Dr. Amr Magdy, who took me by the hand early in my academic

life and helped guide me from undergraduate studies all the way to PhD research. I also extend my sincere thanks to Dr. Ibrahim Sabek, his mentorship and companionship, especially during the early phases of my research, were instrumental in shaping my path.

I would also like to thank Dr. Saleh and Dr. Anans Basalamah from Umm Al-Qura University for their encouragement to pursue a PhD and for setting admirable examples during my undergraduate years. Their belief in me planted early seeds of ambition and academic purpose.

I am equally grateful to my colleagues that I worked with at GISTIC and QCRI research centers. I have learned so much from every single one of them, and their influence has left a lasting mark on my academic and professional identity.

I am equally grateful to my colleagues that I worked with at GISTIC and QCRI research centers. I have learned so much from every single one of them, and their influence has left a lasting mark on my academic and professional identity. I am particularly thankful to Dr. Hicham Elmongui and Dr. Mourad Ouzzani, who mentored me during my internships and played a key role in shaping my early research experience.

Finally, I sincerely thank my committee members: Prof. Shashi Shekhar, Prof. John Krumm, and Dr. Yao-Yi Chiang for their constructive feedback, thoughtful questions, and insightful perspectives. Their guidance significantly enriched this work.

To all my friends and everyone who supported me in ways big and small, thank you. This milestone is not mine alone; it belongs to all of you.

Dedication

To the memory of my father, Abdo Musleh (may Allah have mercy on his soul), who spent his life working tirelessly and making sacrifices to give us a better future. To my mother, Derhemah, whose endless dedication and sleepless nights ensured we were prepared for school every morning and had the opportunity for a better education. To my wife, my sister, and my brother, whose love, support, and encouragement carried me through. And finally, to all my family and friends.

Abstract

Map services such as routing, travel time estimation, and nearby facility search have become integral to a broad range of transportation applications including ride-sharing, food delivery, last-mile logistics, and emergency response. Consequently, significant research efforts have focused on developing algorithms for map service queries, commonly under the implicit assumption that the underlying road network is accurate. As a result, these efforts primarily focus on algorithmic efficiency (i.e., execution time). Unfortunately, this assumption is not true, as road network data suffer from various inaccuracies such as missing roads, incorrect edge weights, and inaccurate traffic information, all of which severely degrade the accuracy of query results. No matter how efficient a shortest path algorithm is, or how accurate the map topology may be, the end result will be only as accurate as the edge weights it relies on. Thus, the primary bottleneck in map services today is no longer the *efficiency*, but rather the *accuracy*, which heavily depends on the quality of the underlying road network used for these services.

The goal of this dissertation is to significantly boost the accuracy of map services and shift the research focus from merely developing more efficient algorithms to improving the accuracy of the underlying map itself. We consider a broader definition of a map that goes beyond traditional topology to a richer one that includes traffic related metadata, without which, most map services would suffer from low accuracy. To this end, we introduce a set of innovative techniques geared towards building the most comprehensive and accurate maps. This dissertation does not aim to develop new algorithms for map services or modify existing ones. Instead, it aims to improve the accuracy of the map itself to provide more accurate input to query systems. This significantly boosts the accuracy of all existing (and future) algorithms without changing their internal logic.

This thesis makes the following contributions: First, we introduce **RASED**, an open-access map analysis tool that we developed to help researchers monitor map updates across the world and gain deeper insights into map quality. Second, we present **QARTA**, a full-fledged machine learning-based system for accurate map services. It leverages machine learning to (i) learn accurate edge weights for the map from sparse

trip data, and (ii) learn and model the map error margins, which are then used to calibrate query answers based on contextual information, including transportation modality, location, and time of day/week. Third, we introduce our vision, **"Let's Speak Trajectories"**, which enables the execution of most (if not all) trajectory analysis tasks essential for constructing accurate maps via one unified framework. We draw a novel analogy between trajectories and natural language statements and innovatively adapt the BERT language model to perform these tasks within a single, model-based framework. Lastly, we present **KAMEL**, a novel system for the trajectory imputation task inspired by our vision. It accurately infers missing points along sparse trajectories, generating dense, high-quality trajectories that enable more reliable map inference for any algorithm using this data as input. In summary, the research contributions in this thesis lay the foundation and provide the means for building comprehensive, high-quality maps that elevate the accuracy of all map-based applications.

Contents

List of Figures

Chapter 1

Introduction

1.1 Map Services in Modern-Day Mobility

The proliferation of GPS-enabled devices has enabled a myriad of essential map services such as finding the shortest or fastest route between two locations, searching for nearby facilities such as restaurants or gas stations, and estimating travel times to plan trips and deliveries. Such map services are widely used by all transportation sectors, including individuals, taxis, logistics and freights. In fact, map services become an integral component of many contemporary transportation services, including ride-sharing (e.g., Uber [246] and Lyft [162]), food delivery (e.g., Uber Eats [247] and Doordash [66]), and last-mile delivery (e.g., Amazon [8], UPS [250], and FedEx [74]). In emergency response systems, where travel time is critical for saving lives, map services are utilized to estimate travel times and efficiently navigate responder vehicles to incident locations [158, 224]. The great demand for map services can also be seen through the competition between enterprises which resulted in having a variety of commercial map products (e.g., Google Maps [88], Bing Maps [20], HERE [103], and Waze [264]) that are ubiquitously used worldwide. Consequently, map services has a direct and significant impact on the reliability of transportation systems and applications.

1.2 Accuracy is the Bottleneck in Map Services

The last few decades have witnessed extensive research in academia and industry for developing efficient algorithms for various map services queries (e.g., shortest path [137, 136, 268, 289], range [38, 120, 255, 304], and k-NN [15, 45, 110, 201, 283] queries). These algorithms have an implicit assumption that the underlying road network is accurate, and therefore, their primary focus was the efficiency. Unfortunately, such an assumption is not always true as road network data suffer from all sorts of inaccuracies that significantly degrade the query result accuracy. For example, Microsoft recently announced that it has found more than one million kilometers of roads missing from current maps [174]. Additionally, a recent independent report on the logistics industry in USA found out that 99% of delivery company drivers believe they would be more efficient if they had better maps [170]. Specifically, they pointed out several issues such as maps are outdated, recommending longer routes, showing wrong drop-off points.

While these accuracy issues might seem minor on an individual level—perhaps only causing someone a few minutes of delay—they quickly accumulate at scale and their cumulative impact becomes substantial, especially for fleet-based, governmental, and commercial applications that rely on map services to support large volumes of users and operations. For example, map inaccuracies cost delivery companies US $6 billion annually [243], due to the significant amount of wasted time that translates into wasted wages, delayed deliveries, customer dissatisfaction, increased service costs, and high vehicle pollutants. This negative impact is expected to grow with the rising number of online shoppers and riders, as traffic shifts from casual users to delivery companies and ride-sharing services. This poses a real challenge for current research because no matter how efficient an algorithm one can develop, it will always fall inaccurate if the underlying road network map is inaccurate.

1.3 Thesis Contributions

The goal of this dissertation is to exploit a new direction by pointing out that the bottleneck in map services queries is no longer the *efficiency* (i.e., their execution time). Instead, it is the *accuracy*, which heavily depends on the quality of the underlying road network used to answer these queries. Hence, we develop a set of innovative techniques

geared towards boosting the accuracy of the map itself, which will ultimately boost the accuracy of existing (and future) algorithms without any need to modify their internals.

This dissertation introduces two main research contributions. The first contribution presents a full-fledged, machine learning-based system that identifies key components essential for building accurate maps. It encapsulates various proposed techniques into an end-to-end system to enable highly accurate map services. The second contribution introduces a novel framework to efficiently analyze trajectory data, which is an important data source for constructing accurate maps. This contribution enables the execution of most (if not all) trajectory analysis tasks via one unified framework. These two research contributions are briefly described in the rest of this section.

1.3.1 Full-fledged ML-based System for Highly Accurate Map Services

In this contribution, we architect and build QARTA [3, 184], a full-fledged machine learning-based system for highly accurate map services. QARTA builds and maintains its own map, which is the system's main asset, and uses it to empower map services. Beyond constructing accurate map topology, QARTA focuses on two main aspects: (1) Learning accurate map metadata, such as edge weights, which are vital for the accuracy of services like routing and ETA queries. (2) Learning and modeling error margins, which are used to calibrate query results and improve their accuracy.

To achieve this, QARTA architecture is composed of three layers: The *Data* layer prepares and cleans various datasets to support accurate map construction. The *Map Inference* layer builds and maintains the map, including the inference of edge weights and other map metadata. The *Query* layer incorporates existing state-of-the-art methods to process map services queries and calibrate their results. Notably, QARTA does not develop new map service algorithms. Instead, it boosts the accuracy of existing ones by: (1) supplying them with a high accuracy map built by the data and map inference layers, and then (2) calibrating their output through the query calibration layer. This is a key advantage of QARTA over existing systems, as it enables more accurate and reliable services without the need to change the internals of existing algorithms.

Our experiments show that QARTA reduces the error in estimated time of arrival (ETA) queries by more than 25% compared to state-of-the-art open-source methods, demonstrating its effectiveness in enhancing the accuracy of map services.

1.3.2 A Novel Unified Framework for Trajectory Analysis Tasks

The widespread use of location-tracking devices has made available a plethora of trajectory datasets (set of consecutive location points for moving objects, represented by their coordinates) [211, 112, 113, 84, 16, 81, 309, 240, 23, 294]. These datasets have become a vital source for data-driven mapping applications [301, 261, 40, 79, 291, 184] including accurate map inference [9, 19, 37, 99, 232, 219]. However, there are two major obstacles that hinder the full utilization of trajectories: **(1) Lack of a unified trajectory analysis system:** working with trajectories often requires executing a myriad of analysis tasks on the dataset such as similarity search, clustering, and classification. This is a major burden when there is no single unified system capable of carrying out these tasks. **(2) Trajectory sparsity:** due to bandwidth, battery, and storage limitations, trajectories often have large spatial and temporal gaps between observed points. This negatively impact the accuracy of the inferred maps. To address these obstacles, we provide the following research contributions:

- **Unified Model-based System for Trajectory Analysis Tasks.** We introduce a vision of a unified, model-based system for most (if not all) trajectory tasks, titled "Let's Speak Trajectories" [183, 189, 190]. Inspired by the tremendous success of BERT [63] deep learning model in various Natural Language Processing (NLP) tasks, our vision is to develop a BERT-like system for trajectory tasks. When trained, BERT core is equipped with the necessary NLP infrastructure to solve various NLP tasks, and only needs to be externally tuned with minimal overhead for each one. Similarly, we envision a model-based system that provides the necessary infrastructure for all trajectory analysis tasks. Whether it is trajectory imputation, similarity, clustering, or any other task, it would be one system that researchers and practitioners can deploy, with a minimal tuning for its BERT-customized model to support the required task. In this vision, we draw a strong analogy between trajectories in a space and statements in a language. We present analogies for several trajectory tasks to their corresponding NLP tasks. We also identify key challenges to realizing this vision and propose potential directions to address them. Finally, we outline a roadmap and research directions to advance this vision toward a practical and high-impact trajectory analysis system.

- **Map-less Trajectory Imputation.** To address the trajectory sparsity issue, we introduce KAMEL [185, 188], a novel trajectory imputation framework that accurately infers the missing points along sparse trajectories. KAMEL is the first realization of our vision to have a unified, model-based system for trajectory analysis. Leveraging a BERT model trained on large volumes of trajectories, KAMEL accurately predicts missing trajectory points, just as BERT in NLP can accurately predict missing words in a sentence after being trained on large textual corpora. Most existing imputation techniques [24, 157, 305, 135] assume the map is accurate and therefore rely on matching trajectories to the underlying road network. In contrast, KAMEL does not require any knowledge of the road network, as it performs imputation solely based on the input trajectories. This is directly inherited from the map-agnostic foundation of our vision, where trajectory tasks are solved using data-driven models. Therefore, beyond being a step toward our broader vision, KAMEL is particularly indispensable for accurate map construction applications where maps are unavailable or suffer from various inaccuracies [174, 170, 243]. Additionally, KAMEL does not require prior highly dense trajectories, distinguishing it from other imputation techniques and making it especially useful for real-world, commonly available trajectory datasets. Towards this goal, KAMEL system identifies and tackles three major challenges in adapting BERT to trajectory data: small training data ratio, spatial awareness, and large trajectory gaps. Our extensive experiments show that KAMEL achieves a recall score of over 89%, even with large trajectory gaps of up to 1 km. This sets a new state-of-the-art, significantly outperforming the latest map-less trajectory imputation technique [69]. With KAMEL generating denser and significantly more accurate trajectories, not only does map inference become more reliable, but all other downstream applications that rely on trajectory data as input also benefit seamlessly from the improved input quality.

1.4 Thesis Outline

The subsequent chapters delve into the research contributions discussed above. We summarize the thesis organization as follows:

- Chapter 2 introduces RASED [186, 187], a map stability and quality analysis tool that we developed to monitors map updates across the world, and offered as an open-access web service to help researchers gain deeper insights into the map.
- Chapter 3 introduces QARTA [3, 184], a full-fledged machine learning-based system for highly accurate map services.
- Chapter 4 presents our "Let's Speak Trajectories" [183, 189, 190] vision of a unified, BERT-like system for trajectory analysis tasks.
- Chapter 5 introduces KAMEL [185, 188], a novel trajectory imputation framework that accurately infers the missing points along sparse trajectories without relying on the underlying road network.
- Chapter 6 highlights the key contributions of our work and concludes the thesis.

Chapter 2

RASED: A Scalable Dashboard for Monitoring Road Network Updates in OSM

2.1 Introduction

It used to be the case that accurate digital maps are only built and sold by major industry, e.g., HERE [80] and TomTom [90]. However, the high cost and proprietary nature of commercial maps along with their inherent inaccuracy due to not being able to be frequently updated, made researchers, developers, practitioners, and enterprises turn their attention towards open-source maps [21, 85, 178, 235]. A prime example of such maps is OpenStreetMap (OSM) [196], known as the Wikipedia of maps. OSM is a platform for crowdsourcing-based maps that has recently replaced commercial providers in various sectors of academia, government, and industry [181, 191]. For example, Amazon Logistics [8] is using OSM data in their delivery programs [195], Apple Maps [11] have been using OSM since iOS 6 [194], Facebook uses OSM as its backbone mapping support [71], Lyft has described OSM as the "Freshet Map for Rideshare" [164], while Tesla [241] uses OSM for its routing [242]. All these companies, and many others including Mapbox, Microsoft, and Uber, are not only using OSM, but are also extensively contributing to it [10, 65, 286].

Meanwhile, though there is extensive research in academia and industry for developing efficient algorithms for a myriad of road network queries (e.g., shortest path [137, 136, 268, 289], range [38, 120, 255, 304], and k-NN [15, 45, 110, 201, 283] queries), all algorithms have the implicit assumption that the underlying road network is accurate. Unfortunately, such an assumption is not always true as road networks suffer from all sorts of inaccuracy that significantly degrade the query result accuracy. While this may be acceptable for casual users where inaccuracy may only mean few minutes of delay, it is not the case for governmental or commercial applications that support map services for large numbers of users. For example, in USA, 99% of delivery company drivers say that they would be more efficient if they had better maps [170]. Problems, identified by those drivers, include: maps recommend longer routes and are not updated. This wastes significant time that translates into wages and high gas consumption, costing delivery companies $6B annually [243]. This trend is just going to increase with the increase of online shoppers and riders, which shifts traffic from casual users to delivery companies and ride sharing services.

Though OSM is deemed more accurate and up-to-date than commercial maps, its accuracy is still far from being acceptable for high-demand map services [72, 163, 165, 176, 248]. This has triggered several research efforts, mostly led by the transportation community, to study the quality of the underlying road network, represented by OSM (e.g., [83, 117, 295]). Unfortunately, all such quality assessment studies have very limited scope and scale, where the focus is only to study a certain city or country road network with heavy manual operations. Up to our knowledge, there is no comprehensive global-scale study for the quality of OSM road network. This is mainly due to its large scale, which makes researchers limit their studies to small regions. For example, OSM road network has more than 180M road segments and 2B nodes, which account for 500GB worth of raw data.

This chapter introduces RASED (`https://rased.cs.umn.edu`); a publicly available scalable dashboard to interactively monitor and analyze all OSM road network updates worldwide. RASED is the first-ever attempt to quantify and visualize all OSM world-wide changes on a daily basis, which gives an idea about road network stability anywhere in the world. RASED provides the necessary infrastructure immensely needed by map analyzers to understand and assess the map quality. Using RASED, map analyzers can

query and visualize various map statistics, including number and percentage of OSM updates per country, comparison between countries, types of updated roads, and temporal evolution of updates. All queries can have several filters including temporal (e.g., time of update), spatial (e.g., country or state), road types, and update types, which would all give a better understanding of map status globally.

RASED is a highly interactive system, where all its analysis queries are supported in milliseconds allowing interactive visualization of the results. This makes RASED a convenient and highly important dashboard for road network map analyzers worldwide. To achieve its scalability and interactivity, the RASED backend employs: (1) *offline daily aggregation*, where the daily crawled OSM updates are analyzed offline to form all sorts of required precomputations, stored in data cubes [92], (2) *hierarchical indexing*, where the offline daily aggregation cubes form a hierarchical index of weekly and monthly updates to support analysis queries over longer time periods, and (3) *caching*, where some of the daily/weekly/monthly data cubes are prefetched in memory for faster access. All together, achieve a milliseconds response time when querying all OSM road network updates during the past 15 years.

The rest of this chapter is organized as follows: Section 2.2 gives a brief background about OSM. Section 2.3 gives RASED system architecture. Section 2.4 shows the queries supported by RASED. RASED three main modules, namely, Data Collection, Indexing, Querying, and User Interface are described in Sections 2.5, 2.6, 2.7, and 2.8, respectively. Section 2.9 experimentally evaluates RASED. The chapter is concluded in Section 2.10.

2.2 Background

OpenStreetMap (OSM) [196], launched in 2004, is a collaborative community project to create a free editable map of the world. Known as the Wikipedia of maps, OSM has 8.5 Million users, with 300K active users per year (users who made at least one edit during the year) [199]. OSM supports 400+ public free open-source OSM-based services [150], 80+ OSM-based commercial services [51], and receives API requests at the rate of 800 requests per second, for only one OSM data center [197]. This section gives a brief and necessary background about OSM data and update representation.

2.2.1 OSM Conceptual Data Model

OSM data is all stored in one big XML file (Plant.osm) presenting a massive list of elements, where each element is one of the following three types: (1) *Node*, which represents a certain point in the space with node identifier and its latitude and longitude coordinates. Objects represented by Nodes include intersection points, traffic lights, stop signs, bus stations, and other Points of Interest (PoI). (2) *Way*, which represents an ordered list of node identifiers making connected road segments. (3) *Relation*, which represents the relations between one or more elements of any type. Relations are used to model complex roads that may contain multiple parts (e.g., multiple Ways). Currently, OSM Planet.osm file is 1.6TB, and includes more than 7.5B nodes, 800M ways, and 9M relations.

2.2.2 OSM Map Updates

OSM is based on crowdsourcing where mappers voluntarily upload geographical data for their surroundings, which results in updating the map by creating new elements or modifying existing ones. OSM stores such updates in three different sets of files, described below:

Diff (`https://wiki.openstreetmap.org/wiki/Planet.osm/diffs`.) OSM creates such a file every minute, day, and hour such that any created or modified element is added (and replicated) to these three files. Only the element's after-image is stored in these files. Currently, OSM has 5M, 82K, and 3.5K minute, hourly, and daily Diff files, respectively, with sizes that range from a few megabytes to a few gigabytes per file.

Changesets (`https://wiki.openstreetmap.org/wiki/Changeset`.) A set of files that provide metadata information about map updates, e.g., user information, bounding box, comments, and sources, described for each changeset; a term used to represent all updates submitted by a particular user in one session (maximum of 24 hours). OSM provides two sources to download such data: (a) A single large file created every week with a dump of all changesets in OSM lifetime, currently of size 50GB. (2) A series of sequentially numbered small files (tens of kilobytes), such that a new file is created for every 1K new changesets. Currently, OSM approximately creates a new such file every minute and has created 5M files.

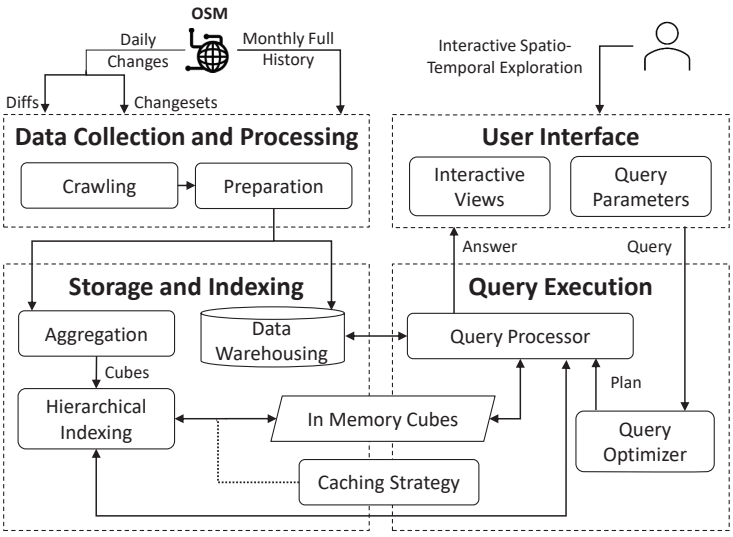


Figure 2.1: RASED Architecture

Full History (`https://wiki.openstreetmap.org/wiki/Planet.osm/full`.) One huge file dumped every few weeks for entire OSM updates. Unlike Diff files, the full history includes the previous state of each update. Currently, this file size is 3+TB and has 12+ Billion elements of all versions.

2.3 RASED Architecture

Figure 2.1 depicts the architecture of RASED, composed of the following four main modules:

Data Collection and Processing. This module is responsible for daily and monthly crawling of the OSM updates and preparing them for consumption by the *Storage and Indexing* module. The output of this module is a long list of daily/monthly updates, termed *UpdateList*, where each update has eight attributes: <*ElementType, Date, Country, Latitude, Longitude, RoadType, UpdateType, ChangesetID*>. *ElementType* is the type of the updated element (i.e., node, way, relation), *Date*, *Country*, *Latitude*, *Longitude* represent the date and location of the update, *RoadType* is the type of the updated road (e.g., highway, service, residential), *UpdateType* is the type of the update (e.g.,

new road, update geometry, deletion), *ChangesetID* is a reference to the changeset (Section 2.2.2) that contains this update. Details are in Section 2.5.

Storage and Indexing. This module takes the output of the *Data Collection* module as its input. Then it goes through two main operations: (a) computing various sorts of precomputations in a form of data cubes [92] and using it to populate its own hierarchical temporal index structure of daily, weekly, monthly, and yearly precomputed statistics, and (b) dumping the input to a traditional data warehouse indexed by both *ChangeSetID* and a spatial index. Details are in Section 2.6.

Query Execution. This module receives RASED queries submitted through the *User Interface* module, and answers them in an interactive way. Internally, it employs two main ideas: (a) *Caching*, where a selected set of aggregate data cubes are cached in memory to efficiently support incoming queries, and (b) *Level optimization*, where it smartly decides which level(s) in the index hierarchy would be better exploited for more efficient query support. Details are in Section 2.7.

User Interface. This module presents the Web Graphical User Interface (GUI) for RASED. It receives a set of interactive online queries from RASED users and sends it to the *Query Execution* module, which responds back with the answer in an interactive way. The query result is then visualized in various ways that allow map analyzers and domain experts to assess OSM stability and changes anywhere in the world. Details are in Section 2.8

2.4 RASED Queries

This section describes the various queries supported by RASED, presented in a SQL format. The most important queries fall under the category of analysis queries, described in Section 2.4.1. Update sample queries are described in Section 2.4.2.

2.4.1 Analysis Queries

RASED analysis queries aim to provide detailed statistics about road network updates. Examples of such queries include: "*finding the number or percentage of road network updates over the last two years for a particular set of countries*", "*finding the number of updates for each road type for a certain country over a certain time period*", and

"*compare the road network evolution for a particular set of countries*". The results of RASED analysis queries can be presented as either absolute numbers or percentages of the country's road network size, and can be visualized as: (a) tabular format sorted on any column, (b) various charts (bar, choropleth, time series), or (c) a timelapse video showing the road network evolution.

Generally speaking, RASED analysis queries are aggregate queries on a subset of the fields from the *UpdateList* relation, namely, *ElementType*, *Date*, *Country*, *RoadType*, and *UpdateType*, described in Section 2.3. In particular, RASED queries would have the following SQL signature:

```
SELECT
  U.ElementType, U.Date, U.Country,
  U.RoadType, U.UpdateType, COUNT(*)
FROM UpdateList U
WHERE
  U.ElementType IN ListofElementTypes
  AND U.Date BETWEEN date1 AND date2
  AND U.Country IN ListofCountries
  AND U.RoadType IN ListofRoadTypes
  AND U.UpdateType IN ListofUpdateTypes
GROUP BY
  U.ElementType, U.Date,
  U.Country, U.RoadType, U.UpdateType
```

Below are few examples of RASED analysis queries and their visualized answer, based on the above query signature:

Example 1: Country Analysis. "*Find the number of newly created or modified element types (node, way, relation) for each country road network in 2021*": Out of the five attributes in the query signature, we would need to group on only two of them (*Country* and *ElementType*) as we need the answer for each country and each element type. We have conditions on both the *Date* and *UpdateType*. No group or constraints on the fifth attribute, *RoadType*. Figures 2.2 and 2.3 give RASED visualization for that query in both bar chart and table formats, respectively.

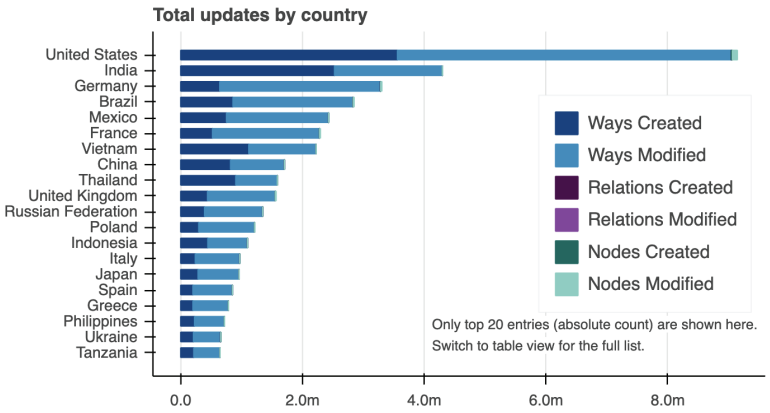


Figure 2.2: Visualized Results for Country Analysis Example in Bar Chart Format

country	All	Ways Created	Ways Modified	Relations Cr	Relations Mc	Nodes Cre	Nodes Moc
United States	9,142,858	3,559,417.0	5,483,190.0	43.0	92.0	19,295.0	80,821.0
India	4,299,546	2,525,864.0	1,771,880.0	3.0	104.0	210.0	1,485.0
Germany	3,302,925	641,372.0	2,639,701.0	79.0	143.0	4,383.0	17,247.0
Brazil	2,847,160	857,852.0	1,971,562.0	4.0	9.0	4,068.0	13,665.0
Mexico	2,432,545	751,154.0	1,677,685.0	0.0	3.0	526.0	3,177.0
France	2,293,375	521,512.0	1,755,216.0	54.0	60.0	4,267.0	12,266.0
Vietnam	2,223,752	1,117,416.0	1,106,088.0	0.0	25.0	157.0	66.0

Figure 2.3: Results for Country Analysis Example in Table Format

```
SELECT U.Country,U.ElementType,COUNT(*)
FROM UpdateList U
WHERE U.Date BETWEEN 2021-01-01
  AND 2021-12-31
  AND U.UpdateType IN [New,Update]
GROUP BY U.Country, U.ElementType
```

Example 2: Road Type Analysis. "*Find the number of newly created or modified elements types (node, way, relation) for each road type in USA since 2018*": We group

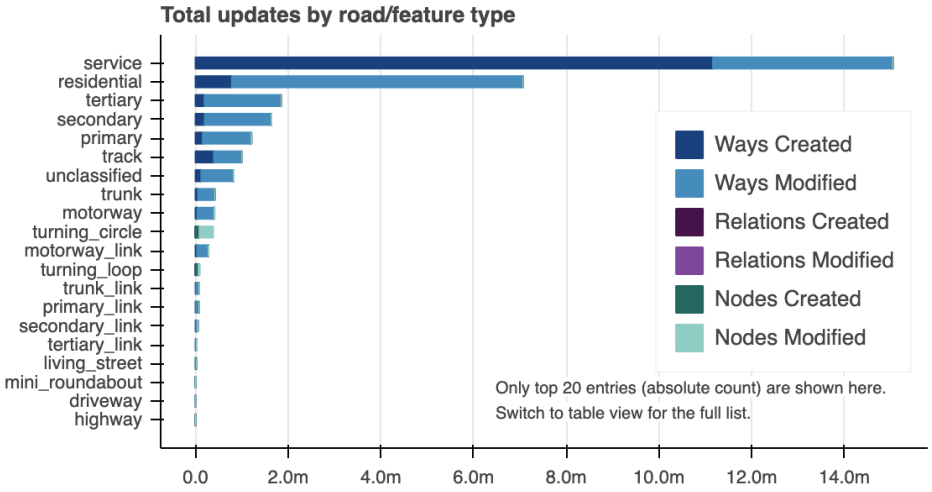


Figure 2.4: Visualized Results for Road Type Analysis Example

on two attributes (*RoadType* and *ElementType*) and have filters on the remaining three attributes, *Date*, *Country*, and *UpdateType*. Figure 2.4 gives RASED visualization for the query answer.

```
SELECT U.RoadType,U.ElementType,COUNT(*)
FROM UpdateList U
WHERE U.Date AFTER 2018-01-01
  AND U.Country = USA
  AND U.UpdateType IN [New,Update]
GROUP BY U.RoadType, U.ElementType
```

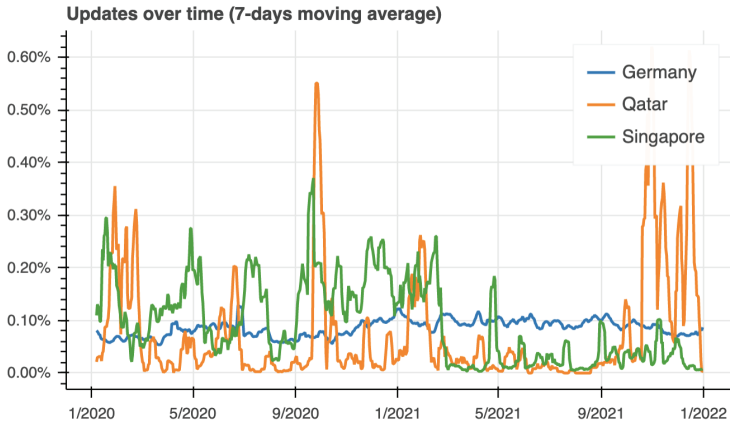


Figure 2.5: Visualized Results for Comparative Time-Series Analysis Example

Example 3: Comparative Time-Series Analysis. "*Compare the percentage of daily changes in road network in Germany, Singapore, and Qatar over 2020 and 2021*". We group and have conditions on *Country* and *Date*. Nothing is needed for the remaining three attributes. Figure 2.5 gives RASED visualization for the query answer.

```
SELECT U.Country, U.Date, Percentage(*)
FROM UpdateList U
WHERE U.Date BETWEEN 2020-01-01
  AND 2021-12-31
  AND U.Country IN [Germany,
    Singapore, Qatar]
GROUP BY U.Country, U.Date
```

2.4.2 Sample Update Queries

RASED users may want to see a sample of the updates that represent a given analysis query. Hence, RASED provides a query interface that visualizes a sample of N (default = 100) such updates on the map based on their *latitude* and *longitude* information. RASED also uses the *ChangesetID* of the samples to call a third-party application that visualizes the details of the sample update.

2.5 Data Collection and Processing

This module crawls OSM update files, described in Section 2.2 to produce the *UpdateList* of eight-attributes tuples: <*ElementType, Date, Country, Latitude, Longitude, RoadType, UpdateType, ChangesetID*>. One way to realize this module is to deploy a monthly crawler of OSM *full history* file. However, this would mean that RASED would have stale statistics, only updated on a monthly basis. Hence, we opt to have daily crawlers of the daily *diff* and *changesets* files, and use them to construct as much as possible of the *UpdateList*. Then, use the monthly crawler to complete the missing information. This ensures the accuracy of most RASED analysis queries.

Daily Crawler. The daily crawler mainly constructs seven out of the eight attributes of the *UpdateList*. For the eighth attribute (*UpdateType*), we can only infer whether an update is a new or updated tuple, but would not know whether this is an update for geometry or for metadata. Hence, we would defer these details to the monthly crawler. For each update record, four out of the seven attributes (in addition to the *UpdateType*) are obtained in a straightforward way from the *diff* files, namely, *ElementType*, *Date*, *RoadType*, and *ChangesetID* attributes. The remaining three attributes, *Country*, *Latitude*, *Longitude*, can only be easily obtained for the *node* elements, but not for the *way* and *relation* elements. To find out such information for each update tuple, we use its *ChangesetID* to retrieve its bounding box from the *changesets* file. We then map the bounding box to its country, and assign latitude and longitude coordinates based on the center point contained in the bounding box.

Monthly Crawler. The monthly crawler is made to go through the *full history* file to compare every two consecutive versions of an element and classify the update type as either *create*, *delete*, *metadata update*, or *geometry update*. Newly created elements will always be their first version, while deleted ones are the last version. Geometry updates occur when there is a change in the latitude/longitude attributes or the list of members of a *way* or *relation* element, while metadata update occurs by changing the element tags.

2.6 Storage and Indexing

The Storage and Indexing module takes the daily and monthly *UpdateList*: <*ElementType, Date, Country, Latitude, Longitude, RoadType, UpdateType, ChangesetID*>, produced from the Data Collection module (Section 2.5) as its input. Then, it builds and maintains a storage infrastructure that can be efficiently accessed by the Query Execution module (Section 2.7) to support RASED queries. This section describes such storage infrastructure per the type of supported queries.

2.6.1 Supporting Analysis Queries

To support RASED analysis queries (Section 2.4.1), we build and maintain a hierarchical temporal index structure that ensures that all queries will be supported with very few I/Os. This gives an interactive user experience navigating through various analysis queries. We describe below the index hierarchy, index nodes, index size, and index maintenance.

Index Hierarchy. Figure 2.6 depicts the hierarchical temporal index structure, employed by RASED. The index does not index the OSM updates itself. Instead, it indexes precomputed statistics (i.e., aggregates) about the OSM updates. These precomputed statistics basically cover everything one could ask for from any RASED analysis query. The index has four levels that represent yearly, monthly, weekly, and daily statistics with one dummy root node at the top that points to the various yearly statistics. All statistics are presented in the form of data cubes [92], each is stored in one-page index node. Each yearly statistics is basically an aggregation of twelve monthly statistics. In turn, the monthly statistics are aggregates of four weekly and zero to three daily statistics, and so on.

Index Nodes. Each index node at any level is basically a four-dimensional data cube [92], where the dimensions correspond to four attributes from the *UpdateList*, namely, *ElementType*, *Country*, *RoadType*, and *UpdateType*. In RASED, we have the following possible values for each dimension: (1) *ElementType*. Three possible values, presenting node, way, and relation elements. (2) *Country*. 300+ values presenting all countries plus some selected zones of interest (e.g., continents and US states). (3) *RoadType*. 150 possible road types, including highway, residential, service, and truck roads.

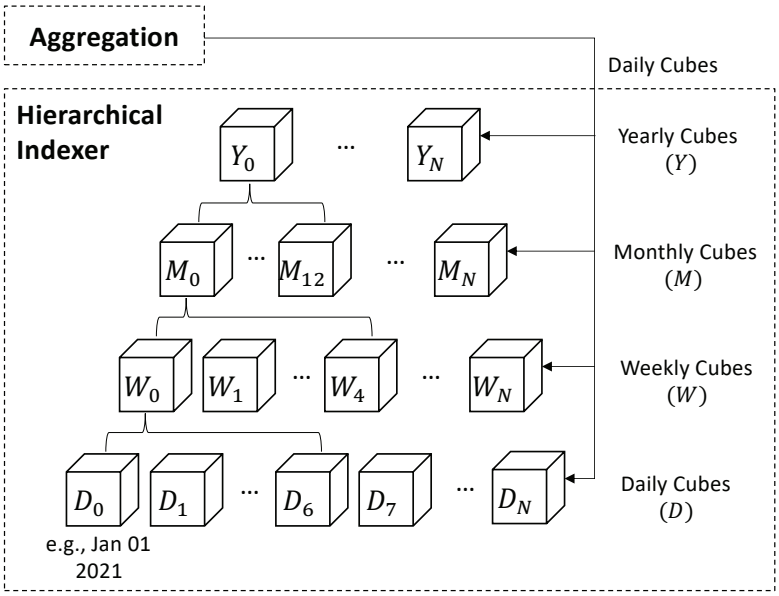


Figure 2.6: Hierarchical Temporal Index for Data Cubes

(4) *UpdateType*. Four kinds of update operations, namely, newly created roads/nodes, deleted roads/nodes, road geometry update, and road metadata update. This means that each cube maintains 540,000 precomputed values. Each cube cell is basically the count of OSM updates that happen in the time window of the cube (year, month, week, day) and match the corresponding value for each of the four dimensions.

Index Size. All nodes at all levels are of fixed size. With 540K values per node, each node takes ∼4MB of storage, which directly fits in one disk page. Considering all the OSM updates since its inception in 2004, we have 6,000+ daily nodes, 850+ weekly nodes, 200+ monthly nodes, and 16 yearly nodes. So, the total required storage is ∼28GB, which accommodate close to 7,000 nodes with 4 billion aggregate values. Though we store all index nodes on disk, the query executor caches some of them in memory for faster processing.

Index Maintenance with Daily Updates. Once the daily *UpdateList* is received from the daily crawler process in Section 2.5, we scan all the updates (10∼20MB), and construct a new data cube of 540,000 aggregate values as described above. Notice that with the daily crawlers, we would have only two possible values for the *UpdateType* dimension, so, in fact, we would calculate only 270,000 aggregate values, while putting

zeros in the rest of the data cube cells. The cube is then stored in a newly allocated disk page and linked to last day cube. If this day is the end of the week, we construct the parent weekly cube by reading the six previous cubes and summing up their corresponding values to build a newly weekly cube. We do so recursively for monthly and yearly cubes, if this day is the end of the month and year, respectively. This process is performed offline, and takes up to 30 minutes. The time is mainly spent in scanning the *UpdateList*, and hence it depends on the number of updates of each day. The process does not consume much I/Os. Normally, we would need only one I/O for daily cubes. If it is the end of the week/month/year, we would need up to 8, 6, and 13 I/Os, respectively.

Index Maintenance with Monthly Updates. Once the monthly *UpdateList* is received from the monthly crawler process in Section 2.5, we scan all the updates and reconstruct all the daily and weekly data cubes in that month. The main reason is that by now we have more detailed information about the *UpdateType* with four possible values. This would be a bit costly operation that would take a few hours due to the size of the monthly *UpdateList* and the number of I/O operations. Yet, the process is completely done offline, and copied to the index structure only when done.

2.6.2 Supporting Sample Update Queries

To support sample update queries (Section 2.4.2), we dump the whole *UpdateList* into a standard database table indexed by: (a) a hash index on *ChangesetID*, which is needed to retrieve a single update for RASED users to see the change that took place for a specific object, and (b) a spatial index on <*Latitude, Longitude*>, which is needed to retrieve the sample updates located in a certain spatial region.

2.7 Query Execution

The Query Execution module supports RASED queries through efficient data retrieval from the index infrastructure laid out by the Storage and Indexing module. This section only focuses on supporting RASED analysis queries (Section 2.4.1), as sample update queries are supported through a straightforward index-based retrieval from a traditional DBMS. RASED query execution goes through two main phases: The first phase is

mostly disk-based as it retrieves the data cubes that include the answer for a given query. The second phase is completely in-memory, where some computations may still be needed to aggregate values within the cube. For example, a query that asks about the number of updates in each country in a certain time window t, would first retrieve the data cubes that satisfy t. Since each cell in each cube represents one value of the four dimensions, *Country*, *ElementType*, *RoadType*, and *UpdateType*), we would then need to aggregate the values across three dimensions as we are only interested in the sum of updates for each country. The first phase is actually the bottleneck of this module, as the second phase is executed all in-memory. To reduce the overhead of the first phase, we employ two optimization techniques, *caching* (Section 2.7.1) and *level optimization* (section 2.7.2), geared towards reducing the number of retrieved data cubes from disk.

2.7.1 Caching Strategy

The idea of caching is to preload into memory some of the very recent data cubes, such that queries over recent data would be either fully or partially answered from in-memory cubes. This would significantly save from the query response time as we reduce the number of disk retrieval of data cubes. The rationale is that RASED is more likely to receive inquiries about recent updates than older ones. The challenge is from which index level we should pick our preloaded data cubes. Hence, we formulate our caching strategy as follows: Given N available memory slots and the sets Y, M, W, D of yearly, monthly, weekly, and daily cubes, we preload the following cubes into memory: $\{D_{|D|-i}\}_{i=0}^{\alpha N} \cup \{W_{|W|-i}\}_{i=0}^{\beta N} \cup \{M_{|M|-i}\}_{i=0}^{\gamma N} \cup \{Y_{|Y|-i}\}_{i=0}^{\theta N}$ where α, β, γ, and θ has a total sum of 1 and present the ratio of the N memory slots that will be allocated to each daily, monthly, weekly, and yearly levels, respectively. Such parameters present a trade-off between aggregation granularity and time coverage. For example, higher α would cache more daily details but less covered period, while higher γ and θ would favor longer period queries.

2.7.2 Level Optimization

A given analysis query could be answered from a mix of data cubes at different temporal levels. For example, an aggregate query for the period Jan 1, 2022 to Feb 15, 2022 can

be answered using either: (a) 46 daily cubes, (b) six weekly cubes (weeks of Jan 2, 9, 16, 23, 30, and Feb 6) and four daily cubes (Jan 1 and Feb 13-15), or (c) one monthly cube (January), one weekly cube (week of Feb 6), and eight daily cubes (Feb 1 to 5 and 13-15). The objective of the level optimization is to find the query plan that would retrieve from disk the least number of data cubes, taking into consideration that some of the data cubes are already in memory due to the deployed caching strategy. For example, trying to reduce the number of data cubes in the previous example would directly advise using either plan (b) or plan (c) as both plans would only require 10 data cubes. However, if the caching strategy is set with a high value of α, then it could be that the last 60 daily cubes are in memory, and none of the other higher temporal level cubes. Hence, plan (a) would be favored here as it has zero disk access, while plans (b) and (c) would require six and two disk cubes, respectively.

2.8 User Interface

The User Interface module is where users can interact with RASED to perform interactive and comprehensive analysis, and navigate through all map updates worldwide. In particular, RASED interactive interface makes the system highly engaging where audience would not feel about the huge scale of data that the system maintains. Interested readers can refer to the live RASED system and interact with its friendly user interface at `https://rased.cs.umn.edu`. Figure 2.7 presents RASED landing page which is organized into two main sections: *Query Input Parameters*, located primarily on the top and left sides of the page, and *Output Visualizations*, which occupy the remaining area. We explain these components in details.

2.8.1 Query Input Parameters

As depicted in Figure 2.7, users can specify the following query parameters in RASED user interface: (1) temporal range to specify the dates of interest, (2) road types of interest (e.g., road networks, bus or cyclists routes), (3) type of OSM data (e.g., nodes, ways, or relations), and (4) nature of updates (e.g., create or modify). Furthermore, users can customize the view of the query results using two main options: (a) absolute/percentage statistics, which is crucial to see the amount of map updates relative to

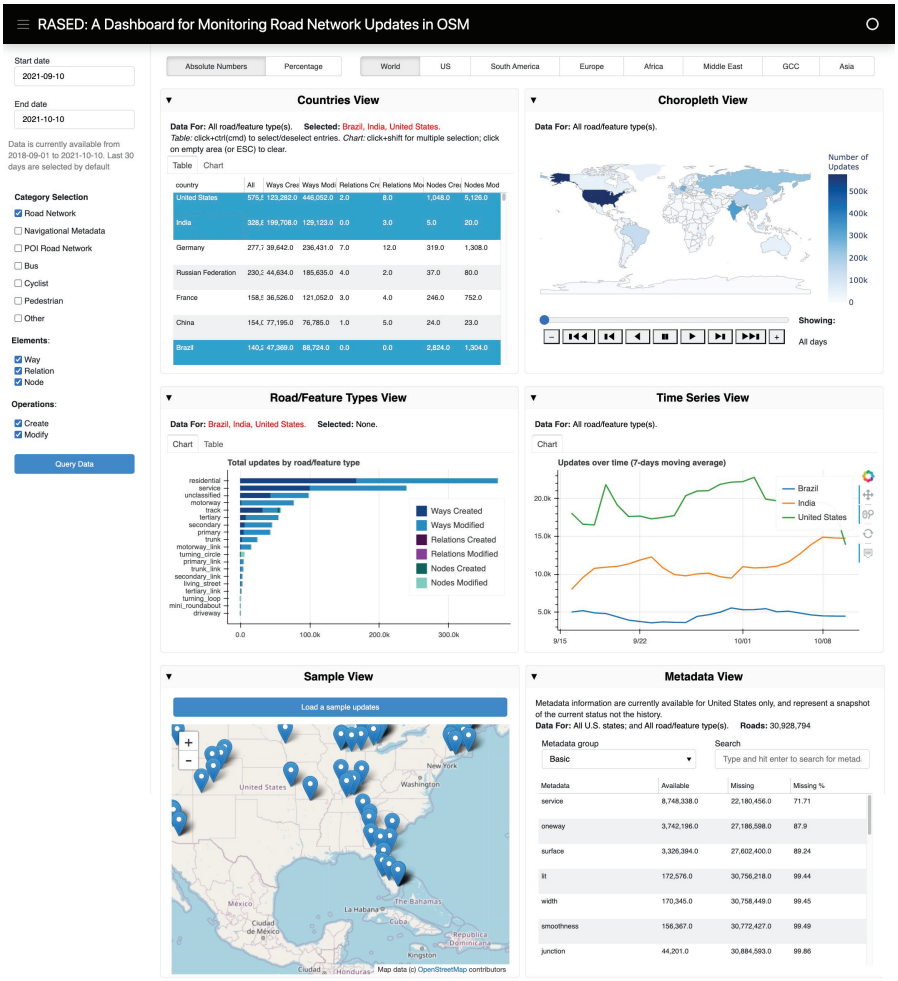


Figure 2.7: User Interface of RASED (`https://rased.cs.umn.edu`)

road network size of each country, and (b) the geographical regions to focus on, which could be the whole world or a specific region such as Europe. RASED users will be able to notice that no matter how large is the query temporal range or spatial region, the query response and all visualizations are highly interactive. This is mainly due to RASED efficient storage and querying techniques.

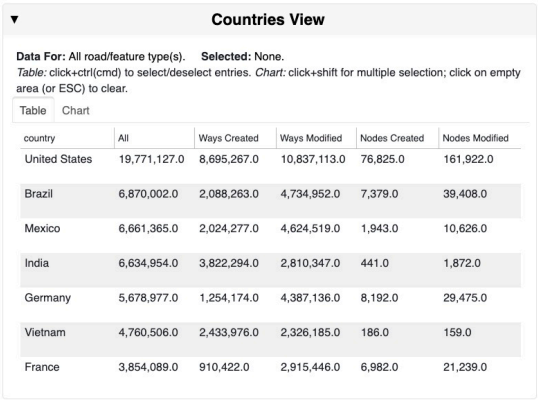


Figure 2.8: Country View

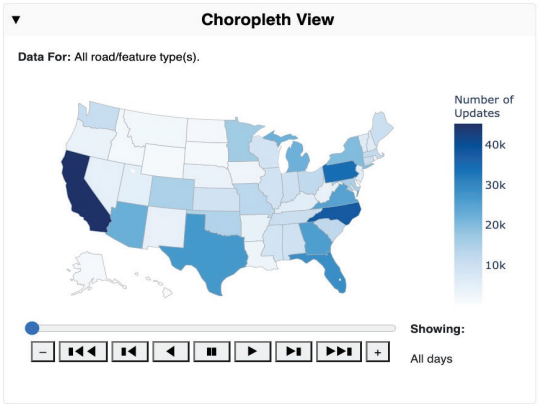


Figure 2.9: Choropleth View

2.8.2 Output Visualizations

Users can see the query results via the followig six output visualizations:

(1) Statistics per Country. Figure 2.8 depicts the Table view of all updates per country. Users can toggle this view as Table or Chart, and can sort entries based on the number/percentage of road changes per country. This helps road network analyzers to understand which countries have more changes in their road network either as an absolute number or percentage, and according to the input query parameters (temporal range, road type, and update type). The Chart view of this visualization is categorized by both the road type and nature of updates. Finally, users can use this view as an additional parameter to filter the results of all other visualizations. For example, one can click on one or more countries in the Table view, which will limit all other visualizations to show only the statistics of the selected countries.

(2) Time-lapse Choropleth View. Figure 2.9 gives a bird view of all map updates according to the query parameters and geographic region selected in the input query parameters. The view is presented as a Choropleth chart, where users can run a time-laps video of it, which helps in understanding the evolution of road network updates through the temporal range of interest.

(3) Statistics per Road/Feature Type: Figure 2.10 shows the Chart view for the update activity for every single road type (e.g., service and residential roads) for the

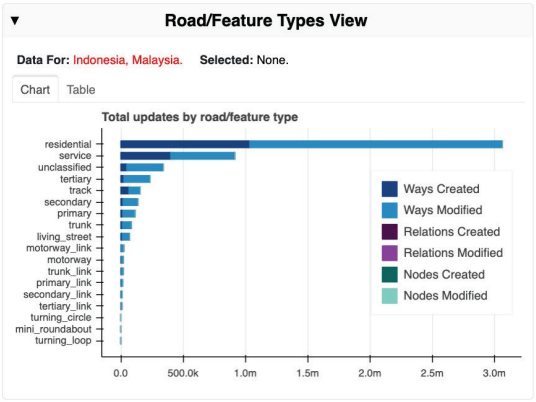


Figure 2.10: Road/Feature Types View

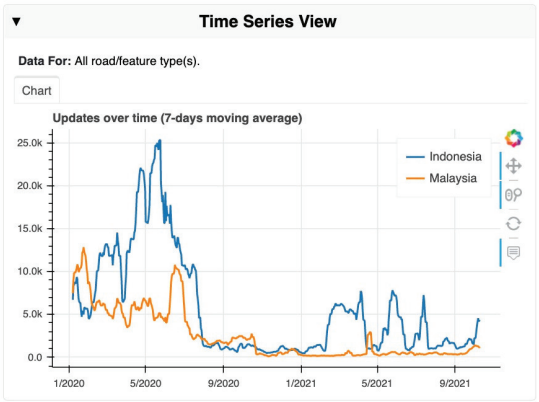


Figure 2.11: Time Series View

input query parameters set and the countries selected in the Country View. Users can toggle this view as Table or Chart, and can sort entries based on various statistics. The Bar Chart view is split into different colors indicating different map elements and different types of updates. This helps map analyzers understand which roads are being updated more than others in which country. In the Table view, selecting one or more road types will reflect on all other visualizations as filters.

(4) Comparison between Countries: Figure 2.11 shows a scenario where users can compare the road network update activities between multiple countries per certain query parameters and road/features types. Users can continue to add/remove countries or scroll to zoom in/out of the chart to inspect the values at different points of the timeline. Users will be able to witness the system scalability through the highly interactive analysis over the large scale data maintained by RASED.

(5) Inspection of Sample Updates: With the Sample View panel, users can retrieve a sample of map updates that satisfy the parameters set in all other visualizations combined. The samples are plotted on the map as pins showing their locations. Clicking on any of these pins would show the "before" and "after" status of this specific update.

(6) Metadata Completeness Check: Figure 2.12 depicts a view where users can check the percentage of roads annotated with certain metadata (e.g., maximum speed, number of lanes) in a tabular view, which can be sorted based on any of the columns.

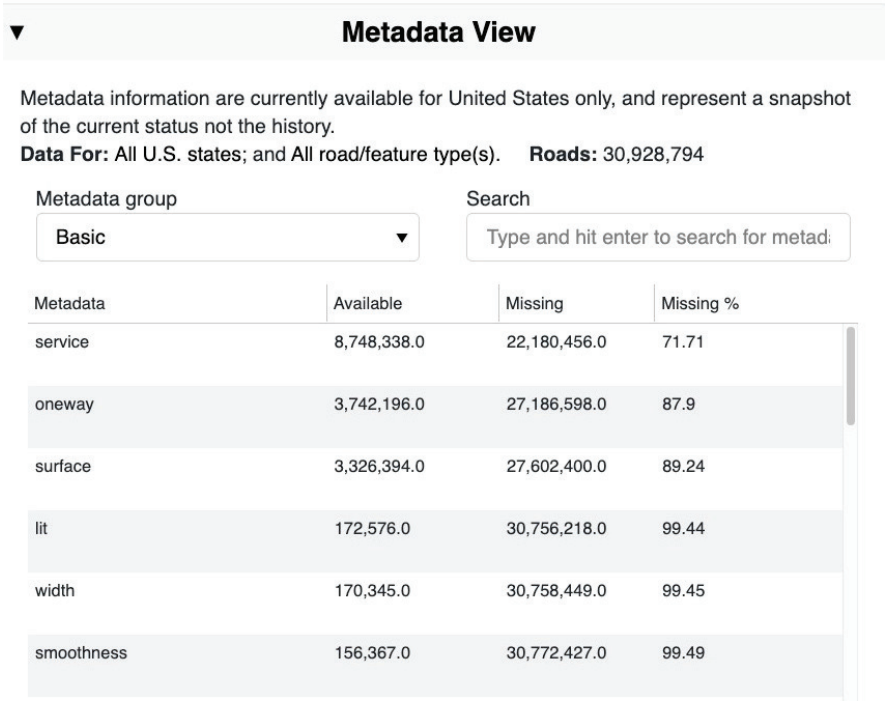


Figure 2.12: Metadata View

Users can also select certain road types (e.g., primary) or specific regions (e.g., Washington State) from other views to evaluate the coverage percentage only against these selected regions and road types.

2.9 Experiments

This section provides experimental evaluation of RASED to: (a) setup RASED parameters in terms of cache size and number of index levels (Section 2.9.1), (b) understand the performance gain from employing caching and level optimization strategies (Section 2.9.2), and (c) compare RASED overall performance against a traditional DBMS implementation (Section 2.9.3). All experiments are done on an actual deployment of RASED as a publicly available web service at: `https://rased.cs.umn.edu`. For evaluation, we use the OSM full history dump which contains more than 12 billion updates with a total size of 3 TB of raw data. We run the whole dataset through RASED Data Collection module to come up with the full *UpdateList* as: *<ElementType, Date,*

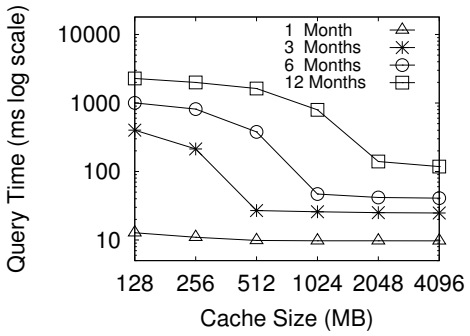


Figure 2.13: Setting Cache Size

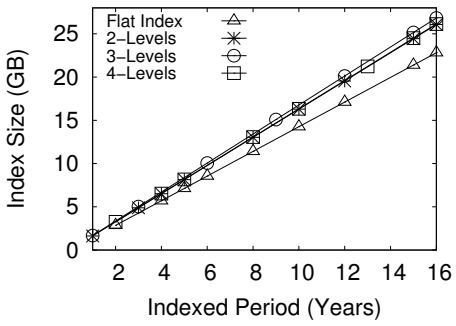


Figure 2.14: Setting Number of Levels

Country, Latitude, Longitude, RoadType, UpdateType, ChangesetID>, then bulk load the list into RASED temporal hierarchical index structure. We focus our experiments only on analysis queries (Section 2.4.1), as sample update queries (Section 2.4.2) are executed in a traditional DBMS way, so, there is nothing much to report about it. Our main performance measure is the query response time, which needs to be in order of milliseconds to ensure an interactive user experience of RASED analysis queries. Each point reported in all performance experiments is an average of 100 query execution. Unless mentioned otherwise, each query retrieves only one data cube cell to focus our performance results on the disk retrieval time, the default cache size N is 2GB, with α, β, γ, and θ are set to 0.4, 0.35, 0.2, and 0.05 respectively. All experiments are done using an Ubuntu system running on 8-core Intel(R) i7-4790 CPU @ 3.60GHz and 32GB of memory.

2.9.1 Setting RASED Parameters

This experiment aims to set the parameters of RASED index structure, namely the cache size and the number of levels in the hierarchical index. Figure 2.13 gives the query response time of RASED when varying the cache size from 128MB to 4GB, which can fit from 32 to 1,000 data cubes. We perform this experiment using various query loads with a time span of 1, 3, 6, and 12 months, which would reflect on the number of data cubes needed to answer each query. Clearly, the larger the cache size, the better the performance as higher numbers of data cubes can be retrieved from memory. For each query temporal window, there is a saturation point where increasing the cache

size will not have significant enhancement, e.g., 512MB, 1024MB, and 2048MB for the queries with 3, 6, and 12 months, respectively. Since RASED supports queries with large time windows, we opt to choose 2048MB cache size in RASED deployment. Figure 2.14 gives the size needed for each additional hierarchy level for RASED index when varying the covered period from one to 16 years, where a flat index means one level of daily cubes, while extra levels are for weekly, monthly, and yearly cubes. Apparently, the extra levels do not add much beyond the storage already needed for the first daily level. In particular, a four-levels index for a 16-years period would only take 1.15 of storage taken by a flat index for the same period. Hence, we opt to have our hierarchical index with four levels.

2.9.2 RASED Query Execution Strategies

This section aims to understand the performance gain from employing caching and level optimization strategies in RASED. In particular, Figure 2.15 gives the performance of three variants of RASED when varying the query time window from one to 16 years. The first variant (RASED-F) is a one-level flat index with neither caching nor level optimization. The second variant (RASED-O) is the full RASED index with level optimization, but no caching. The third variant is the full RASED system with both level optimization and caching. The more than two orders of magnitude performance gain from RASED-F to RASED-O shows the impact of having the index hierarchy, along with the level optimizer. Meanwhile, the order of magnitude performance gain from RASED-O to RASED shows the impact of deploying the caching strategy. Overall, both index hierarchy and caching boost RASED performance by three orders of magnitude.

2.9.3 Overall Performance

This section evaluates RASED against PostgreSQL implementation of the RASED analysis queries. To ensure fairness, we set PostgreSQL buffer size to 2GB similar to RASED cache size. Figure 2.16 gives the performance of RASED and PostgreSQL when varying the query time window from one to 16 years. PostgreSQL constantly takes around 1000 seconds to answer the analysis queries regardless of the query period or the aggregation size. This is mainly because it requires scanning the whole data since the query involves

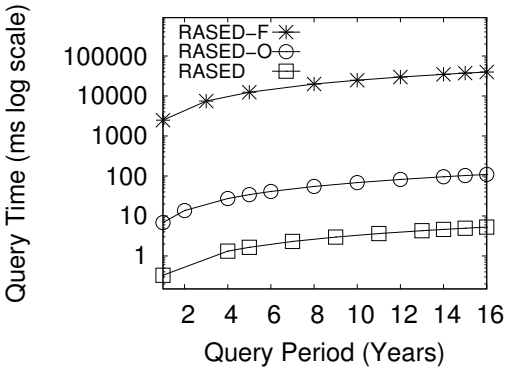


Figure 2.15: Effect of Each Component

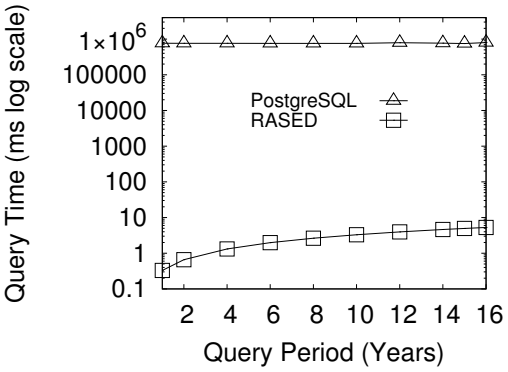


Figure 2.16: RASED Overall Performance

multiple attributes in the *Group By* clause. Meanwhile, RASED consistently achieves five to six orders of magnitudes better performance, reaching up to 10 milliseconds in its longest query period, which is due to its powerful index structure.

2.10 Conclusion

This chapter presented RASED; a publicly available scalable dashboard to interactively monitor and analyze all OpenStreetMap (OSM) road network daily updates worldwide. RASED supports a myriad of analysis queries that provide detailed statistics about road network daily updates activity, e.g., finding the number or percentage of road network updates over the last two years for a particular set of countries, finding the number of updates for each road type for a certain country over a certain time period, and comparing the road network evolution for a particular set of countries. RASED is equipped with a hierarchical temporal index structure and caching strategy that efficiently retrieve precomputed statistics needed for analysis queries. Results of RASED queries are visualized as either tabular format, various charts, or a timelapse video. RASED is highly interactive with milliseconds response to all its analysis queries. Realization of RASED has orders of magnitudes better performance than realizing similar ideas using traditional PostgreSQL DBMS.

Chapter 3

QARTA: An ML-based System for Accurate Map Services

3.1 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 [137, 136, 202, 262, 268, 289]. 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 [19, 37, 219, 232] or satellite images [17, 44, 237]. 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 chapter presents QARTA [1] ; an open-source full-fledged system that employs machine learning 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, is outdated in both 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 basic routing service to accommodating other important queries such as range and k-nearest neighbor queries. All such queries 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 the answer.

[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.

QARTA is built with two principles in mind: (1) *Map-centric.* QARTA believes that a key point to the success of all map services 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 3.2 describes QARTA system architecture. The three layers of QARTA, *data layer*, *map making*, and *query calibration*, are described in Sections 3.3-3.5. Experimental evaluation is in Section 3.6. Related work and conclusion are in Sections 3.7 and 3.8.

3.2 QARTA Architecture

Figure 3.1 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

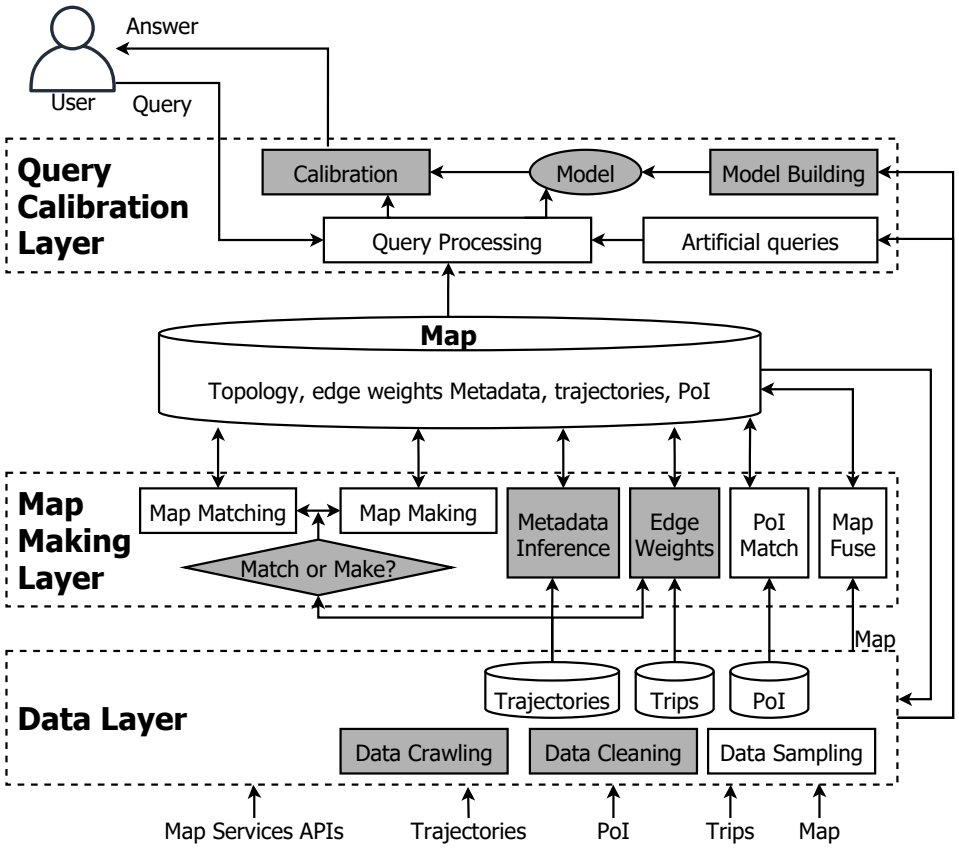


Figure 3.1: QARTA Architecture

layers, namely *Data layer*, *Map Making layer*, and *Query Calibration layer*, described briefly below. The gray modules in Figure 3.1 represent the new components designed only by QARTA, while other modules employ off-the-shelf solutions.

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 3.3.

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 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 3.4.

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 3.5.

3.3 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.

3.3.1 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 techniques for general data cleaning [116], they all fall short when dealing with spatial and trajectory data, mainly due to the spatial data distinguishing characteristics [98]. Recent attempts to trajectory data cleaning [73, 138] 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:

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 chapter.

3.3.2 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 [246] that provide access

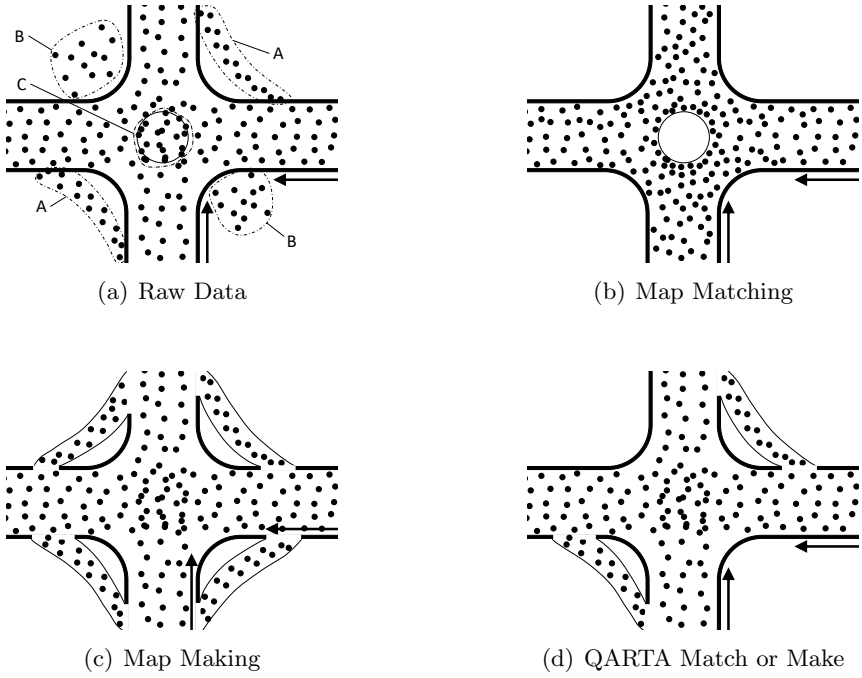
to limited datasets or explicitly from commercial services [231] by sending API routing calls with (*origin*, *destination*, *timestamp*). In case of commercial map services, it is crucial to optimize the number of API calls to accommodate low-budget enterprises. Hence, QARTA employs its own smart crawler 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.

3.4 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 [233]. Since there is already a plethora of techniques for building the road network and map matching (see Section 3.7), 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 3.4.1), which smartly decides whether we need to deploy map-making or map-matching, *Edge Weight Inference* (Section 3.4.2), which finds road segments edge weights, and *Metadata inference* (Section 3.4.3), which finds road segments metadata .

3.4.1 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 [33] have an implicit assumption that the underlying road network is ground truth, and



(a) Raw Data (b) Map Matching

(c) Map Making (d) QARTA Match or Make

Figure 3.2: Match or Make

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 [32, 119]. Figure 3.2a 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 3.2b 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 3.2c 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 3.2d 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 [104] using the scikit-learn Python library [206] 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.

3.4.2 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 [49], plate recognition [123], private GPS traces [114], or cell phone data [50], and are considered proprietary information. In addition, edge weights are usually presented as multiple values per edge (a.k.a time-dependent [61] or time-aggregated [82] 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 3.7), 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 [230]. 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:

$$\sum_{\tau \in \Gamma} (\sum_{e \in P_\tau} W_e l_e - \delta_\tau)^2, \tag{3.1}$$

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 [105].

Tuning Step 1: Heavy edges inference. The main problem in equation 3.1 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 3.1 becomes:

$$\sum_{\tau \in \Gamma} (\sum_{e \in P_\tau \cap H} W_e l_e + W_0 \sum_{e \in P_\tau \setminus H} l_e - \delta_\tau)^2. \tag{3.2}$$

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 [196], 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 3.2, 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 3.2 becomes:

$$\sum_{\tau \in \Gamma} (\sum_{g : P_\tau \cap H_g \neq \emptyset} W_g L_g + W_0 \sum_{e \in P_\tau \setminus H} l_e - \delta_\tau)^2. \tag{3.3}$$

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 3.3 by adding a regularization term that penalizes weights deviating from average speed:

$$\sum_{\tau \in \Gamma} \Big(\sum_{g: P_\tau \cap H_g \neq \emptyset} W_g L_g + W_0 \sum_{e \in P_\tau \setminus H} l_e - \delta_\tau \Big)^2 + \alpha \sum_g (W_g - \sigma)^2. \tag{3.4}$$

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 [206] for constrained ridge regression to find W that minimizes Equation 3.4. From our experiments, more than 99% of edge weights satisfy physical constraints. For the rest, we set edge weights to the minimum possible value $1/maxspeed$.

3.4.3 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 [296]. One way to fill in missing data is to model the metadata inference as a Graph Convolutional Network [47, 108, 126], 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.

3.5 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 [77] 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.

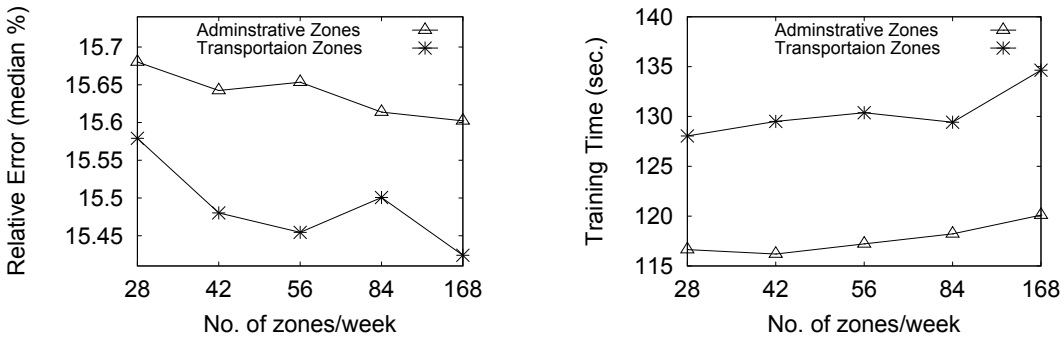
Query Calibration. Whenever QARTA receives a shortest path query, we pass it to two simultaneously working modules: (1) *Query processing*, where an off-the-shelf shortest path algorithm is applied to get the query answer with ETA. (2) *Calibration*, where we extract the feature vector v from the query 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.

3.6 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 ($\sim$ 4K vehicles) and a food delivery company ($\sim$ 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) [211], (3) NYC. 1.5M Taxi trips for a period of 6 months in New York City (250K nodes and 644K edges) [193]. 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 3.3), 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 [198], (c) QARTA uses the Open Source Routing Engine (OSRM) [200] for its off-the-shelf query library.

3.6.1 Setting QARTA parameters

This section aims to study and set the query calibration parameters in terms of the temporal and spatial zoning granularity. Figure 3.3 gives the effect of a finer granularity on the median relative travel time error (Figure 3.3(a)) and the model training time (Figure 3.3(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



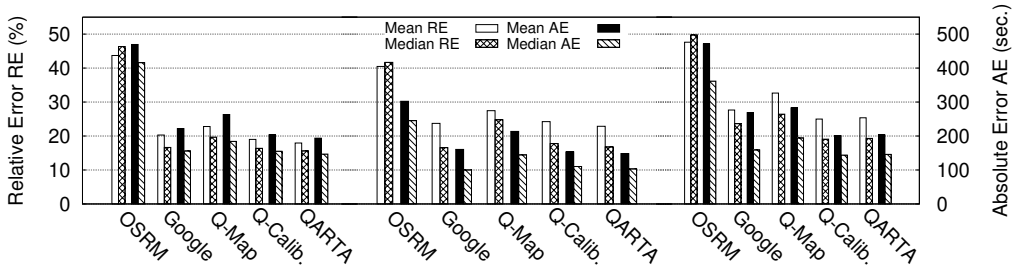
(a) Accuracy (b) Training Time

Figure 3.3: QARTA Spatial and Temporal Zoning

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.

3.6.2 Travel Time Accuracy

Figure 3.4 compares the overall accuracy of QARTA, Google Maps [89], and OSRM [200] 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 [196] (i.e., calibrating OSRM results), and the full QARTA. It is clear that OSRM has the worst performance in all datasets, mainly due to lack of traffic information, and hence its



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

Figure 3.4: QARTA vs Other Map Services

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 3.5 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 3.6 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

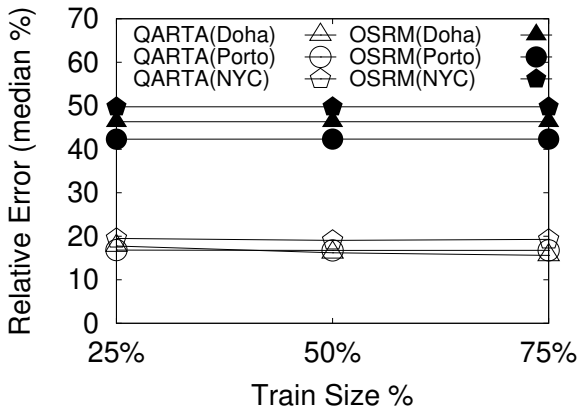
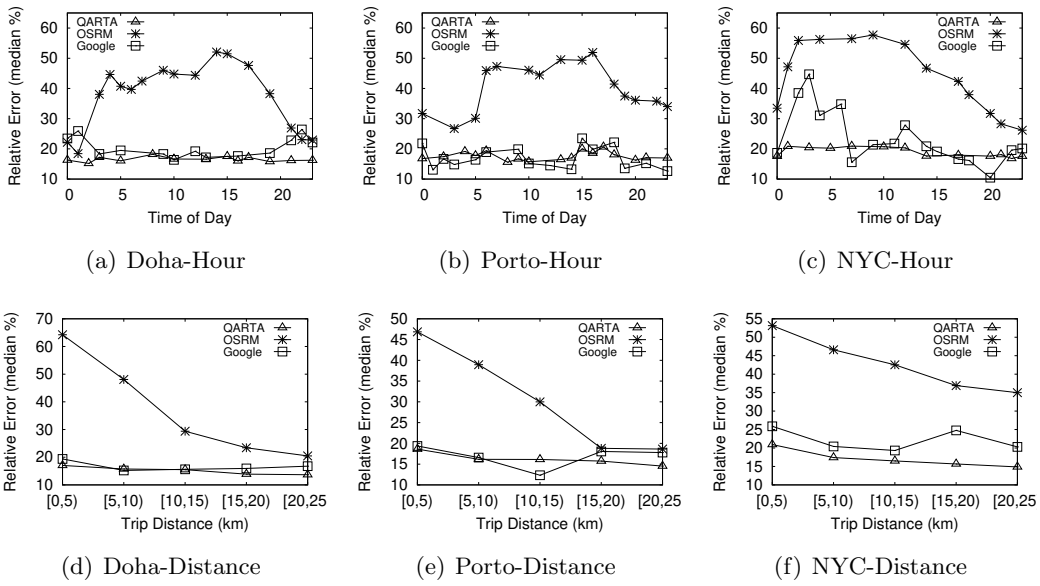


Figure 3.5: Accuracy vs Training Size

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.

3.6.3 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 3.7(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 3.7(b) uses the Normalized Discounted Cumulative Gain (NDCG), as the quality measure of the K-NN list. NDCG [118] is a widely used



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

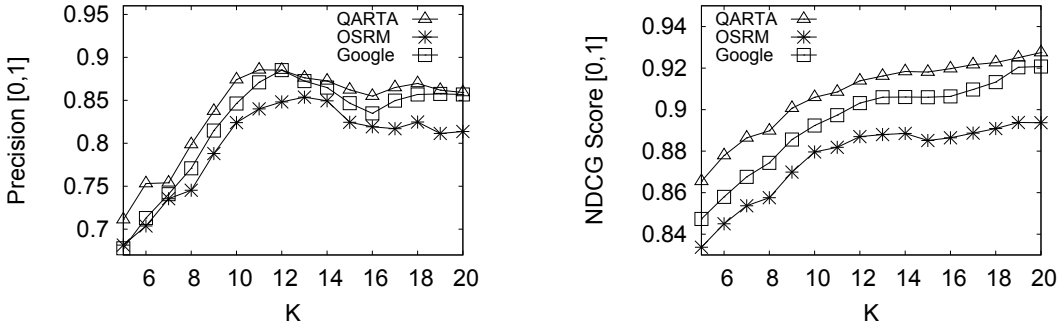
(d) Doha-Distance (e) Porto-Distance (f) NYC-Distance

Figure 3.6: Accuracy by Hour of Day and Trip Distance

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.

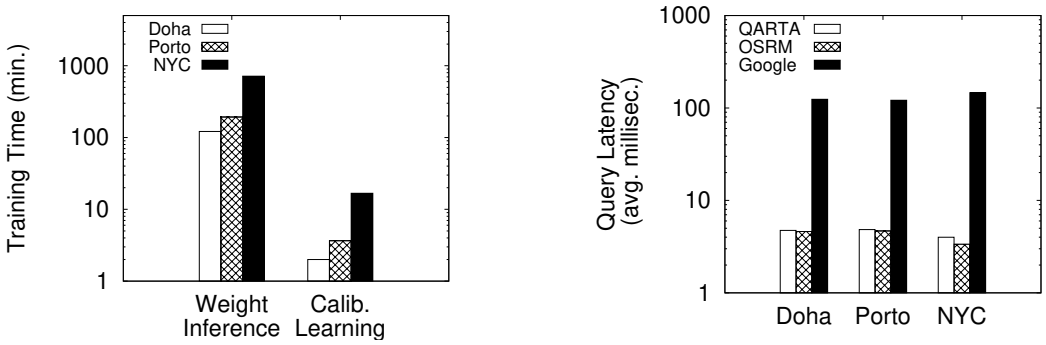
3.6.4 Query Performance and Training time

Figure 3.8(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 3.4, which depends on both the number of edges and number of trips, NYC needs the most time. Overall, the time to solve Equation 3.4 is acceptable, given 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.



(a) Precision (b) NDCG

Figure 3.7: K-NN Accuracy



(a) Training Time (b) Response Time

Figure 3.8: Performance of QARTA

Figure 3.8(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.

3.7 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 [73, 138, 293] or densifying trajectory points (a.k.a trajectory interpolation [156, 305], trajectory completion [144], trajectory restoration [135], or trajectory imputation [36]). QARTA can employ any of these techniques in its data layer. In addition, QARTA adds its own rule-based trajectory cleaning module that addresses 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 [19, 29, 37, 68, 99, 219, 232, 260]. (2) *Map Matching* [33], 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 [24, 31, 107, 143, 215, 265]. 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 [115, 182, 301], time-dependent [276, 303], or stochastic [108] weights for each edge, or *path-centric* [56, 275], 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 [56, 275] 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 [115, 303, 301] or tens of thousands edges [108, 182, 276]). Meanwhile, QARTA develops its own tuning steps to scale up Equation 3.1 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 [182, 301], static edge weights [115, 182, 301], and/or limited number of time-dependent weights [276, 303], 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 [137, 268], time-dependent routing [202, 262, 289], batch routing [136, 208], personalized routing [95, 141], and eco-routing [6, 93], 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 [221] are mainly geared towards replacing existing algorithms with machine learning models (e.g., [94, 108, 161, 142, 244, 279]). All these algorithms would fit in the Query Processing module of the Query Calibration layer, marked as white box in Figure 3.1, 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 3.1. *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 [109, 203, 205] that return probabilistic result distribution is planned for next QARTA release.

3.8 Conclusion

This chapter 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.

Chapter 4

Let's Speak Trajectories: A Vision To Use NLP Models For Trajectory Analysis Tasks

4.1 Introduction

Enabled by location tracking technologies, there have been a vast amount of trajectories collected by industry and academia. Many of such datasets have been publicly released for various cities around the world, including Athens [16], Beijing [281], New York [193], Porto [211], Rio de Janeiro [64], Rome [23], San Francisco [84], Shenzhen [254], and Singapore [113]. This has enabled a myriad of trajectory analysis techniques that have been the focus of the spatial community for years (e.g., see [236, 259, 306] for surveys). Such techniques have empowered numerous applications with high impact across many sectors. For example, in transportation, trajectory analysis has been vital in data-driven routing [204, 292], traffic monitoring [128, 172], and traffic forecasting [106, 251]. In location-based services, trajectory analysis was used in trip planning [25, 310], route recommendations [40, 79], and map services [184]. In urban planning, trajectory analysis has been a key to map inference [99, 232], deciding on the locations of bike lanes and EV charging stations [100, 145], and understanding human mobility [149, 272]. In the health domain, trajectory analysis has played an important role in contact tracing [7, 177, 271]

and understanding the pandemic spread [30, 238].

All such applications have to tackle a wide range of trajectory problems, including: (a) trajectory similarity search, where the objective is to find those trajectories that are considered similar to each other according to some predefined similarity measure [39, 140, 210, 60, 290, 278, 151], (b) trajectory imputation, where the objective is to add artificial points to a trajectory as a means of filling in the gaps between actual trajectory points [156, 135, 144, 267, 258, 305], (c) trajectory classification and clustering, where the objective is to cluster or classify trajectories either based on their modality, similarity, location, or other characteristics [234, 180, 217, 253, 307, 308], (d) trajectory prediction, where the objective is to predict the next few locations of the current trajectory points [75, 121, 152, 257, 288], and (e) trajectory simplification, where the objective is to sample some of the trajectory points without losing its main characteristics [122, 131, 154, 155, 263, 300].

However, with all of these techniques, there is an apparent lack of full-fledged systems that provide the infrastructure support for trajectory analysis tasks. Existing attempts to build such systems (e.g., see [5, 54, 132, 227]) are limited to providing the underlying index and retrieval storage, but did not reach to the stage for providing complete data analysis functionality. The main reason is that the focus of trajectory analysis techniques is mainly on the algorithmic part of the analysis, which gives much less weight to the need of having a solid system infrastructure. Hence, despite the fact that all of trajectory analysis techniques deal with the same trajectory data, each of the proposed solutions is entirely designed to solve one problem of interest. This makes it hard and not practical to have a unified efficient system that is capable of supporting most (if not all) trajectory problems, if each solution is entirely different.

Trying to learn from other communities, the research landscape of Natural Language Processing (NLP) was very recently in a similar situation. There have been decades of research in pushing the accuracy and efficiency of various NLP tasks, e.g., text similarity, text classification, sentence completion, and sentiment analysis. This has led to a myriad of different solutions for each of these problems, even though all tasks are for the same textual data. This is mainly for the same reasons that trajectory analysis techniques are entirely different from each other. Yet, most recently, in 2018, the BERT deep learning model [63] (Bidirectional Encoder Representations from Transformers), is

proposed by Google to act as a unified solution infrastructure for a wide variety of NLP tasks. BERT, at its core, is equipped with the necessary NLP infrastructure to solve various NLP tasks, which only needs to be externally tuned with minimal overhead for each task. Examples of NLP tasks that used the BERT model include sentiment analysis [67], question answering [43], spell checking [297], text classification [53], text generation[41], text summarization [212], among others [134, 207]. BERT has also been used for similar problems with respect to speech processing, where the words are spoken instead of written [130, 245]. As a testimony to the importance and ubiquity of BERT to NLP research, the main BERT paper [63] has been cited 60+K times within five years.

This chapter presents our vision towards using the same idea of BERT to magically deal with almost all trajectory analysis tasks. The goal is that BERT (or a customization of it) will do for trajectory analysis what it did already for NLP tasks. Should we be able to do so, various trajectory analysis ideas will be just about how to tune that BERT customized model to support the required analysis. Such vision will lead to a long-waited-for full-fledged trajectory data management system that does not only store and index trajectory data, but natively supports all its analysis needs. Our vision is grounded by the fact that we can actually think of trajectories as statements. In a nutshell, a statement is composed of a set of words drawn from a set of limited words (language), while a trajectory is represented by a set of GPS points, which is also drawn from a set of possible points (space). Statements follow rules imposed by the underlying language, while trajectories follow rules imposed by both the underlying road network and the physical world. Words in a statement should be semantically related, while points in a trajectory should be spatially and temporally related.

While it is theoretically possible to think of trajectories as statements, and hence trajectories can be fed to BERT as is to support various trajectory analysis tasks, this may not practically work due to various inherent limitations in both the BERT model structure and the nature of trajectory data. Hence, our vision is composed of two orthogonal directions. The first direction is to overcome the limitations coming from the nature of trajectory data, where we can customize trajectory data to be fit for BERT, and then use BERT as is. The second direction is to overcome the limitations of the BERT model itself, and make it spatially- and temporally-aware in a way that

it can support trajectory data. Both directions can be sought after together for the best performance. In all cases, the outcome of the vision is a BERT-like model that is not specific to one trajectory problem. Instead, it will act as a Swiss army knife that supports a myriad of diverse trajectory operations.

The rest of this chapter is organized as follows: Section 4.2 outlines the similarities between trajectories and statements, which presents the solid ground of our vision. Section 4.3 presents how BERT can be applied as is to five widely used trajectory analysis tasks, namely, trajectory imputation, trajectory prediction, trajectory classification, trajectory simplification, and trajectory similarity. The challenges that face our vision, which emerge from our attempts to use BERT as is for various trajectory analysis tasks, are outlined in Section 4.4. Section 4.5 presents our vision with its two orthogonal but complementary directions to use BERT for trajectories. Initial exploratory experimental results that show the promise of our vision are presented in Section 4.6. Section 4.7 presents the related work to our vision. Finally, Section 4.8 concludes the chapter.

4.2 Points in Trajectories vs Words in Statements

This section presents the rationale behind our vision, where we see that points in trajectories follow very similar properties to words in statements. Hence, tools and techniques that are being used in NLP tasks (e.g., BERT) can be employed to support trajectory analysis tasks. We outline four such common properties, namely, *limited domain*, *domain constraints*, *intra-relationship constraints*, and *clear context*.

4.2.1 Limited Domain

Both words in a statement and points in trajectory are drawn from a limited domain defined by the underlying language or space, respectively. In particular, a *statement* is composed of an ordered set of *words* drawn from a finite pool of *words* (domain) per the underlying *language*. Similarly, a *trajectory* is composed of an ordered set of *points* drawn from a finite pool of *points* (domain) per the underlying *space*. This particular property is key to BERT functionality when dealing with statements. In particular, when using BERT for a certain language, it is first fed with large numbers of statements from that language as training examples, which could be coming from any

set of online documents, e.g., Wikipedia articles. Then, BERT uses these documents to learn the domain of all possible words in the given language. That domain is then used to control BERT internal operations allowing it to understand any given set of statements, and hence perform NLP operations over it. Our vision is based on the fact that trajectories exhibit the same limited domain property as statements. Hence, one can feed BERT with large number of trajectories in a certain city instead of statements in a certain language. Then, BERT can use these trajectories to learn the domain of all possible points in the city and use that knowledge to understand any given set of trajectories and perform various analysis tasks on it.

4.2.2 Domain Constraints

Although statements and trajectories must pick their words and points from their corresponding domains, the order of such words and points must adhere to some domain constraints. In particular, *words* in a *statement* must adhere to the constraints imposed by the underlying *language grammar*. Similarly, *points* in a trajectory must adhere to the constraints imposed by the underlying *road network and physical space*. BERT is taking advantage of this to guide its operations. For example, when fed by large number of documents, BERT would understand the grammar by knowing that the verb "are" is only used with plural nouns. Hence, when given a new statement that does not follow this constraint, BERT may raise a flag of grammatical error. Also, BERT can use this knowledge to predict the next word. Our vision is based on the analogy that along the same lines, when fed by large number of trajectories, BERT can understand several constraints. Examples include understanding that the speed of movement never exceeds a certain limit or that the change of speed from one trajectory segment to another is within a certain range. Even further, it can infer parts of the underlying road network from the large set of trajectory GPS points it has. Hence, when receiving a new set of trajectories, BERT can validate it against its learned constraints and flag for errors if any. Also, it can predict the next point or impute missing points accordingly without invalidating the trajectory domain constraints.

4.2.3 Intra-Relationship Constraints

Both words in a statement and points in a trajectory must be intra related. In particular, *words* in a *statement* are *semantically* related, where random *words* cannot make a *statement*. Similarly, *points* in a *trajectory* are *spatially and temporally* related, where random *points* cannot make a *trajectory*. For example, the statement "John is drinking steak" satisfies the domain constraints (i.e., correct grammar), however, it does not satisfy the intra relationship constraint, where "drinking" is not semantically related to "steak". BERT takes advantage of such constraints as it learns them from its input documents. Hence, when used to check typos, find missing words or predict the next words, BERT will use its learnt intra relationship constraint and suggest the use of the word "eating" instead of "drinking" as it is more semantically related to the word "steak". Our vision is based on the analogy of the intra relationship constraint to the case of trajectories. A sequence of GPS points that still match the domain constraints (i.e., road network and physical) may not satisfy the intra relationship constraint, where one point of such sequence could be either from a nearby road or from the same road but in a different direction. When fed by trajectories, BERT should be able to learn such constraints and use them for various trajectory analysis tasks.

4.2.4 Clear Context

There is always a clear context that impacts the sequence of words in a statement or points in a trajectory. In particular, the *words* used in a *statement* would differ based on the *topic of discussion* (context). Similarly, the *points* used in a *trajectory* would differ based on the *driving modality* (context). For example, the words used in a medical document are pretty different than those words used in a political document, even though both documents are in the same language. BERT uses this property to learn the underlying context of statements in a document and uses this knowledge to perform various NLP tasks. If BERT can identify that a certain statement or document is coming from a medical context, it will act differently (in terms of looking at different vocabulary) from the case where the statement is coming from a political context. With a similar analogy, our vision is based on the fact that trajectories also follow a clear context. For example, the driving modality (e.g., vehicles, buses, motorbikes, bikes)

impact the sequence of points used in a trajectory in terms of different speeds, different driving patterns, and even different lanes and roads. Hence, when fed by trajectories, BERT could understand the underlying context of these trajectories, and then use this knowledge to understand the context of any new given trajectory and perform various tasks on it. For example, if BERT can identify that a certain trajectory belongs to a bike, it will act differently from the case if the trajectory is coming from a bus.

4.3 BERT for Trajectory Analysis Tasks

This section discusses the first step towards our vision, by showing that five different widely used trajectory analysis tasks are pretty analogous to corresponding five widely NLP analysis tasks. In particular, we show that *trajectory imputation* is analogous to *find the missing word* problem (Section 4.3.1), *trajectory prediction* is analogous to *next sentence prediction* (Section 4.3.2), *trajectory classification* is analogous to *text classification* (Section 4.3.3), *trajectory simplification* is analogous to *text summarization* (Section 4.3.4), and *trajectory similarity* is analogous to *text similarity* (Section 4.3.5). Since BERT is widely used to solve all these five NLP tasks, with the same analogy we also show that BERT can *potentially* be used to solve their corresponding five trajectory analysis tasks. We use the word *potentially* here as this may not practically work right away. This is due to many challenges that we will outline later in Section 4.4 and need to address to realize our vision of a BERT-like model to execute various trajectory analysis tasks.

4.3.1 BERT For Trajectory Imputation

This section presents the analogy between the "*trajectory imputation*" problem and "*Finding the missing word*" NLP task, which is commonly solved using BERT.

Trajectory Task: Imputation. Trajectory data are inherently sparse, with large and frequent spatial and temporal gaps between every two consecutive GPS readings. This is mainly either due to low GPS sampling rate to preserve the bandwidth, battery, and/or storage in location tracking devices, or due to loss of GPS signal in some areas such as tunnels and near high-rise buildings. Such gaps present an inherent uncertainty of the object's whereabouts between each two GPS readings, which affects all applications that

rely on trajectory data. The higher the sparsity (i.e., the larger and more frequent such gaps spatially and temporally), the lower the accuracy and quality of both trajectory data and the applications that rely on it. To address the sparsity issue, and as a means of boosting the accuracy of trajectories and their applications, several recent efforts were dedicated to insert artificial location points between each two consecutive trajectory points. The promise is that these artificially imposed points are as accurate as if they were obtained by actual GPS readings of trajectory data. Such a process had various names, including trajectory interpolation [156, 305], trajectory completion [144], trajectory data cleaning [293], trajectory restoration [135], trajectory map matching [24, 157], trajectory recovery [267, 258], and trajectory imputation [36]. Without loss of generality, we will use the term "trajectory imputation" in this chapter. With the exception of few techniques [69, 144, 185], the large majority of existing trajectory imputation techniques rely on matching the trajectories on the underlying road network, and hence they have an implicit assumption that the underlying road network is available and reliable, which is not always true. Road networks, like any other data, suffer from all sorts of inaccuracy, and may not be even available in many places [186, 170, 243, 174]. This calls for developing new imputation techniques that do not require the knowledge of the underlying road networks.

NLP Task: Finding the Missing Word. Consider, for example, the incomplete English statement "Paris, the ... of France, is ... Summer Olympics in 2024", where each blank "..." represents a missing word. *Finding the missing word problem* (a.k.a cloze test) aims to infer the words that are replaced by a blank. In this case, the first missing word would be "capital" and the second missing word is "hosting". This is one of the main and very common tasks in NLP, as there are several practical scenarios behind having missing words in a statement. For example, speech and image recognition techniques may partially recognize a statement, with some (missing) words that could not be recognized and left as blank. A translation task may miss translating some words and leave them as blank or mark them as low confidence. A typo in a text could be replaced by a blank if it was not corrected b a spell checker. In such scenarios, finding the missing word problem aims to fill in the blanks. BERT has been used to solve finding the missing world problem, where it is first trained by hundreds of thousands of true statements that will make it able to understand the context of any sentence and

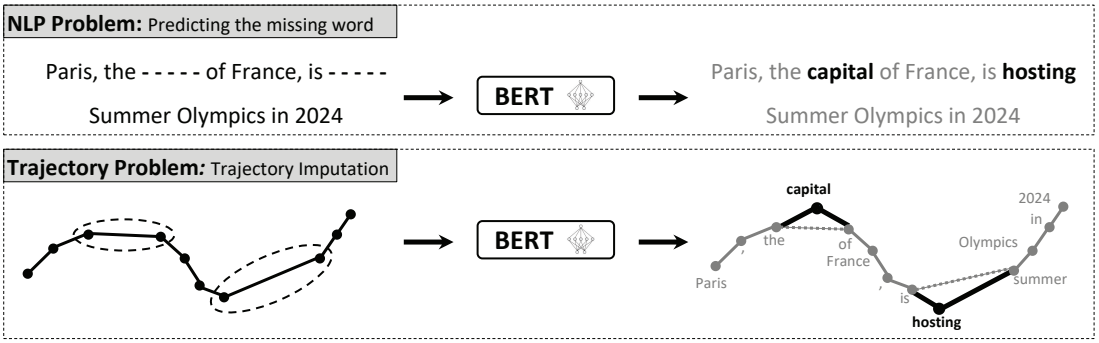


Figure 4.1: Trajectory Imputation Using BERT Model

accurately find out its missing words.

The Analogy. Figure 4.1 shows the analogy between *trajectory imputation* and *finding the missing word* problems. Instead of the blanks "...", we circle the segments that have significant gaps between their end points. For clarity, the example aligns the statement and trajectory together where the statement is composed of 11 words (including two punctuation marks) with two more words marked as blank, while the trajectory is composed of 11 points with two segments identified as need to be imputed. The question that we are asking in our vision here is that: If BERT, when trained with large number of statements, can identify the two missing words as "Capital" and "Hosting", can BERT be trained with large number of trajectories, and then used to impute a trajectory by identifying its missing points. Should we be able to do so, we would be solving the trajectory imputation problem, without the need for the knowledge of the underlying road network, making the solution applicable to much wider set of problem settings.

4.3.2 BERT For Trajectory Prediction

This section presents the analogy between the "*trajectory prediction*" problem and "*next sentence prediction*" NLP task, which is commonly solved using BERT.

Trajectory Task: Prediction. Trajectory prediction is the task of predicting the next few points of a current trajectory. This is one of the very common tasks in trajectory analysis as it is a cornerstone to many practical applications. For example, knowing

where current vehicle trajectories are heading to enables traffic monitoring and forecasting where congestions can be expected ahead of time, and hence an appropriate action can be taken [125, 120]. It also enables events forecasting where one can predict expected gathering events at certain locations [251, 124]. Wireless communications can strengthen cellular connectivity by preparing the next few predicted cell towers to admit mobile devices based on the predicted workload [58, 192, 229]. Personalized services and recommendations, which include offering location-based information or advertisements, benefit from predicting vehicle trajectories to personalize their offering [57, 239]. Overall, the ability to do trajectory prediction has enabled a whole area of research in spatio-temporal predictive query processing, where the objective is to support spatio-temporal queries asking about a future time rather than the current state [102, 101]. Due to its importance, significant efforts have been dedicated to support various forms of trajectory prediction including short-term prediction (next few minutes) [75], long-term prediction (next 20-30 minutes) [288, 121], or prediction for extended periods (e.g., the rest of the day) [222]. These solutions are based on several frameworks including statistical patterns [87, 179], machine and deep learning [152, 121, 75], and Markov models [14, 171].

NLP Task: Next Sentence Prediction. Consider, for example, the English statement "Paris is hosting Summer Olympics in 2024", the *next sentence prediction* aims to predict, with probability, what would be the next sentence among a set of options. For example, if the following three statements are potential ones: "Biden announced a Tax Relief program", "Milan is hosting Winter Olympics in 2026", and "Paris is the capital of France", a *next statement prediction* algorithm may give probabilities 1%, 70%, and 29% for these statements, respectively, which recommends that the second statement is the most likely one to come next after the input statement. This has practical applications including sentence auto completion and text generation. The *next sentence prediction* problem is usually solved using a BERT model. To do so, BERT is first trained and fine-tuned using pairs of ¡*input*, *target*¿ sentences. It gets such pairs from its input training data, composed of hundreds of thousands of statements in documents. This will make BERT understand a target statement, given an input one, and hence is able to accurately predict that next target statement.

The Analogy. Figure 4.2 shows the analogy between *trajectory prediction* and *next*

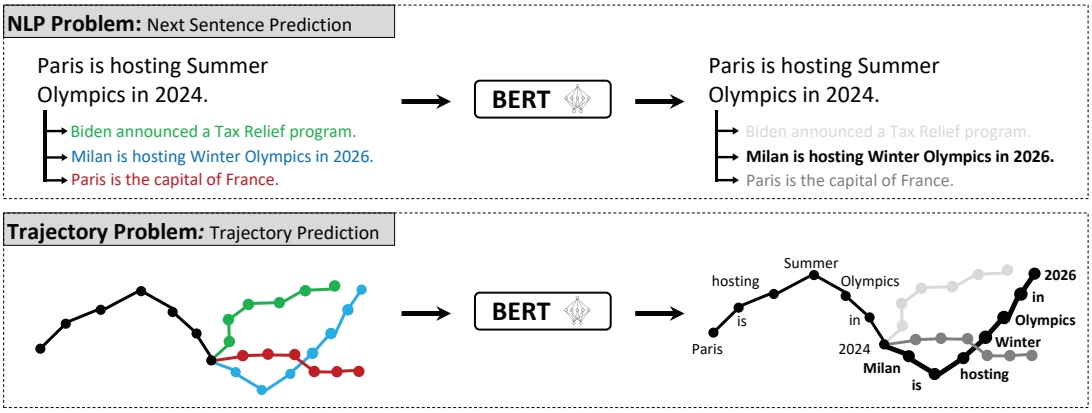


Figure 4.2: Trajectory Prediction Using BERT Model

statement prediction problems. For the latter one, we plot the three candidate next statements in three different colors. For the case of trajectory prediction, we plot the three potential trajectories in the same three colors to their corresponding statements. For clarity, each (potential) trajectory has the same number of points as the number of words of its corresponding statement. Hence, one way to solve the trajectory prediction problem is to train a BERT model with real trajectories. Then, BERT can split these trajectories into large numbers of pairs of sequence trajectories in the form of *input* and *target* trajectories. In a way analogous to what BERT is doing for the *next sentence prediction*, it can use its trajectory-based trained model to predict (with probability) the next trajectory among the three possible options. Similar to the case of trajectory imputation, a great advantage of using BERT for trajectory prediction is that it can do so without the need to know the underlying road network, which would distinguish it from all existing trajectory prediction approaches.

4.3.3 BERT For Trajectory Classification

This section presents the analogy between the "*trajectory classification*" problem and "*text classification*" NLP task, which is commonly solved using BERT.

Trajectory Task: Classification. Trajectory classification is the process of associating a trajectory with one class from a predefined set of classes. A prime example of such

operation is associating a trajectory with a driving modality that could be either bike, bus, vehicle, or even a walking trajectory [180, 217, 234, 253, 307, 308]. Such classification is crucial in many practical applications. For example, to infer an accurate travel time estimation (ETA) out of a large set of trajectories, a pre processing step needs to separate vehicle, bus, and motorbike trajectories from each other. Then, use each set of trajectories separately to have a modality-based ETA. Map inference algorithms that are used to infer the underlying vehicle, bus, and bike routing maps would need to first classify a given set of trajectories based on their modality, and use them separately to discover the corresponding map. Another example of trajectory classification is associating a trajectory with its travel purpose that could be either work commute, shopping, education, or recreation [22, 166, 167, 228, 59, 270]. This has a direct application in urban planning as a means of helping decision makers understand travel demands and relationships among neighborhoods. Existing approaches for trajectory classification rely on either hand-crafted rules for each class, heavily engineered features to train supervised machine learning models, or customized deep learning models.

NLP Task: Classification. Text classification is one of the widely used analysis tasks in NLP for a large number of important and practical applications. The objective is that given a set of text, associate each text with one class from a predefined set of classes. A prime example would be classifying social media posts (e.g., tweets) into their topic category, e.g., sports, politics, news, technology, and health. This is an important preprocessing operation for various data analysis procedures that need to have the analysis based on only posts that belong to a certain category. Another example is sentiment analysis, where social media posts are classified per their sentiment category, e.g., happy, angry, sarcastic, and lukewarm. This is important in market study to understand the reaction of users to a certain product, advertisement, or article. The problem of text classification is usually solved using a BERT model. To do so, BERT is first trained on a large number of *unlabeled* sentences to learn about words in general and how they relate to each other. Then, it is fine-tuned using relatively smaller *labeled* sentences as ¡*tweet*, *category*¿. Such trained model is then used to decide on the category of a given tweet.

The Analogy. Figure 4.3 shows the analogy between *trajectory classification* and *text classification* problems, where text classification is used to classify tweets into politics,

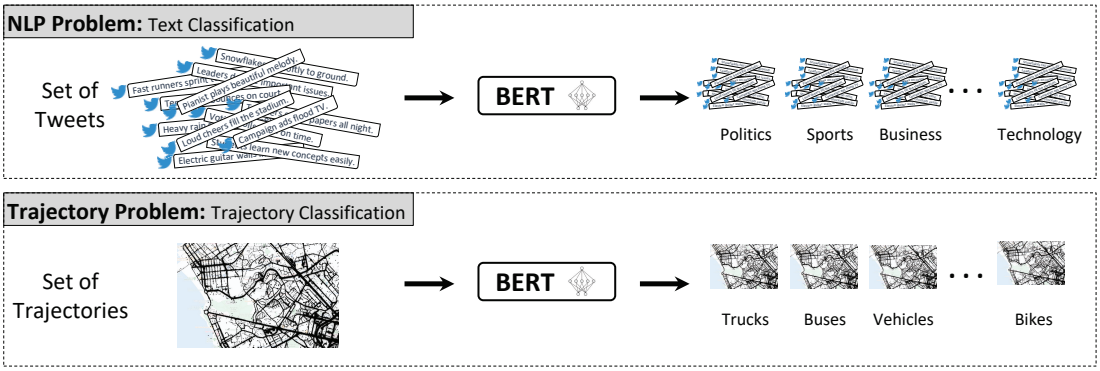


Figure 4.3: Trajectory Classification Using BERT Model

sports, business, and technology, while trajectory classification is used to classify trajectories into trucks, buses, vehicles, and bikes. Hence, one way to solve the trajectory classification problem is to go through a similar procedure of using BERT for the text classification problem. In particular, we start by training a BERT model using large numbers of *unlabeled* trajectories. Then, fine-tune the model using a relatively smaller set of *labeled* trajectories of the form ¡*trajectory*, *modality*¿. Finally, use the refined model to infer the modality of a given trajectory.

4.3.4 BERT For Trajectory Simplification

This section presents the analogy between the "*trajectory simplification*" problem and "*text summarization*" NLP task, which is commonly solved using BERT.

Trajectory Task: Simplification. Trajectory simplification, sometimes seen as the opposite of trajectory imputation, is the task of reducing the number of trajectory GPS points while preserving their essential information. Trajectories can be vast and complex, making it costly and challenging to transmit (e.g., bandwidth in cellular phones is limited), store (e.g., managing and storing these large datasets can become prohibitively expensive), and process (e.g., executing queries on such datasets becomes more complex). Trajectory simplification lowers the number of points and thus significantly reduces the cost of query processing, storage, and data transmissions of complex trajectories. It can be employed either in online or offline mode. In online mode, during

data collection, trajectory simplification helps tracking devices transmit only the most important points in real time. In offline mode, a simplified trajectory is obtained from the full trajectory and used either for storage or as a filter step during query processing, with refinements done using the full trajectory data. Due to the importance of trajectory simplification problem, significant research efforts were dedicated to solve it (e.g., see [122, 131, 154, 155, 263, 300]). Most of these approaches aim to maintain a certain distance or direction error threshold when dropping some of the original points. Some of these approaches aim to match trajectories to the underlying road network, and then simplify each trajectory by selecting representative road network points (e.g., intersections) [127, 280, 139].

NLP Task: Text Summarization. Consider, for example, the verbose English statement "Paris, the capital of France, is hosting the Summer Olympics in 2024". A *text summarization* procedure would make a shorter version of this statement to be: "Paris is hosting 2024 Olympics". Given a document of words, a text summarization analysis task aims to summarize the document by a short description which can be used for newsletters, video descriptions, or brief highlights. Such short description uses less number of words while preserving the main ideas of the original text and minimizing the information loss due to the removed text. The outcome summary can be either *extractive*, i.e., uses only words from the original text, or *abstractive*, i.e., can come up with entirely new words and phrases that were not present in the original text. In our example, we used an extractive summarization as all the words are taken from the original statement. BERT has been widely used for the text summarization problem. To do so, similar to the case of text classification, BERT has to be first trained on large number of documents to build its initial model. Then, the model is fine-tuned using another smaller set of document and summary pairs to learn what would be the words to use or omit to come up with a document summary. Finally, given a document, BERT would use its learnt model to produce the summary.

The Analogy. Figure 4.4 shows the analogy between *trajectory simplification* and *text summarization* problems. For clarity, we plot the trajectory before and after simplification with the same number of points that corresponds to the statement before and after summarization. Same like the case of text summarization where the summary still preserves the full meaning of the statement, the simplified trajectory still preserves

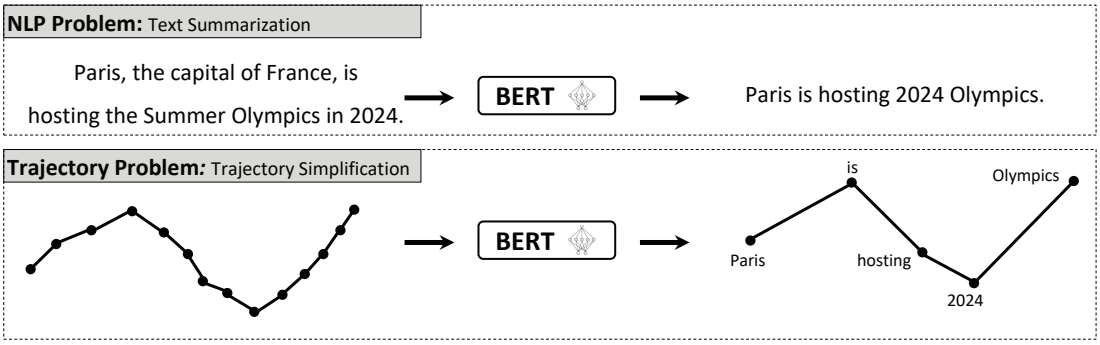


Figure 4.4: Trajectory Simplification Using BERT Model

the characteristics and overall structure of the full trajectory where main turns are still kept. As BERT is used for text summarization, one way to solve the trajectory simplification problem is to train a BERT model with large number of real trajectories, and then fine tune the model using pairs of the form ¡*raw trajectory*, *simplified trajectory*¿. Though our example shows an *extractive* simplification, where the simplified trajectory uses only points from the full trajectory, but the idea can also be applied to *abstractive* simplification. Similar to previous trajectory analysis tasks, using BERT would allow trajectory simplification to be done without the need to be aware of the underlying road network.

4.3.5 BERT For Trajectory Similarity

This section presents the analogy between the "*trajectory similarity*" problem and "*text similarity*" NLP task, which is commonly solved using BERT.

Trajectory Task: Similarity. Trajectory similarity is the process of computing a similarity score between two trajectories based on their sampled GPS points. This is one of the commonly performed tasks in trajectory analysis as it is a fundamental step to many practical applications. For instance, in routing and navigation applications, given a historical dataset of trajectories, a source point, and a destination point, it is often needed to compute the estimated travel times (ETA) for a certain route between source and destination points [184, 261]. Such information is highly valuable to help plan routes more efficiently, or to add travel time information to the map. In this case,

trajectory similarity is utilized to find trajectories that closely resemble the given route, then use these trajectories to help estimate the route travel time. In addition, several other trajectory analysis tasks rely on trajectory similarity. For example, in clustering and classification tasks (Section 4.3.3), computing the similarity score between two or more trajectories is essential. In particular, such tasks use the similarity scores to correctly assign each trajectory to an appropriate cluster based on its similarity to the other trajectories in each cluster. The higher the score to a certain cluster, the closer the trajectory to it and the higher the trajectory probability to belong to it. Another example of analysis tasks that utilize trajectory similarity is outlier detection. This is a very important preprocessing step for any downstream application that uses trajectories as it can help in cleaning such trajectories and ensure the application receives high-quality input data. In this task, given a dataset of trajectories, similarity scores are computed for each trajectory with respect to the reset of trajectories in the dataset. Then, those trajectories that are significantly less similar to the majority of the data are flagged as outliers, which then can be eliminated or processed independently via data cleaning tools. Due to the importance of trajectory similarity in various applications, significant research efforts have been devoted to it (e.g., see [140, 151, 60, 278]). Many of these approaches rely on pairwise computations between the sampled GPS points to find the similarity score, which may become prohibitively expensive with large datasets and large numbers of GPS points.

NLP Task: Text Similarity. Consider, for example, an input English statement such as "The capital city of France", the *text similarity* aims to compute how similar (or relevant) this statement is to a set of available statements or documents. For example, if the following three statements are the available ones: "Paris is the largest city in France", "Marseille is the second largest city in France", and "Paris is France's capital city", a *text similarity* algorithm may find the similarity scores between the input statement and the three available statements as 0.12, 0.03, and 0.85, respectively, which recommends that the last one is the most similar statement to the input. This has many practical applications including search and information retrieval, results ranking, and question answering. This is in addition to clustering applications, e.g., categorizing a set of tweets into topics where tweets in each cluster are highly similar to each other. Similar to many other NLP tasks, *text similarity* is usually solved using a BERT model.

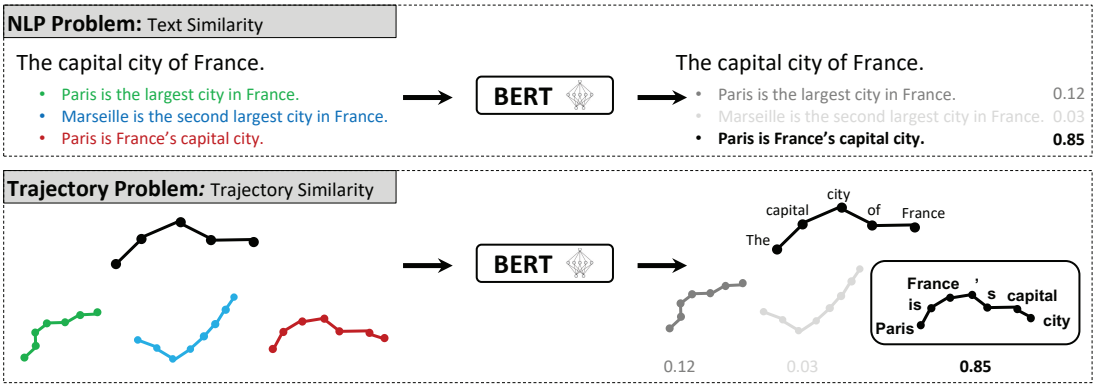


Figure 4.5: Trajectory Similarity Using BERT Model

Given two statements, BERT represents them as two vectors of the same size, regardless of the number of words in each statement. The vectors then go through a mathematical similarity measure, e.g., cosine similarity or Euclidean distance, to measure their similarity. The similarity score is then used to represent how both statements are similar to each other. To do so, BERT is first trained on large datasets of statements so it can learn relationships between words and then be able to represent similar statements by similar vectors.

The Analogy. Figure 4.5 shows the analogy between *trajectory similarity* and *text similarity* problems. For the latter one, we have an input statement and three other available ones, which we plot in different colors. Similarly, for trajectory similarity, we have an input trajectory and three available ones. For clarity, each of the three trajectories has the same number of points as the number of words of its corresponding statement. We also plot the three trajectories in the same three colors to their corresponding statements. The goal is to compute their similarity to the input. Hence, one way to solve the trajectory similarity problem is to train BERT on a large trajectory dataset so it can learn how to represent similar trajectories by similar vectors. Then, we can use these vectors to easily compute the similarity between their corresponding trajectories. This has the potential to scale up since the vector size is constant regardless of the number of GPS points, which would distinguish it from other existing pairwise similarity approaches.

4.4 Challenges in using BERT for Trajectory Analysis

The clear analogy between trajectories and statements, and hence between their corresponding tasks calls for applying state-of-the-art NLP models, e.g., BERT, for trajectory operations. The hope is that given the apparent analogy, BERT will be as successful for trajectory analysis as it is already successful for NLP tasks. However, this may not be practically applicable. In fact, if we apply BERT as is to trajectory analysis tasks, it will yield highly inaccurate results, and may not even work in most cases. This is mainly due to several fundamental differences between trajectories and statements, which hinder the applicability of BERT to trajectory operations. This section presents the below three main challenges that stem out from these differences, which need to be addressed first to pave the way for our vision of a BERT-like model for a myriad of trajectory analysis tasks.

Challenge I: Ratio of Training Datasets To Possible Words. BERT is designed to be used for languages, where: (a) the number of possible words is of limited size, and (b) there is an abundance of publicly available data, including Wikipedia pages, news articles, and various websites. In fact, BERT was actually trained on ∼3.3B word corpus (2.5B of them are from Wikipedia and 800M from Books Corpus [314]) composed of ∼30K distinct words [63]. This means that, on average, each English word appears ∼100K times in the BERT training set. This gives BERT the ability to see each word in various contexts, and hence would be able to understand various context-based meanings of each word as well as linking words together when they co-occur often. This is basically due to the law of large numbers that BERT heavily relies on. Meanwhile, trajectory datasets exhibit a very different behavior. First, the number of possible points that can patriciate in a trajectory is huge and highly depends on the resolution of the GPS tracking devices. For example, assuming a 5 meters accuracy of a GPS device, then a $1km^2$ area would have 40,000 GPS points. Considering a state like Minnesota, which has an area of around $225{,}000km^2$, we would have 9B possible GPS points in only one state. Of course, the numbers would be much less if we only focus on roads, biking trails, and sidewalks, which are the ones that appear in trajectory datasets, yet, still the numbers would be orders of magnitudes large than language datasets where the English word has only around 30K words (points). Second, due to many factors,

including privacy and proprietary ownership, trajectory data are not widely publicly available like language data. With such limited data, distinct GPS points would hardly appear in training datasets. For example, a trajectory dataset from Oregon State, obtained from UCR STAR [249] has ∼1.3M distinct GPS points with ∼1.75M total points. This means that, on average, each trajectory point appears only once in the training datasets, which is five orders of magnitude *less* than what English words appear in their training datasets (100K times). Apparently, such low ratio of training datasets to possible words is not suitable for BERT. In its core, BERT heavily relies on having each word appears hundreds of thousands of time in the training data, which would allow BERT to understand various contexts and witness different scenarios of each word. This makes BERT able to support various NLP tasks that all rely on the context understanding of BERT for each word in a statement. Hence, applying BERT as is to trajectory data, with its scarce data availability and large numbers of possible words, is not practical.

Challenge II: Ratio of Noisy Data. Language data is subject to noise that takes place in typos, grammatical errors, words that could not be extracted from images, or words that were not well translated from another language. Overall, noisy data is considered as outliers in the whole language dataset. BERT is kind of used to this, as given its large numbers of training datasets, it can identify the outlier noisy parts of the data and act accordingly. Meanwhile, noise in trajectory data is inherent, and it could be even more than accurate data. This is due to the inherent inaccuracy and the low sampling rate of GPS tracking devices as a means to accommodate bandwidth and battery limitations. Such high ratio of noisy data makes it very hard for BERT to learn from its own training set. Noisy data will increase the number of distinct points in a dataset, which will decrease the number of times that each point appears in the training set. Besides affecting BERT training process, noisy data significantly impacts the applicability of BERT to some trajectory analysis tasks. For example, consider the trajectory imputation task, which is analogous to finding the missing word NLP problem (Section 4.3.1). In the NLP problem, BERT is usually deployed to find only one missing word. Yet, errors that lead to trajectory imputation are usually coming from low sampling rates, which would drop out several points between each two consecutive points. Applying BERT as is to trajectory imputation will end up having only one point

between the two segment end points. While this would increase the trajectory accuracy, but it is not enough as, practically speaking, we would need to fill in multiple points in between two segment end points.

Challenge III: Long and Unrelated Consecutive Trajectories. Statements are usually composed of few words, typically 15-20 words per statement. Then, paragraphs are composed of a set of *related* statements, typically 5-8 statements per paragraph. Meanwhile, a trajectory may include hundreds of points, and subsequent trajectories may be *unrelated*, e.g., a series of taxi trips. BERT, by design, has a limit on the input length of each statement it can process, mainly due to its computationally expensive attention mechanism that exponentially grows with the number of words it needs to take into account. That limit for BERT is usually big enough to cover most language statements. In the case of long statements, BERT truncates them to keep only a window of the last N words, where N is the maximum input length. Apparently, the number of points in a trajectory is mostly above the limit that BERT can accommodate in its attention mechanism. Hence, it may not be practical for BERT to be applied as is for trajectories that are already long enough without splitting these trajectories into short ones, and hence reduce their context information. Along the same lines, BERT takes full advantage of the fact that subsequent statements are related to each other. Hence, using its training set, BERT is able to understand the relations between subsequent statements, and use it for various NLP tasks, including predicting the next statement task (Section 4.3.2). Since subsequent trajectories may not be that related, it would be harder for BERT to perform trajectory tasks that link a trajectory to its next one, e.g., trajectory prediction.

4.5 The Vision: A BERT-like Model for Trajectory Analysis

Our vision is that the spatial community would work together towards a full-fledged BERT-like system for a myriad of trajectory tasks. *We envision that, in a few years, we will have such system, where no one needs to worry again about each specific trajectory analysis operation. Whether it is trajectory imputation, similarly, clustering, or whatever, it would be one system that researchers, developers, and practitioners can*

deploy to get high accuracy for their tasks. The system would always be extensible in a way that can accommodate new operations contributed by the community at large.

This envisioned system does not have to be a completely new one built from scratch. Instead, it can be an adaptation of the current BERT system to make it more amenable to trajectory data analysis. Towards our vision, we believe that the community needs to explore two orthogonal, but complementary, directions. The first direction is to customize both trajectory data and BERT usage to match each other, without the need to change the core of BERT itself. This will produce a quicker system solution that is way more accurate than using BERT as is as in Section 4.3, but still does not exploit the full power of BERT for more accurate results and system extensibility. The second direction is to inject spatial- and temporal-awareness inside the core of BERT itself. This will make BERT deals with spatial data in general and trajectories in particular as first-class citizens and support their special characteristics. As a result, this direction will produce a more accurate system as it exploits the full potential of BERT at its core. However, significant efforts would need to be put in to realize such system. Exploiting and applying both directions together would produce a system with the ultimate desired performance for trajectory applications. Below are our initial thoughts on how to tackle each direction:

Direction I: Customized Trajectories and BERT. This direction aims to customize either the trajectories or the use of BERT or both together, without changing BERT core, in a way that can potentially overcome some of the challenges listed in Section 4.4. Here are three examples of trajectory dataset customizations that belong to this direction: (1) Partition the space into a set of fine-grained hexagons, using Uber's H3 Hexagonal Hierarchical Spatial Index [26]. This way, all points within the same hexagon will be assigned to the same GPS value, which is the hexagon centroid. Then, trajectories become a sequence of hexagons instead of points, and the hexagons become the words to BERT. This customization brings the number of possible words/points in the example Oregon State dataset mentioned in Challenge I in Section 4.4 down from $\sim$1.3M to $\sim$18K, where now each point appears $\sim$100 times on average during training. This is a two-order of magnitude improvement compared to the original dataset where each trajectory point appears only once in the training data. Such customization can potentially overcome both Challenge I and Challenge II, while still preserving accuracy

due to the fine-grained nature of the hexagons. (2) Generate synthetic trajectories to enrich our corpus. To do so, we can employ existing trajectory simulation techniques (e.g., [214, 302]), which basically take our available real trajectory data to generate additional trajectories that resemble the behavior of existing trajectories over different parts of the road network. The will enrich our corpus, which can be then used to train BERT, while avoiding more noisy data. Hence, this can potentially overcome both Challenge I and Challenge II. (3) Split long trajectories into a set of shorter subtrajectories. This will ensure that consecutive trajectories are both short and related, which would overcome Challenge III in Section 4.4. In other words, with this, trajectories would be actually analogous to paragraphs rather than statements. Recall that a *paragraph* is a collection of subsequent short and related *statements*. With splitting, a *trajectory* too becomes a collection of subsequent short and related *subtrajectories*.

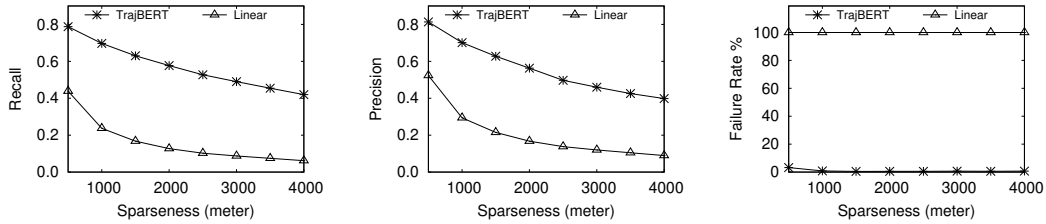
Meanwhile, customizing the use of BERT may be task-specific. So, we give an example for the trajectory imputation task (Section 4.3.1). Since BERT is designed to find only one missing word in a statement, it may not be suitable as is for trajectory imputation, where we would need to impute several points between each two GPS readings. A possible customization to overcome this issue is to call BERT iteratively. For example, in the second trajectory gap in Figure 4.1, we first call BERT to predict one missing point, which is the one shown in the figure corresponding to the word "hosting". Then, if we need an additional imputed point, we can call BERT again for the same gap by including the point/word that we just found ("hosting") in the input as if it was originally there. This is to get an additional imputed point for the same gap after the word "hosting". Once we get a new point, we can use it again in the input to call BERT for a third imputed point. We can iteratively repeat the process until reaching sufficient granularity. This customized usage helps overcome Challenge II, which is related to GPS noise and low sampling rates.

Direction II: Spatially- and Temporally-Aware BERT. This direction aims to inject spatial- and temporal-awareness inside BERT core engine, as a means of a better and more accurate support for trajectory analysis tasks. One of the components that could be a key to injecting spatial- and temporal- awareness into BERT is the *loss function*. This function is responsible for evaluating BERT predictions during training along with penalizing or rewarding the model accordingly. BERT is typically trained

by randomly masking or covering one word in a statement and attempting to guess it. The current loss function that BERT is using is a kind of binary, where only predictions that exactly match the masked word are considered correct and rewarded, while all other predictions are considered incorrect and penalized. There is no notion that some words are closer to the correct answer than others. The model adjusts its parameters towards the correct one and hopefully comes back in the next iteration and guesses it correctly. Applying this training procedure in the case of trajectories using GPS points instead of words means that predicting a GPS point that does not exactly match the masked/covered point is treated as a completely wrong prediction. This also means that predicting a point that is a few feet away from the actual point is as wrong as predicting a point that is miles away because both penalize the model the same way. This is mainly because the loss function (and BERT) has no spatial awareness and hence there is no sense of being almost close to the correct answer. As a result, this makes it hard for the model to learn unless it guesses correctly, which is almost impossible given the trajectory characteristics such as the presence of noisy data and the small ratio of training examples compared to the number of possible GPS points. To overcome this issue, we can adjust the loss function via an additional spatial penalty component that correlates with this distance. In particular, it ensures that the further the prediction from the actual point the higher the penalty. By implementing this approach, the model is encouraged to closely match the actual points and receives cues about its proximity to the correct answer. Such spatial guidance helps the model to overcome the noisiness and lack of large training examples as it would help in converging with good accuracy even with small training data.

4.6 Initial Results

This section provides initial results that show the promise of our vision. In particular, we are doing exploratory evaluation for only the first direction of our vision, i.e., customizing trajectories and BERT (Section 4.5), and for one particular trajectory analysis task, namely, *trajectory imputation* (Section 4.3.1). For the training and evaluation dataset, we use a real trajectory dataset from the city of Porto, Portugal [211], composed of 1.7M trajectories for real taxi trips covering ∼83M GPS points, driven for a total length



(a) Recall (b) Precision (c) Failure Rate

Figure 4.6: Performance Results

of ∼8.8M km spanning an area of ∼500 km^2. We use 80% of these trajectories for training and 20% for testing. Then, we sparsify the testing trajectories by keeping the first point in the trajectory, then, removing all points within distance $Sparse_{gap}$, then, keep the next point, and so on. Hence, all testing trajectories have gap distance $Sparse_{gap}$ between each two consecutive GPS points. We compare our initial technique for trajectory imputation, termed TrajBERT, against a base line approach, termed linear interpolation, that will just impute each trajectory segment by a straight line between its two end points.

Figure 4.6 gives our initial experiments showing the impact of varying $Sparse_{gap}$ from 500 to 4,000 meters on both TrajBERT and linear interpolation with respect to three performance measures, *recall*, *precision*, and *failure rate*. To measure the *recall* and *precision*, for each imputed segment, we discretize the ground truth path into G artificial points, by placing one point each 100m. Similarly, we discretize the imputed path into I artificial points, by placing one point each 100m. Hence, the *recall* is computed as the ratio of points in G that are correctly recovered/recalled, within an accuracy threshold $\delta = 25m$ from the imputed path. The higher the recall the more accurate is the algorithm. The *precision* is computed as the ratio of points in I that are within an accuracy threshold $\delta = 25m$ from the ground truth. The higher the precision the more accurate is the algorithm. For the *failure rate*, when TrajBERT fails to finish the process of imputing a trajectory segment, we resort to linear interpolation between the two segments end points, and count this as one failure for TrajBERT. Hence, the failure rate is the ratio of trajectory segments that TrajBERT could not properly impute. With this, the baseline linear interpolation method has a 100% failure rate as it always resorts for linear line between each two segment end points.

Figure 4.6(a) gives the *recall* ratio of both TrajBERT and linear interpolation. TrajBERT consistently outperforms the linear interpolation baseline with 2x to 7x better recall. For example, at a sparsity gap of 1000m, the TrajBERT recall is 0.7 which is three time better than the recall for linear interpolation. The performance increase is getting higher with large sparse gaps, where for a 4,000m gap, TrajBERT recall is 0.43 while linear interpolation recall is only 0.06. This shows that TrajBERT is more resilient to larger gaps. Figure 4.6(b) gives the *precision* ratio of both TrajBERT and linear interpolation. The results for the *precision* exhibits similar performance to that of the *recall*, where TrajBERT consistently outperforms linear interpolation in both short and large sparsity gaps. Figure 4.6(c) gives the failure rate of TrajBERT compared to the 100% linear interpolation basesline. TrajBERT failure rate is almost 0% regardless of the sparseness, meaning that it is always capable of finding imputation points between the two segment end points. This is mainly because our training dataset is kind of more dense than typical trajectory data. TrajBERT failure rate would definitely go up if the dataset is more sparse. However, the experiment shows that if given a good trajectory dataset, TrajBERT can have almost 0% failure rate.

Though the experiments here are initial and not conclusive as it is made for only one trajectory operation, one data set, and not trying to vary many parameters, yet, it shows the promise of our vision. The objective here is to show that there is a road to our vision, and even a strawman implementation of some initial ideas would achieve highly promising results.

4.7 Related Work

This section discusses related work to our vision. Since we have already discussed related work to each trajectory operation in Section 4.3, we are limiting the discussion in this section to those studies that aim to deploy ideas from the NLP domain to spatial data. We are classifying such work into the two below categories:

Pre-BERT Era: Embedding and Representation Learning. Many researchers in the spatial domain were inspired by the discovery of Word Embedding in NLP, with its classical and popular Word2Vec approach, proposed in 2013 [175]. The main idea of Word2Vec is to map each vocabulary word to a numerical vector that represents it and

thus enables computations and mathematical operations on words. Hence, several efforts in the spatial community have taken similar approaches for various spatial datasets (e.g., see [78, 151, 169, 256, 285, 284, 298, 312, 311]). Many of these approaches have even named their techniques in an analogous way to Word2Vec, e.g., GPS2Vec [287], Loc2Vec [225, 266], Location2Vec [313], Place2Vec [274], and POI2Vec [76]. The main goal of these studies is to learn and obtain embedding (i.e., vectors or numerical representations) for a wide range of spatial elements such as GPS points [287], points of interest PoI(s) [169, 225, 266, 274, 312], trajectories [78, 151, 311], road network nodes and edges [42], geographical areas and regions such as neighborhoods, cities, or zones [256, 285, 313, 298]. The term embedding refers to the fact that some of the hidden information about the element (e.g., meaning and semantics in the case of a vocabulary word, or location type in the case of a PoI or GPS point) are discovered and learned via deep neural networks and then embedded/encoded in a numerical form in the vector representation. Despite the importance of the embedding process, it is limited to basic element-wise operations such as computing the difference or similarity between two words or locations. Embedding alone is not sufficient for more complex tasks such as the next sentence prediction or next trajectory prediction. This is why all the efforts for spatial data embedding are mainly concerned with a very specific task and can not be extended to other tasks. For example, Place2Vec is only concerned with place type similarity [274], SERM and Loc2Vec [225, 284] are only concerned with next point prediction, while another work [151] is only concerned with activity similarity.

The BERT era: Language Models for Spatial Data. Due to the limitations of the word embedding approach, the NLP community came up with the idea of BERT as a language model that is able to process complete documents and sentences. Then, learning the embedding becomes only one component among many other components of BERT. Research efforts in employing language models for the spatial domain are mainly geared towards utilizing it, as a *pretrained* model, in its lingual form by verbally asking it questions of spatial nature [111, 209, 273]. The main idea is to use language as a medium to make the model perform certain tasks such as correcting an address or forecasting the name of the next PoI from an input sentence. This is the opposite of utilizing the model architecture only and then training it from scratch on spatial data. Hence, this approach is limited by the ability to verbally represent the data. It is also subject to

the model ability to understand and comprehend geospatial semantics from the text. To overcome these limitations, a follow up work proposed to add external supervised training [111, 209]. For example, use existing spatial datasets and spatial knowledge graphs to artificially generate new sentences about places, PoIs, distances between them, their locations, types, and add these new sentences to the training corpus in an effort to increase the language model ability to do spatial reasoning from language. A more recent work has investigated using language models for spatial data of multiple forms [168]. In particular, it looks at how to combine multiple datasets of multiple forms (e.g., satellite images, graphs, text) that contain spatial data and then use this combination to train a geospatial model for various spatial tasks. With respect to trajectory analysis, few research efforts have recently considered using a language model like BERT as is for various operations, including similarity [62, 140], prediction [28, 86, 148], imputation [52, 213], and classification [146].

Our Vision: BERT-like Models for Trajectory Analysis Our vision goes beyond the word embedding and focuses more on BERT and similar language models. Unlike all existing work, our vision is not only concerned with only one specific trajectory operation. Instead, our vision aims to build a BERT-like model that can support a wide variety of trajectory analysis tasks. This goes along with the spirit of BERT itself that was not designed for one specific NLP task. Instead, it is made to support a wide variety of NLP tasks.

4.8 Conclusion

We have presented our vision to utilize state-of-the-art language models in Natural Language Processing (NLP), e.g., BERT, to support trajectory analysis operations. The promise is that doing so will pave the way towards a long-waited-for full-fledged trajectory data management system that natively supports all trajectory analysis operations. We show that trajectories in a space exhibit a very similar behavior to statements in a language, and hence many of the NLP tasks on statements are analogous to spatial analysis tasks on trajectories. However, we also point out several challenges that hinder the applicability of BERT as is to trajectory analysis. This helps in outlining the road to realizing our vision by exploiting two orthogonal, but complementary, directions. The

first direction is to customize both trajectory data and BERT usage to match each other, without the need to change the core of BERT. The second direction is to inject spatial- and temporal-awareness inside the core of BERT itself to achieve better results. Initial exploratory results for one trajectory operation and one dataset confirm the promise and possibility of our vision.

Chapter 5

KAMEL: A Scalable BERT-based System for Trajectory Imputation

5.1 Introduction

Numerous important applications heavily rely on trajectory data generated from GPS devices. Such applications include path finding (routing) [292, 99, 40, 184], traffic monitoring and forecasting [128, 172, 251, 106], location-based services [25, 79, 310]), contact tracing [271, 177, 7], map inference [19, 37, 219, 232], and urban planning [145, 147, 100]. Unfortunately, due to bandwidth, battery, and storage limitations, trajectory data are inherently sparse, i.e., there are frequent spatial and temporal gaps between every two consecutive readings. Such gaps present an inherent uncertainty of the object's whereabouts between each two GPS readings. The higher the sparsity (i.e., the larger such gaps spatially and temporally), the lower the accuracy and quality of all applications that rely on trajectory data.

As a means of boosting the accuracy of trajectory data and hence their applications, several techniques have been proposed to impute trajectory data by trying to predict the trajectory whereabouts within the gaps (e.g., see [24, 135, 157, 305]). The large majority of such techniques rely on matching the trajectories on the underlying road network, where the imputation process becomes mainly about finding the road network shortest path between each two consecutive trajectory points. Unfortunately, all such techniques have the implicit assumption that the underlying road network is available and reliable,

which is not always true. Road networks, like any other type of data, suffer from all sorts of inaccuracy, and may not be even available in many places [186, 170, 243]. In fact, Microsoft has recently announced that it has found more than one million kilometers of roads missing from current maps [174], Amazon has its own map inference techniques [9], Uber and Lyft build their own map on top of OpenStreetMaps [163, 248], while Apple has recently completely rebuilt its map [12]. This is a very practical problem that triggered a multi-billion dollars industry for constructing accurate maps [48, 91] and a whole area of industrial and academic research of map inference, which aims to infer (all or missing parts) of a road network from trajectory data [19, 37, 219, 232]. This calls for new trajectory imputation techniques that do not rely on the underlying road network, and hence can generate accurate trajectories that can be used to infer the road network. This chapter is concerned with imputing trajectories under this setting, which is when the road network is not used as an input in any way. There are only a couple of such approaches [144, 69], but, they are only applicable for small road networks and assume abundance of trajectory data. This makes them not applicable to large-scale road networks.

In this chapter, we present KAMEL; a new framework for scalable trajectory imputation that is the first to combine the following distinct features: (1) does not require the knowledge of the underlying road networks as it makes its imputations solely based on input trajectories, (2) does not require prior highly *dense* trajectory data (i.e., large number of trajectories in a small area), (3) scales up to support large geographical areas beyond junction or small city areas, and (4) supports trajectory imputation in both offline bulk mode and online mode for incoming streams of trajectories.

The main idea of KAMEL is to map the trajectory imputation to the *"finding the missing word"* in Natural Language Processing (NLP), which is usually solved using the widely used BERT model [63]. Given a statement like "Paris is the ... of France", where "..." represents a missing word (due to speech recognition, translation, or typo), BERT finds out that the missing word is "capital". To do so, BERT is first trained by large numbers of statements. KAMEL deals with trajectories as statements, where a trajectory/statement is composed of an ordered set of points/words drawn from a finite pool of possible points/words. Also, points/words in a trajectory/statement are spatially/semantically related and are constrained by rules imposed by the underlying

road network/language grammar. Hence, at its core, KAMEL is equipped with a BERT model trained by a set of trajectories, and then used to impute sparse trajectories.

However, using BERT as is within KAMEL is not straightforward and results in a very poor accuracy and performance. This is mainly due to three main challenges: (1) *Spatial awareness*. BERT is not spatially aware, where it may poorly train its model by including datasets that are not spatially related and/or produce results that are not spatially feasible. (2) *Training Data factor*. This is the average number of times each word appears in the training dataset. The original BERT [63] is trained on ∼3.3B word corpus composed of ∼30K distinct words, so each word appears ∼100K times on average in the training set. Meanwhile, trajectory data is no where close to these numbers. A typical trajectory dataset (e.g., Portland dataset [249]) would have ∼2M GPS points with ∼1.5M distinct points, so each point appears only once on average in the training dataset. Such very low training data factor significantly degrades the quality of BERT to be almost useless. (3) *Multiple missing points*. BERT usually aims to find one missing word in a statement, while trajectory imputation may need to find several missing points between two known points. Applying BERT repetitively to do so would significantly degrade the imputation quality and performance.

KAMEL overcomes all the challenges of using BERT through an architecture composed of five main modules, *Tokenization*, *Partitioning*, *Spatial Constraints*, *Multipoint Imputation*, and *Detokenization*. The *Partitioning* and *Spatial Constraints* modules address the *spatial awareness* challenge by injecting spatial awareness into both BERT training process and output result, respectively. The *Tokenization* and *Detokenization* modules address the *training data factor* challenge by clustering points into tokens to increase their appearance in the training dataset. The *Multipoint Imputation* module addresses the *multiple missing points* challenge by estimating the probability of multiple points together to form an imputed segment.

Our goal is to show that NLP models (e.g., BERT), trained with trajectory data, have the potential to push the state-of-the-art in trajectory imputation. We do not aim to find the best NLP model suited for trajectories. We have chosen BERT as it is one of the most commonly used models. Yet, other BERT variants or language models can be also used with different adaptations. We opted for a design that employs BERT *as is* as an architecture, trained on trajectories, rather than language, which makes

KAMEL attractive for industrial and open-source market that are already familiar with it and would need less disturbance to their systems. The experimental evaluation based on system deployment of KAMEL and real datasets shows that KAMEL achieves a high recall score of 89% of the missing points from the ground truth trajectories, with high precision. Comparison with other approaches shows that KAMEL achieves nearly three times the score of the baselines and the state-of-the-art methods, and can work on various straight and curved trajectories, with gaps as large as 2.5 km.

The rest of this chapter is organized as follows: Section 5.2 presents KAMEL architecture. The five modules of KAMEL are described in Section 5.3 to 5.7. Experimental results are presented in Section 5.8. Section 5.9 discusses related work. Section 5.10 concludes the chapter.

5.2 KAMEL Architecture

Figure 5.1 depicts the architecture of KAMEL, where it takes two types of input: (1) *training data*; new trajectory datasets which does not produce any output, instead KAMEL uses it to enrich its models. (2) *sparse trajectories*; the trajectories that the users wants to impute. KAMEL receives such data either in bulk offline mode or as a stream of incoming trajectories. In both cases, KAMEL outputs the set of imputed dense trajectories that correspond to the sparse input. KAMEL is a BERT-based system, where BERT is used as a core component and is depicted as a black box at the bottom of Figure 5.1. However, as mentioned earlier, using BERT as is results in a poor accuracy and performance, due to three main challenges, *spatial awareness*, *training data factor*, and *multiple missing points*. Hence, internally, KAMEL is composed of five main modules to address these challenges, namely: *Tokenization*, *Partitioning*, *Spatial Constraints*, *Multipoint Imputation*, and *Detokenization*, described briefly below:

Tokenization. This module is a gateway to KAMEL, where all input go through it first. It addresses the *training data factor* challenge by converting each input point to a token that covers a specific spatial area. This reduces the number of distinct items and increases their appearance in the training set. The output of this module is a set of tokens sent to the *Partitioning* module. Details are in Section 5.3.

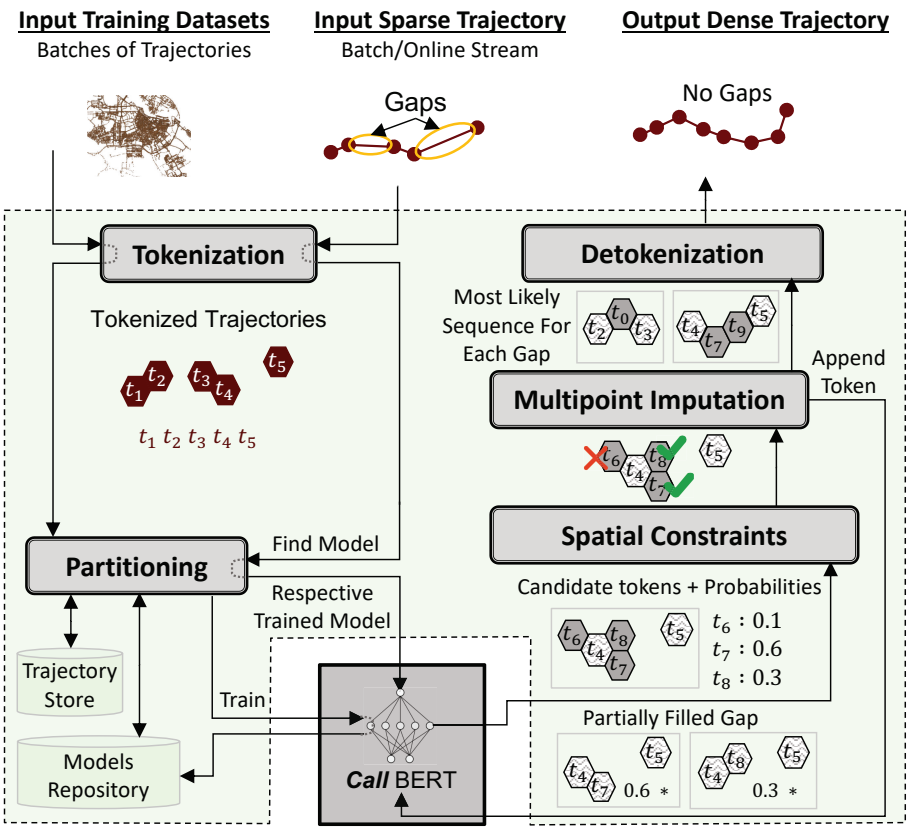


Figure 5.1: KAMEL Architecture

Partitioning. This module addresses the *spatial awareness* challenge as it builds various BERT models based on the spatial coverage of the input data. It takes its input as a set of tokens produced from the *Tokenization* module. In case these tokens represent training data, the *Partitioning* module stores such data in its raw trajectory store. Then, it decides whether to enrich or expand one of its existing models, or even build a new model, and calls BERT accordingly. The new or updated models will be stored in a dedicated model repository. In case the input tokens represent sparse trajectories that need to be imputed, the *Partitioning* module finds out which BERT model needs to be used to impute each trajectory. For each trajectory gap represented by two end tokens, it calls the selected BERT model to impute one token between the two end tokens. The output of BERT is a set of candidate tokens (with probabilities) sent to the *Spatial Constraint* module. Details are in Section 5.4.

Spatial Constraints. This module addresses the *spatial awareness* in BERT output. Its input is the set of candidate tokens produced from BERT. Then, it drops off some of these tokens that do not satisfy a set of spatial constraints. Remaining tokens are passed to the *Multipoint Imputation* module. Details are in Section 5.5.

Multipoint Imputation. This module addresses the *multiple missing points* challenge. It converts its input from a candidate imputed token to a *sequence* of tokens through calling BERT iteratively. It outputs the most likely sequence of tokens for each trajectory gap and sends it to the *Detokenization* module. Details are in Section 5.6.

Detokenization. This module addresses the *training data factor* challenge, and to some extent it reverses the *Tokenization* module. The input to this module is the imputed trajectory, represented as tokens. The output is the imputed trajectory represented as points, which is the final output of KAMEL. Details are in Section 5.7.

5.3 Tokenization

This module is the gateway for KAMEL where all input data (training data or sparse trajectories) have to go through before any other KAMEL module. A typical trajectory input is composed of points, each represented by its latitude and longitude coordinates. With the high precision of such coordinates, a certain point does not often appear more than once or twice in a training dataset, which is not enough to train a BERT model. Contrast this to the language that BERT is used to work with, where each word may appear 100+K times during the training. This module aims to address this challenge by partitioning the whole space into a set of non-overlapping cells, where all points located in the same cell are given the same token, which is the cell ID. This means that a trajectory dataset will be presented as a set of tokens (cell IDs) instead of a set of points. So, different points will have the same token, and hence they can appear more often in the training dataset, which is also composed of tokens. To increase accuracy and efficiency, the *tokenization* module has to decide on how to partition the space into cells (Section 5.3.1) and the optimal cell size for each dataset (Section 5.3.2).

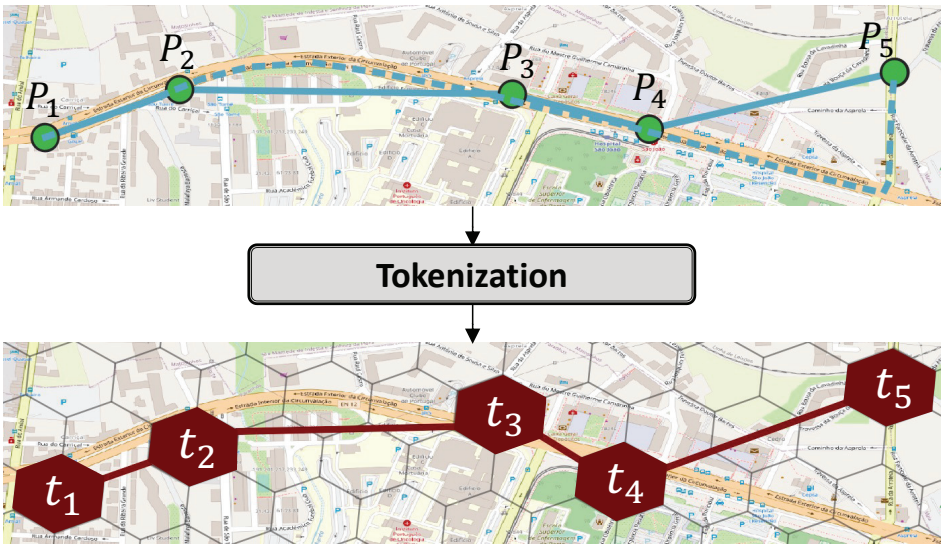


Figure 5.2: Example Output of the Tokenization Module

5.3.1 Hexagonal Space Partitioning

For its tokenization scheme, KAMEL uses a flat *hexagonal* grid structure based on Uber's H3 Hexagonal Hierarchical Spatial Index [26, 96]. In this index, the whole world geographical area is divided into a set of non-overlapping hexagons, where each hexagon has a unique ID h_i. Figure 5.2 shows an example of the tokenization process using a hexagonal grid. The upper part of the figure shows an input trajectory that consists of five points P_1 to P_5, and the bottom part shows the output tokenized trajectory which consists of five tokens t_1 to t_5, where each token corresponds to one of the input points. It is important to note that the road network in the background is shown only for illustration, but the imputation process does not rely on (or even know about) it. Other tokenization alternatives such as Google S2 squares [220] partitioning, which is basically a form of a grid structure is also a possible alternative. However, hexagons help achieve more accurate imputations as confirmed by our experiments in Section 5.8.6.

The rationale of using hexagons over common traditional square or rectangular partitioning is that: With a hexagonal grid, all the six neighbors of each cell C have the same exact properties in terms of distance to C centroid and the length of the shared border with C. This is in contrast to rectangular partitioning, where each cell would have four corner neighbors sharing a point, two neighbors sharing the length, and two

(a) $\mathcal{H}$= 25m (b) $\mathcal{H}$= 75m (c) $\mathcal{H}$= 200m (d) Accuracy vs Cell Size

Figure 5.3: Selection Between Different Cell Sizes

other neighbors sharing the width. Ensuring that all neighbors have the same properties makes it more suitable for BERT as going from one token (hexagon) to its neighbor would not be influenced by our partitioning. Meanwhile, we acknowledge that unlike rectangles, hexagons are not hierarchical, where we cannot fit a set of neighbor hexagonal cells into one bigger hexagonal cell. Yet, this is not a concern to KAMEL, as there is no need for such hierarchy. Hexagons are needed only to tokenize points to cells, and later to detokenize cells to points (Section 5.7). The cost of mapping a point to its hexagonal cell has a constant time, as it is done through a series of coordinate system conversions [97].

5.3.2 Cell Size Optimization

Deciding on the right cell size would significantly impact KAMEL accuracy. Figures 5.3(a)-(c) depict three different sizes, with edge length $\mathcal{H} = 25$, 75, and 200 meters, respectively, laid on top of the same part of a road network. The cell size is inversely proportional to the number of distinct cells (tokens); the larger the size the less the number of distinct tokens, and hence token will appear more in the training set. Hence, large sizes would address the *training data factor* challenge and push for better imputation accuracy. Meanwhile, with large sizes, each cell becomes not really representative of the points inside it, as too many points would map to the same cell, which would negatively affect the accuracy of both the *tokenization* and *detokenization* modules (Section 5.7). In addition, large sizes make it difficult for BERT to learn distinguishable contexts between trajectories since a large cell would capture trajectories from many roads and directions. This may call for having small sizes, which would highly increase the number of distinct

tokens in the dataset. This makes the decision on the right cell size an optimization problem, as illustrated in Figure 5.3(d), where both ends of the spectrum lead to low accuracy. KAMEL is shipped with a default cell size obtained from our exhaustive experiments in Section 5.8. However, KAMEL acknowledges that the optimal cell size might be different for each dataset, as it depends on both the trajectories and area characteristics. Hence, KAMEL is equipped with an auto tuning module that sets the system cell size for a given *training* dataset. When the input to the *tokenization* module is a training dataset, we sample the input data and try training BERT models for various cell sizes, and then pick the size that achieves the highest accuracy.

5.4 Partitioning

The original BERT is trained separately for each language. For example, the BERT model used for English is pretty different from the BERT model used for Korean, as each is trained by different datasets. While the boundaries between languages are clear, it is not the case for spatial areas. Trajectory datasets may come from nearby, overlapping, or far spatial areas. The *Partitioning* module in KAMEL is responsible for having spatial-awareness in BERT models by maintaining various models for various spatial areas even though their boundaries are not well defined. One can look at each BERT model in KAMEL as a model for a different language. Hence, we maintain two data stores: A simple trajectory store [55, 227] that maintains the set of tokenized trajectories that KAMEL received as input training dataset and a model repository that maintains all BERT models that KAMEL has built for various spatial areas. Building and/or updating such models is completely done offline, where it may take hours depending on the number of models and trajectories. However, this does not affect the scalability of the imputation process itself, which is done online and only usees the precomputed models in its process. This way, KAMEL scales to support large geographical areas. The tokenized input trajectories received by the *Partitioning* module either represent a set of sparse trajectories that need to be imputed or training trajectories. In the former case, the *Partitioning* module basically consults its model repository (Section 5.4.1) to retrieve the BERT model that is best suited for the imputation process. For training trajectories, the *Partitioning* module uses it to update its model repository (Section 5.4.2).

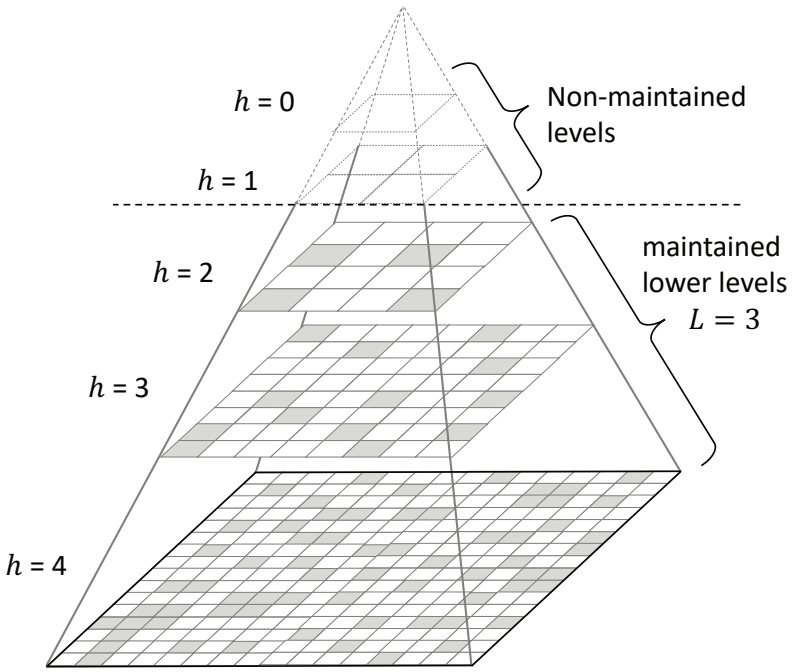


Figure 5.4: Pyramid Data Structure for Models Repository

5.4.1 Models Repository Structure and Retrieval

KAMEL maintains its model repository in a disk-based hierarchical pyramid data structure of H levels [13], where each level h is decomposed to 4^h equal cells. The pyramid root is of height zero and has only one cell that covers the whole space. The pyramid is built bottom up, and not all levels have to be maintained. In fact, we only maintain the lowest L levels of the pyramid, because there is usually not enough trajectory data to build models for higher levels. The number of levels H and L are parameters that balance between the high resolution of the model and the maintenance overhead of the pyramid structure. Figure 5.4 gives an example of such structure with $H = 5$ and $L = 3$. Shaded cells are the ones that have models, while blank ones have nothing to maintain.

We maintain two kinds of models: (1) Single-cell models, which are built based on the contents of a single cell, and (2) Neighbor-cells models, which are built based on the contents of two neighboring cells sharing an edge, and stored in either the north or west cell of the two neighbors. The goal of neighbor-cell models is to help in boundary cases, where we need a model to cross the contents of two neighboring cells, especially, if they

do not share a parent, or if their parent cell does not have a model. To ensure the model accuracy, we build a model at cell C at level l only if there are at least $k \times 4^{(H-l)}$ tokens in that cell, where k is a system parameter (default 20K) and H is pyramid height. This means that we need at least k tokens to build a model at a leaf node and $4k$ tokens to build a model at a cell just above the leaf level. For the neighbor cell models, we double that threshold. Each cell C contains one or more of the following: (1) The number of tokens in the trajectory store that lie within C, (2) A single-cell BERT model for the contents of C, along with its metadata, which include model statistics and last update date, (2) Up to two neighbor-cell BERT models (with their metadata) for the contents of C and its east and/or south neighbor cells, and (3) Up to two pointers to neighbor-cell BERT models stored at the north and/or west neighbor cells.

When the input to the *Partitioning* module is a sparse trajectory to be imputed, we find the smallest cell C or pair of neighbor cells C_i and C_j that fully enclose the trajectory minimum bounding rectangle, and pick BERT model accordingly for the imputation process. Calling the model does not scan or read any trajectory data after it has been trained offline, which makes KAMEL highly scalable. In case there is no single or pair of cells with models to cover the trajectory, we split it into sub-trajectories that can be enclosed within a model. For those sub-trajectories that cannot fit in any of our models, we just impute them by a simple straight line.

5.4.2 Models Repository Maintenance

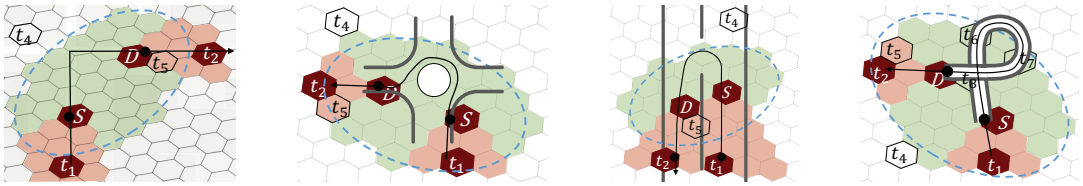
Whenever a new training trajectory dataset $\mathcal{T}$ is received, we first add it to our trajectory store. Then, we find the pyramid cell C as the smallest cell that fully encloses the minimum bounding rectangle of all trajectories in $\mathcal{T}$. Then, we enrich $\mathcal{T}$ by adding to it the set of trajectories in our trajectory store that are fully enclosed in C. With C, we do the following four steps: (1) If the number of tokens in $\mathcal{T}$ is above the model threshold, we build a BERT model and store it at C, which will either update an existing model or create a new one. (2) For each cell C_i among the four neighbors of C, if the total number of tokens in C and C_i is above our model threshold, we retrieve all trajectories in C_i and use it with $\mathcal{T}$ to build a neighbor-cell model. The model is stored in the north or west cell of C and C_i, with a pointer to it from the other cell. (3) If the total number of tokens in C and its three siblings is above the threshold to build a model at their

parent, we do so. This would be done recursively for C's ancestors till we reach the lowest maintained level. (4) If C is a non-leaf node, we split the trajectories in $\mathcal{T}$ among the four children cells of C, and check if this would warrant building a new model at any of these children cells. We do so recursively till there are not enough tokens to build a model. Note that this does not need to happen for every single trajectory. Instead, it is scheduled as a background process when needed for a batch of new trajectories, without causing any downtime to the system.

5.5 Spatial Constraints

The original BERT does not have any restrictions in its output as any word in the vocabulary can be the answer as long as it is the most likely one. However, in case of trajectory imputation, we would need to impose some constraints, due to two main reasons: (1) Imputed points have to respect physical movement constraints. For example, an imputed point p between two segment end points S and D should be within a spatial range that respects the locations and timestamps of both S and D. Unfortunately, BERT as is, may end up in predicting p outside that range as BERT may have seen that point following S for another segment that does not end at D. Moreover, unlike the original BERT that imputes only one point, and hence only picks the top possibility, we impute multiple points, and hence we pick the top k possible candidates (details in Section 5.6). With large value of k, it is more likely to have candidate imputed points that do not respect the physical movement constraints. (2) Cycles. As we aim to impute multiple points for each segment, BERT may end up producing points that go into cycles and hence do not converge to the segment end point.

The *Spatial Constraints* module takes the output from BERT as its input and filters out those tokens that (a) do not respect the physical movement constraints (Section 5.5.1), or (b) result in cycles in the imputed segment (Section 5.5.2). The output of the *Spatial Constraints* module is passed to the *Multipoint Imputation* module.



(a) Right Turn (b) Roundabout Turn (c) U-Turn (d) Overpass Exit

Figure 5.5: Spatial Constraints Using Different Road Examples

5.5.1 Physical Movement Constraints

The physical movement constraints mainly specify an area where any imputed token t between the two segment end tokens S and D must be located in. Figure 5.5 gives examples for applying such constraints for four road cases, namely, right turn, roundabout, U-turn, and overpass exit. In all cases, we need to impute a trajectory segment between tokens S and D, where t_1 and t_2 are the tokens that just come before S and after D, respectively. There are two types of physical movement constraints: *speed* and *direction* constraints. In all four road cases in the figure, the blue dashed ellipse presents an area computed per the *speed* constraints, where physically, a token cannot take place outside such area. The set of red hexagons (tokens) are picked per the *direction* constraints, where none of them can be an imputed token between S and D. This makes the set of green tokens are the only acceptable ones for any imputed token between S and D, where they present the set difference between the tokens within the blue ellipse and the red tokens. Below are the details of how to compute the area of each constraint.

Speed Constraints Area. The rationale behind such area is that travel distance correlates with duration, and a vehicle has a physical limit on where it can travel to, given a time span. This area is depicted as a blue dashed ellipse in the four cases of Figure 5.5, where the centers of tokens S and D are the ellipse foci points. The maximum total sum of distances from foci points to any point p in the ellipse is $speed_{max}$ $\times$ *TimeDiff* (S,D), where *TimeDiff* is the timestamp difference between S and D. While the token timestamps are already given as part of the input data, there are several ways to compute the maximum speed. One way is to use a fixed speed limit based on the city we are trying to impute trajectories in. Another way is to consider the speed of the preceding imputed segment multiplied by a conservative factor. KAMEL currently uses

a fixed speed inferred from its training trajectory data. In the four cases of Figure 5.5, token t_4 would be rejected as it is outside the dashed ellipse, and hence it is physically impossible to reach there.

Direction Constraints Area. The rationale behind such area is that an imputed token should respect the direction from S to D, and not jump ahead of D towards t_2 or before S towards t_1. This area is depicted as a set of red hexagonal tokens in Figure 5.5, where it is computed as all tokens deviated within a certain angle (default 45°) from S towards its previous token t_1 and from D towards its next token t_2. In the four cases of Figure 5.5, token t_5 is rejected as it is within 45°angle from the direction of D to t_2.

5.5.2 Cycles Prevention

It is important to note that when imputing a gap between two tokens S and D, it is most likely that we will need to insert multiple tokens in between. Hence, during the imputation of one segment, BERT will be called many times, which may result in having cycles. A cycle is formed if the sequence of the last x tokens are repeated. A trivial cycle would take place when x=1, which means that BERT has returned the same token as the last imputed token. In this case, the *Spatial Constraints* module would just reject this outcome. For a non-trivial case of $x > 1$, the *Spatial Constraints* module always checks the last x imputed tokens, and rejects all of them together if a cycle is detected. KAMEL uses a default value of x=6, which is very reasonable to detect almost all possible cycles.

Figure 5.5(d) gives an interesting case for an overpass route, where four imputed tokens t_8, t_6, t_7, t_8 are added between S and D. Although t_8 appears twice, but this is not considered a cycle as there is no repeated sequence of any length x. This shows the ability of KAMEL to impute such non-trivial trajectory.

5.6 Multipoint Imputation

For each trajectory segment, the input to this module is a set of candidate tokens returned from BERT and filtered out from the *Spatial Constraints* module. Each token is associated with a probability indicating how likely it is to be the imputed token. Since BERT is designed to predict only *one* missing word in a statement, it just picks the

Algorithm 1 Iterative BERT Calling

Procedure IterativeBERT(SourceToken S, DestinationToken D)

1: $Segment \leftarrow \{S, D\}$; $GapPointer \leftarrow S$
2: **while** $GapPointer$ **is not** null **do**
3: $CandTokens \leftarrow$ BERT($Segment$, $GapPointer$)
4: $CandTokens \leftarrow$ SConstraints($GapPointer$, $CandTokens$)
5: $Segment \leftarrow$ Insert($Segment$, **Top**($CandTokens$), $GapPointer$)
6: $GapPointer \leftarrow$ FindFirstGap($Segment$)
7: **end while**
8: **return** $Segment$

token with the highest probability. However, in case of trajectories, one token is not enough to impute a segment. Hence, the goal of the *Multipoint Imputation* module is to adapt BERT to support imputing multiple tokens between each two consecutive trajectory tokens, such that the distance between any two consecutive tokens is less than a certain maximum gap distance, max_{gap}. KAMEL employs two approaches: (1) iterative BERT calling (Section 5.6.1), which resembles calling BERT, as is, iteratively and (2) Bidirectional beam search (Section 5.6.2), which employs a spatially modified version of the classical Beam search algorithm [216] to guide multiple BERT calls. In both approaches, we put a hard limit on the maximum number of times that BERT can be called. If exceeded, we declare failure and resort to linear line imputation between the segment end points. For each segment, the output of the *Multipoint Imputation* module is a complete imputed trajectory segment as a sequence of hexagonal tokens.

5.6.1 Iterative BERT Calling

This approach calls BERT iteratively to fill in multiple imputed tokens between the segment end tokens S and D, such that the distance between any two consecutive tokens is less than a certain gap distance, max_{gap}. Algorithm 1 gives the pseudo code for this approach, where it starts by initializing the output segment to its two end tokens, and a *GapPointer* variable to the starting token, indicating where to insert our next imputed token. Then, it calls BERT, which returns a set of candidate tokens to pick one of them as the next imputed one. The candidate tokens are passed by the *Spatial Constraints* module (*SConstraints*) to filter out tokens that do not satisfy the *speed* and *direction*

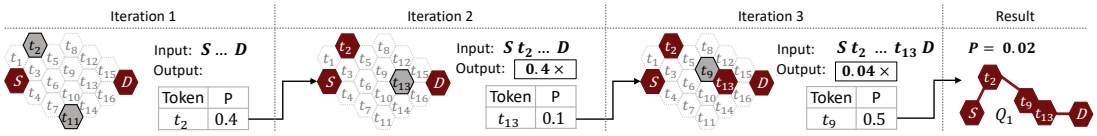


Figure 5.6: Iterative BERT Calling

constraints as described in Section 5.5. Then, we insert the candidate token with the highest probability just after S. Finally, we call the function *FindFirstGap* to find the first gap in the segment with a length more than max_{gap}. If such gap exists, we repeat the cycle by calling BERT to find the next imputed token and so on. The algorithm concludes when there is no such gap, and returns the complete imputed segment.

Example. Figure 5.6 gives an example of the iterative BERT calling approach to impute the trajectory segment between tokens S and D with maximum allowed gap distance max_{gap}=2. In the first iteration, token t_2 is returned from BERT as the candidate token with highest probability (0.4) and passed all spatial constraints. Hence, it is inserted as the first imputed point between S and D. Then, we try to find if there is still any gap in the segment. Apparently, S and t_2 are within two tokens from each other, which means that there is no gap in between to fill. However, t_2 and D are still of distance more than two tokens from each other. Hence, the *FindFirstGap* function will return t_2 as the token that we still need to add (at least) one more point after it. This means that we need to go through a second iteration, which returns token t_{13} as the top candidate valid token to be inserted between t_2 and D. Then, calling the *FindFirstGap* function will again return t_2 as the distance between t_2 and t_{13} is still within two tokens. Then, in a third iteration, t_9 will be added between t_2 and t_{13}. Then, *FindFirstGap* will return *null* as all tokens are within a distance less than two tokens from each other. Hence, the algorithm returns the imputed segment as S, t_2, t_9, t_{13}, D.

5.6.2 Bidirectional Beam Search

The iterative calling approach is kind of a greedy approach as it only considers the topmost probable token in each iteration. That may not be the right decision as it may lead to less probable later options. Hence, we introduce the *bidirectional beam search* approach that aims to pick the most probable *sequence* of tokens between the source and destination tokens.

Main Idea. Our main idea is to adapt the classical Beam Search [216], designed for searching graphs, to work for trajectory imputation. Instead of searching all possible paths in a graph, beam search limits its search to only the top B promising paths, where a higher B gives more accurate results, but more expensive search. We then employ the following to adopt the beam search idea for trajectory imputation: (1) Instead of searching in only one direction, we make the beam search bidirectional, as trajectory imputation could go in several directions to impute a given segment. So, we keep track with the most probable B sequence of tokens of all directions that can contribute to the final imputed segment. (2) As the probability of a sequence of tokens is computed by multiplying each token probability, longer segments are penalized by having smaller probabilities. Since we are looking for the most probable path, not the shortest one, we counterweight this penalty via length normalization [269]. In particular, a segment S with a probability P would have its normalized probability as $P \times |S|^{\alpha}$ where $0<= \alpha <=1$ is the normalization strength, which we set to 1 as a default, and $|S|$ is the number of tokens in segment S.

Algorithm. Algorithm 2 gives the pseudo code for our bidirectional beam search approach. It starts by initializing the list *AllGaps* by one gap for the source and destination tokens and a pointer to the source token, indicating where we need to insert our next token. Unlike the case for iterative calling, the gap segment is associated with a probability, initialized by 1. Then, for each gap in *AllGaps*, we call BERT followed by the *Spatial Constraints* module to get the list of candidate tokens, along with their probabilities. Unlike the case of iterative calling where we only care about the top token, here we care about the top B probable tokens (Line 8 in Algorithm 2). Hence, for each such token T, we construct a new segment by inserting T in the gap, along with updating the segment probability. Among all such new segments constructed for all gaps, we pick only the top B of them according to their probability, bounded by an upper limit (initially set to ∞) (Line 13 in Algorithm 2). For each of the remaining B segments, we aim to find all the gaps that are still there to address in the next iteration. If any of such segments has no gaps, we add it to our answer set with its *normalized* probability score, and use that score to adjust our upper limit of considering any further segments. Once there are no gaps in any of the segments, we conclude by returning the segment with the highest probability.

Algorithm 2 Bidirectional Beam Search

Procedure BeamBERT(Token S, Token D, BeamSize B)

```
1:  Gap.Segment ← {(S, D), 1}; Gap.Pointer ← S
2:  AllGaps ← {Gap}; ProbLimit ← ∞; AnswerSet ← φ;
3:  while AllGaps is not null do
4:      NewSegments ← φ
5:      for each Gap in AllGaps do
6:          CandTokens ← BERT(Gap.Segment, Gap.Pointer)
7:          CandTokens ← SConstraints(Gap.Pointer,CandTokens)
8:          for each token T in Top(CandTokens,B) do
9:              Segment ← Insert(Gap.Segment,T,Gap.Pointer,Prob(T))
10:             NewSegments ← NewSegments ∪ Segment
11:         end for
12:     end for
13:     NewSegments ← TopB(NewSegments,B,ProbLimit)
14:     AllGaps ← φ
15:     for each Segment in NewSegments do
16:         Gaps ← FindGaps(Segment)
17:         if Gaps is empty then
18:             AnswerSet ← AnswerSet ∪ Normalized(Segment)
19:             ProbLimit ← Min(ProbLimit, Segment.Probability)
20:         else
21:             AllGaps ← AllGaps ∪ Gaps
22:         end if
23:     end for
24: end while
25: return Top(AnswerSet)
```

Example. Figure 5.7 gives an example of the bidirectional beam search algorithm to impute the gap between S and D with beam size $B = 3$. In the *first iteration*, we call BERT to get the top-B probable tokens, which returns t_2, t_{11}, and t_{13}, with probabilities 0.4, 0.3, and 0.2, respectively. Having t_2 between S and D leaves one gap between t_2 and D as the distance between them is more than our max_{gap} threshold. Meanwhile, having t_{11} leaves two gaps between S and t_{11} and between t_{11} and D. Then, having t_{13} leaves only one gap between S and t_{13}. So, in total, we have four gaps to consider further. Hence, in the *second iteration*, we make four BERT calls, one for each gap. The first returns three tokens t_8, t_{13}, and t_{15} with a probability of 0.1 each. The

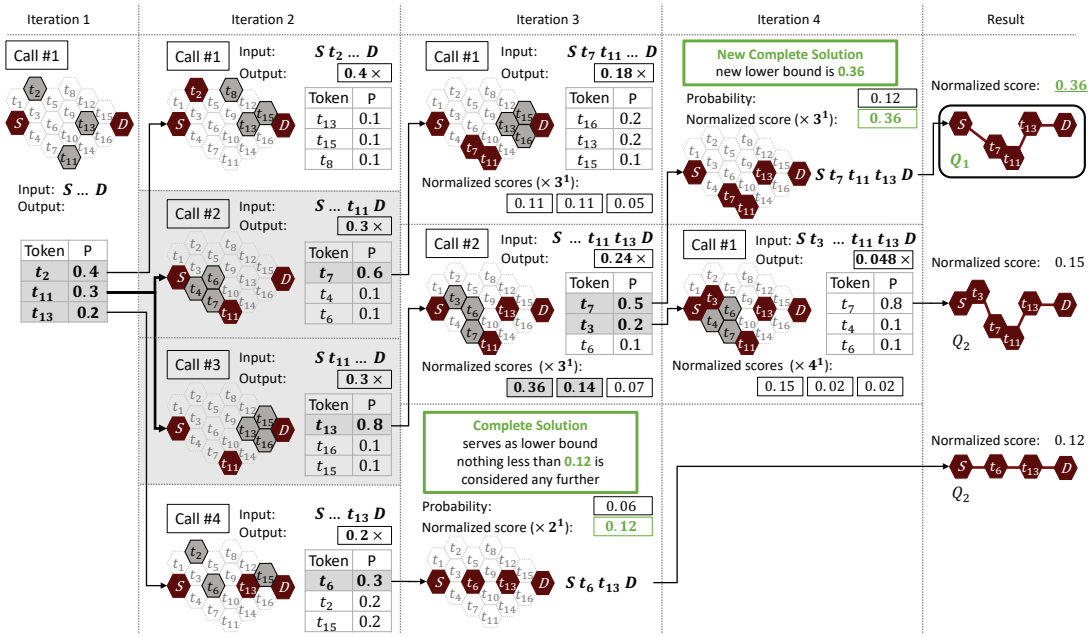


Figure 5.7: Bidirectional Beam Search

segment probability is 0.04 as they will be multiplied by 0.4, which is the probability of the previous iteration. The second gap (between S and t_{11}) returns t_4, t_6, and t_7 with probabilities 0.1, 0.1, and 0.6, respectively. Multiplied by 0.3, the segment probability for these tokens are 0.03, 0.03, and 0.18, respectively. Similarly, The third call returns tokens t_{13} t_{15}, t_{16} with segment probabilities 0.24, 0.03, 0.03, and the fourth call returns tokens t_2 t_6, t_{15} with segment probabilities 0.04, 0.06, 0.04. Out of the 12 new segments, we only pick the top 3 probable ones, namely, segments $\{S, t_7, t_{11}, D\}$, $\{S, t_{11}, t_{13}, D\}$, and $\{S, t_6, t_{13}, D\}$.

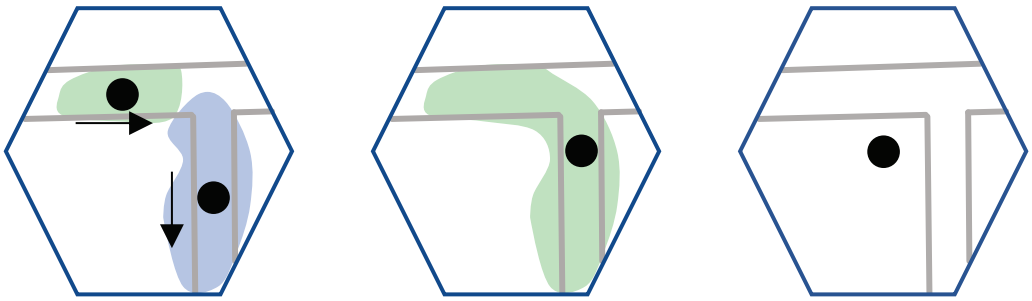
In the *third iteration* to find if there are still gaps in the three segments we still have. The first segment has a gap between t_{11} and D. Similarly, the second segment has a gap between S and t_{11}. Meanwhile, there are no gaps in the third segment, and hence it is concluded as one possible complete imputed segment, with an updated normalized probability $0.06 \times 2^1 = 0.12$. We then use this probability as a lower limit of any segment to be considered further. Hence, we call BERT for only two gaps. The first call suggests t_{13}, t_{15}, and t_{16} with probabilities 0.2, 0.1, and 0.2. To get the segment probabilities, we multiply each by 0.18 as the probability given from the previous iteration. To get the

normalized probability, we multiply each segment probability by three as each segment would have three imputed tokens. This makes the normalized segment probabilities for these tokens are 0.11, 0.05, and 0.11. Similarly, the second call suggests tokens t_7, t_3, and t_6 with normalized probabilities 0.36, 0.14, 0.07. Among these six segments, we pick only the top three ones that are above our lower limit of 0.12. There are only two such segments, namely, $\{S, t_7, t_{11}, t_{11}, D\}$ with probability 0.36 and $\{S, t_7, t_{11}, t_{13}, D\}$ with probability 0.14. In the *fourth iteration*, we find out that the first segment has no more gaps, hence we add it to the answer set with its probability 0.36. We call BERT for the second segment to get a concluded one with a normalized score of 0.15. Finally, among the three concluded segments, we return the one with the highest probability of 0.36. Contrast this to what we get from the *Iterative Calling* approach (Figure 5.6), where we end up with a segment with a normalized probability of 0.06, which shows how powerful is our bidirectional beam search approach.

5.7 Detokenization

The output of the *Multipoint Imputation* module is an imputed trajectory presented as a set of hexagonal tokens. Then, the objective of the *Detokenization* module is to take this output and convert each of its tokens to be a point. The output of the *Detokenization* module is a complete imputed trajectory presented as a sequence of GPS points. The functionality of the *Detokenization* module is composed of the below *offline* and *online* operations:

Offline Operations. When training data is uploaded to KAMEL and passed through its *Tokenization* module, KAMEL triggers the classical DBSCAN clustering algorithm [70] to spatially cluster the contents of each token, based on each point's direction. Since we do not have any information on the underlying road network, the clustering would help in understanding where the points in each token are usually located within the token. Figure 5.8 gives three outcome cases of running a DBSCAN clustering algorithm on a hexagonal token where the road network within the hexagon has a right turn. It is important to note that the road network here is just drawn for illustration, but it is not actually available to KAMEL. In the first case (Figure 5.8(a)), there were enough data points that made the clustering algorithm identify two separate clusters, one going in a



(a) Cluster Centroid (b) Data Centroid (c) Hexagon Centroid

Figure 5.8: Three Outcomes for Clustering Points in a Token

horizontal direction, and one in a vertical direction. In the second case (Figure 5.8(b)), there was not enough data to get two clusters, so, all data in the hexagon is considered as one cluster. In the third case (Figure 5.8(c)), there were almost no data in the token to run a clustering algorithm. In the first two cases, the centroid of each cluster is presented by a solid dot. In the third case, the solid dot represents the hexagon centroid. The information for all clusters along with their centroids are stored per each token as its metadata, to be used later in the online part.

Online Detokenization. When receiving the output of the *Multipoint Imputation* module, the *Detokenization* module goes through each token, and replaces it with a centroid point using the following procedure: If the token T has multiple clusters (Figure 5.8(a)), we first compute the token direction angle as the average of the incoming angle between T and its preceding token and the outgoing angle between T and its next token. Then, we pick the cluster that has a closer direction angle to T, and return its centroid point. If the token has only one cluster (Figure 5.8(b)), we just return the centroid point of that cluster. Finally, if there are no clusters in the token (Figure 5.8(c)), we just return the hexagon centroid. It is important to note that this latter case is unlikely to take place, as BERT is very unlikely to recommend a hexagon token that has not appeared much in its training data, due to the lack of points there.

5.8 Experimental Evaluation

Baselines. KAMEL is designed as a pre-processing step for map inference applications, a practical case where the road network is not used as input either because it is not available or not trusted. Hence, we compare KAMEL, based on a real system implementation [185] against TrImpute [69] as the state-of-the-art and the only trajectory imputation technique that does not rely on an underlying map and scale up to large networks. For a baseline, we also compare against linear interpolation, where trajectories are imputed by a simple linear line. For complete analysis, we include Map Matching [277] results as an example of techniques that rely on road networks.

Datasets. (1) Porto, Portugal [211]: 1.7M trajectories for real taxi trips, composed of ∼83M GPS points, driven for a total length of ∼8.8M km spanning an area of ∼500 km^2, and (2) Jakarta, Indonesia [112]: 56K trajectories for real ride-sharing trips, composed of ∼56M GPS points, driven for a total length of ∼500K km spanning an area of ∼660 km^2. For each dataset, we use 80% for training and 20% for testing. We sparsify testing trajectories by imposing gaps where we keep the first trajectory point, then, remove all points within distance $Sparse_{distance}$, keep the next point, and so on.

Performance metrics. (1) *Recall*. We discretize ground truth trajectories by placing points P as one point every max_{gap} distance (same threshold used in the imputation process). The *Recall* is the ratio of points in P that the algorithm has correctly recalled within the accuracy threshold δ from the imputed trajectory. (2) *Precision*. We discretize imputed trajectories by placing points Q as one point every max_{gap} distance. *Precision* is the ratio of points in Q that are within accuracy threshold δ from the ground truth. (3) *Failure Rate*. An imputation technique fails to impute a trajectory segment when it just inserts a linear line between the segment end points. *Failure rate* is the ratio of segments imputed by a linear line. (4) *Time Overhead*. Training and imputation time overhead of each algorithm.

Default values and parameter tuning. Unless mentioned otherwise, we set the maximum allowed gap max_{gap} to 100m, the accuracy threshold δ to 50m for Porto and 25m for Jakarta, and the imposed sparsity distance $Sparse_{distance}$ to 1km. For KAMEL parameter tuning, our experiments for both datasets (omitted for space constraints) show that setting hexagon size $\mathcal{H}$ to 75m, beam size $\mathcal{B}$ to 10, and pyramid levels L and

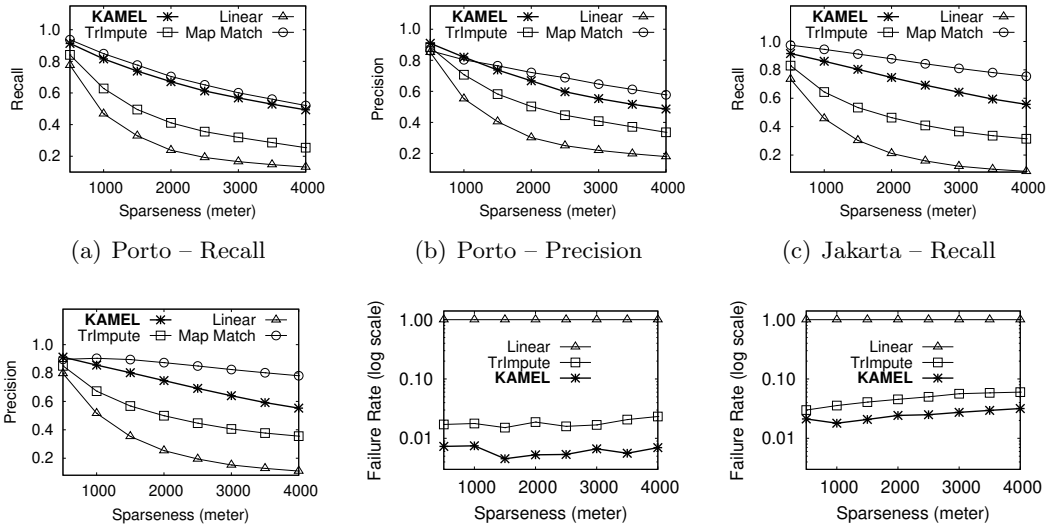
H to 3 and 10 give the best trade-offs between accuracy and overhead. For Porto, we only needed to build three single-cell models, as one in each of the lowest three pyramid levels. For Jakarta, we had to build 20 models as 7 single-cell and 7 neighbor-cells models in the lowest level, two single-cell and one neighbor-cell models in each of the other two levels. We use BERT original architecture as described in [63] and its open-source implementation [18], with 768 hidden dimensions, 12 attention heads, and 12 hidden layers. The average vocabulary size in our experiments is ∼80K, which creates ∼165M trainable parameters.

Experiments Design. Sections 5.8.1 and 5.8.2 study the impact of sparse distance $Sparse_{distance}$ and accuracy threshold δ, respectively. The overhead of training and imputation time is discussed in Section 5.8.3. Section 5.8.4, 5.8.5, and 5.8.6 study the impact of road type, training data properties, and grid type, respectively. Section 5.8.7 performs an ablation study for KAMEL components. Training was conducted using a single Google Cloud TPU. All other experiments were conducted using an Intel(R) Xeon(R) Silver 4112 CPU running @ 2.60GHz, with 196GB of memory and 1TB of SSD storage.

5.8.1 Impact of Data Sparseness

Figure 5.9 gives the impact of data sparseness on the recall, precision, and failure rate of KAMEL and its competitors for both datasets. For reference, we also report the performance of map matching techniques that have the knowledge of the full underlying network, however, we do not consider map matching as a competitor, as KAMEL, and its competitors, work on the environments where road network is not reliable. For all experiments, we vary $Sparse_{distance}$ from 500 to 4,000m. For Porto data in Figures 5.9(a) and 5.9(b), KAMEL consistently achieves much higher accuracy than competitors. It is even very close to the performance of map matching, which shows how powerful is KAMEL where it acts *as if* it knows the underlying road network, while it really has no knowledge about it.

For lower gaps (e.g., 500m), KAMEL gives the best performance, but other techniques, even linear interpolation, still give acceptable performance, as it is not that hard to impute small gaps. For medium gaps (e.g., 1,500 - 2,500m), KAMEL is 1.5 to 3 times better than its competitors. TrImpute was unable to cope with such gaps as it



(a) Porto – Recall (b) Porto – Precision (c) Jakarta – Recall

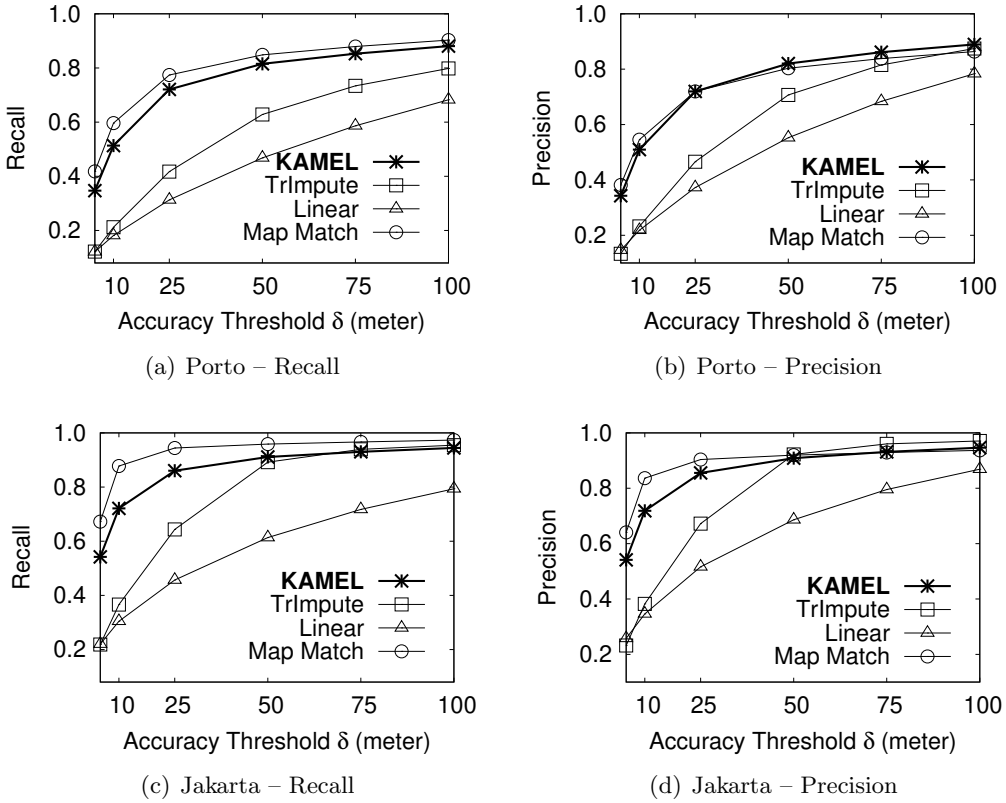
(d) Jakarta – Precision (e) Porto – Failure (f) Jakarta – Failure

Figure 5.9: Impact of Data Sparseness on Recall, Precision, and Failure Rate

only works when there are highly dense prior trajectories, which is not practical. It is important to note that these medium gap sizes are the most practical ones, given the current sampling rates of location-tracking devices. We stretched our experiments to very large sparse gaps for up to 4km, which is analogous to asking BERT to predict a full 200 words paragraph given only the first and last words. Even with this, KAMEL still gives more than 50% recall and precision, while TrImpute and linear interpolation gives around 25% and 10% recall, respectively. This shows that KAMEL is still able to do some useful imputation even for unusually very large gaps.

Figures 5.9(c) and 5.9(d) repeat the same experiments for Jakarta. Similarly, KAMEL consistently outperforms its competitors. However, KAMEL has a better performance than Porto, both in terms of absolute recall and precision and in its relative performance to its competitors. The main reason is that even though Jakarta dataset has significantly less number of trajectories than Porto, each trajectory, on average, has 1,000 points compared to 50 points in Porto. Longer trajectories give more semantics on what points could come after others, and only KAMEL takes full advantage of this.

Figures 5.9(e) and 5.9(f) give the impact of data sparseness on the failure rate for both datasets. By definition, linear interpolation has a 100% failure rate. KAMEL



(a) Porto – Recall

(b) Porto – Precision

(c) Jakarta – Recall

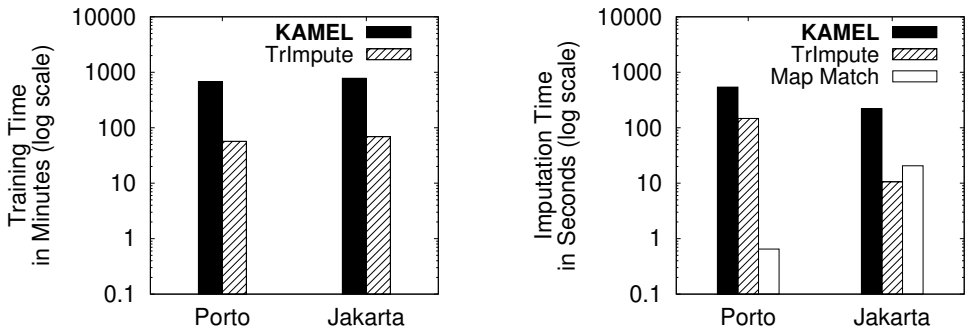
(d) Jakarta – Precision

Figure 5.10: Impact of Accuracy Threshold on Recall and Precision

significantly outperforms TrImpute in both datasets and for all sparse gaps. For Porto, KAMEL has consistently less than 1% failure rate, compared to up to 2.5% for TrImpute. For Jakarta, as the data is more sparse, failure rate is higher for both with up to 3% for KAMEL and 6% for TrImpute. In all cases, failure rate is still within 2% for KAMEL for the medium practical gaps, which is highly acceptable for the trajectory imputation.

5.8.2 Impact of Accuracy Threshold

Figure 5.10 gives the impact of varying the accuracy threshold δ from 5 to 100m on the recall and precision of KAMEL and its competitors for both Porto and Jakarta datasets. We also plot the performance of map matching as a point of reference. The objective is to understand the accuracy of each method when used for either relaxed applications that can entertain large thresholds or sensitive applications that require few



(a) Training Time

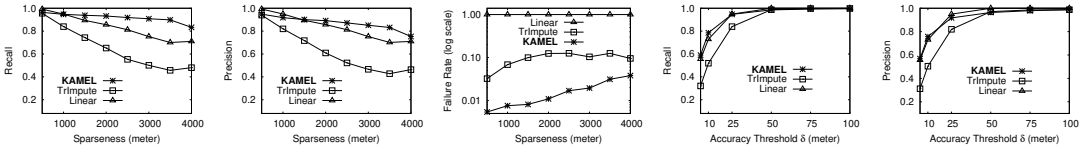
(b) Imputation Time

Figure 5.11: Timing Analysis

meters of accuracy. For Porto dataset, KAMEL consistently achieves higher accuracy than its competitors, with a performance that is almost identical to map matching. For high δ (e.g., 75-100m), though KAMEL gives higher recall and precision than its competitors, the difference is not that much. The main reason is that by increasing δ, lower accuracy techniques like TrImpute would still be able to catch. However, a 100m accuracy threshold may not be practical to consider, especially in dense cities where 100m distance may just lead to a different road. For medium δ (e.g., 25-50m), which are more practical and acceptable by the large majority of trajectory applications, KAMEL has $\sim$80% recall and precision, which is significantly higher than that of TrImpute and linear interpolation. We stretched our experiments to very tight accuracy threshold less than 10m. While TrImpute and linear interpolation become almost useless, KAMEL is still able to cope with $\sim$40% recall and precision. For Jakarta dataset, we get a similar analysis and conclusion. The only difference is that all techniques give higher accuracy for the corresponding δ for Porto. The main reason is that Jakarta spans a large area, hence a higher value of δ would be acceptable as roads are not as close to each other as in Porto. However, for lower δ, KAMEL has significantly higher accuracy.

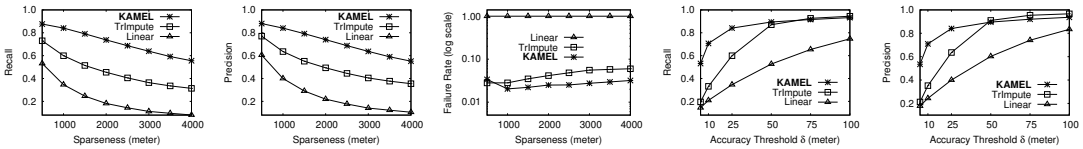
5.8.3 Training and Imputation Time

Figure 5.11 gives the training and imputation time for Porto and Jakarta datasets. For training time, apparently KAMEL takes more time than TrImpute, with 680 minutes for Porto and 780 minutes for Jakarta. This is mainly because KAMEL inherits the complex



(a) (b) (c) (d) (e)

Figure 5.12: Performance Analysis on Straight Segments



(a) (b) (c) (d) (e)

Figure 5.13: Performance Analysis on Curved Segments

training model from BERT, while TrImpute training basically computes a simple set of stats and lookup indices. However, we do not see this as a major issue since training is an offline process and only happens periodically whenever a bulk of new trajectories is received, and without affecting the imputation or causing any system downtime. Figure 5.11(b) gives the average imputation time for trajectories. KAMEL takes longer with 540 seconds for Porto and 220 seconds for Jakarta, which is mainly due to the Multipoint Imputation module that trades-off accuracy with imputation time. Since our main goal in KAMEL is higher accuracy, we tune our modules to achieve higher accuracy, while accepting the imputation time overhead.

5.8.4 Impact of Road Type

This section aims to understand the behavior of KAMEL and its competitors for straight and curved road segments.Hence, as we have the ground truth for our test trajectories, we classify each test trajectory segment into two types: *straight* and *curved*. A segment is identified as *straight* if the Euclidean distance between its two end points is within a very small threshold (5m by default) from their road network distance, otherwise, the segment is identified as *curved*. Figures 5.12 and 5.13 repeat the same experiments of Sections 5.8.1 and Section 5.8.2, when only focusing on *straight* and *curved* segments, respectively. Experiments are only shown for Jakarta dataset. Porto dataset gives a similar analysis and conclusion.

For *straight* segments (Figure 5.12), KAMEL outperforms its competitors in all measures, sparsity gaps, and accuracy thresholds. For sparsity gaps (Figures 5.12(a)-5.12(b)), TrImpute gives the worst performance, as it gets distracted with various directions of surrounding GPS points and misses the easiest case of going in a straight direction. KAMEL outperforms linear interpolation, mainly due to our definition of a *straight* segment, which allows for a 5m threshold. For failure rate (Figure 5.12(c)), KAMEL significantly outperforms others. For various thresholds δ (Figures 5.12(d)-5.12(e)), both KAMEL and linear interpolation exhibit similar high performance, while TrImpute has a much worse performance for lower δ.

For *curved* segments (Figure 5.13), KAMEL consistently has the highest performance in all measures. For the most practical medium gaps, KAMEL significantly outperforms TrImpute, which shows the resilience of KAMEL towards curved segments. For failure rates (Figure 5.13(c)), KAMEL consistently outperforms its competitors. For various thresholds δ (Figures 5.13(d)-5.13(e)), it is expected that the performance of TrImpute and linear interpolation catch with KAMEL for large δ, which is a very relaxed value that is not practical in most applications. For practical medium δ, KAMEL is clearly better, showing its applicability to most trajectory applications. For very tight values of $\delta < 10m$, both TrImpute and linear interpolation are useless, while KAMEL has more than 50% recall and precision.

5.8.5 Impact of Training Data Properties

Figure 5.14 studies the impact of the training data size for Jakarta dataset (Porto dataset gives similar behavior), where we compare four variants of KAMEL trained on 100%, 75%, 50%, and 25% of the available training trajectories. In all metrics, the three variants 100%, 75%, and 50% perform almost identically with only some minor differences. Only the 25% variant shows a noticeable reduction in performance. This shows that KAMEL can still achieve its good results even with as little as half of the data it has. Figure 5.15 studies the impact of the density of training data. We train KAMEL using four variations of Jakarta dataset, all have the same number of trajectories but differ in their GPS density. Those variants include the original dense dataset which has a sampling rate of 1 second, and three other sparse variants with sampling rates 15, 30, and 60 seconds. For all metrics, KAMEL still achieves almost the same performance for

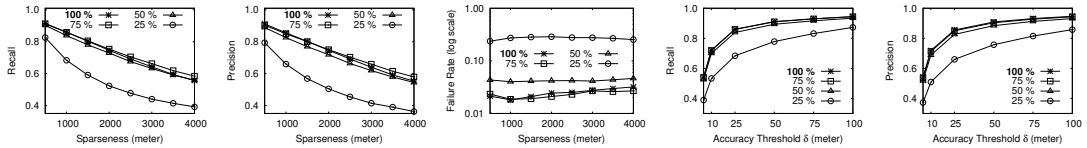


Figure 5.14: Performance Analysis Using Different Training Size

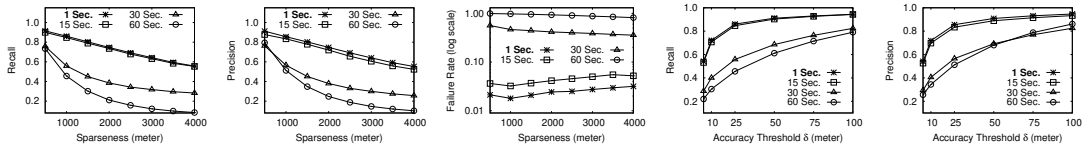


Figure 5.15: Performance Analysis Using Different Training Density

the 1 and 15 seconds sampling rates. It is important to note that the 15 seconds sampled data is basically 15 times less (7%) of the original data. While having dense data is much better for training, this experiment shows that KAMEL can still work perfectly fine with only 7% of its available data for Jakarta.

5.8.6 Impact of Grid Type

Figure 5.16 compares two tokenization alternatives for KAMEL, namely, Uber H3 Hexagons [96] and Google S2 Squares [220]. For S2, we set the edge length to 120m to ensure a similar area coverage as that of hexagons with edge length of 75m. In all measures, a hexagonal grid gives better performance due to its unique properties. In particular, all neighbors of a certain hexagonal cell have identical properties in terms of the length of shared borders and the distance between their centroids, which makes the transition patterns between cells more consistent and easier to learn.

5.8.7 Ablation Analysis

Figure 5.17 performs an ablation study of KAMEL using Jakarta dataset, to understand the impact of each of its components by disabling them one at a time and then measure all performance metrics. In particular, we compare four system variants: (a) KAMEL:

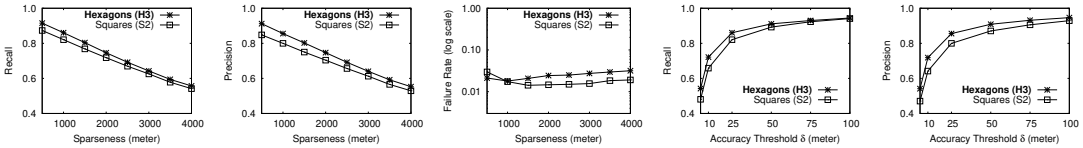


Figure 5.16: Performance Analysis Using Different Grid Types

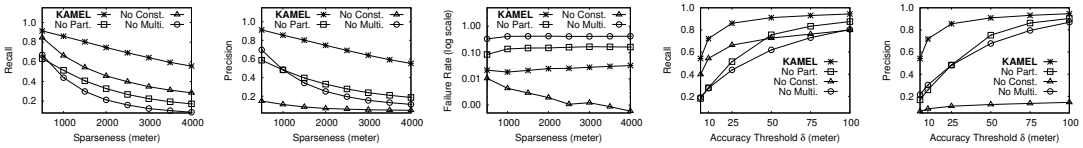


Figure 5.17: Ablation Study on KAMEL System

our full system as explained in this chapter, (b) *No Part.*: disables the Spatial Partitioning module (Section 5.4) by training one BERT model for the entire data, (c) *No Const.*: disables the Spatial Constraints module (Section 5.5) by accepting any BERT prediction, and (d) *No Multi.*: disables the Multipoint Imputation module (Section 5.6) by calling BERT only once to get a single imputation point.

For recall and precision across various $Sparse_{distance}$ (Figures 5.17(a) and 5.17(b)), removing any of KAMEL components significantly reduces its accuracy. For the recall, removing the multipoint imputation affects the performance the most because the system now predicts only one point for each gap and leaves the rest of it, hence reduces the recall. In contrast, removing the spatial constraints affects the precision the most while affecting the recall the least. This is because removing these constraints does not prevent the system from still predicting some accurate imputations, but will allow it to predict so many noisy points. For the failure rate (Figure 5.17(c)), removing any module causes KAMEL to fail more except when removing the spatial constraints, which causes less failure rate. This is because BERT is now allowed to make any prediction regardless of how imprecise it is. Removing the multipoint imputation module causes KAMEL to fail the most as this makes KAMEL predict only one point for each gap. Similarly, in Figures 5.17(d) and 5.17(e), the recall and precision for the full KAMEL for all accuracy thresholds δ is higher than it is when removing any of its components. For the

recall, one observation here is that the variant with no spatial partitioning starts lower than other variants at lower thresholds δ, but then outperforms them with the increase of δ. This shows the maximum effect of the spatial partitioning is at lower thresholds. This is where the locally trained models are mostly needed to improve the accuracy. For precision, without the spatial constraints module, the precision is consistently low around 10% or less regardless of δ.

5.9 Related Work

Trajectory Imputation with Road Network. The immense need for high resolution trajectories motivated the efforts to insert points between each two consecutive trajectory points. Such a process had various names, including trajectory interpolation [156, 305], completion [144], cleaning [293], restoration [135], recovery [267, 258], and imputation [69, 185]. The large majority of such work mainly rely on matching trajectory points on the underlying road network [24, 157], followed by a shortest path algorithm between consecutive points. Unfortunately, none of this work is applicable to us as they all rely on an underlying road network. In our case, we aim for imputation without road network knowledge as our target applications include map inference, which needs to infer the *unknown* road network.

Trajectory Imputation without Road Network. Motivated by the unreliability of road networks [48, 91] and the need to develop trajectory-based map inference techniques [1], it becomes highly important to develop trajectory imputations techniques that do not rely on the road network, and hence can be used as a preparation step before any map inference technique [19, 37, 219, 232]. Unfortunately, existing techniques suffer from one or more of the following: (1) only impute one or two points between each two trajectory points [258], (2) only applicable to impute high granularity zones [129, 213], (3) only applicable to junction-level road network [144]. Our proposed KAMEL system overcomes all of these issues as it scales up to impute tens of points between each two trajectory points, applicable to a very fine granularity of GPS points, and applicable to a city-scale road network. The closest work to ours is TrImpute [69], which uses historical data to find the most likely imputed points for each trajectory segment. However, it only works when there are significant amounts of highly dense historical data, which

is not a practical case. We consider TrImpute as our direct competitor and compare against it in our experiments.

NLP Models and Trajectories. The BERT language model was introduced in 2018 [63] as an infrastructure for a myriad of complex linguistic tasks, including sentence completion. It utilizes the Transformer neural network [252], trained on vast textual corpora by removing random words from sentences and tasks the model with predicting each missing word based on surrounding context (left and right words). Since then, several new models and variations were proposed, including XLNet [282], RoBERTa [153], DistilBERT [223], ALBERT [133], and GPT [27], each focusing on a certain aspect such as training size, objectives, and masking procedures. Research efforts in employing language models for the spatial domain mainly utilize it as a *pretrained* model in its lingual form by verbally asking it questions of spatial nature [111, 209, 273]. Our system KAMEL does not use a pretrained BERT model. Instead, it trains BERT with trajectory data through its system architecture and five components built around BERT. We have chosen the BERT model as it is the first and most commonly used language model, yet other BERT variants can be also used. Our goal is not to find which language model is best suited for trajectories. Instead, our goal is to show that language models, in general, trained with trajectory data, can be adopted to solve trajectory imputation with higher accuracy than state-of-the-art techniques. We opted for a design that employs BERT as is. Another design alternative would be to embed spatial awareness inside the core of each internal BERT component, as outlined in our recent vision paper [190]. Our rationale is that our design is more attractive to a large sector of systems that already deploy the BERT model and would need less disturbance to their systems. Future work would explore the second design option by carefully looking into the internal components of BERT one by one.

Time-series Imputation. In the same way we have mapped the trajectory imputation problem to the missing word problem and used NLP models to solve it, one can also consider mapping it to the multivariate time series imputation problem, and use any of its solutions that are based on Generative Adversarial Networks [159, 160, 226], Generative Learning Models [35, 173, 218], Recurrent Neural Networks [34], Graph Neural Networks [46], or attention mechanisms [299]. Similar to the case of language models, solutions to the multivariate time series imputation: (a) assume only few missing data

points, hence to support trajectory imputation, it would need to go through a multipoint imputation module similar to that of Section 5.6, (b) mainly impute sensor readings, which have a very limited domain of values, hence to support trajectory imputation, it would need Tokenization and Detokenization modules similar to that of Sections 5.3 and 5.7, (c) are not spatially-aware, hence to support trajectory imputation, it would need spatial partitioning and spatial constraints modules similar to that of Sections 5.4 and 5.5. In essence, an orthogonal approach to ours is to start from a multivariate time series imputation algorithm and adopt it in a similar way to what we have done in KAMEL for NLP models.

5.10 Conclusion

This chapter has presented KAMEL; a scalable BERT-based system for trajectory imputation. KAMEL maps the trajectory imputation problem to *finding the missing word* problem in natural language processing (NLP). Hence, KAMEL starts from using BERT, the widely used language model for NLP problems. However, BERT, as is, does not lend itself to the special characteristics of the trajectory imputation problem. Hence, KAMEL architecture is composed of five main modules to make BERT applicable for trajectory imputation. These modules adapt the nature of trajectory data to be more fit for BERT, and inject the spatial-awareness in both the input and output of BERT. Experimental results based on real system implementation and real datasets show that KAMEL significantly outperforms its competitors. In addition, unlike all related approaches, KAMEL was able to impute city-scale trajectories, large sparse gaps that need tens of imputed points, all within tight accuracy thresholds.

Chapter 6

Thesis Conclusion

The widespread availability of GPS-enabled devices has fueled the demand for various essential map services such as routing, travel time estimation, and nearby facility search. While there exists a plethora of efficient algorithms to process these services, the output accuracy remains the primary bottleneck in map services. The key reason lies in the quality of the underlying road network data, which includes not only the map topology but also traffic metadata such as edge weights and travel times. Unfortunately, recent findings and independent studies have shown that current maps suffer from a variety of inaccuracies that directly degrade the accuracy of map services. In this thesis, we laid the foundation and provided the means to significantly improve the accuracy of map services by improving the accuracy of the map data itself. In particular, we presented two main research directions.

The first research direction focuses on architecting a novel end-to-end system for accurate map services. We introduced **QARTA** (Chapter 3); which achieves the following: (1) learns its own highly accurate map, with accurate edge weights that reflect detailed traffic throughout the week, and (2) calibrates the result of map services through its built-in machine learning modules that continuously understand the error margin of various query processors, and use it to adjust the results. Our experiments show that QARTA reduces the error in estimated time of arrival (ETA) queries by more than 25% compared to state-of-the-art open-source methods, demonstrating its effectiveness in enhancing the accuracy of map services.

The second research direction focuses on providing novel frameworks to efficiently analyze trajectory data, which is an important data source for constructing accurate maps. We presented our vision **"Let's Speak Trajectories"** (Chapter 4), which leverages state-of-the-art natural language processing (NLP) models like BERT to support trajectory analysis via a single unified framework. We showed that trajectories in a space exhibit a very similar behavior to statements in a language, and hence many of the NLP tasks on statements are analogous to spatial analysis tasks on trajectories. We also highlighted key challenges that limited the direct application of BERT and proposed two paths forward: customizing trajectory data to fit BERT, and injecting the BERT core with spatial and temporal awareness. As a case study, we presented **KAMEL** system (Chapter 5), which maps the *trajectory imputation* problem to *finding the missing word* problem in NLP. We defined the main modules that make BERT applicable for trajectory imputation. Specifically, these modules adapt the nature of trajectory data to be more fit for BERT, and inject spatial awareness in both the input and output of BERT. Experimental results based on real system implementation and real datasets show that KAMEL significantly outperforms its competitors.

In summary, the research contributions presented in this thesis established the foundation and offered practical tools for constructing detailed, high-quality maps, thereby improving the accuracy of map services and enhancing the quality of all derived map-based applications.

References

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

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

[3] Sofiane Abbar, Rade Stanojevic, Mashaal Musleh, Mohamed ElShrif, and Mohamed Mokbel. 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):2723–2726, 2021.

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

[5] Louai Alarabi, Mohamed F. Mokbel, and Mashaal Musleh. ST-Hadoop: A MapReduce Framework for Spatio-temporal Data. *GeoInformatica*, 22(4):785–813, 2018.

[6] Reem Y. Ali, Venkata M. V. Gunturi, and Shashi Shekhar. 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):19–25, 2014.

[7] Rakan Alseghayer. Racoon: Rapid Contact Tracing of Moving Objects Using Smart Indexes. In *Proceedings of the International Conference on Mobile Data Management, MDM*, 2021.

[8] Amazon Delivery and Logistics. https://www.aboutamazon.com/what-we-do/delivery-logistics.

[9] Road Inference From GPS Trajectories. https://www.amazon.science/publications/ring-net-road-inference-from-gps-trajectories-using-a-deep-segmentation-network/.

[10] Jennings Anderson, Dipto Sarkar, and Leysia Palen. Corporate Editors in the Evolving Landscape of OpenStreetMap. *ISPRS International Journal of Geo-Information*, 8(5):232, 2019.

[11] Apple Maps. https://www.apple.com/maps/.

[12] Apple rolls out all-new map across Belgium, Liechtenstein, Luxembourg, the Netherlands, and Switzerland. https://www.apple.com/cm/newsroom/2022/12/apple-rolls-out-all-new-map-across-belgium-liechtenstein-luxembourg-the-netherlands-and-switzerland/.

[13] Walid G. Aref and Hanan Samet. Efficient Processing of Window Queries in The Pyramid Data Structure. In *Proceedings of the ACM Symposium on Principles of Database Systems, PODS*, 1990.

[14] Akinori Asahara, Kishiko Maruyama, Akiko Sato, and Kouichi Seto. Pedestrian-Movement Prediction Based on Mixed Markov-Chain Model. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2011.

[15] Jie Bao, Chi-Yin Chow, Mohamed F. Mokbel, and Wei-Shinn Ku. Efficient Evaluation Of K-Range Nearest Neighbor Queries In Road Networks. In *Proceedings of the International Conference on Mobile Data Management, MDM*, pages 115–124, May 2010.

[16] Emmanouil Barmpounakis and Nikolas Geroliminis. On the New Era of Urban Traffic Monitoring with Massive Drone Data: The pNEUMA Large-scale Field Experiment. *Transportation Research Part C: Emerging Technologies*, 111:50–71, 2020.

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

[18] Bert implementation by google. https://github.com/google-research/bert/.

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

[20] Bing Maps. `https://www.bing.com/maps`.

[21] Bloomberg CityLab. Who Owns the Digital Map of the World? `https://www.bloomberg.com/news/articles/2015-06-25/mapbox-openstreetmap-and-the-future-of-the-global-digital-mapping-industry`.

[22] Wendy Bohte and Kees Maat. Deriving and Validating Trip Purposes and Travel Modes for Multi-Day GPS-Based Travel Surveys: A Large-Scale Application in the Netherlands. *Transportation Research Part C: Emerging Technologies*, 17(3):285–297, 2009.

[23] Lorenzo Bracciale, Marco Bonola, Pierpaolo Loreti, Giuseppe Bianchi, Raul Amici, and Antonello Rabuffi. CRAWDAD dataset roma/taxi (v. 2014-07-17). Downloaded from `https://crawdad.org/roma/taxi/20140717`, July 2014.

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

[25] Igo Ramalho Brilhante, José Antônio Fernandes de Macêdo, Franco Maria Nardini, Raffaele Perego, and Chiara Renso. Planning Sightseeing Tours Using Crowdsensed Trajectories. *ACM SIGSPATIAL Special*, 7(1), 2015.

[26] Isaac Brodsky. Uber's Hexagonal Hierarchical Spatial Index. `https://eng.uber.com/h3`.

[27] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Annual Conference on Neural Information Processing Systems, NeurIPS*, 2020.

[28] Dun Cao, Kai Zeng, Jin Wang, Pradip Kumar Sharma, Xiaomin Ma, Yonghe Liu, and Siyuan Zhou. BERT-Based Deep Spatial-Temporal Network for Taxi Demand Prediction. *IEEE Trans. Intell. Transp. Syst.*, 23(7):9442–9454, 2022.

[29] Lili Cao and John Krumm. 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*, pages 3–12, January 2009.

[30] Mário Cardoso, André Cavalheiro, Alexandre Borges, Ana Filipa Duarte, Amílcar Soares, Maria João Pereira, Nuno Jardim Nunes, Leonardo Azevedo, and Arlindo L. Oliveira. Modeling the Geospatial Evolution of COVID-19 using Spatio-temporal Convolutional Sequence-to-sequence Neural Networks. *ACM Transactions on Spatial Algorithms and Systems, TSAS*, 8(4):28:1–28:19, 2022.

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

[32] Pingfu Chao, Wen Hua, and Xiaofang Zhou. 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):47–73, 2020.

[33] Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. A Survey on Map-Matching Algorithms. In *Australasian Database Conference, ADC*, pages 121–133, February 2020.

[34] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent Neural Networks for Multivariate Time Series with Missing Values. *Nature Scientific Reports*, 2018.

[35] Zhengping Che, Sanjay Purushotham, Max Guangyu Li, Bo Jiang, and Yan Liu. Hierarchical Deep Generative Models for Multi-Rate Multivariate Time Series. In *Proceedings of the International Conference on Machine Learning, ICML*, 2018.

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

[37] Chen Chen, Cewu Lu, Qixing Huang, Qiang Yang, Dimitrios Gunopulos, and Leonidas J. Guibas. City-Scale Map Creation and Updating using GPS Collections. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 1465–1474, August 2016.

[38] Ling Chen, Yanlin Tang, Mingqi Lv, and Gencai Chen. Partition-Based Range Query For Uncertain Trajectories In Road Networks. *GeoInformatica*, 19(1):61–84, 2015.

[39] Lisi Chen, Shuo Shang, Christian S. Jensen, Bin Yao, and Panos Kalnis. Parallel Semantic Trajectory Similarity Join. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2020.

[40] Lisi Chen, Shuo Shang, Christian S. Jensen, Bin Yao, Zhiwei Zhang, and Ling Shao. Effective Efficient Reuse of Past Travel Behavior for Route Recommendation. In *Proceedings of*

the ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD, 2019.

[41] Yen-Chun Chen, Zhe Gan, Yu Cheng, Jingzhou Liu, and Jingjing Liu. Distilling Knowledge Learned in BERT for Text Generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL*, 2020.

[42] Yile Chen, Xiucheng Li, Gao Cong, Zhifeng Bao, Cheng Long, Yiding Liu, Arun Kumar Chandran, and Richard Ellison. Robust Road Network Representation Learning: When Traffic Patterns Meet Traveling Semantics. In *Proceedings of the International Conference on Information and Knowledge Management, CIKM*, pages 211–220, November 2021.

[43] Yuhao Chen and Farhana H. Zulkernine. BIRD-QA: A BERT-based Information Retrieval Approach to Domain Specific Question Answering. In *Proceedings of the IEEE International Conference on Big Data, BigData*, 2021.

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

[45] Mirla Rafaela Rafael Braga Chucre, Samara Martins do Nascimento, José Antônio Fernandes de Macêdo, José Maria Monteiro, and Marco Antonio Casanova. Taxi, Please! A Nearest Neighbor Query In Time-Dependent Road Networks. In *Proceedings of the International Conference on Mobile Data Management, MDM*, pages 180–185, June 2016.

[46] Andrea Cini, Ivan Marisca, and Cesare Alippi. Filling the G_ap_s: Multivariate Time Series Imputation by Graph Neural Networks. In *International Conference on Learning Representations, ICLR*, 2022.

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

[48] CNN Buisness. The Billion Dollar War over Maps. `https://money.cnn.com/2017/06/07/technology/business/maps-wars-self-driving-cars/index.html`.

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

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

[51] Commercial OSM Software and Services. `https://wiki.openstreetmap.org/wiki/Commercial_OSM_Software_and_Services`.

[52] Alessandro Crivellari, Bernd Resch, and Yuhui Shi. TraceBERT - A Feasibility Study on Reconstructing Spatial-Temporal Gaps from Incomplete Motion Trajectories via BERT Training Process on Discrete Location Sequences. *Sensors*, 22(4), 2022.

[53] Danilo Croce, Giuseppe Castellucci, and Roberto Basili. GAN-BERT: Generative Adversarial Learning for Robust Text Classification with a Bunch of Labeled Examples. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL*, 2020.

[54] Philippe Cudré-Mauroux and Eugene Wu andvSamuel Madden. TrajStore: An Adaptive Storage System for Very Large Trajectory Data Sets. In *Proceedings of the International Conference on Data Engineering, ICDE*, pages 109–120, Long Beach, CA, USA, March 2010.

[55] Philippe Cudré-Mauroux, Eugene Wu, and Samuel Madden. TrajStore: An Adaptive Storage System for Very Large Trajectory Data sets. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2010.

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

[57] Sajal K. Das, Diane J. Cook, Amiya Bhattacharya, Edwin O. Heierman III, and Tze-Yun Lin. The Role of Prediction Algorithms In The Mavhome Smart Home Architecture. *IEEE Wirel. Commun.*, 9(6):77–84, 2002.

[58] Sajal K. Das and Sanjoy K. Sen. Adaptive Location Prediction Strategies Based on a Hierarchical Network Model in a Cellular Mobile Environment. *Comput. J.*, 42(6):473–486, 1999.

[59] Elton Figueiredo de S. Soares, Kate Revoredo, Fernanda Baião, Carlos Alvaro de M. S. Quintella, and Carlos Alberto Vieira Campos. A Combined Solution for Real-Time Travel Mode Detection and Trip Purpose Prediction. *IEEE Trans. Intell. Transp. Syst.*, 20(12):4655–4664, 2019.

[60] Roniel S. de Sousa, Azzedine Boukerche, and Antonio A. F. Loureiro. Vehicle Trajectory Similarity: Models, Methods, and Applications. *ACM Comput. Surv.*, 53(5), 2020.

[61] Ugur Demiryurek, Farnoush Banaei Kashani, and Cyrus Shahabi. 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*, pages 474–477, November 2010.

[62] Liwei Deng, Hao Sun, Rui Sun, Yan Zhao, and Han Su. Efficient and Effective Similar Subtrajectory Search: A Spatial-aware Comprehension Approach. *ACM TIST*, 13(3):35:1–35:22, 2022.

[63] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805, 2018.

[64] Daniel Dias and Luis Henrique Maciel Kosmalski Costa. CRAWDAD dataset coppe-ufrj/riobuses (v. 2018-03-19). Downloaded from `https://crawdad.org/coppe-ufrj/RioBuses/20180319`, March 2018.

[65] Corey Dickinson. Inside the Wikipedia of Maps Tensions Grow Over Corporate Influence. Bloomberg. `https://www.bloomberg.com/news/articles/2021-02-19/openstreetmap-charts-a-controversial-new-direction`.

[66] Doordash. `https://www.doordash.com/`.

[67] Chunning Du, Haifeng Sun, Jingyu Wang, Qi Qi, and Jianxin Liao. Adversarial and Domain-Aware BERT for Cross-Domain Sentiment Analysis. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL*, 2020.

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

[69] Mohamed M. Elshrif, Keivin Isufaj, and Mohamed F. Mokbel. Network-less trajectory imputation. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2022.

[70] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, 1996.

[71] Facebook AI. Mapping roads through deep learning and weakly supervised training. `https://ai.facebook.com/blog/mapping-roads-through-deep-learning-and-weakly-supervised-training/`.

[72] Facebook Engineering. MaRS: How Facebook keeps maps current and accurate. https://engineering.fb.com/2019/09/30/ml-applications/mars/.

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

[74] FedEx. https://www.fedex.com/en-us/delivery-manager.html.

[75] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. DeepMove: Predicting Human Mobility with Attentional Recurrent Networks. In *Proceedings of the International Conference on World Wide Web, WWW*, 2018.

[76] Shanshan Feng, Gao Cong, Bo An, and Yeow Meng Chee. POI2Vec: Geographical Latent Representation for Predicting Future Visitors. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, pages 102–108, February 2017.

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

[78] Tao-Yang Fu and Wang-Chien Lee. Trembr: Exploring Road Networks for Trajectory Representation Learning. *ACM Trans. Intell. Syst. Technol.*, 11(1), 2020.

[79] Qiang Gao, Goce Trajcevski, Fan Zhou, Kunpeng Zhang, Ting Zhong, and Fengli Zhang. DeepTrip: Adversarially Understanding Human Mobility for Trip Recommendation. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2019.

[80] Geo Awesomeness. What does the acquisition of HERE mean for Nokia, carmakers, TomTom, Google and the industry? https://geoawesomeness.com/what-does-the-acquisition-of-here-mean-for-nokia-carmakers-tomtom-google-and-the-industry/.

[81] GeoLife GPS Trajectories. https://www.microsoft.com/en-us/download/details.aspx?id=52367.

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

[83] Jean-Francois Girres and Guillaume Touya. Quality Assessment of the French OpenStreetMap Dataset. *Trans. in GIS*, 14(4):435–459, 2010.

[84] Similarity Challenge. ACM SIGSPATIAL Cup 2017. https://sigspatial2017.sigspatial.org/giscup2017/download.

[85] GIS Lounge. Businesses Using Open Source GIS. https://www.gislounge.com/businesses-using-open-source-gis/.

[86] Francesco Giuliari, Irtiza Hasan, Marco Cristani, and Fabio Galasso. Transformer Networks for Trajectory Forecasting. In *Proceedings of the IEEE International Conference on Pattern Recognition, ICPR*, pages 10335–10342, January 2020.

[87] Marta C. González, César A. Hidalgo, and Albert-László Barabási. Understanding Individual Human Mobility Patterns. *Nature*, 453(7196):779–782, 2008.

[88] Google Maps. https://www.google.com/maps.

[89] Google Maps API. https://developers.google.com/maps.

[90] GPS World. TomTom-Tele Atlas Merger a Done Deal. https://www.gpsworld.com/consumer-oemnewstomtom-tele-atlas-merger-a-done-deal-2911/.

[91] Grand View Research. Abolute Reports. Global High Accuracy Map Market Size, Status and Forecast 2021-2027, March 2020. https://www.grandviewresearch.com/industry-analysis/digital-map-market.

[92] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In *Proceedings of the International Conference on Data Engineering, ICDE*, pages 152–159, February 1996.

[93] Chenjuan Guo, Bin Yang, Ove Andersen, Christian S. Jensen, and Kristian Torp. EcoSky: Reducing Vehicular Environmental Impact through Eco-routing. In *Proceedings of the International Conference on Data Engineering, ICDE*, pages 1412–1415, April 2015.

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

[95] Chenjuan Guo, Bin Yang, Jilin Hu, Christian S. Jensen, and Lu Chen. Context-aware, Preference-based Vehicle Routing. *VLDB Journal*, 29(5):1149–1170, 2020.

[96] H3: Hexagonal Hierarchical Geospatial Indexing System. https://h3geo.org/.

[97] Conversion from latitude/longitude to containing H3 Cell Index. https://h3geo.org/docs/core-library/latLngToCellDesc/.

[98] Glen Hart and Catherine Dolbear. *What's So Special about Spatial?*, pages 39–44. Springer, 01 2007. Arno Scharl and Klaus Tochtermann.

[99] Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. 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*, pages 3–12, November 2018.

[100] Tianfu He, Jie Bao, Sijie Ruan, Ruiyuan Li, Yanhua Li, Hui He, and Yu Zheng. Interactive Bike Lane Planning Using Sharing Bikes' Trajectories. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 32(8), 2020.

[101] Abdeltawab Hendawi and Mohamed F. Mokbel. Panda: A Predictive Spatio-Temporal Query Processor. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 13–22, Redondo Beach, CA, November 2012.

[102] Abdeltawab Hendawi and Mohamed F. Mokbel. Predictive Spatio-Temporal Queries: A Comprehensive Survey and Future Directions. In *Proceeding of the ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems, MobiGIS, co-located with SIGSPATIAL GIS*, pages 97–104, Redondo Beach, CA, November 2012.

[103] HERE. `https://www.here.com/platform`.

[104] Tin Kam Ho. Random Decision Forests. In *Proceeding of the IEEE International Conference on Document Analysis and Recognition ICDAR*, pages 278–282, August 1995.

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

[106] Bushra Hossain, Kazi Abir Adnan, Md. Fazle Rabbi, and Mohammed Eunus Ali. Modelling Road Traffic Congestion from Trajectories. In *Proceedings of the ACM International Conference on Data Science and Information Technology, DSIT*, 2020.

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

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

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

[110] Ling Hu, Yinan Jing, Wei-Shinn Ku, and Cyrus Shahabi. Enforcing K Nearest Neighbor Query Integrity On Road Networks. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 422–425, November 2012.

[111] Jizhou Huang, Haifeng Wang, Yibo Sun, Yunsheng Shi, Zhengjie Huang, An Zhuo, and Shikun Feng. ERNIE-GeoL: A Geography-and-Language Pre-trained Model and its Applications in Baidu Maps. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 3029–3039, August 2022.

[112] Xiaocheng Huang, Yifang Yin, Simon Lim, Guanfeng Wang, Bo Hu, Jagannadan Varadarajan, Shaolin Zheng, Ajay Bulusu, and Roger Zimmermann. Grab-posisi: An extensive real-life GPS trajectory dataset in southeast asia. In *PredictGIS@SIGSPATIAL*, 2019.

[113] Xiaocheng Huang, Yifang Yin, Simon Lim, Guanfeng Wang, Bo Hu, Jagannadan Varadarajan, Shaolin Zheng, Ajay Bulusu, and Roger Zimmermann. Grab-posisi: An extensive real-life GPS trajectory dataset in southeast asia. In *Proceedings of the ACM SIGSPATIAL International Workshop on Prediction of Human Mobility, PredictGIS 2019*, pages 1–10, Chicago, IL, USA, November 2019.

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

[115] Tsuyoshi Idé and Masashi Sugiyama. Trajectory Regression on Road Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, August 2011.

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

[117] Kent T. Jacobs and Scott W. Mitchell. OpenStreetMap Quality Assessment using Unsupervised Machine Learning Methods. *Trans. in GIS*, 24(5):1280–1298, 2020.

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

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

[120] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, and Christian S. Jensen. Path Prediction And Predictive Range Querying In Road Network Databases. *VLDB Journal*, 19(4):585–602, 2010.

[121] Antonios Karatzoglou, Adrian Jablonski, and Michael Beigl. Seq2Seq Learning Approach Modeling Semantic Trajectories and Predicting Next Location. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2018.

[122] Panagiota Katsikouli, Rik Sarkar, and Jie Gao. Persistence Based Online Signal and Trajectory Simplification for Mobile Devices. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2014.

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

[124] Amin Vahedian Khezerlou, Xun Zhou, Ling Tong, Yanhua Li, and Jun Luo. Forecasting Gathering Events through Trajectory Destination Prediction: A Dynamic Hybrid Model. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 33(3):991–1004, 2021.

[125] Sang-Wook Kim, Jung-Im Won, Jong-Dae Kim, Miyoung Shin, Junghoon Lee, and Hanil Kim. Path Prediction of Moving Objects on Road Networks Through Analyzing Past Trajectories. In *Proceedings of the International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES*, volume 4692, pages 379–389, September 2007.

[126] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations, ICLR*, April 2017.

[127] Satoshi Koide, Yukihiro Tadokoro, Chuan Xiao, and Yoshiharu Ishikawa. CiNCT: Compression and Retrieval for Massive Vehicular Trajectories via Relative Movement Labeling. In *Proceedings of the International Conference on Data Engineering, ICDE*, pages 1097–1108, April 2018.

[128] Benjamin B. Krogh, Ove Andersen, Edwin Lewis-Kelham, Nikos Pelekis, Yannis Theodoridis, and Kristian Torp. Trajectory Based Traffic Analysis. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2013.

[129] John Krumm. Maximum Entropy Bridgelets for Trajectory Completion. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2022.

[130] Chia-Chih Kuo, Shang-Bao Luo, and Kuan-Yu Chen. An Audio-Enriched BERT-Based Framework for Spoken Multiple-Choice Question Answering. In *Interspeech*, 2020.

[131] Moritz Laass, Marie Kiermeier, and Martin Werner. Improving Persistence Based Trajectory Simplification. In *Proceedings of the International Conference on Mobile Data Management, MDM*, 2021.

[132] Hai Lan, Jiong Xie, Zhifeng Bao, Feifei Li, Wei Tian, Fang Wang, Sheng Wang, and Ailin Zhang. VRE: A Versatile, Robust, and Economical Trajectory Data System. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 15(12):3398–3410, 2022.

[133] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *Proceedings of the International Conference on Learning Representations, ICLR*, 2020.

[134] Bing Li, Yukai Miao, Yaoshu Wang, Yifang Sun, and Wei Wang. Improving the Efficiency and Effectiveness for BERT-based Entity Resolution. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2021.

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

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

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

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

[139] Tianyi Li, Ruikai Huang, Lu Chen, Christian S. Jensen, and Torben Bach Pedersen. Compression of Uncertain Trajectories in Road Networks. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 13(7):1050–1063, 2020.

[140] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. Deep Representation Learning for Trajectory Similarity Computation. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2018.

[141] Yaguang Li, Han Su, Ugur Demiryurek, Bolong Zheng, Kai Zeng, and Cyrus Shahabi. PerNav: A Route Summarization Framework for Personalized Navigation. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, pages 2125–2128, June 2016.

[142] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations, ICLR*, May 2018.

[143] Yang Li, Qixing Huang, Michael Kerber, Lin Zhang, and Leonidas J. Guibas. 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*, pages 214–223, November 2013.

[144] Yang Li, Yangyan Li, Dimitrios Gunopulos, and Leonidas J. Guibas. 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*, pages 33:1–33:10, October 2016.

[145] Yanhua Li, Jun Luo, Chi-Yin Chow, Kam-Lam Chan, Ye Ding, and Fan Zhang. Growing Charging Station Network For Electric Vehicles With Trajectory Data Analytics. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2015.

[146] Yuxuan Liang, Kun Ouyang, Yiwei Wang, Xu Liu, Hongyang Chen, Junbo Zhang, Yu Zheng, and Roger Zimmermann. TrajFormer: Efficient Trajectory Classification with Transformers. In *Proceedings of the International Conference on Information and Knowledge Management, CIKM*, pages 1229–1237, October 2022.

[147] Fandel Lin and Hsun-Ping Hsieh. An Intelligent And Interactive Route Planning Maker For Deploying New Transportation Services. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2018.

[148] Yan Lin, Huaiyu Wan, Shengnan Guo, and Youfang Lin. Pre-training Context and Time Aware Location Embeddings from Spatial-Temporal Trajectories for User Next Location Prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2021.

[149] Jonathan Liono, Zahraa S. Abdallah, A. Kai Qin, and Flora D. Salim. Inferring Transportation Mode and Human Activity from Mobile Sensing in Daily Life. In *Proceedings of the ACM International Conference on Mobile and Ubiquitous Systems, MobiQuitous*, pages 342–351, New York City, NY, November 2018.

[150] List of OSM-based services. `https://wiki.openstreetmap.org/wiki/List_of_OSM-based_services`.

[151] An Liu, Yifan Zhang, Xiangliang Zhang, Guanfeng Liu, Yanan Zhang, Zhixu Li, Lei Zhao, Qing Li, and Xiaofang Zhou. Representation Learning With Multi-Level Attention for Activity Trajectory Similarity Computation. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 34(5):2387–2400, 2022.

[152] Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. Predicting the Next Location: A Recurrent Model with Spatial and Temporal Contexts. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2016.

[153] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692, 2019.

[154] Cheng Long, Raymond Chi-Wing Wong, and H. V. Jagadish. Direction-Preserving Trajectory Simplification. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 6(10), 2013.

[155] Cheng Long, Raymond Chi-Wing Wong, and H. V. Jagadish. Trajectory Simplification: On Minimizing the Direction-based Error. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 8(1), 2014.

[156] Jed A. Long. Kinematic Interpolation of Movement Data. *International Journal of Geographical Information Science*, 30(5):854–868, 2016.

[157] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. Map-Matching for Low-Sampling-Rate GPS Trajectories. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2009.

[158] Lili Lu and Shuaian Wang. Literature Review of Analytical Models on Emergency Vehicle Service: Location, Dispatching, Routing and Preemption Control. In *Proceedings of the IEEE International Intelligent Transportation Systems Conference, ITSC*, pages 3031–3036, 2019.

[159] Yonghong Luo, Xiangrui Cai, Ying Zhang, Jun Xu, and Xiaojie Yuan. Multivariate Time Series Imputation with Generative Adversarial Networks. In *Advances in Neural Information Processing Systems, NeurIPS*, 2018.

[160] Yonghong Luo, Ying Zhang, Xiangrui Cai, and Xiaojie Yuan. E^2GAN: End-to-End Generative Adversarial Network for Multivariate Time Series Imputation. In *IJCAI*, 2019.

[161] Zhongjian Lv, Jiajie Xu, Kai Zheng, Hongzhi Yin, Pengpeng Zhao, and Xiaofang Zhou. LC-RNN: A Deep Learning Model for Traffic Speed Prediction. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 3470–3476, July 2018.

[162] Lyft. `https://www.lyft.com/`.

[163] Lyft Engineering. How Lyft Creates Hyper-Accurate Maps from Open-Source Maps and Real-Time Data. `https://eng.lyft.com/how-lyft-creates-hyper-accurate-maps-from-open-source-maps-and-real-time-data-8dcf9abdd46a`.

[164] Lyft Engineering. How Lyft discovered OpenStreetMap is the Freshest Map for Rideshare. `https://eng.lyft.com/how-lyft-discovered-openstreetmap-is-the-freshest-map-for-rideshare-a7a41bf92ec`.

[165] Lyft mapping team. ground truth evaluation of openstreetmap quality in north american cities. `https://drive.google.com/file/d/1Sb-dOUjeP1Ljqz4ra931D3Pe8B5C3pde/view`.

[166] Suxing Lyu, Tianyang Han, Yuuki Nishiyama, Kaoru Sezaki, and Takahiko Kusakabe. A Plug-in Memory Network for Trip Purpose Classification. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 34:1–34:12, November 2022.

[167] Suxing Lyu and Takahiko Kusakabe. Graph-aware Chained Trip Purpose Inference. In *Proceedings of the IEEE International Intelligent Transportation Systems Conference, ITSC*, pages 3691–3697, September 2021.

[168] Gengchen Mai, Chris Cundy, Kristy Choi, Yingjie Hu, Ni Lao, and Stefano Ermon. Towards A Foundation Model for Geospatial Artificial Intelligence (Vision Paper). In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 106:1–106:4, 2022.

[169] Gengchen Mai, Krzysztof Janowicz, Bo Yan, Rui Zhu, Ling Cai, and Ni Lao. Multi-Scale Representation Learning for Spatial Feature Distributions using Grid Cells. In *Proceedings of the International Conference on Learning Representations, ICLR*, April 2020.

[170] Mapillary. Unveiling the Mapping in Logistics Report: The Impact of Broken Maps on Last-Mile Deliveries. https://blog.mapillary.com/update/2020/02/14/mapping-in-logistics.html.

[171] Wesley Mathew, Ruben Raposo, and Bruno Martins. Predicting Future Locations with Hidden Markov Models. In *Proceedings of the ACM International Conference on Ubiquitous Computing, Ubicomp*, 2012.

[172] Chuishi Meng, Xiuwen Yi, Lu Su, Jing Gao, and Yu Zheng. City-wide Traffic Volume Inference with Loop Detector Data and Taxi Trajectories. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2017.

[173] Xiaoye Miao, Yangyang Wu, Jun Wang, Yunjun Gao, Xudong Mao, and Jianwei Yin. Generative Semi-supervised Learning for Multivariate Time Series Imputation. In *AAAI*, 2021.

[174] Discover New Roads with Bing Maps. https://blogs.bing.com/maps/2022-12/Bing-Maps-is-bringing-new-roads/.

[175] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the International Conference on Learning Representations, ICLR*, 2013.

[176] Marco Minghini and Francesco Frassinelli. OpenStreetMap History for Intrinsic Quality Assessment: Is OSM up-to-date? *Open Deospatial Data, Software, and Standards*, 4(9):1–17, 2019.

[177] Mohamed F. Mokbel, Sofiane Abbar, and Rade Stanojevic. Contact Tracing: Beyond the Apps. *ACM SIGSPATIAL Special*, 12(2), 2020.

[178] Money Control News. Uber may shun Google Maps for open source ones: Report. https://www.moneycontrol.com/news/business/uber-may-shun-google-maps-for-open-source-ones-report-2764111.html.

[179] Anna Monreale, Fabio Pinelli, Roberto Trasarti, and Fosca Giannotti. Wherenext: A Location Predictor on Trajectory Pattern Mining. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, 2009.

[180] David Montoya, Serge Abiteboul, and Pierre Senellart. Hup-Me: Inferring and Reconciling a Timeline of User Activity from Rich Smartphone Data. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2015.

[181] Joe Morrison. OpenStreetMap is Having a Moment: The Billion Dollar Dataset Next Door. Medium Artcile. `https://joemorrison.medium.com/openstreetmap-is-having-a-moment-dcc7eef1bb01`.

[182] Sankarshan Mridha, Niloy Ganguly, and Sourangshu Bhattacharya. 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*, pages 71:1–71:4, November 2017.

[183] Mashaal Musleh. Towards A Unified Deep Model For Trajectory Analysis. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 109:1–109:2, 2022.

[184] Mashaal Musleh, Sofiane Abbar, Rade Stanojevic, and Mohamed Mokbel. QARTA: An ML-based System for Accurate Map Services. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 14(11):2273–2282, 2021.

[185] Mashaal Musleh and Mohamed Mokbel. A Demonstration of KAMEL: A Scalable BERT-based System for Trajectory Imputation. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2023.

[186] Mashaal Musleh and Mohamed F. Mokbel. RASED: A Scalable Dashboard for Monitoring Road Network Updates in OSM. In *Proceedings of the International Conference on Mobile Data Management, MDM*, 2022.

[187] Mashaal Musleh and Mohamed F Mokbel. A Demonstration of RASED: A Scalable Dashboard for Monitoring Road Network Updates in OSM. In *IEEE International Conference on Data Engineering, ICDE*, 2022. To Appear.

[188] Mashaal Musleh and Mohamed F. Mokbel. KAMEL: A Scalable BERT-based System for Trajectory Imputation. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 17(3):525–538, 2023.

[189] Mashaal Musleh and Mohamed F. Mokbel. Let's Speak Trajectories: A Vision to Use NLP Models for Trajectory Analysis Tasks. *ACM Transactions on Spatial Algorithms and Systems, TSAS*, 10(2):15, 2024.

[190] Mashaal Musleh, Mohamed F. Mokbel, and Sofiane Abbar. Let's Speak Trajectories. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 37:1–37:4, Seattle, WA, USA, November 2022.

[191] Nextbillion.ai. OpenStreetMap for Businesses: A Primer. White Paper. https://nextbillion.ai/whitepapers/OpenStreetMap-for-Businesses-A-Primer.

[192] Anthony J. Nicholson and Brian D. Noble. BreadCrumbs: Forecasting Mobile Connectivity. In *Proceedings of the International Conference on Mobile Computing and Networking, MOBICOM*, pages 46–57, September 2008.

[193] Kaggle. New York City Taxi Trip Duration. https://www.kaggle.com/c/nyc-taxi-trip-duration/data.

[194] OpenStreetMap Blog. Apple Maps. https://blog.openstreetmap.org/2012/10/02/apple-maps/.

[195] OpenStreetMap Wiki. Organised Editing/Activities/Amazon. https://wiki.openstreetmap.org/wiki/Organised_Editing/Activities/Amazon.

[196] OpenStreetMap. http://www.openstreetmap.org/.

[197] OSM API Calls Dashbord per Server (Culebre). https://prometheus.openstreetmap.org/d/9xY_21OMk/apache?orgId=1&refresh=1m&var-instance=culebre&from=now-7d&to=now.

[198] OSM Boundaries. https://osm-boundaries.com/.

[199] OSM Statistics. https://wiki.openstreetmap.org/wiki/Stats.

[200] Open Source Routing Machine (OSRM). http://project-osrm.org/.

[201] Dian Ouyang, Dong Wen, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. Progressive Top-K Nearest Neighbors Search in Large Road Networks. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, pages 1781–1795, June 2020.

[202] Dian Ouyang, Long Yuan, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 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):602–615, 2020.

[203] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. Anytime Stochastic Routing with Hybrid Learning. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 13(9):1555–1567, 2020.

[204] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. Anytime Stochastic Routing with Hybrid Learning. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 13(9):1555–1567, 2020.

[205] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. Fast stochastic routing under time-varying uncertainty. *VLDB Journal*, 29(4):819–839, 2020.

[206] 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. Scikit-learn: Machine learning in Python. *Journal of machine Learning research*, 12(85):2825–2839, 2011.

[207] Nicole Peinelt, Dong Nguyen, and Maria Liakata. tBERT: Topic Models and BERT Joining Forces for Semantic Similarity Detection. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL*, 2020.

[208] Shangfu Peng, Jagan Sankaranarayanan, and Hanan Samet. SPDO: High-throughput road distance computations on Spark using Distance Oracles. In *Proceedings of the International Conference on Data Engineering, ICDE*, pages 1239–1250, May 2016.

[209] Vamsi Krishna Penumadu, Nitesh Methani, and Saurabh Sohoney. Learning Geospatially Aware Place Embeddings via Weak-supervision. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 80:1–80:10, 2022.

[210] Lucas May Petry, Carlos Andres Ferrero, Luis Otávio Alvares, Chiara Renso, and Vania Bogorny. Towards Semantic-Aware Multiple-Aspect Trajectory Similarity Measuring. *Transactions in GIS*, 23(5), 2019.

[211] Taxi Service Trajectory. Prediction Challenge. ECML PKDD 2015. `http://www.geolink.pt/ecmlpkdd2015-challenge/dataset.html`.

[212] Reinald Adrian Pugoy and Hung-Yu Kao. Unsupervised Extractive Summarization-Based Representations for Accurate and Explainable Collaborative Filtering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL*, 2021.

[213] Kyle Kai Qin, Yongli Ren, Wei Shao, Brennan Lake, Filippo Privitera, and Flora D. Salim. Multiple-level Point Embedding for Solving Human Trajectory Imputation with Prediction. *ACM Transactions on Spatial Algorithms and Systems, TSAS*, 2023.

[214] Jinmeng Rao, Song Gao, and Xiaojin Zhu. VTSV: A Privacy-Preserving Vehicle Trajectory Simulation and Visualization Platform Using Deep Reinforcement Learning. In *ACM SIGSPATIAL Workshop on AI for Geo Knowledge Discovery, GeoAI*, 2021.

[215] Efstratios Rappos, Stephan Robert, and Philippe Cudré-Mauroux. 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*, pages 319–328, November 2018.

[216] D Raj Reddy et al. Speech Understanding Systems: A Summary of Results of the Five-Year Research Effort. *Department of Computer Science. Carnegie Mellon University, Pittsburgh, PA*, 17, 1977.

[217] Sasank Reddy, Min Y. Mun, Jeff Burke, Deborah Estrin, Mark H. Hansen, and Mani B. Srivastava. Using Mobile Phones to Determine Transportation Modes. *ACM Transactions on Sensors Networks*, 6(2), 2010.

[218] Xiaobin Ren, Kaiqi Zhao, Patricia J. Riddle, Katerina Taskova, Qingyi Pan, and Lianyan Li. DAMR: Dynamic Adjacency Matrix Representation Learning for Multivariate Time Series Imputation. *Proceeding of ACM Management of Data*, 1(2), 2023.

[219] Sijie Ruan, Cheng Long, Jie Bao, Chunyang Li, Zisheng Yu, Ruiyuan Li, Yuxuan Liang, Tianfu He, and Yu Zheng. Learning to Generate Maps from Trajectories. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, pages 890–897, February 2020.

[220] S2 Spherical Geometry Library. `https://s2geometry.io/`.

[221] Ibrahim Sabek and Mohamed Mokbel. Machine Learning Meets Big Spatial Data (Tutorial). In *Proceedings of the International Conference on Data Engineering, ICDE*, pages 1782–1785, April 2020.

[222] Amin Sadri, Flora D. Salim, Yongli Ren, Wei Shao, John Krumm, and Cecilia Mascolo. What Will You Do for the Rest of the Day?: An Approach to Continuous Trajectory Prediction. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(4):186:1–186:26, 2018.

[223] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[224] Shankha Shubhra Sarkar, Anindya Sen, A. Krishnamoorthy, and V. Vijayarajan. Route Planning Service for Emergency Vehicles with Increased Accuracy and Efficiency for Online Platforms. *SN Computer Science*, 3(5), 2022.

[225] Abdessamed Sassi, Mohammed Brahimi, Walid Bechkit, and Abdelmalik Bachir. Location Embedding and Deep Convolutional Neural Networks for Next Location Prediction. In *IEEE LCN Symposium on Emerging Topics in Networking*, pages 149–157, October 2019.

[226] Reza Shahbazian and Sergio Greco. Generative Adversarial Networks Assist Missing Data Imputation: A Comprehensive Survey and Evaluation. *IEEE Access*, 11, 2023.

[227] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. DITA: Distributed In-Memory Trajectory Analytics. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2018.

[228] Li Shen and Peter R Stopher. A Process for Trip Purpose Imputation from Global Positioning System Data. *Transportation Research Part C: Emerging Technologies*, 36:261–267, 2013.

[229] Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating Next-Cell Predictors with Extensive Wi-Fi Mobility Data. *IEEE Trans. Mob. Comput.*, 5(12):1633–1649, 2006.

[230] Rade Stanojevic, Sofiane Abbar, and Mohamed Mokbel. 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*, pages 424–427, November 2018.

[231] Rade Stanojevic, Sofiane Abbar, and Mohamed Mokbel. MapReuse: Recycling Routing API Queries. In *Proceedings of the International Conference on Mobile Data Management, MDM*, pages 279–287, June 2019.

[232] Rade Stanojevic, Sofiane Abbar, Saravanan Thirumuruganathan, Sanjay Chawla, Fethi Filali, and Ahid Aleimat. Robust Road Map Inference through Network Alignment of Trajectories. In *Proceedings of the SIAM International Conference on Data Mining, SDM*, pages 135–143, May 2018.

[233] Rade Stanojevic, Sofiane Abbar, Saravanan Thirumuruganathan, Gianmarco De Francisci Morales, Sanjay Chawla, Fethi Filali, and Ahid Aleimat. Road Network Fusion for Incremental Map Updates. In *Processigns on the Intenranional Conference on Progress in Location Based Services, LBS*, pages 91–109, January 2018.

[234] Leon Stenneth, Ouri Wolfson, Philip S. Yu, and Bo Xu. Transportation Mode Detection Using Mobile Phones and GIS Information. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2011.

[235] Studio Software Blog. Google Maps vs OpenStreetMap: Which One Is Better for Your Project? `https://studiosoftware.com/blog/google-maps-vs-openstreetmap/`.

[236] Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. A Survey of Trajectory Distance Measures and Performance Evaluation. *VLDB Journal*, 29(1):3–32, 2020.

[237] Tao Sun, Zonglin Di, Pengyu Che, Chun Liu, and Yin Wang. Leveraging Crowdsourced GPS Data for Road Extraction From Aerial Imagery. In *IEEE International Conference on Computer Vision and Pattern Recognition, CVPR*, pages 7509–7518, June 2019.

[238] Christoph Sydora, Faiza Nawaz, Leepakshi Bindra, and Eleni Stroulia. Building Occupancy Simulation and Analysis under Virus Scenarios. *ACM Transactions on Spatial Algorithms and Systems, TSAS*, 8(3):1–20, 2022.

[239] Kohei Tanaka, Yasue Kishino, Tsutomu Terada, and Shojiro Nishio. A Destination Prediction Method Using Driving Contexts and Trajectory for Car Navigation Systems. In *ACM Symposium on Applied Computing (SAC)*, pages 190–195, March 2009.

[240] T-Drive Trajectory Data Sample. `https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/`.

[241] Tesla. `https://www.tesla.com/`.

[242] Tesmanian. tesla and spacex news. tesla owners improve smart summon routes by updating open street maps. `https://www.tesmanian.com/blogs/tesmanian-blog/tesla-owners-smart-summon-routes-open-street-maps-full-self-driving`.

[243] Traffic Technology Today. Poor maps costing delivery companies US $6bn annually. `https://www.traffictechnologytoday.com/news/mapping/poor-maps-costing-delivery-companies-us6bn-annually.html`.

[244] Luan Tran, Minyoung Mun, Matthew Lim, Jonah Yamato, Nathan Huh, and Cyrus Shahabi. 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):2957–2960, 2020.

[245] Kazuki Tsunematsu, Johanes Effendi, Sakriani Sakti, and Satoshi Nakamura. Neural Speech Completion. In *Interspeech*, 2020.

[246] Uber. `https://www.uber.com/`.

[247] Uber Eats. `https://www.ubereats.com/`.

[248] Uber engineering. enhancing the quality of uber maps with metrics computation. `https://eng.uber.com/maps-metrics-computation/`.

[249] UCR STAR: The UCR Spatio-temporal Active Repository. OSM/GPS Dataset for Portland, OR, USA. `https://star.cs.ucr.edu/?OSM/GPS#center=45.5428,-122.6544`.

[250] UPS. `https://www.ups.com/us/en/services/professional-services/delivery-services.page`.

[251] Amin Vahedian, Xun Zhou, Ling Tong, Yanhua Li, and Jun Luo. Forecasting Gathering Events through Continuous Destination Prediction. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2017.

[252] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Annual Conference on Neural Information Processing Systems, NeurIPS*, 2017.

[253] Bijun Wang, Yulong Wang, Kun Qin, and Qizhi Xia. Detecting transportation modes based on lightgbm classifier from GPS trajectory data. In *Geoinformatics*, 2018.

[254] Guang Wang, Xiuyuan Chen, Fan Zhang, Yang Wang, and Desheng Zhang. Experience: Understanding Long-Term Evolving Patterns of Shared Electric Vehicle Networks. In *Proceeding of the International Conference on Mobile Computing and Networking, MobiCom*, pages 1–12, Los Cabos, Mexico, October 2019.

[255] Haojun Wang and Roger Zimmermann. Processing of Continuous Location-Based Range Queries on Moving Objects in Road Networks. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 23(7):1065–1078, 2011.

[256] Hongjian Wang and Zhenhui Li. Region Representation Learning via Mobility Flow. In *Proceedings of the International Conference on Information and Knowledge Management, CIKM*, pages 237–246, November 2017.

[257] Jingyuan Wang, Jiawei Jiang, Wenjun Jiang, Chao Li, and Wayne Xin Zhao. LibCity: An Open Library for Traffic Prediction. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2021.

[258] Jingyuan Wang, Ning Wu, Xinxi Lu, Wayne Xin Zhao, and Kai Feng. Deep Trajectory Recovery with Fine-Grained Calibration using Kalman Filter. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 33(3), 2021.

[259] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, and Gao Cong. A Survey on Trajectory Data Management, Analytics, and Learning. *ACM Comput. Surv.*, 54(2), 2021.

[260] Suyi Wang, Yusu Wang, and Yanjie Li. 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*, pages 1–10, November 2015.

[261] Yilun Wang, Yu Zheng, and Yexiang Xue. Travel Time Estimation of A Path Using Sparse Trajectories. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 25–34, August 2014.

[262] Yong Wang, Guoliang Li, and Nan Tang. Querying Shortest Paths on Time Dependent Road Networks. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 12(11):1249–1261, 2019.

[263] Zheng Wang, Cheng Long, and Gao Cong. Trajectory Simplification with Reinforcement Learning. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2021.

[264] Waze. `https://www.waze.com/`.

[265] Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. 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*, pages 434–437, November 2013.

[266] Yan Wen, Jiansong Zhang, Qingtian Zeng, Xin Chen, and Feng Zhang. Loc2Vec-Based Cluster-Level Transition Behavior Mining for Successive POI Recommendation. *IEEE Access*, 7:109311–109319, 2019.

[267] Hao Wu, Jiangyun Mao, Weiwei Sun, Baihua Zheng, Hanyuan Zhang, Ziyang Chen, and Wei Wang. Probabilistic Robust Route Recovery with Spatio-Temporal Dynamics. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, 2016.

[268] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest Path and Distance Queries on Road Networks: An Experimental Evaluation. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 5(5):406–417, 2012.

[269] Yonghui Wu et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016.

[270] Guangnian Xiao, Zhicai Juan, and Chunqin Zhang. Detecting Trip Purposes from Smartphone-Based Travel Surveys with Artificial Neural Networks and Particle Swarm imization. *Transportation Research Part C: Emerging Technologies*, 71:447–463, 2016.

[271] Li Xiong, Cyrus Shahabi, Yanan Da, Ritesh Ahuja, Vicki Hertzberg, Lance Waller, Xiaoqian Jiang, and Amy Franklin. REACT: Real-Time Contact Tracing and Risk Monitoring Using Privacy-enhanced Mobile Tracking. *ACM SIGSPATIAL Special*, 12(2), 2020.

[272] Hao Xue, Flora D. Salim, Yongli Ren, and Nuria Oliver. MobTCast: Leveraging Auxiliary Trajectory Forecasting for Human Mobility Prediction. In *Annual Conference on Neural Information Processing Systems, NeurIPS*, 2021.

[273] Hao Xue, Bhanu Prakash Voutharoja, and Flora D. Salim. Leveraging Language Foundation Models for Human Mobility Forecasting. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 90:1–90:9, November 2022.

[274] Bo Yan, Krzysztof Janowicz, Gengchen Mai, and Song Gao. From ITDL to Place2Vec: Reasoning About Place Type Similarity and Relatedness by Learning Embeddings From Augmented Spatial Contexts. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 35:1–35:10, November 2017.

[275] Bin Yang, Jian Dai, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. PACE: a PAth-CEntric paradigm for stochastic path finding. *VLDB Journal*, 27(2):153–178, 2018.

[276] Bin Yang, Manohar Kaul, and Christian S. Jensen. Using Incomplete Information for Complete Weight Annotation of Road Networks. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 26(5):1267–1279, 2014.

[277] Can Yang and Gyozo Gidofalvi. Fast Map Matching, An Algorithm Integrating Hidden Markov Model With Precomputation. *International Journal of Geographical Information Science*, 32(3), 2018.

[278] Peilun Yang, Hanchen Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. T3S: Effective Representation Learning for Trajectory Similarity Computation. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2021.

[279] Sean Bin Yang, Chenjuan Guo, and Bin Yang. Context-aware Path Ranking in Road Networks. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 2020.

[280] Xiaochun Yang, Bin Wang, Kai Yang, Chengfei Liu, and Baihua Zheng. A Novel Representation and Compression for Queries on Trajectories in Road Networks. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 30(4):613–629, 2018.

[281] Yu Yang, Fan Zhang, and Desheng Zhang. SharedEdge: GPS-Free Fine-Grained Travel Time Estimation in State-Level Highway Systems. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(1):48:1–48:26, 2018.

[282] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Annual Conference on Neural Information Processing Systems, NeurIPS*, 2019.

[283] Bin Yao, Zhongpu Chen, Xiaofeng Gao, Shuo Shang, Shuai Ma, and Minyi Guo. Flexible Aggregate Nearest Neighbor Queries In Road Networks. In *Proceedings of the International Conference on Data Engineering, ICDE*, pages 761–772, April 2018.

[284] Di Yao, Chao Zhang, Jian-Hui Huang, and Jingping Bi. SERM: A Recurrent Model for Next Location Prediction in Semantic Trajectories. In *Proceedings of the International Conference on Information and Knowledge Management, CIKM*, pages 2411–2414, November 2017.

[285] Zijun Yao, Yanjie Fu, Bin Liu, Wangsu Hu, and Hui Xiong. Representing Urban Functions through Zone Embedding with Human Mobility Patterns. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 3919–3925, July 2018.

[286] Deborah Yates. How Facebook, Apple and Microsoft are contributing to an openly licensed map of the world. The Open Data Institute (ODI). `https://theodi.org/article/how-are-facebook-apple-and-microsoft-contributing-to-openstreetmap/`.

[287] Yifang Yin, Zhenguang Liu, Ying Zhang, Sheng Wang, Rajiv Ratn Shah, and Roger Zimmermann. GPS2Vec: Towards Generating Worldwide GPS Embeddings. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, pages 416–419, November 2019.

[288] Josh Jia-Ching Ying, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S. Tseng. Semantic Trajectory Mining for Location Prediction. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2011.

[289] Ziqiang Yu, Xiaohui Yu, Nick Koudas, Yang Liu, Yifan Li, Yueting Chen, and Dingyu Yang. Distributed Processing of k Shortest Path Queries over Dynamic Road Networks. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, pages 665–679, June 2020.

[290] Haitao Yuan and Guoliang Li. Distributed In-memory Trajectory Similarity Search and Join on Road Network. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2019.

[291] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. Effective Travel Time Estimation: When Historical Trajectories over Road Networks Matter. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, pages 2135–2149, June 2020.

[292] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. T-Drive: Driving Directions Based on Taxi Trajectories. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS*, 2010.

[293] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S. Yu. Time Series Data Cleaning: From Anomaly Detection to Anomaly Repairing. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 10(10):1046–1057, 2017.

[294] Desheng Zhang. Description for Electric Vehicle Data Release V0. Downloaded from `https://www.cs.rutgers.edu/~dz220/Data/ETData.rar`.

[295] Hongyu Zhang and Jacek Malczewski. Accuracy Evaluation of the Canadian OpenStreetMap Road Networks. *International Journal of Geospatial and Environmental Research*, 5(2):1:1–1:14, 2018.

[296] Hongyu Zhang and Jacek Malczewski. *Quality Evaluation of Volunteered Geographic Information: The Case of OpenStreetMap*, chapter 58, pages 1173–1201. IGI Global, 2019.

[297] Shaohua Zhang, Haoran Huang, Jicong Liu, and Hang Li. Spelling Error Correction with Soft-Masked BERT. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, ACL*, 2020.

[298] Yunchao Zhang, Yanjie Fu, Pengyang Wang, Xiaolin Li, and Yu Zheng. Unifying Inter-region Autocorrelation and Intra-region Structures for Spatial Embedding via Collective Adversarial Learning. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pages 1700–1708, August 2019.

[299] Jingqi Zhao, Chuitian Rong, Chunbin Lin, and Xin Dang. Multivariate time series data imputation using attention-based mechanism. *Neurocomputing*, 542, 2023.

[300] Yan Zhao, Shuo Shang, Yu Wang, Bolong Zheng, Quoc Viet Hung Nguyen, and Kai Zheng. REST: A Reference-based Framework for Spatio-temporal Trajectory Compression. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, 2018.

[301] Fangfang Zheng and Henk Van Zuylen. Urban Link Travel Time Estimation based on Sparse Probe Vehicle Data. *Transportation Research Part C: Emerging Technologies*, 31:145–157, 2013.

[302] Guanjie Zheng, Hanyang Liu, Kai Xu, and Zhenhui Li. Learning to Simulate Vehicle Trajectories from Demonstrations. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2020.

[303] Jiangchuan Zheng and Lionel M. Ni. Time-Dependent Trajectory Regression on Road Networks via Multi-Task Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, July 2013.

[304] Kai Zheng, Goce Trajcevski, Xiaofang Zhou, and Peter Scheuermann. Probabilistic Range Queries For Uncertain Trajectories On Road Networks. In *Proceedings of the International Conference on Extending Database Technology, EDBT*, pages 283–294, March 2011.

[305] Kai Zheng, Yu Zheng, Xing Xie, and Xiaofang Zhou. Reducing Uncertainty of Low-Sampling-Rate Trajectories. In *Proceedings of the International Conference on Data Engineering, ICDE*, pages 1144–1155, April 2012.

[306] Yu Zheng. Trajectory Data Mining: An Overview. *ACM TIST*, 6(3), 2015.

[307] Yu Zheng, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma. Understanding Transportation Modes Based on GPS Data for Web Applications. *ACM Tran. on Web*, 4(1), 2010.

[308] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. Learning Transportation Mode from Raw GPS Data for Geographic Applications on the Web. In *Proceedings of the International Conference on World Wide Web, WWW*, 2008.

[309] Yu Zheng, Xing Xie, and Wei-Ying Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engineering Bulletin*, 33(2):32–39, 2010.

[310] Fan Zhou, Hantao Wu, Goce Trajcevski, Ashfaq A. Khokhar, and Kunpeng Zhang. Semi-supervised Trajectory Understanding with POI Attention for End-to-End Trip Recommendation. *ACM Transactions on Spatial Algorithms and Systems, TSAS*, 6(2), 2020.

[311] Ningnan Zhou, Wayne Xin Zhao, Xiao Zhang, Ji-Rong Wen, and Shan Wang. A General Multi-Context Embedding Model for Mining Human Trajectory Data. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 28(8), 2016.

[312] Yang Zhou and Yan Huang. DeepMove: Learning Place Representations through Large Scale Movement Data. In *Proceedings of the IEEE International Conference on Big Data, BigData*, pages 2403–2412, December 2018.

[313] Minfeng Zhu, Wei Chen, Jiazhi Xia, Yuxin Ma, Yankong Zhang, Yuetong Luo, Zhaosong Huang, and Liangjun Liu. Location2vec: A Situation-Aware Representation for Visual Exploration of Urban Locations. *IEEE Trans. Intell. Transp. Syst.*, 20(10):3981–3990, 2019.

[314] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. In *ICCV*, 2015.