# Enriching Digital Maps with Aerial Imagery and GPS Data

by

Songtao He

Master of Science, Massachusetts Institute of Technology (2018)
Bachelor of Engineering, University of Science and Technology of China (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Department of Electrical Engineering and Computer Science
May 13, 2022

Certified by. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Hari Balakrishnan
Fujitsu Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

# Enriching Digital Maps with Aerial Imagery and GPS Data

by

Songtao He

Submitted to the Department of Electrical Engineering and Computer Science
on May 13, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

## Abstract

Digital street maps with rich features are the foundation of many applications. However, creating and maintaining up-to-date digital maps often involve many labor-intensive tasks, making the mapping process time-consuming and expensive. This thesis explores automated techniques for enriching digital street maps from aerial imagery and GPS data.

Digital street maps consist of a collection of geometry structures such as a road graph and the semantics associated with the structures, such as the lane count and the speed limit of a road segment. This thesis first proposes two solutions, RoadRunner and Sat2Graph, to automatically extract road-level street maps from GPS trajectory data and aerial imagery, respectively. Road-level street maps serve as the base maps in digital street maps, providing the basic yet fundamental way-finding service to the map users. However, road-level street maps don't have lane structure information, which is essential for lane-to-lane navigation and autonomous vehicles. Therefore, this thesis proposes a mapping pipeline that extracts lane-level street maps from aerial imagery. Besides road structure extraction, this thesis proposes RoadTagger to infer road attributes such as the lane count and road type of road segments from aerial imagery. Finally, this thesis proposes a mapping solution to create high-resolution traffic accident risk maps that can enrich the semantics of existing digital maps and enable new applications such as safety-aware routing and precise insurance.

Thesis Supervisor: Hari Balakrishnan
Title: Fujitsu Professor of Electrical Engineering and Computer Science

# Acknowledgments

Six years ago, I came to Cambridge and started my journey at MIT, feeling excited about the future, but little was I anticipating all the adventures ahead. Throughout this journey, I'm grateful to meet so many kind and wise people; it was them who made my past six years at MIT a truly rewarding and remarkable experience for me. This thesis cannot be completed without their kindest help and greatest support.

First, I would like to thank my advisor, Hari Balakrishnan. He gave me the opportunity to come to MIT, study in a Ph.D. program, and conduct my research work. During my Ph.D. study, Hari gave me a lot of time to learn, grow, and explore different ideas that attracted me; sometimes, he also created opportunities for me to do so. He always encourages me to conduct research work as doing science - keep asking why, focus on explaining the unknowns, and let my curiosity guide me forward. Looking back, there were great moments but also down times; it was Hari's continuous trust and encouragement that kept me on the right track. I'm deeply grateful to Hari.

I would like to thank my committee member, Mohammad Alizadeh and Samuel Madden. They were there in my first project meeting at MIT back in 2016. Since then, I've been so lucky to work with them on almost every project I did at MIT. They gave me numerous valuable guidance, many of which turned out to be the most critical push to the success of several projects. I'm hugely grateful that I can work with them during my Ph.D. study.

Next, I want to thank Favyen Bastani - a strong collaborator in work and a good friend in life. Favyen and I joined the Ph.D. program in the same year. In my first project meeting at MIT six years ago, he was also there. After that, we collaborated on numerous projects, starting from the very beginning, when we were struggling to land our first project, all the way to the end of this journey. He gave me enormous help and support in my research work. We also had many good times discussing random research ideas and interesting thoughts.

I would like to thank my collaborators from Qatar Computing Research Institute, Sanjay, Amin, and Sofiane. We collaborated on the map-making project for more than five years, and their novel ideas and insights continuously inspired me. Our work cannot be done without their vital contribution.

Also, I would like to thank all my other collaborators: Mehrdad and Hongzi, in our project on applying reinforcement learning to active queue management; Arjun, Karthik, and Ziwen, in our drone sensing and intelligence project; Edward and Satvat in our map-making project. They diversified the scope of my research, and I learned a lot from them.

I would like to thank everyone I met in the NMS group: Ahmed, Akshay, Alex, Amy, Anirudh, Arash, Arjun, Deepti, Frank, Hongzi, Inho, James, Lei, Manya, Mehrdad, Mingran, Moein, Pari, Peter, Pouya, Prateesh, Radhika, Ravi, Ravichandra, Seo Jin, Sheila, Sudarsanan, Tiffany, Venkat, Vibhaa, Vikram, Weiyang, Will, and Zhizhen. Each time I got a chance to chat with someone in the group, either while we were making coffee in the 9-th floor's kitchen or at a random event, the conversation we had always gave me substantial energy. I'm grateful for being one of this group.

Equally important, I want to thank all my friends around me during my Ph.D. study. They definitely made my Ph.D. life much more colorful.

Next, I would like to express my eternal gratitude to my beloved fiancèe, Zhuyun. Our paths merged in Cambridge shortly after I started at MIT. Since then, she has been there with me during all my ups and downs, supporting and guarding my dream, and being a big fan of all my research work. She made me better than I was and made my life unprecedentedly wonderful. I'm indebted to her forever.

Lastly, I would like to express my deepest gratitude to my parents, Huiming and Chen, for their perpetual support and unconditional love. They always support and trust my decisions, encourage me to pursue what I feel is the most important, and give me their best.

*To my beloved fiancée and parents*

## Previously Published Material

Chapter 2 revises a previous publication [54]: Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden. RoadRunner: Improving the Precision of Road Network Inference from GPS Trajectories. ACM SIGSPATIAL, Seattle, WA, November 2018.

Chapter 3 revises a previous publication [56]: Songtao He, Favyen Bastani, Satvat Jagwani, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Mohamed M. Elshrif, Samuel Madden, Amin Sadeghi. Sat2Graph: Road Graph Extraction through Graph-Tensor Encoding. ECCV, Glasgow, Scotland, August 2020.

Chapter 4 revises a previous publication [53]: Songtao He, Hari Balakrishnan. Lane-Level Street Map Extraction from Aerial Imagery. WACV, WAIKOLOA, Hawaii, Jan 2022.

Chapter 5 revises a previous publication [58]: Songtao He, Favyen Bastani, Satvat Jagwani, Edward Park, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Samuel Madden, Mohammad Amin Sadeghi. RoadTagger: Robust Road Attribute Inference with Graph Neural Networks. AAAI, New York, NY, February 2020.

Chapter 6 revises a previous publication [59]: Songtao He, Mohammad Amin Sadeghi, Sanjay Chawla, Mohammad Alizadeh, Hari Balakrishnan, Samuel Madden. Inferring High-Resolution Traffic Accident Risk Maps based on Satellite Imagery and GPS Trajectories. ICCV, virtual, Oct 2021.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Digital street maps are collections of information that describe the road infrastructure around us. The most basic element of a digital street map is the road network, which contains information about where the roads are and how the roads connect. This information provides a simple but fundamental service to the map users - helping them find routes going from one location to another. Though simple, this service has played a critical role in our everyday life and work.

We consider the road network as the base map in a digital street map. Starting from this base map, digital street maps have evolved drastically in the past decades. In this evolution, more and more features have been incorporated into digital street maps, including a rich set of road attributes such as the road type (e.g., residential road or highway road), the lane count, the surface type of the road, biking lane type, etc. These new features have enabled many new applications to improve map users' experience. For example, customized routing becomes possible with rich road attribute information; new drivers can set their routing preference to avoid primary and highway roads, cyclists can find routes with separated bike lanes, and runners can find routes with the best surfaces for their knees.

Besides road attributes, digital street maps also start to incorporate more subtle road structures, which enable the support for new applications and further improve the user experience. For example, the position of each driving lane of a road and the positions of the turning lanes at intersections are critical map elements for lane-to-lane navigation and self-driving car routing. A map that contains crosswalk and sidewalk information can significantly improve the navigation experience for pedestrians. Beyond describing the concrete subjects on the road, modern digital street maps also include information such as real-time traffic, crash and road closure reports, and even carbon emission estimation of routes. All these bring the importance of digital street maps to a new level.

## 1.1 Challenges in Map-Making

Nowadays, digital street maps with rich features have become the foundation of many applications, benefiting all of us directly and indirectly. However, creating and maintaining digital street maps is time-consuming and labor-intensive. For example, the most popular crowdsourced digital street map provider, OpenStreetMap (OSM), uses iD editor as its default map editor. The contributors can create roads by drawing polylines over satellite imagery background and entering the road attributes through a GUI. The whole process is manual and slow. Even though OSM has over 8.3 million (2022-01-10) map contributors, the base map still has limited coverage in some remote places. Many essential road attributes are still missing even in popular areas.

The reason behind the incompleteness of OSM is not only about the time-consuming mapping process but also due to the nature of maps; *road networks are growing.* For example, in Figure 1-1, we show the satellite images (years 2000, 2010, and 2020) of Doha, Qatar, a rapidly developing city. The road network grew dramatically from 2000 to 2020. Therefore, creating a near-complete map requires the map makers to map all the existing roads and keep tracking all the newly constructed roads.

In the meantime, *Road networks are changing.* In Figure 1-2, we show the satellite images of a roundabout in Doha from 2017 to 2020. At the very beginning, there is only one roundabout. After a few months, a new intersection was constructed near it. Then, a new roundabout was constructed, people started to use it, and the old intersection was removed. This example implies that the map makers also need to monitor the changes of existing roads to create a near-complete map.

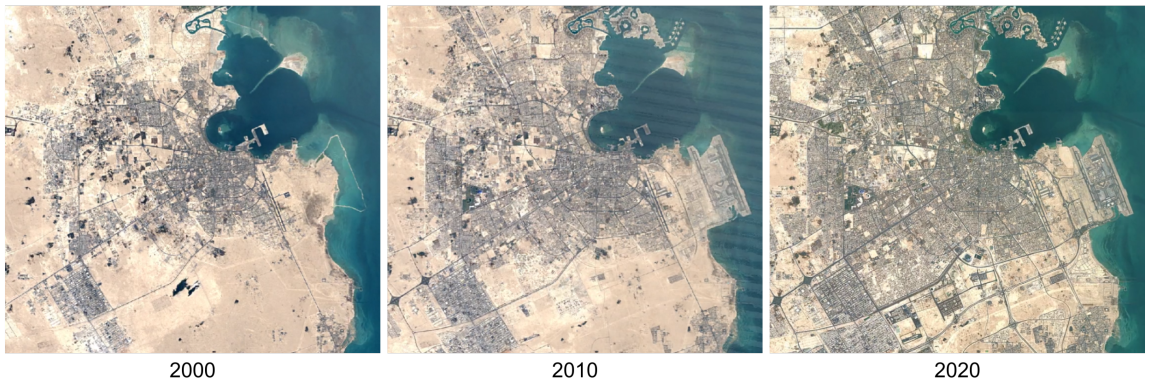


Figure 1-1: Satellite images of Doha, Qatar in years 2000, 2010 and 2020. Road network grew dramatically from 2000 to 2020.

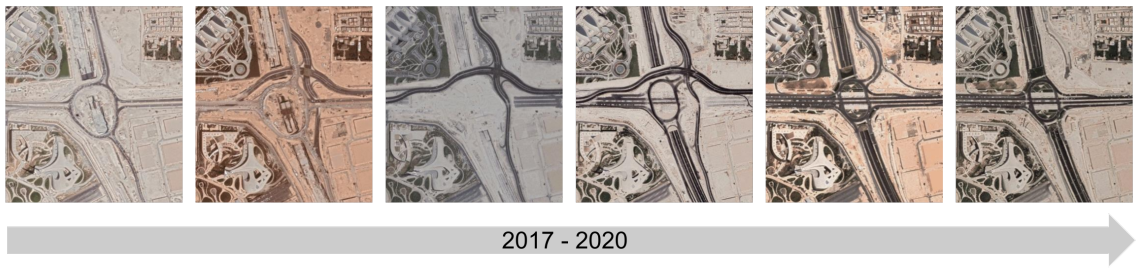


Figure 1-2: Satellite images of a roundabout in Doha from 2017 to 2020. Road network is changing over time. For example, a roundabout changed into an intersection then changed back to a roundabout.

Road networks are growing and changing, making it highly challenging to create and maintain digital street maps on a global scale. Because of the high mapping costs, mapping resources are more dedicated to high-value areas. As a result, the base map coverage is limited in many remote areas. Many new map features such as lane-level maps, crosswalks, and sidewalk maps are only available in popular cities. To make digital maps accessible to everyone globally, we have to develop low-cost and scalable mapping solutions. This thesis focuses on developing such mapping solutions. Specifically, this thesis focuses on automatic mapping techniques that rely on largely available and low-cost data sources and can help enrich digital street maps at a global scale.

## 1.2 Automatic Mapping Techniques

Automatic mapping techniques fall into two categories based on their data sources. One type of mapping solution relies on the sensors mounted on vehicles, e.g., GPS receivers, IMUs, cameras, and lidars. This mapping solution can provide decent accuracy and cover a rich set of road features such as speed limits, street names, and turning restrictions. However, mapping costs are often high because it relies on sensor-equipped vehicles, especially when maintaining an up-to-date map covering a large region. Among all the possible data sources from vehicles, GPS trajectory data has many potentials to achieve relatively low-cost mapping as the acquisition of GPS trajectory data becomes cheaper and more scalable with the ubiquitous of GPS-equipped smartphones. However, GPS data are often noisy. As a result, existing GPS-based mapping solutions often have low precision.

Another type of mapping solution relies on aerial imagery collected from airplanes or satellites. This type of mapping solution can quickly scale up to a large region at a low cost. However, aerial imagery has a top-down view. This top-down view provides a global view of the region of interest, but some important objects such as the vertical road signs are invisible from the top-down view. Besides this limitation, many road surfaces and important lane markers are often occluded by trees, buildings, or even vehicles driving on the roads. As a result, extracting digital maps from aerial imagery data is challenging, and many prior works yield limited accuracy.

Although many challenges exist, automatic mapping techniques still hold the potential to benefit many applications by providing low-cost maps that can span the globe and quickly reflect road network changes. Therefore, this thesis focuses on automated mapping techniques with aerial imagery and GPS data and proposes techniques to improve mapping accuracy and enrich digital maps. As shown in Figure 1-3, we propose RoadRunner (Figure 1-3.a and §2) and Sat2Graph (Figure 1-3.b and §3) to extract road-level street maps (the base maps) from GPS trajectory and aerial imagery (satellite imagery), respectively. Meanwhile, we propose a mapping pipeline based on aerial imagery to extract lane-level street maps that are essential for lane-to-lane

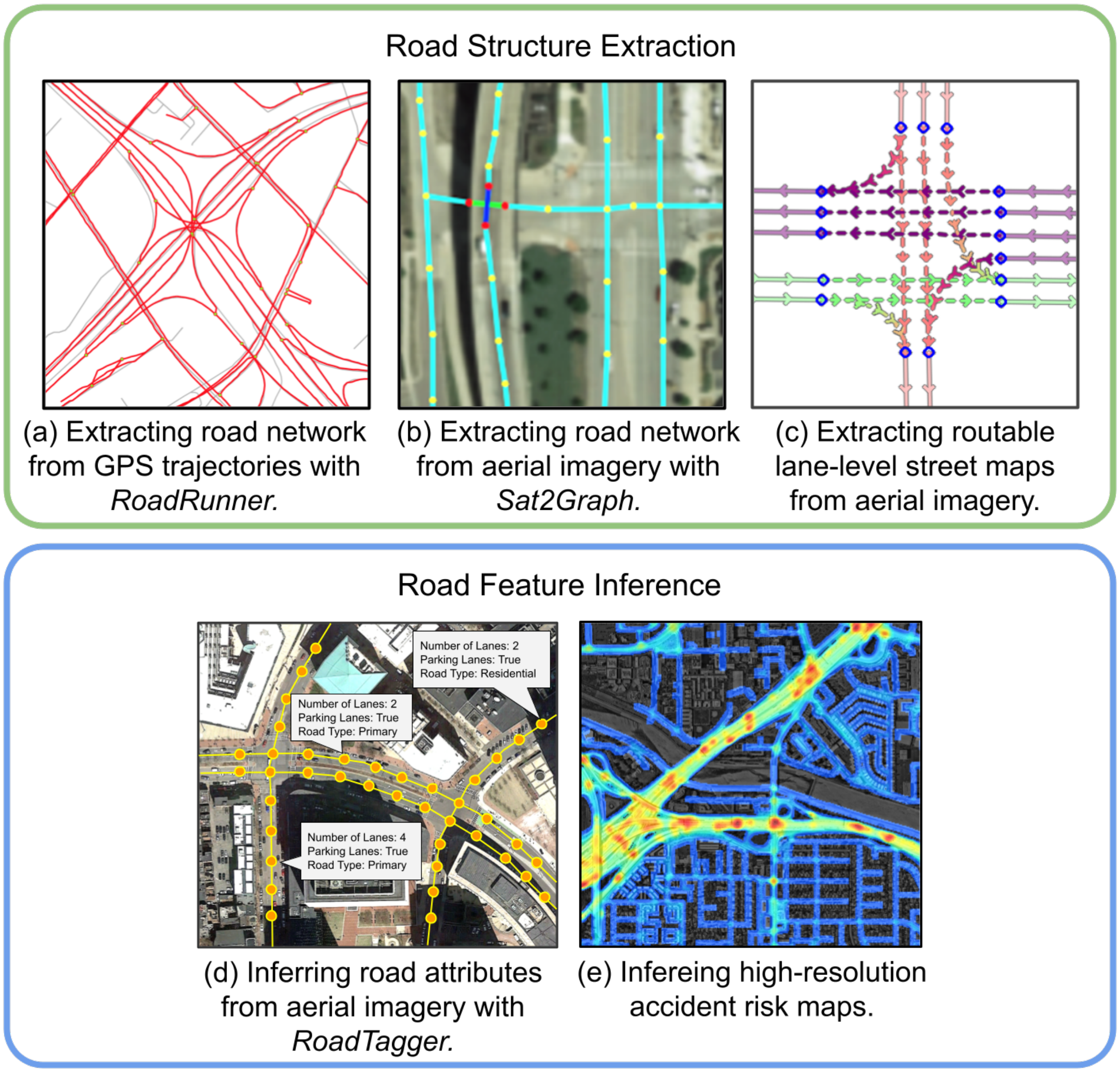

Road Structure Extraction
(a) Extracting road network from GPS trajectories with *RoadRunner.*
(b) Extracting road network from aerial imagery with *Sat2Graph.*
(c) Extracting routable lane-level street maps from aerial imagery.
Road Feature Inference
Number of Lanes: 2
Parking Lanes: True
Road Type: Residential
Number of Lanes: 2
Parking Lanes: True
Road Type: Primary
Number of Lanes: 4
Parking Lanes: True
Road Type: Primary
(d) Inferring road attributes from aerial imagery with *RoadTagger.*
(e) Infereing high-resolution accident risk maps.

Figure 1-3: This thesis proposes a set of automated techniques to enrich digital maps.

navigation and autonomous vehicles (Figure 1-3.c and §4). Besides road structure extraction, we propose RoadTagger (Figure 1-3.d and §5) to infer road attributes such as the lane count and road type of road segments. Finally, beyond inferring traditional digital map features, we propose a mapping solution to create high-resolution traffic accident risk maps that can enrich the semantics of existing digital maps and support new applications such as safety-aware routing and precise insurance (Figure 1-3.e and §6).

## 1.3 Technical Challenges

In the design of the above five automation techniques, we find two common technical challenges, *data modality conversion* and *geospatial reasoning*.

**Data modality conversion.** In our proposed mapping solutions, a typical mapping task takes geospatial data with one or more data modalities (e.g., trajectories, graphs, and arrays), processes the input data, and produces the map features. In this procedure, the input geospatial data and the output map features may have different data modalities. Therefore, the mapping task often needs to convert data modality, either for intermediate data processing, final result generation, or both. For example, producing a road graph from an aerial image (a 3D data array), producing a road graph from a set of GPS trajectories, producing a data array representing a road graph or a set of GPS trajectories, etc. Many of such conversions are non-trivial, and good conversion solutions are very critical to the accuracy of the automation solutions. Of course, such good conversion solutions often require novel designs.

This thesis demonstrates a few examples of such modality conversion designs. In Table 1.1, we summarize the modality conversions involved in each work.

| Solutions | Input Data Type(s) | Output Data Type |
|---|---|---|
| RoadRunner (§2) | Trajectory | Graph |
| Sat2Graph (§3) | Imagery | Graph |
| Lane Extraction (§4) | Imagery | Graph |
| RoadTagger (§5) | Imagery and Graph | Vertex Label |
| Crash Risk Map (§6) | Imagery, Graph and Trajectory | Imagery |

Table 1.1: Data modality conversions in each work.

**Geospatial reasoning.** In many of the problems we are solving, we find that the expected output result at one geo-location is often not only determined by the local information at that location but also information from its neighboring locations or even information from far away places. This property makes it challenging to design robust algorithms and neural network models because they have to exploit both local and non-local information effectively.

This thesis demonstrates a few examples of how we address this challenge. More specifically, RoadRunner (§2) demonstrates how we extract non-local information from GPS trajectories. RoadTagger (§4) demonstrates a neural network architecture that can effectively combine and process visual information scattering in a wide area.

# Chapter 2

# Road Graph Extraction from GPS Trajectories

The availability of accurate and up-to-date road maps is critical for many applications, including GPS-based navigation services, disaster relief efforts, and autonomous transportation. However, creating and maintaining road network maps is currently human-intensive, expensive, and slow. Thus, automating parts or all of the map creation and maintenance process holds the potential to benefit many applications by providing maps that quickly reflect road network changes. Crowdsourced GPS trajectories are a useful data source for this task. These trajectories are sequences of GPS observations collected as vehicles travel along the road network, and are available at scale nowadays from smartphones.

Inferring road network maps from GPS data has received considerable attention in recent years [19, 71, 5, 42]. Broadly, existing inference schemes infer a high-coverage but low-accuracy initial graph, and then apply refinement heuristics (e.g., graph spanners pruning [90] and $k$-means refinement [20]) to improve precision. However, we find that these methods fail to produce maps of high quality, especially in dense urban areas and in areas with complex intersections. In the two left panels of Figure 2-1, we show the results from two such approaches on selected regions of three cities.

The resulting maps suffer from three significant problems:

- They connect overpasses with underpasses in a planar graph, despite the un-

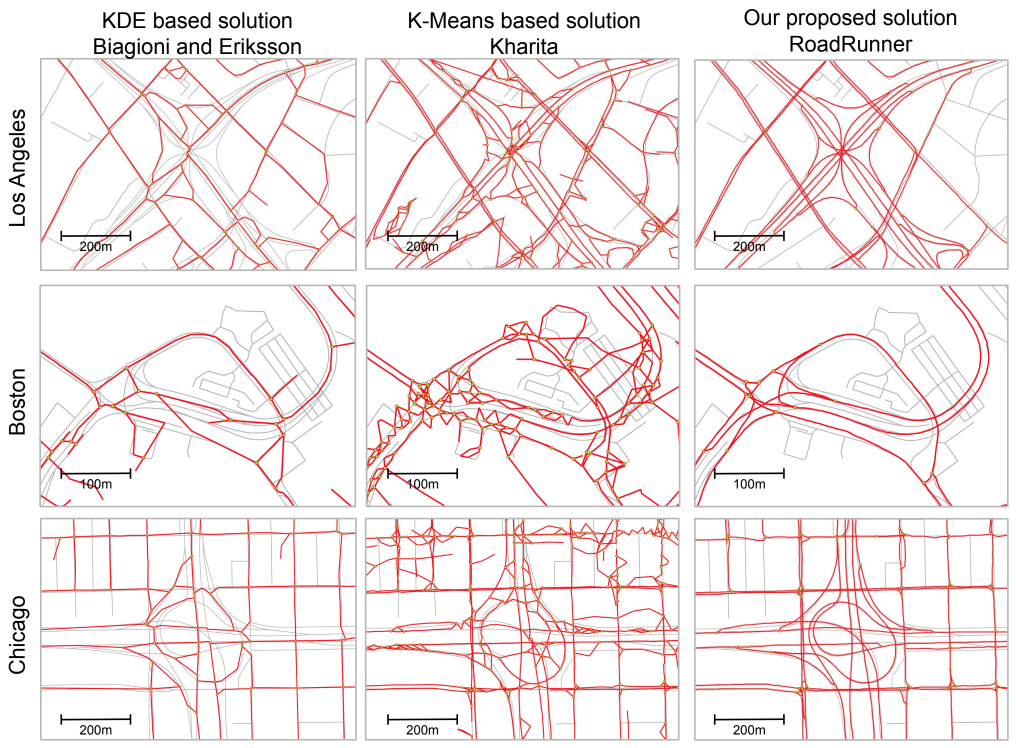


Figure 2-1: Motivation. GPS-trajectory based mapmaking algorithms running on complex interchanges in major cities. Red lines show algorithm output overlaid on underlying ground truth map in gray. Two existing algorithms are on left, our proposed RoadRunner algorithm is on right. Both existing approaches connect parallel segments on highways, often with many small, spurious crossing segments, and also add incorrect intersections where roads of different heights cross.

derlying techniques using heuristics to avoid these spurious connections.

- They connect adjacent roads that do not intersect.
- They fail to capture detailed topology such as highway interchanges.

Because prior schemes developed over a period of several years suffer from these problems, we believe that strategies that start with a low-accuracy initial graph and then refine it will not be able to regain high precision.

Thus, in this work, we turn the strategy on its head. We propose a two-stage architecture that focuses on *precision first*, and *recall second*. In the first stage, we infer a high-accuracy road network to accurately capture road topology in complex regions like overpasses/underpasses, stacked roads, parallel roads, multi-road intersections, and dense urban areas. Then, in the second stage, rather than developing new refinement heuristics, our approach regains recall by simply running any of several prior schemes, but taking care not to disrupt the segments and interconnections computed in the first stage. The result is much higher precision with recall similar to the second-stage scheme.

Accurately inferring road topology in complex regions is difficult even when high recall is not a concern. For example, in Figure 2-2, we cannot discern whether the two roads meet from a point cloud of GPS observations across the trajectories (where we discard the connectivity between observations defined by the trajectories). A common technique of taking a histogram over observations (*kernel density estimation* [40]), in which each cell is weighted by the number of GPS samples contained in the cell, does not reveal the underlying network topology. On the other hand, the topology becomes clear if we bring back the trajectory connectivity: because no trajectory that starts from the left road exits along the right road and vice versa, we can conclude that the two roads do not intersect.

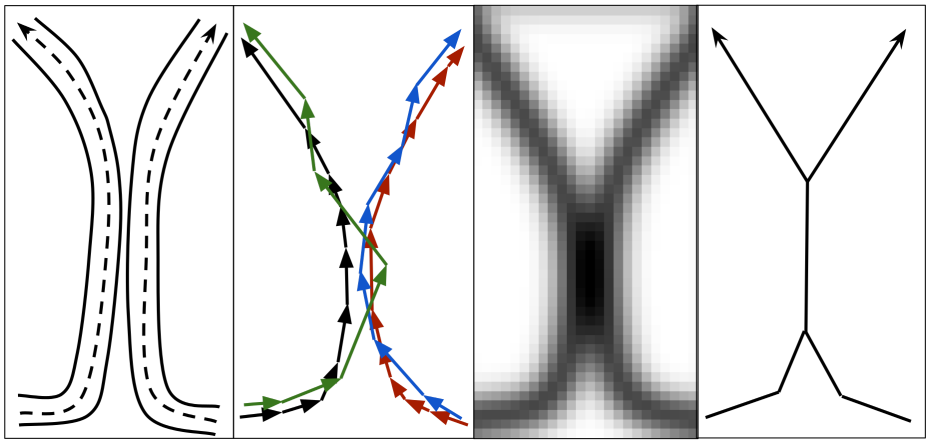


Figure 2-2: Illustration of the importance of accounting for the association between observations in a trajectory. Two roads that pass near each other but do not meet are shown on the left. In the center, we show a set of trajectories that pass these roads, along with an example histogram over GPS observations in those trajectories. We cannot tell whether the roads intersect from the histogram, although it is clear from the original trajectories. Existing approaches often produce a road network graph similar to the one on the right.

We present *RoadRunner*, a method that exploits trajectory connectivity to generate accurate maps in challenging regions where existing methods fail. RoadRunner constructs a road network graph *incrementally* by following the flow of GPS trajectories. This iterative process enables RoadRunner to consider GPS observations on each iteration not in isolation, but in the context of several predecessor and successor observations in the same trajectory. This context, which we use in a trajectory filtering algorithm, is crucial to precisely separating out nearby but distinct roads, and doing so in a way that is robust to GPS noise and complex road topologies. The right panel of Figure 2-1 exemplifies how RoadRunner produces maps with significantly higher precision than prior work.

Although our incremental procedure and filtering strategy allow RoadRunner to infer roads with high precision, they also cause RoadRunner to miss roads that are covered by few GPS trajectories, such as roads in residential areas. Fortunately, existing methods already perform well in such regions. Thus, after running RoadRunner to produce a high-precision map covering the most challenging areas of the road network, we run an existing high-recall method in the second stage. Then, we apply a merging procedure to identify segments found by the high-recall method but not by RoadRunner, and integrate them into the inferred map.

In summary, we make the following contributions:

- We propose a two-stage map inference architecture that enables us to generate high precision road network graphs without sacrificing the coverage. At the core of this architecture is RoadRunner, our high-precision first-stage method that uses the connectivity of GPS trajectories to produce accurate maps in dense urban areas and at complex intersections where current approaches perform poorly.
- We show how to integrate several prior schemes as the second stage in our two-stage architecture with a merging procedure to combine the road segments inferred by RoadRunner with those inferred by existing approaches. We apply the merging procedure to two current state-of-the-art GPS-based approaches, Biagioni-Eriksson (BE) [19] and Kharita [90], and to RoadTracer [15], which processes aerial imagery.
- We evaluate our two-stage architecture over 4 km $\times$ 4 km regions of four U.S.

cities containing 1,864 kilometers of roads on a GPS trajectory dataset of over 60,000 trajectories. We find that our proposed two-stage architecture significantly improves the quality of the inferred maps compared with the state-of-art solutions. We summarize the evaluation result in table 2.1.

| Schemes | Precision | Recall | Routing Error |
|---|---|---|---|
| BE | 84.5% | 37.2% | 76.1 meters |
| RoadRunner+BE | 89.7% | 42.8% | 24.8 meters |
| Kharita | 60.0% | 43.0% | 73.7 meters |
| RoadRunner+Kharita | 84.3% | 44.7% | 41.4 meters |

Table 2.1: Summary of the evaluation results

## 2.1 Background

The rapid adoption of GPS-enabled smartphones has led to considerable interest in the problem of automatic road network inference from GPS trajectories [20, 71, 5]. However, as we demonstrated in Figure 2-1, prior schemes yield noisy maps with low precision on complex topologies such as highway interchanges. We show that precision can be improved by exploiting the connectivity between GPS observations at different times along the same trajectory.

Broadly, road network inference schemes can be divided into three categories [20]. *k-means* approaches begin by clustering the GPS observations. Cluster centers become vertices in the inferred road network graph, and edges are added between the clusters of successive observations in each trajectory [43, 86, 4]. During clustering, the longitude, latitude, heading, and speed of each observation may be considered, but the connectivity between observations in the same trajectory is ignored. Although this connectivity is used to add edges, this stage still only considers pairs of consecutive observations rather than longer trajectory subsequences.

*Kernel density estimation (KDE)* approaches first generate a spatial histogram where each cell is weighted by the number of GPS trajectories that pass through the cell. Kernel smoothing is applied on the histogram (typically using a Gaussian distribution), followed by morphological thinning or similar skeletonization techniques to extract centerlines and produce a graph [33, 40, 88]. These techniques do not consider the connectivity between GPS observations.

*Trajectory merging* approaches merge GPS trajectories one at a time into an initially empty road network graph [27, 7]. Again, while these approaches iterate over a trajectory to merge it into the graph, the merging process operates only on pairs of successive observations at a time.

Recently, a number of methods have been proposed that combine or extend these techniques. Biagioni et al. [19] propose a hybrid pipeline using KDE with an adaptive thresholding scheme to obtain an initial road network graph, followed by geometry and topology refinement and map-matching-based pruning to further improve accuracy. Chen et al. [34] propose a supervised learning framework that leverages prior knowledge on real-world road networks to learn the shape of different junctions, and integrate this with a cluster based algorithm. Stanojevic et al. [90] develop a novel model in which map construction is framed as a network alignment problem. The derived optimization problem is then mapped into a hybrid algorithm combining $k$-means clustering and graph spanners. Zheng and Zhu [117] propose a revisited version of the trace merging method, applying a novel clustering algorithm that uses a partial curve matching method based on Fréchet distance to measure the partial similarity between any trajectory and a previously created link. While these methods improve on earlier schemes, none utilize the long-term connectivity between observations in the same trajectory when constructing the road network graph.

Other data sources, aerial imagery in particular, have also been leveraged for automatic road network inference. DeepRoadMapper [72] applies CNN-based segmentation followed by an extensive post-processing pipeline to extract a road network graph from aerial imagery. RoadTracer [15] uses an iterative graph construction strategy to obtain a graph directly from a CNN, thereby attaining higher precision that segmentation-based approaches. However, because of occlusion by tall buildings, shadows, and overpasses, accurately inferring roads from aerial imagery in dense urban areas and complex intersections is challenging—indeed, these methods assume a fully planar road network graph, and thus yield low precision in these regions.

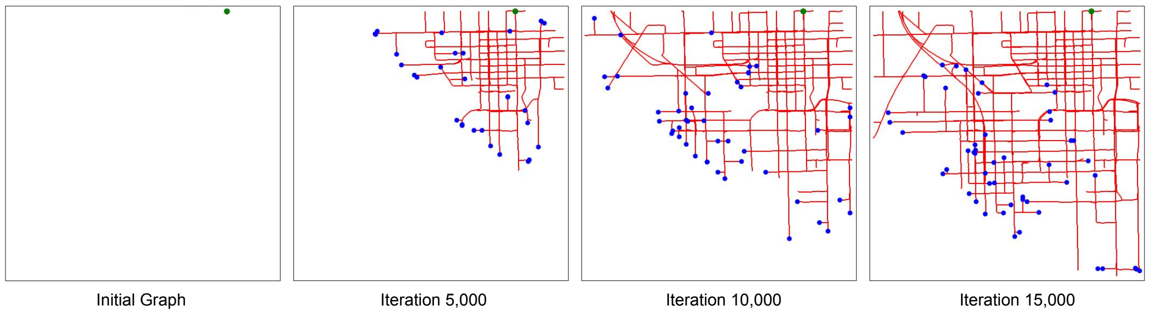


Figure 2-3: We visualize the partially constructed graph $G$ (red) and the active vertices queue $Q$ (blue) from different iterations in the construction procedure. In this example, the algorithm begins with a single vertex (green) at the upper-right corner of the region.

## 2.2 RoadRunner

RoadRunner iteratively constructs the road network from an initial graph by following the flow of GPS trajectories. We show the structure of RoadRunner in Algorithm 1. The algorithm begins with an initial graph; this may be obtained from an existing graph or a graph inferred by another approach. We first push all the vertices in this initial graph into a queue $Q$. We call the vertices in the queue *active vertices.* On each iteration, RoadRunner picks an active vertex from the queue and extends the current graph from this active vertex by performing two key operations — adding new road segments (tracing), or connecting the active vertex with an existing vertex when two physical roads join together (merging).

We show an example of how RoadRunner works in Figure 2-3. The algorithm starts with an initial graph $G$ which has only one single vertex (the green vertex near the upper-right corner). We visualize the partially constructed graph and the active vertices at different iterations. The construction procedure uses the GPS data to gradually extend the graph by following the road network, including forking at intersections and merging when two roads come together. The algorithm stops tracing a road when it reaches a dead end, leaves the region of interest, or joins with other roads. Finally, after the search queue is empty, we obtain the road network graph for the whole region.

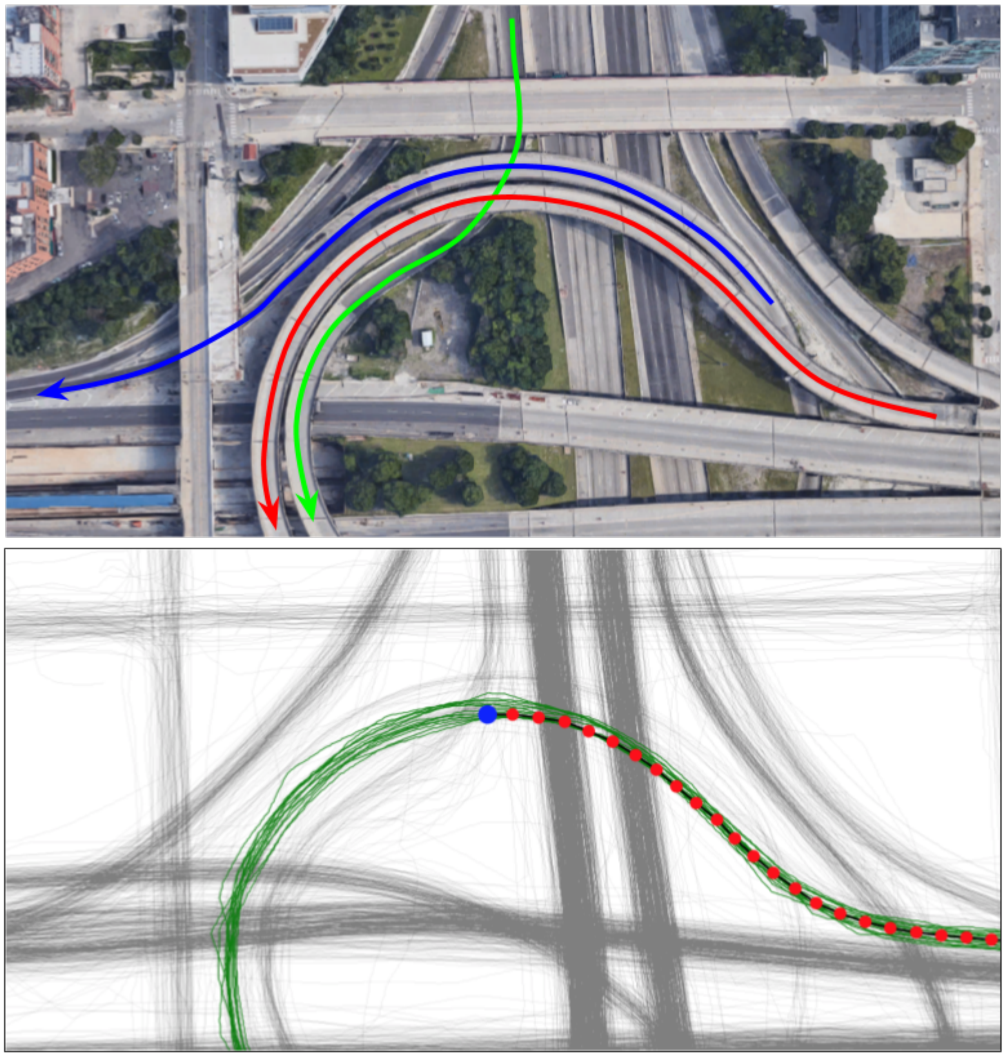


Figure 2-4: We show the GPS trajectories (grey) near a complex highway junction as well as the corresponding aerial image of the same region. The red vertices represent the partially constructed graph and the blue vertex is the active vertex from where we are going to extend the graph. We highlight the trajectories that pass near the partially constructed graph in green. In the aerial image, we highlight three roads that are challenging for map inference algorithm.

**Algorithm 1** RoadRunner

```
1: procedure RoadRunner(InitialGraph)
2:     $G \leftarrow$ InitialGraph
3:     $Q \leftarrow$ Push all vertices of $G$ into a queue
4:     while $Q$ is not empty do
5:         $v \leftarrow Q.pop()$
6:         $\mathcal{D} \leftarrow$ Trace$(G, v)$        ▷ Return the direction(s) of the GPS flow(s) around vertex $v$.
7:         for each direction $\theta$ in $\mathcal{D}$ do
8:             $loc \leftarrow v.loc + d \cdot (cos\theta, sin\theta)$
9:             $u \leftarrow G.initVertexAtLocation(loc)$
10:            $u.predecessor \leftarrow v$
11:            $G.addEdge(v, u)$
12:            $succ \leftarrow$ Merge$(G, u)$        ▷ Try to merge the vertex $u$ with the current graph $G$, return the state of merging.
13:            if $succ$ is False then
14:                $Q.push(u)$
15:    return $G$ as the inferred road network graph
```

Constructing the road network graph in an iterative manner enables RoadRunner to exploit the *long-term connectivity* between GPS observations to accurately capture road geometry and topology as it adds each segment to the graph. By contrast, most prior approaches rely on a histogram over observations [40], the position of individual or pairs of GPS observations [27], or the local heading of observations [90, 34, 42] to construct the road graph. Some prior approaches use long-term trajectory information *after* an initial road network graph has been produced. For example, BE [19] leverages the trajectory sequences in a post-processing phase to prune edges from an initial low-accuracy graph. However, fixing a low-accuracy graph is very difficult, particularly, in regions with complex road topology or high GPS noise.

At each iteration, RoadRunner compares trajectory sequences with the partially constructed graph to filter trajectories that are not related to the road of interest. Consider Figure 2-4, where we are extending the graph from the blue vertex corresponding to a highway ramp. Since the three highlighted roads are close to each other and have almost the same heading, if we take GPS observations from all of the trajectories into account, we may connect the underpass and overpass of the green road and the red road, or merge the red road with the blue road. However, by excluding trajectories that do not pass near a sequence of edges in the partially constructed graph corresponding to the current road, we obtain a much cleaner subset of GPS trajectories that covers only the red road. Our iterative graph construction architecture enables us to construct such a subset of trajectories on each step, and to use it to accurately capture the road network.

To realize this idea, we introduce a primitive called a *way path filter*. Given a sequence of circles centered at locations along a road, the way path filter prunes the GPS trajectory data to retain only trajectories that pass through the circles in order. Thus, the circles act as waypoints that we require the trajectories to traverse.

Formally, given the trajectories $T$ and a sequence of circles $P = \langle (p_1, r_1), (p_2, r_2), \ldots, (p_n, r_n) \rangle$, where the $i$th circle is centered at $p_i$ and has radius $r_i$, the way path filter is a function $filter(T, P)$ that computes a subset of $T$ containing trajectories that pass through the circles in $P$ sequentially. A trajectory $t \in T$ is in this subset if there exists a subsequence of GPS observations in the trajectory, $\langle t_{j_1}, \ldots, t_{j_n} \rangle$ such that the distance from $t_{j_i}$ to $p_i$ is at most $r_i$ for all $i$.

For a particular active vertex, we apply the way path filter to retain only trajectories that are likely to have traversed the road corresponding to the active vertex. To do so, we compute a path in $G$ of length $k$ that terminates at the active vertex, and then apply the filter with circles drawn around vertices along this path. The radius of these circles should correspond to the width of the road at the vertex, so that the circle covers all trajectories that traverse the road. We estimate this width from the trajectory data (section 2.5.1).

The length of the path, $k$, impacts the precision of the inferred map. A larger $k$ applies a filter through a longer path, enabling more accurate tracing and merging operations. In practice, we set $k$ to be 75 meters, that is at the same scale of a typical city block. We find this $k$ is long enough for us to accurately capture the complex road structures.

Next, we discuss RoadRunner's tracing operation (section 2.3), merging operation (section 2.4), and some implementation details (section 2.5).

## 2.3 Tracing

The tracing operation extends the road network graph by adding new vertices and edges to an active vertex in the direction of the GPS trajectories that pass through that vertex. RoadRunner simultaneously follows all peak directions indicated by the trajectories. It predicts these directions using Algorithm 2.

Consider the example in Figure 2-5, where we are predicting the peak directions of GPS trajectories near the active vertex $v$ (blue). Let $P = \langle(p_0, r_0), (p_1, r_1), \ldots, (p_{k-1}, r_{k-1})\rangle$ be a sequence of circles centered at vertex $v$ and its $k-1$ predecessors in the partially constructed graph (line 2-3). In the figure, we highlight the trajectories in $filter(T, P)$. We can clearly see that the trajectories split into three groups at the intersection.

**Algorithm 2** Trace Operation

```
1: procedure TRACE(Graph $G$, Vertex $v$)
2:     $\mathbf{u} = \langle u_i \rangle \leftarrow$ the first $k-1$ predecessors of $v$.
3:     $\mathbf{P} \leftarrow$ the circle sequence of path $\mathbf{u} + \langle v \rangle$
4:     $N, Score \leftarrow$ arrays indexed by different angles.
5:     for each $\theta$ in evenly spaced angles from 0 to $2\pi$ do
6:         $w \leftarrow$ circle $(v.loc + D \cdot (cos\theta, sin\theta), r)$
7:         $T' \leftarrow filter(T, \mathbf{P} + \langle w \rangle)$
8:         $N[\theta] \leftarrow min(0, |T'| - M)$
9:     $Score \leftarrow smooth(N)$
10:    return angles of local peaks of $Score[\theta]$
```

To predict the directions of these outgoing groups of trajectories, we check 72 evenly spaced angles from 0 to $2\pi$. For each direction $\theta$, we create a circle $w = (v + D(cos\theta, sin\theta), r)$ located at a distance $D$ from $v$ in the direction $\theta$. We count the number of GPS trajectories that pass near the path $\mathbf{P} + \langle w \rangle$ using the way path filter (line 7). We subtract the counted number by $M$. The parameter $M$ helps control the precision of constructed maps by excluding low confidence (low density) GPS flows.

We define the score function as the smoothed version of this counter over different angles. In the *smooth* function, we convolve the input array with a Gaussian kernel. Smoothening helps remove small changes in the score function and lets us focus on only major peaks. We extract the directions of all outgoing GPS flows by identifying

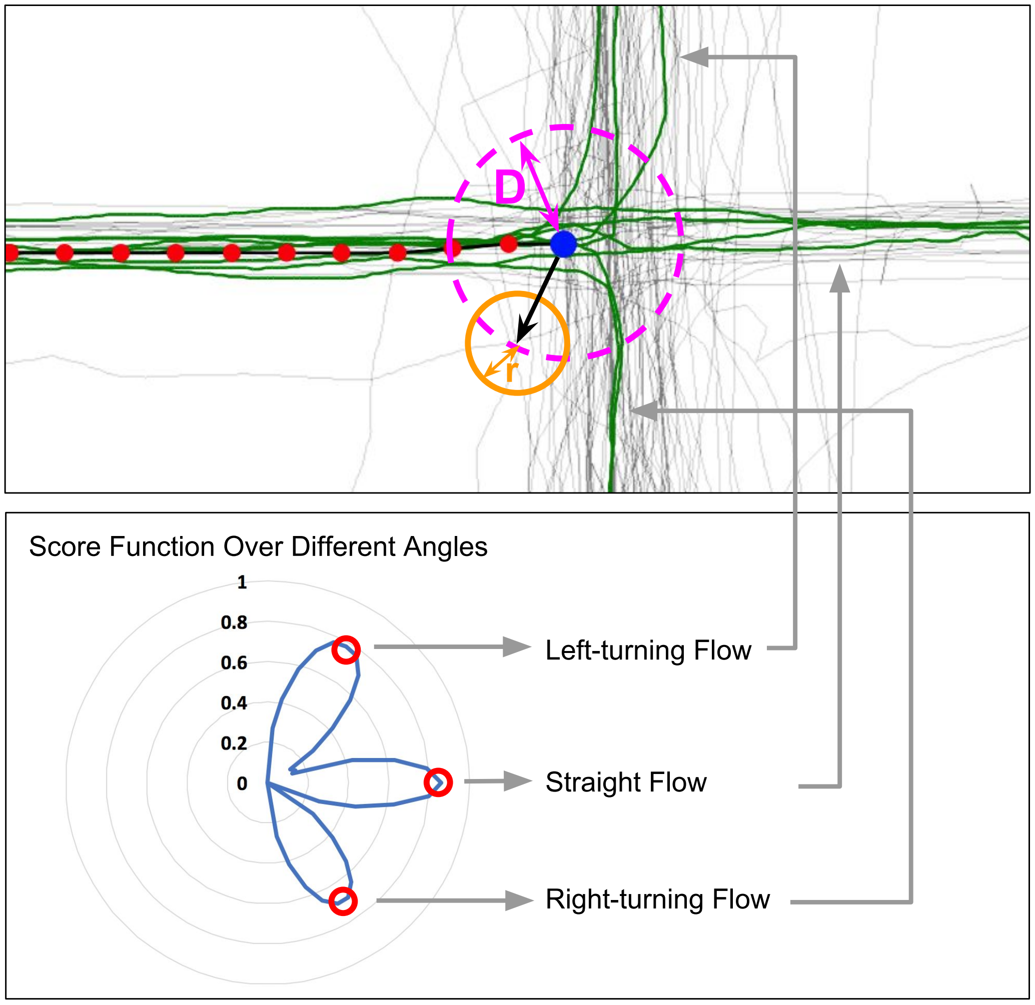


Figure 2-5: We show an example of the tracing operation near a four-way intersection. In the above figure, the red vertices are the partially constructed graph and the blue vertex is the active vertex. We highlight the GPS trajectories that pass near this partially constructed graph in green. The green GPS flow splits into three groups at the intersection. We predict the directions of these outgoing GPS flows through a score function. Here, we can clearly see three local peaks in the score function over different angles.

local peaks of this score function. We show an example of this score function in Figure 2-5. There are three local peaks in the score function, corresponding to the left-turning flow, the straight flow, and the right-turning flow. After we get the directions of the outgoing flows, we extend the graph $G$ by adding new vertices with a small fixed distance from $v$ toward each direction.

## 2.4 Merging

When the construction procedure encounters a road segment that has already been explored in the graph $G$, we need to merge the current road with the previous path. However, merging is challenging: we need to merge roads that connect while ensuring that overpasses/underpasses, parallel roads, and multilayer roads remain separated. Correctly capturing connectivity is crucial because even a small number of incorrect connections lead to a large number of navigation errors.

Existing approaches [90, 19] aggressively merge road segments, considering only local information such as distance and heading. This yields numerous incorrect connections in challenging regions like dense urban areas and highway intersections. We find that deciding whether we should merge two road segments with only local information such as distance and heading is not enough. As the example shown in Figure 2-6, a universal merging threshold may not exist if we only consider the local information.

To overcome this challenge, we introduce a novel merging criteria that can accurately decide whether road segments should merge or not. When two road segments are close to each other, instead of only considering the local information of these two road segments such as distance and heading, we look at the GPS trajectories that pass through these two road segments. We merge the road segments if and only if the distributions of these two groups of GPS trajectories match in the future (after traveling the two road segments).

We show the distributions of GPS trajectories that pass after the blue and green road segments in Figure 2-6. We can clearly see that the two distributions of example

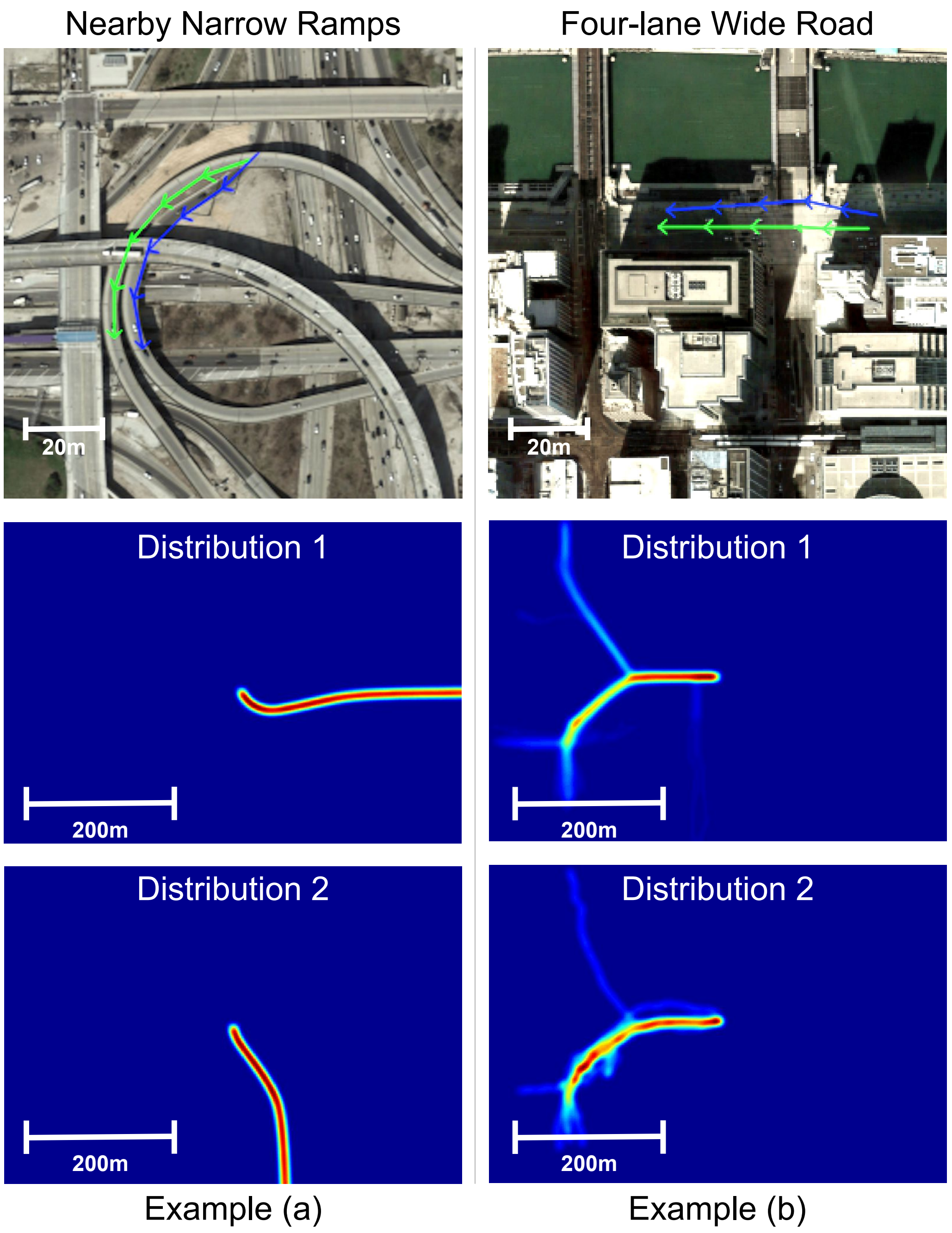


Figure 2-6: Suppose we are considering to merge the green road and the blue road in the above two examples. Simply considering the distance between the two roads cannot yield valid results. In fact, the distance between the two roads in example (b) is larger than the distance in example (a). However, we need to merge the two roads in example (b) rather than the two roads in example (a). Below the satellite images, we show the distribution of GPS trajectories after they pass through the blue road segments (distribution 1) and the green road segments (distribution 2).

(a) disagree with each other, while the two distributions are highly similar in example (b). This design enables us to make correct decisions on whether the two roads should be merged or not in a very general way, supporting a wide range of road types. Again, we use the partially constructed graph as a trajectory filter to generate the

**Algorithm 3** Merge Operation

```
1: procedure MERGE(Graph $G$, Vertex $v$)
2:     $D_v \leftarrow GenDistribution(G, v)$
3:     for each vertex $u$ near $v$ do
4:         $D_u \leftarrow GenDistribution(G, u)$
5:         if $D_u$ is similar with $D_v$ then
6:             $G.addEdge(u, v)$
7:             return $True$
8:     return $False$
9: procedure GENDISTRIBUTION(Graph $G$, Vertex $v$)
10:    $\mathbf{P_v} \leftarrow$ the path formed by $v$'s predecessors
11:    $T_v \leftarrow filter(T, \mathbf{P_v})$
12:    $T'_v \leftarrow$ for each $t$ in $T_v$, we only keep a portion of it; this portion starts near $v$
       and continues for a fixed distance, e.g., 300 meters.
13:    return the spatial distribution of $T'_v$
```

two distributions. The details of the merging algorithm are shown in Algorithm 3.

## 2.5 Implementation Details

In this section, we discuss the implementation details of RoadRunner, including the vertex radius estimation, tracing acute branches and the parameter settings.

### 2.5.1 Vertex Radius Estimation

Each vertex in the road network graph has a *radius* attribute. Recall the *way path filter*, we filter GPS trajectories based on this radius attribute. Instead of using a fixed radius for all vertices, we estimate this radius for each vertex during the graph construction process. The basic idea is to estimate the width of the road at vertex $v$ from the GPS trajectories and use that as the radius.

Specifically, we consider the vertex sequence from $v$'s $2k$th predecessor to its $k$th predecessor. Then, we apply the way path filter on this vertex sequence. We look at the distribution of the filtered trajectories along the perpendicular direction of the edge $v.predecessor \rightarrow v$. Here, we assume this distribution follows Gaussian mixture model, where the component closest to the center line corresponds to the road through $v.predecessor \rightarrow v$, while other components capture nearby roads that may have just forked from the main road. We estimate the number of the mixture components as well as the mean and standard deviation of each component. Then we use the standard deviation of the component located closest to the center line as the radius of vertex $v$. Note that this estimation algorithm depends on the radius of $v$'s predecessors. Thus, we set the radius of all vertices in the initial graph to a fixed initial value, e.g., 5 meters.

This dynamic radius estimation solution enables us to handle a wide range of road types, i.e., highway and residential roads, and different GPS noise levels.

### 2.5.2 Tracing Acute Branches

The tracing algorithm in section (2.3) may fail to trace all branches when the angle between two branches are very acute. This often happens on highways. When there is a ramp coming out from the highway, the large traffic volume disparity together with the very acute angle of the branches make it hard to identify the GPS flow through the ramp. We solve this problem by re-using the Gaussian mixture components estimated in the vertex radius estimation (Section 2.5.1). In the road width estimation, each mixture component, located along the perpendicular direction of the current road, represents a nearby road that just forked from the current road. We check all these nearby roads. If they are not covered by the road network graph, we add them to the graph and push their last vertices into the active vertices queue.

### 2.5.3 Parameter Setting

RoadRunner involves several configurable parameters. These parameters are important to the quality of inferred maps. Here, we explain our choices of three major parameters. Each time we add an edge to the graph, we set the length of this edge (step size) to be 7.5 meters ($d = 7.5$). We find this number is a good trade-off between the algorithm's running time and the map coverage; a larger step size may speed up the running time but it may jump over some intersections. We set the history length $k$ to be 75 meters, as we find this is long enough for us to distinguish the most challenging road structures. In fact, in our experiments, we find that a larger k yields a higher precision but a slightly lower recall. After the k reaches 75 meters the precision stops improving. Finally, we set the default minimum number of GPS trips $M$ in the scoring function to 2. This parameter decides the minimal number of GPS trajectories required for each inferred road segment. A larger $M$ yields higher precision but lower coverage.

## 2.6 Two-stage Map Inference

As shown in Figure 2-1, RoadRunner can capture complex portions of the road network with high accuracy. However, we find that RoadRunner may miss roads in regions that are covered by very few GPS trajectories, such as residential areas. The reason is that RoadRunner aggressively filters trajectories that don't conform to the structure of the currently explored road at each step of the search; this strategy enables very high precision but misses lightly covered roads. Fortunately, these are the very regions where current state-of-the-art inference algorithms excel.

Thus, we develop a merging procedure to get the best of both worlds. We obtain a highly accurate road network graph covering noisy and topologically-complex areas from RoadRunner, and then merge segments inferred by an existing approach that correspond to new roads.

Let $G_1$ be a road network graph inferred by RoadRunner, and $G_2$ be one inferred by an existing scheme that captures sparsely covered roads. We prune edges and portions of edges in $G_2$ that lie within $R_{\text{merge}}$ meters of $G_1$ to obtain $G'_2$; this eliminates segments corresponding to roads that RoadRunner has already captured. We then compute $G$ as the union of $G_1$ and $G'_2$. However, roads added from $G'_2$ will be disconnected from the rest of the road network. To add back connections, we iterate through dead-end vertices in $G$, i.e., vertices with exactly one incident edge. We merge a dead-end vertex $v$ with an edge $(u, w)$ if both of the following hold:

- The distance from $v$ to $(u, w)$ is less than $R_{\text{merge}}$.
- $|filter(T, \langle v, p, u \rangle)|$ or $|filter(T, \langle v, p, w \rangle)|$ exceed a threshold, where $p$ is obtained by projecting $v$ onto the line segment $(u, w)$.

Together, these conditions prevent the introduction of spurious connections.

In Figure 2-7, we show an example of the map inferred by this two-stage map inference algorithm. We merge the output map of RoadRunner with the output map of a KDE based map inference scheme[20]. As shown in Figure 2-7, RoadRunner accurately captures the complex road structures in challenge regions such as highway junctions and parallel roads, whereas the KDE scheme fills up the missing roads where the GPS data is sparse. These combination yields a map with both good coverage and very high precision.

## 2.7 Evaluation

In this section, we compare our two-stage map inference scheme with RoadRunner against two GPS trajectory map inference algorithms, BE [19] and Kharita [90], and one aerial imagery inference approach, RoadTracer [15], on 16 sq km regions of four cities. BE applies kernel density estimation followed by several refinement steps. Kharita uses graph spanners to prune redundant edges from an initial graph constructed with k-means clustering. RoadTracer infers roads from aerial imagery with a computer vision approach.

### 2.7.1 Dataset

We evaluate the approaches on a large dataset of over 60 thousand GPS trajectories spanning 4km by 4km regions at the centers of four cities: Los Angeles, Boston, Chicago, and New York City. These regions contain diverse road structures, from complex highway interchanges to small residential roads, and a wide range of GPS

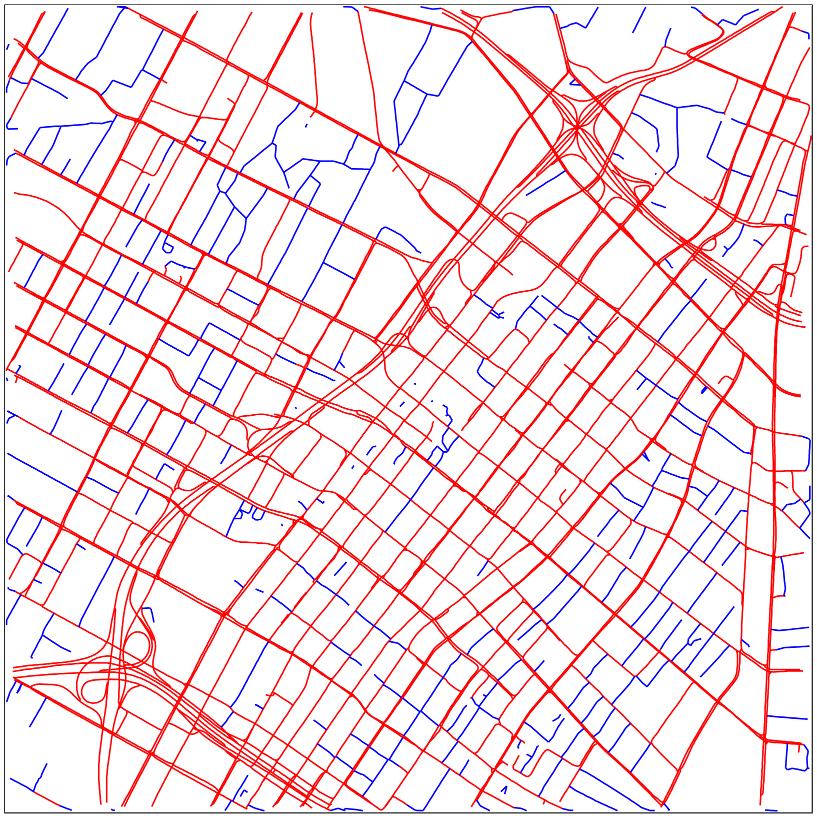


Figure 2-7: The merge result of RoadRunner (in red color) and an KDE based solution (in blue color) in Los Angeles

noise, from near-perfect GPS accuracy in open areas to heavy noise in the downtown core. The sampling interval of our GPS data is 1 second. A summary of this dataset is shown in table 2.2.

| City | Number of Trips | Number of GPS samples |
|---|---|---|
| Los Angeles | 8,422 | 2,582,195 |
| Boston | 30,422 | 7,805,400 |
| Chicago | 9,940 | 4,832,594 |
| New York City | 13,729 | 6,151,150 |

Table 2.2: GPS Dataset

We use OpenStreetMap [50] as the ground truth map for all evaluations.

### 2.7.2 Metrics

We evaluate inferred maps on two metrics: TOPO [20], which is commonly used in related work, and a shortest-path metric, which we develop by combining ideas from several existing path-comparison-based metrics [5, 98].

#### TOPO Metric

TOPO captures both geometry and topology differences between two maps. We first drop "seeds" at 50-meter intervals on every road in the ground truth map. For each seed, we try to find a corresponding point in the inferred map with similar distance and orientation (i.e., the angles of the edges that the seed and point fall on). If there exists such a point, we say the seed is valid.

For each valid seed, we drop "holes" every 5 meters on the edges of the ground truth map that can be reached within 300 meters from the seed. We then drop "marbles" in the same way from the nearest corresponding point in the inferred map. Then, we compute the maximum one-to-one matching between the marbles and holes. A marble and a hole can be matched only if the distance between them is smaller than 15 meters and the difference between the orientations of the edges they belong to is smaller than 45 degrees. From this matching, we compute a precision and recall for the seed:

$$precision = \frac{\#\ of\ matched\ marbles}{\#\ of\ all\ marbles} \quad recall = \frac{\#\ of\ matched\ holes}{\#\ of\ all\ holes}$$

We compute the overall precision and recall in a region as the average of the precision and recall from all seeds in this region. In the computation of the overall precision, we ignore the invalid seeds in the region. In contrast, we consider the recall of an invalid seed to be 0 in the computation of the overall recall of a region. This yields a fair comparison when two inferred maps have different numbers of valid seeds.

**Shortest-Path Metric**

We propose a shortest-path metric to evaluate the correctness of navigation routes in the inferred map. In each city, we randomly sample 10,000 origin-destination pairs using the GPS trajectory data; each origin and destination is sampled uniformly from the origins and destinations of the trajectories. Then, for each sampled pair, we find the closest points to the origin and the destination in the inferred map. We compute the shortest path $P_{inferred}$ in the inferred map between these two points.

To evaluate the correctness of this path, we find the path $P_{GT}$ in the ground truth map that minimizes the Fréchet distance [8] between $P_{inferred}$ and $P_{GT}$. If $P_{inferred}$ uses an invalid road or connection (e.g., the path jumps to a highway road from another non-connected road), the corresponding portion of $P_{GT}$ may involve a long detour, yielding a large Fréchet distance between the two paths.

To aggregate results across all origin-destination pairs, we define the routing error as the median Fréchet distance.

### 2.7.3 Parameters

For all the schemes we evaluated, we fix most of their parameters except one tuning parameter, which we vary to explore the trade-off between precision and recall (Table 2.3). We explain the choice of this parameter for each different scheme below.

**RoadRunner (RR).** We vary the threshold of the minimum trajectory number $M$ in the scoring function (recall that we only add a new edge toward angle $\theta$ if there are at least $M$ trajectories on this direction). Increasing this threshold makes RoadRunner more cautious when adding new roads.

**Biagioni and Eriksson (BE).** During map-matching-based pruning, roads with fewer than $L_{mm}$ matched GPS trajectories are removed from the road network graph. We vary $L_{mm}$. When $L_{mm}$ is higher, more roads are pruned. The authors set $L_{mm} = 2$.

**Kharita.** We vary the initial seed radius for $k$-means clustering. This radius corresponds to the maximum degree of GPS noise that Kharita can handle. A larger radius increases precision by merging noisy GPS observations with other observations

| Scheme | Parameter Notation | Parameter Set |
|---|---|---|
| BE | Minimum GPS traversals | 2, 5, 10, 30, 50 |
| Kharita | Initial seed radius | 20, 50, 75, 100 |
| RoadRunner | Minimum GPS traversals | 2,3,4,6,10 |

Table 2.3: The parameter set we used for different schemes

from the road, but may also merge observations from two nearby roads. The authors propose two seed radiuses, 20m and 75m.

In the following sections, we denote particular parameter settings by suffixing the approach name with this setting. For example, in BE-2, we set the minimum number of matched trajectories $L_{mm} = 2$.

### 2.7.4 Geometry and Topology Correctness

In this section, we focus on the geometry and topology correctness of the inferred map. We evaluate BE, Kharita and RoadRunner schemes as well as the RoadRunner-based two-stage schemes with TOPO metric. The evaluation covers all the GPS data in our dataset (approximately 4km by 4km for each city.)

In the two-stage map inference scheme, we first use RoadRunner (Section 2.2) to generate the high precision map, covering the most challenging areas. Then, we use the algorithm introduced in Section 2.6 to merge this high precision map with maps generated by BE or Kharita.

**Error-Rate Frontier of Different Schemes.** To obtain a comprehensive understanding of the potential performance of all the schemes, we evaluated the schemes with a wide range of parameters. The parameters we used for all the schemes are shown in Table 2.3. For two-stage schemes, we evaluate them with all the pairs of parameter combinations.

In Figure 2-8, we show the error-rate frontier of different schemes. The error-rate frontier of a scheme is generated from the output maps with all its different parameter configurations.

We would like to use this error-rate frontier to demonstrate the best achievable performance of different algorithms. More specifically, we can compare the error

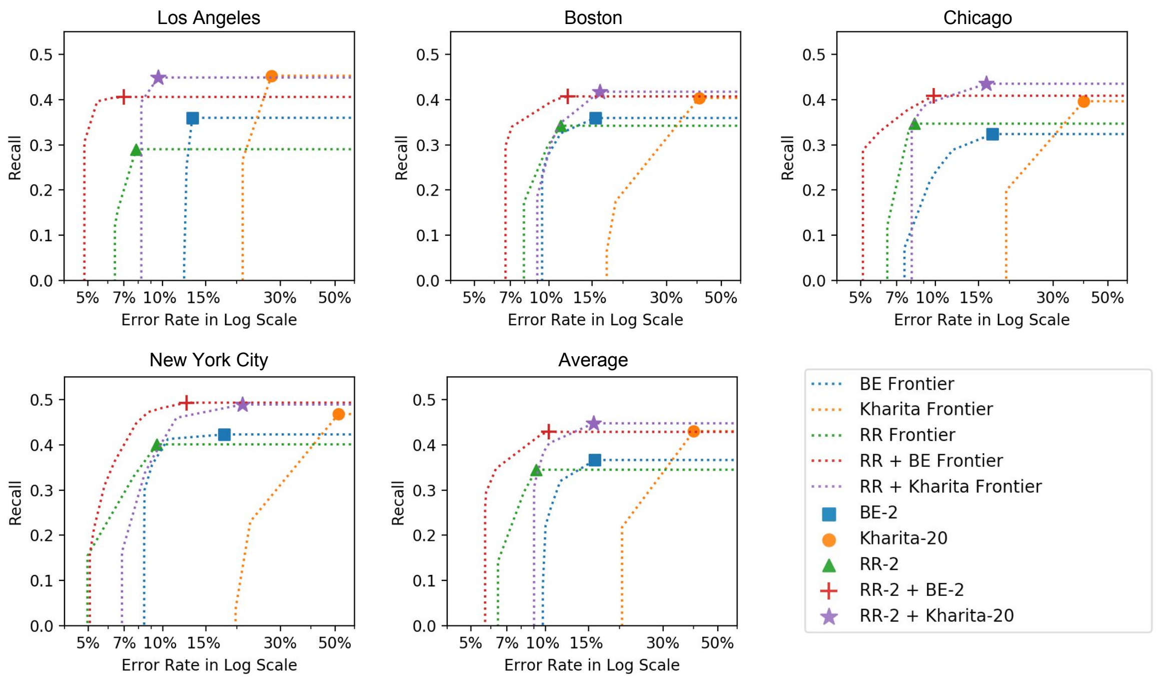


Figure 2-8: The error-rate frontier of RR, BE, Kharita schemes and the two-stage schemes, RR plus BE and RR plus Kharita. The markers show the point on the frontier corresponding to the specific parameters that produce the best average $F_1$ scores.

rates of different schemes conditioned on recall by looking at the cross points of the schemes' error-rate frontiers and a horizontal line corresponding to a certain recall.

As shown in Figure 2-8, tuning the parameters of the existing map inference algorithms can yield improvements in error rate. However, due to the inherent limitations of these algorithms, the improvement in the error rate often comes at the expense of a sharp decrease in recall. The best achievable error rate is also limited.

By contrast, as a standalone scheme, RoadRunner can achieve a significantly lower error-rate compared with other schemes in all four cities. However, since RoadRunner sometimes fails to infer roads when there are not enough GPS trajectories, its highest achievable recall is lower compared to other schemes. As we discussed in §2.6, our merging procedure is intended to yield a two-stage solution that achieves both high precision and high recall. Indeed, Figure 2-8, shows that the RoadRunner-based two-stage solutions significantly reduce the error rate with no impact on recall. For example, the error-rate frontier of the hybrid of RoadRunner and BE completely surpasses the frontier of BE scheme alone in each of the four cities.

These results show that when merging the RoadRunner scheme with an existing map inference algorithm, the merging procedure effectively replaces the high error-rate regions of the output of the inference algorithm with RoadRunner's accurate map. Meanwhile, the roads missed by RoadRunner but found by the map inference algorithm are mostly retained in the final inferred map.

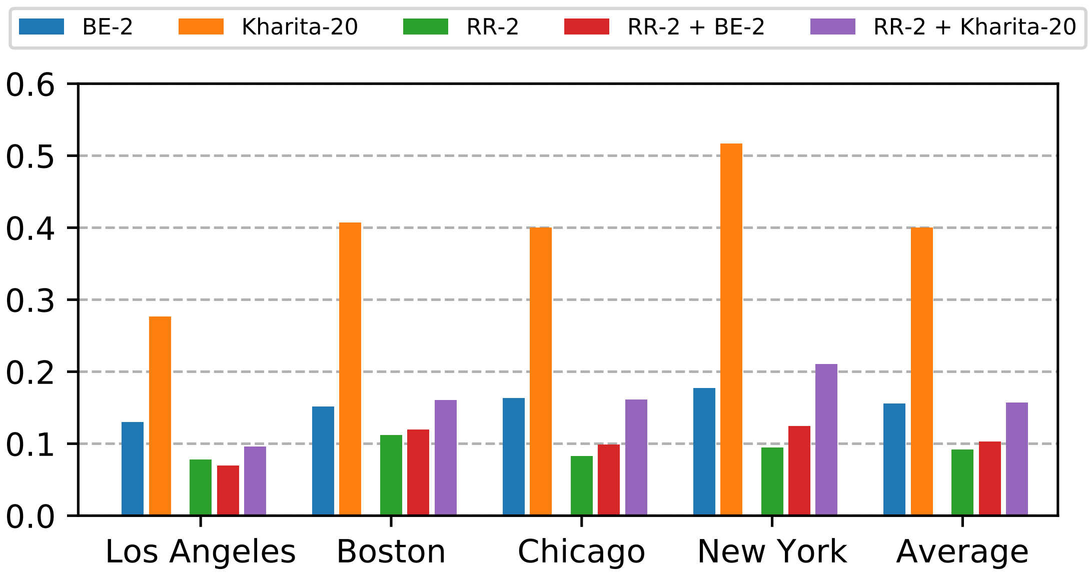


Figure 2-9: The TOPO error rates of different schemes with their best parameter settings.

**Comparison with the best parameter settings.** We take the parameter setting that can produce the best average $F_1$ score for each scheme as their best parameter setting. We compare our two-stage schemes against BE and Kharita with the best parameter settings. As shown in Figure 2-9 and Figure 2-10, the two-stage scheme significantly reduces the error rates by 5.2 points(33.6%) versus BE, and 24.3 points (60.7%) versus Kharita on average over the four cities, with a slight improvement in recall.

Note that recall here is limited because the ground truth OpenStreetMap road network includes alleys, service roads, and other minor roads that are not covered by

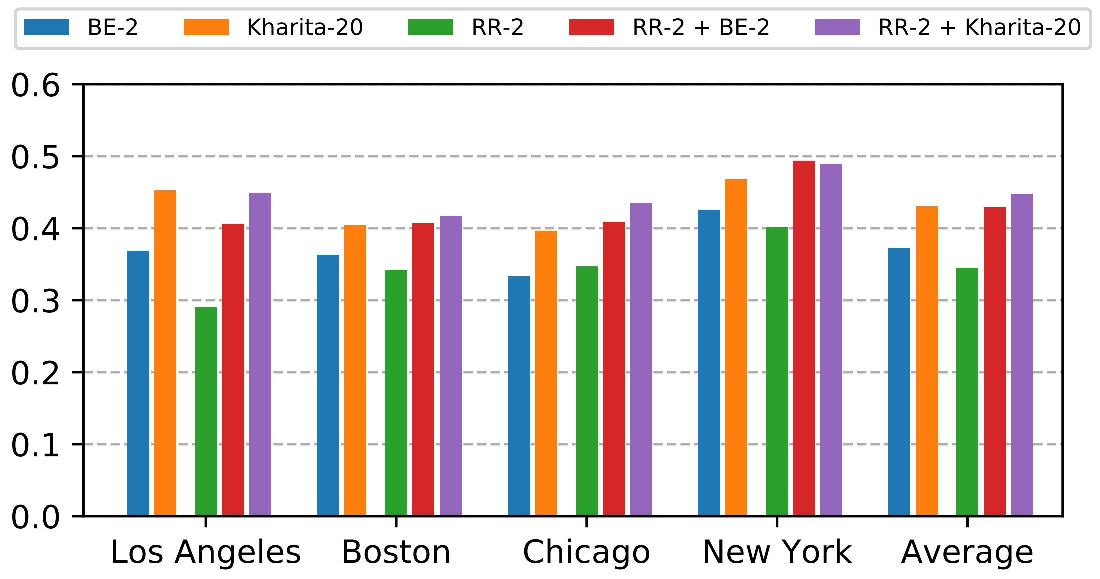


Figure 2-10: The TOPO recalls of different schemes with their best parameter settings.

our GPS trajectory dataset.

**High Error Rate Areas in Existing Schemes** We study the distribution of high error rate areas in the existing schemes. For each city, we split the whole evaluation region $A$ into 100-meter by 100-meter grids. We ignore grids that include no road inside them in the ground-truth map. For each grid $G$, we compute the error rate ($1 -$ precision) of the inferred map in $G$ using the TOPO metric. We visualize the grids with high (top-5, top-10 and top-20) error rates in Figure 2-11.

The distribution of the high error rate grids demonstrate the areas where existing approaches failed. As shown in Figure 2-11, most of the high error rate grids consist of highway interchanges, overpasses and underpasses, complex road structures or downtown areas where GPS is noisy; these places are exactly the target scenarios of RoadRunner's high precision design.

### 2.7.5 Usability in Navigation Scenario

We evaluate the navigation usability of the inferred maps from RoadRunner, BE, Kharita and the RoadRunner based two-stage scheme. As shown in Figure 2-12,

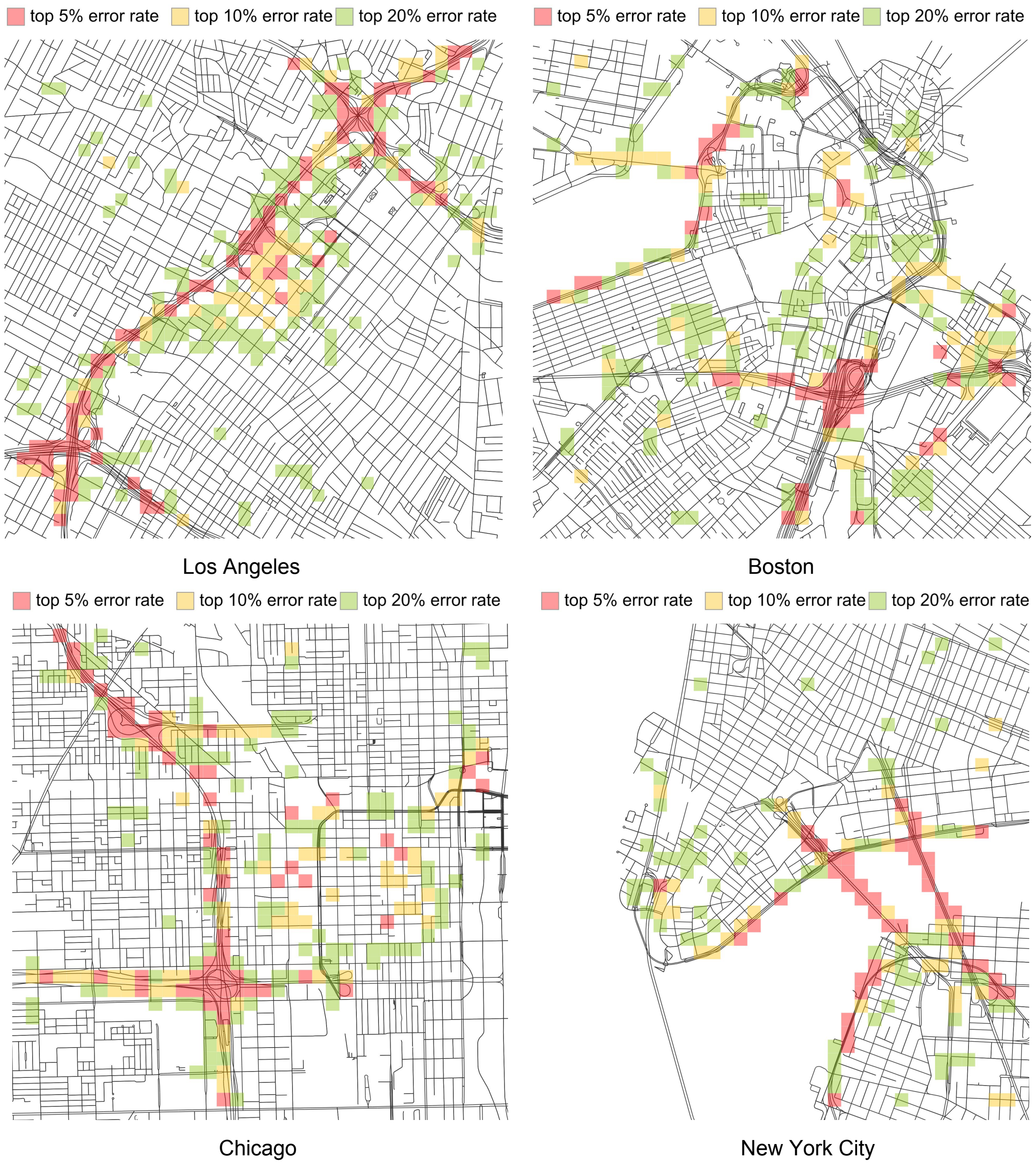


Figure 2-11: The distribution of the areas where existing schemes failed to infer the road network correctly.

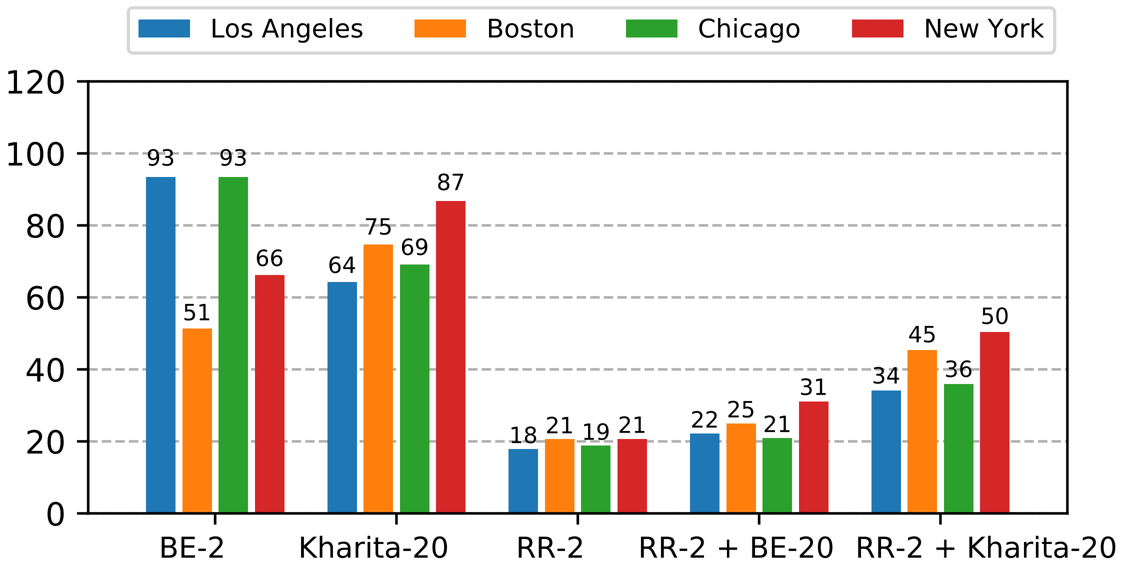


Figure 2-12: The median of the minimal Fréchet Distances in the shortest-path metric. The unit of the y-axis is in meter.

our RoadRunner scheme significantly reduces the routing error by 3.9x versus BE, and 3.8x versus Kharita. The two-stage scheme also yields a significant reduction in the routing error, by 3.1x versus BE and 1.8x versus Kharita, even with a slight improvement on the map coverage (from TOPO metric).

These results show that RoadRunner captures the major road network very accurately. The two-stage scheme inherits this major road network from RoadRunner, yielding higher confidence routes in navigation scenario. The results also imply the importance of getting road network correctly in challenging areas; this areas often have a large volume of traffic and play an important role in a city's road network.

### 2.7.6 Integration with Computer Vision-based Solution

We also study the impact of RoadRunner when combined with a computer-vision-based map inference solution that uses satellite imagery to infer the location of roads. Specifically, we compare the overall quality of the output map from RoadTracer[15] with a RoadRunner-enhanced two-stage solution. Because RoadTracer is also an incremental algorithm, in the two-stage solution, instead of merging two maps into

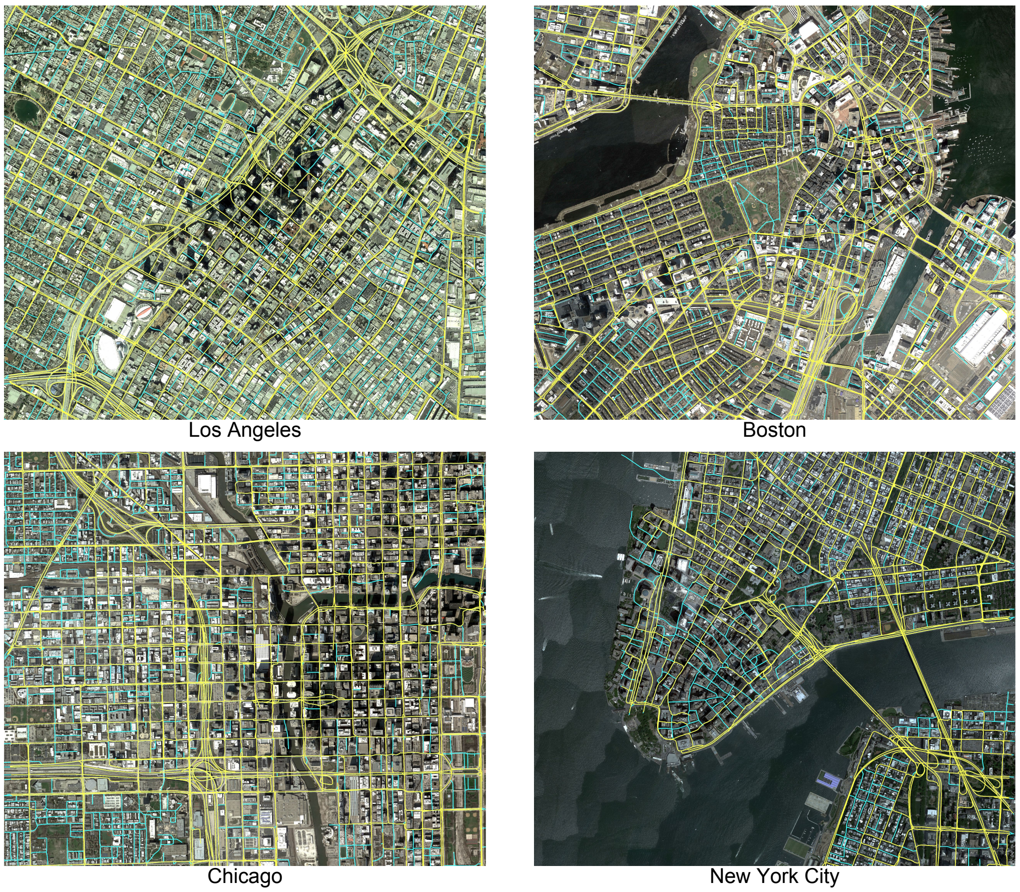


Figure 2-13: The generated map of the two-stage scheme with RoadRunner RR-2 (yellow) and RoadTracer (cyan).

one, we first use RoadRunner to generate a high precision map; then we use this map as the initial input to RoadTracer, which fills in missing roads in the RoadRunner output based on satellite imagery.

We use $F_1$ score of TOPO metric to quantify the overall quality of the output maps. We show the comparison results in Table 2.4. For the overall $F_1$ score, our two-stage solution achieves an improvement of 16.43% on average over the four cities against the satellite imagery alone solution.

We show the generated map from our two-stage solution in Figure 2-13. We find this combination of RoadRunner and imagery based solutions enables us to produce maps with much higher quality than prior solutions: the GPS based solution RoadRunner accurately captures the road structures in challenge areas where existing computer-vision based solutions fail, on the other hand, the imagery based solutions fill up the missing roads in areas where GPS data is very sparse or not available. We envision that this combination is the right way to take toward fully automated map generation systems.

| City | RoadTracer | RR-2 + RoadTracer | Gain |
|---|---|---|---|
| Los Angeles | 0.634 | 0.673 | 6.17% |
| Boston | 0.540 | 0.608 | 12.68% |
| Chicago | 0.585 | 0.653 | 11.56% |
| New York City | 0.497 | 0.673 | 35.33% |

Table 2.4: $F_1$ score of RoadTracer and RR-2 + RoadTracer

## 2.8 Conclusion

In this work, we proposed a two-stage map inference architecture that enables us to generate high precision road network graphs without sacrificing coverage. As the core of this architecture, we presented RoadRunner, an automatic road network inference method that leverages the long-term structure of GPS trajectories to generate accurate maps in the dense urban areas and complex intersections where existing methods fail. We evaluated RoadRunner together with two state-of-the-art GPS-based methods and a recent computer vision-based solution on 64 $km^2$ from four U.S. cities. Compared with the existing map-making methods, our RoadRunner based two-stage scheme yields an improvement in the overall quality of the inferred map by 15.29%[1] on average in TOPO $F_1 score$. This quality improvement represents up to a 60.7% error rate reduction of road segments and up to a 3.1x reduction of routing errors in navigation scenario, with a small increase in recall.

Insufficient precision of current automated map-making solutions is perhaps the biggest obstacle that prevents them from real-world deployment. We believe that our approach, which significantly improves the precision of automatically generated maps while not reducing recall, marks an important step towards automating the process of road map generation.

[1]12.17% for RR+BE, 17.28% for RR+Kharita, 16.43% for RR+RoadRunner

# Chapter 3

# Road Graph Extraction from Aerial Imagery

Accurate and up-to-date road maps are critical in many applications, from navigation to self-driving vehicles. However, creating and maintaining digital maps is expensive and involves tedious manual labor. In response, automated solutions have been proposed to automatically infer road maps from different sources of data, including GPS tracks, aerial imagery, and satellite imagery. In this chapter, we focus on extracting road network graphs from satellite imagery.

Although many techniques have been proposed [14, 17, 36, 37, 61, 69, 72, 99, 115, 118, 24, 77], extracting road networks from satellite imagery is still a challenging computer vision task due to the complexity and diversity of the road networks.

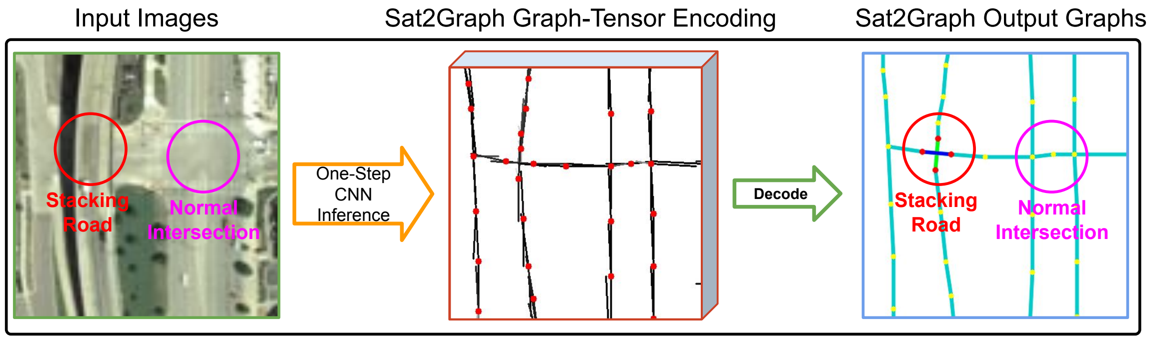


Figure 3-1: Highlight of Sat2Graph.

Prior solutions fall into two categories: pixel-wise segmentation-based approaches and graph-based approaches. Segmentation-based approaches assign a *roadness* score to each pixel in the satellite imagery. Then, they extract the road network graph using heuristic approaches. Here, the road segmentation acts as the intermediate representation of the road network graph. In contrast, graph-based approaches construct a road network graph directly from satellite imagery. Recently, Bastani *et al.* [14], as well as several follow-up works [37, 69], utilize graph-based solutions that iteratively add vertices and edges to the partially constructed graph.

We observe that the approaches in these two categories often tradeoff with each other. Segmentation-based approaches typically have a wider receptive field but rely on an intermediate non-graph representation and a post-processing heuristic (e.g., morphological thinning and line following) to extract road network graphs from this intermediate representation. The usage of the intermediate non-graph representation limits the segmentation-based approaches, and they often produce noisy and lower precision road networks compared with the graph-based methods as a result. To encourage the neural network model to focus more on the graph structure of road networks, recent work [17] proposes to train the road segmentation model jointly with road directions, and the approach achieves better road connectivity through this joint training strategy. However, a postprocessing heuristic is still needed.

In contrast, graph-based approaches [14, 37, 69] learn the graph structure directly. As a result, graph-based approaches yield road network graphs with better road connectivity compared with the original segmentation-based approach [14]. However, the graph generation process is often iterative, resulting in a neural network model that focuses more on local information rather than global information. To take more global information into account, recent work [37, 69] proposes to improve the graph-based approaches with a sequential generative model, resulting in better performance compared with other state-of-art approaches.

Recent advancements [17, 37, 69] in segmentation-based approaches and graph-based approaches respectively are primarily focused on overcoming the inherent limitations of their baseline approaches, which are exactly from the same aspects that the methods in the competing baseline approach (i.e., from the other category) claim as advantages. Based on this observation, a natural question to ask is if it is possible to combine the segmentation-based approach and the graph-based approach into one unified approach that can benefit from the advantages of both?

Our answer to this question is a new road network extraction approach, Sat2Graph, which combines the inherent advantages of segmentation-based approaches and graph-based approaches into one simple, unified framework. To do this, we design a novel encoding scheme, *graph-tensor encoding* (GTE), to encode the road network graph into a tensor representation, making it possible to train a simple, non-recurrent, supervised model that predicts graph structures holistically from the input image.

In addition to the tensor-based network encoding, this work makes two contributions:

1. Sat2Graph surpasses state-of-the-art approaches in a widely used topology-similarity metric at all precision-recall trade-off positions in an evaluation over a large city-scale dataset covering 720 $km^2$ area in 20 U.S. cities and the popular SpaceNet roads dataset [96].

2. Sat2Graph can naturally infer stacked roads, which prior approaches don't handle.

## 3.1 Background

**Traditional Approaches.** Extracting road networks from satellite imagery has long history [47, 97]. Traditional approaches generally use heuristics and probabilistic models to infer road networks from imagery. For examples, Hinz *et al.* [61] propose an approach to create road networks through a complicated road model that is built using detailed knowledge about roads and the environmental context, such as the nearby buildings, vehicles and so on. Wegner *et al.* [99] propose to model the road network with higher-order conditional random fields (CRFs). They first segment the aerial images into super-pixels, then they connect these super-pixels based on the

CRF model.

**Segmentation-Based Approaches.** With the increasing popularity of deep learning, researchers have used convolutional neural networks (CNN) to extract road network from satellite imagery [115, 118, 24, 36, 72, 17]. For example, Cheng *et al.* [36] use an end-to-end cascaded CNN to extract road segmentation from satellite imagery. They apply a binary threshold to the road segmentation and use morphological thinning to extract the road center-lines. Then, a road network graph is produced through tracing the single-pixel-width road center-lines. Many other segmentation-based approaches proposed different improvements upon this basic graph extraction pipeline, including improved CNN backbones [24, 118], improved post-processing strategy [72], improved loss functions [72, 77], incorporating GAN [114, 87, 39], and joint training [17].

In contrast with existing segmentation-based approaches, Sat2Graph does not rely on the road segmentation as intermediate representation and learns the graph structure directly.

**Graph-Based Approaches.** Graph-based approaches construct a road network graph directly from satellite imagery. Recently, Bastani *et al.* [14] proposed RoadTracer, a graph-based approach to generate road network in an iterative way. The algorithm starts from a known location on the road map. Then, at each iteration, the algorithm uses a deep neural network to predict the next location to visit along the road through looking at the surrounding satellite imagery of the current location. Recent works [37, 69] advanced the graph-based approach through applying sequential generative models (RNN) to generate road network iteratively. The usage of sequential models allows the graph generation model to take more context information into account compared with RoadTracer [14].

In contrast with existing graph-based approaches, Sat2Graph generates the road graphs in one shot (holistic). This allows Sat2Graph to easily capture the global information and make better coordination of vertex placement. The non-recurrent property of Sat2Graph also makes it easy to train and easy to extend (e.g., combine Sat2Graph with GAN). We think this simplicity of Sat2Graph is another advantage over other solutions.

**Using Other Data Sources and Other Digital Map Inference Tasks.** Extracting road networks from other data sources has also been extensively studied, e.g., using GPS trajectories collected from moving vehicles [21, 6, 44, 41, 28, 91, 55]. Besides road topology inference, satellite imagery also enables inference of different map attributes, including high-definition road details [73, 74, 57], road safety [78] and road quality [25].

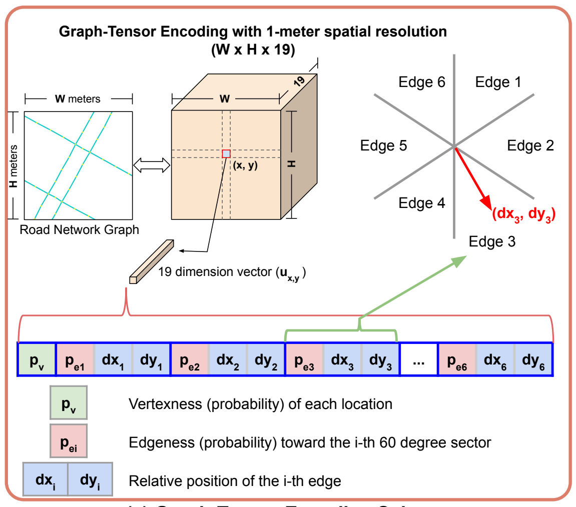
## 3.2 Sat2Graph

In this section, we present the details of our proposed approach - Sat2Graph. Sat2Graph relies on a novel encoding scheme that can encode the road network graph into a three-dimensional tensor. We call this encoding scheme Graph-Tensor Encoding (GTE). This graph-tensor encoding scheme allows us to train a simple, non-recurrent, neural network model to directly map the input satellite imagery into the road network graph (i.e., edges and vertices). As noted in the introduction, this graph construction strategy combines the advantages of segmentation-based and graph-based approaches.

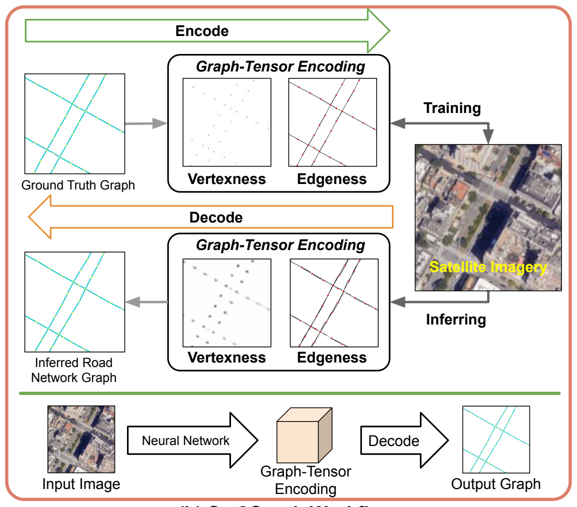
### 3.2.1 Graph-Tensor Encoding (GTE)

We show our graph-tensor encoding (GTE) scheme in Figure 3-2(a). For a road network graph $G = \{V, E\}$ that covers a $W$ meters by $H$ meters region, GTE uses a $\frac{W}{\lambda} \times \frac{H}{\lambda} \times (1 + 3 \cdot D_{max})$ 3D-tensor (denoted as $T$) to store the encoding of the graph. Here, the $\lambda$ is the spatial resolution, i.e., one meter, which restricts the encoded graph in a way that no two vertices can be co-located within a $\lambda \times \lambda$ grid, and $D_{max}$ is the maximum edges that can be encoded at each $\lambda \times \lambda$ grid.

The first two dimensions of $T$ correspond to the two spatial axes in the 2D plane. We use the vector at each spatial location $u_{x,y} = [T_{x,y,1}, T_{x,y,2}, \cdots, T_{x,y,(1+3 \cdot D_{max})}]^T$


Graph-Tensor Encoding with 1-meter spatial resolution
(W x H x 19)
W meters
H meters
Road Network Graph
W
H
19
(x, y)
19 dimension vector ($u_{x,y}$)
Edge 6
Edge 1
Edge 5
Edge 2
Edge 4
Edge 3
($dx_3$, $dy_3$)
$p_v$ $p_{e1}$ $dx_1$ $dy_1$ $p_{e2}$ $dx_2$ $dy_2$ $p_{e3}$ $dx_3$ $dy_3$ ... $p_{e6}$ $dx_6$ $dy_6$
$p_v$ Vertexness (probability) of each location
$p_{ei}$ Edgeness (probability) toward the i-th 60 degree sector
$dx_i$ $dy_i$ Relative position of the i-th edge

(a) Graph-Tensor Encoding Scheme



(b) Sat2Graph Workflow

Figure 3-2: Graph-Tensor Encoding and Sat2Graph workflow.

to encode the graph information. As shown in Figure 3-2(a), the vector $u_{x,y}$ has $(1+3\cdot D_{max})$ elements. Its first element $p_v \in [0,1]$ (vertexness) encodes the probability of having a vertex at position $(x, y)$. Following the first element are $D_{max}$ 3-element groups, each of which encodes the information of a potential outgoing edge from position $(x, y)$. For the $i$-th 3-element group, its first element $p_{e_i} \in [0,1]$ (edgeness) encodes the probability of having an outgoing edge toward $(dx_i, dy_i)$, i.e., an edge pointing from $(x, y)$ to $(x + dx_i, y + dy_i)$. Here, we set $D_{max}$ to six as we find that vertices with degree greater than six are very rare in road network graphs.

To reduce the number of possible different isomorphic encodings of the same input graph, GTE only uses the $i$-th 3-element group to encode edges pointing toward a $\frac{360}{D_{max}}$-degree sector from $(i-1)\cdot\frac{360}{D_{max}}$ degrees to $i\cdot\frac{360}{D_{max}}$ degrees. We show this restriction and an example edge (in red color) in Figure 3-2(a). This strategy imposes a new restriction on the encoded graphs – for each vertex in the encoded graph, there can only be at most one outgoing edge toward each $\frac{360}{D_{max}}$-degree sector. However, we find this restriction does not impact the representation ability of GTE for most road graphs. This is because the graphs encoded by GTE are *undirected*. We defer the discussion on this in Section 3.4.

### Encode

Encoding a road network graph into GTE is straightforward. For road network extraction application, the encoding algorithm first interpolates the segment of straight road in the road network graph. It selects the minimum number of evenly spaced intermediate points so that the distance between consecutive points is under $d$ meters. This interpolation strategy regulates the length of the edge vector in GTE, making the training process stable. Here, a small $d$ value, e.g., $d < 5$, converts GTE back to the road segmentation, making GTE unable to represent stacking roads. A very large $d$ value, e.g. $d = 50$, makes the GTE hard to approximate curvy roads. For these reasons, we think a $d$ value between 15 to 25 can work the best. In our setup, we set $d$ to 20.

For stacked roads, the interpolation may produce vertices belonging to two overlapped road segments at the same position. When this happens, we use an iterative conflict-resolution algorithm to shift the positions of the endpoint vertices of the two edges. The goal is to make sure the distance between any two vertices (from the two overlapping edges) is greater than 5 meters. During training, this conflict-resolution pre-processing also yields more consistent supervision signal for stacked roads - overlapped edges tend to always cross near the middle of each edge. After this step, the encoding algorithm maps each of the vertices to the 3D-tensor $T$ following the scheme shown in Figure 3-2(a). For example, the algorithm sets the vertexness ($p_v$) of $u_{x,y}$ to 1 when there is a vertex at position $(x, y)$, otherwise the vertexness is set to 0.

### Decode

GTE's Decoding algorithm converts the predicted GTE (often noisy) of a graph back to the regular graph format (G = {V,E}). The decoding algorithm consists of two steps, (1) vertex extraction and, (2) edge connection. As both the vertexness predictions and edgeness predictions are real numbers between 0 and 1, we only consider vertices and edges with probability greater than a threshold (denoted as $p_{thr}$).

In the vertex extraction step, the decoding algorithm extracts the potential vertices through localizing the local maximas of the vertexness map (we show an example of this in Figure 3-2(b)). The algorithm only considers the local maximas with vertexness greater than $p_{thr}$.

In the edge connection step, for each candidate vertex $v \in V$, the decoding algorithm connects its outgoing edges to other vertices. For the $i$-th edge of vertex $v \in V$, the algorithm computes its distance to all nearby vertices $u$ through the following distance function,

$$\begin{aligned} d(v, i, u) = & ||(v_x + dx_i, v_y + dy_i) - (u_x, u_y)|| \\ & + w \cdot cos_{dist}((dx_i, dy_i), (u_x - v_x, u_y - v_y)) \end{aligned} \tag{3.1}$$

, where $cos_{dist}(v_1, v_2)$ is the cosine distance of the two vectors, and $w$ is the weight of the cosine distance in the distance function. Here, we set $w$ to a large number, i.e., 100, to avoid incorrect connections. After computing this distance, the decoding algorithm picks up a vertex $u'$ that minimizes the distance function $d(v, i, u)$, and adds an edge between $v$ and $u'$. We set a maximum distance threshold, i.e., 15 meters, to avoid incorrect edges being added to the graph when there are no good candidate vertices nearby.

### 3.2.2 Training Sat2Graph

We use cross-entropy loss (denoted as $\mathcal{L}_{CE}$) and $L_2$-loss to train Sat2Graph. The cross-entropy loss is applied to vertexness channel ($p_v$) and edgeness channels ($p_{e_i}$ $i \in \{1, 2, ..., D_{max}\}$), and the $L_2$-loss is applied to the edge vector channels ($(dx_i, dy_i)$ $i \in \{1, 2, ..., D_{max}\}$). GTE is inconsistent along long road segments. In this case, the same road structure can be mapped to different ground truth labels in GTE representation. Because of this inconsistency, we only compute the losses for edgeness and edge vectors at position $(x, y)$ when there is a vertex at position $(x, y)$ in the ground truth. We show the overall loss function below ($\hat{T}, \hat{p_v}, \hat{p_{e_i}}, \hat{d_{x_i}}, \hat{d_{y_i}}$are from ground truth),

$$\begin{aligned}\mathcal{L}(T, \hat{T}) = \sum_{(x,y)\in[1..W]\times[1..H]} \Bigg( & \mathcal{L}_{CE}(p_v, \hat{p_v}) \\ & + \hat{T}_{x,y,1} \cdot \Big( \sum_{i=1}^{D_{max}} \big( \mathcal{L}_{CE}(p_{e_i}, \hat{p_{e_i}}) + \mathcal{L}_2((dx_i, dy_i), (\hat{dx_i}, \hat{dy_i})) \big) \Big) \Bigg)\end{aligned} \tag{3.2}$$

In Figure 3-2(b), we show the training and inferring workflows of Sat2Graph. Sat2Graph is agnostic to the CNN backbones. In this work, we choose to use the Deep Layer Aggregation (DLA) [109] segmentation architecture as our CNN backbone. We use residual blocks [52] for the aggregation function in DLA. The feasibility of training Sat2Graph with supervised learning is counter-intuitive because of the GTE's inconsistency. We defer the discussion of this to Section 3.4.

## 3.3 Evaluation

We now present experimental results comparing Sat2Graph to several state-of-the-art road-network generation systems.

### 3.3.1 Datasets

We conduct our evaluation on two datasets, one is a large city-scale dataset and the other is the popular SpaceNet roads dataset [96].

**City-Scale Dataset.** Our city-scale dataset covers 720 $km^2$ area in 20 U.S. cities. We collect road network data from OpenStreetMap [50] as ground truth and the corresponding satellite imagery through Google static map API [48]. The spatial resolution of the satellite imagery is set to one meter per pixel. This dataset enables us to evaluate the performance of different approaches at city scale, e.g., evaluating the quality of the shortest path crossing the entire downtown of a city on the inferred road graphs.

The dataset is organized as 180 tiles; each tile is a 2 km by 2 km square region. We randomly choose 15% (27 tiles) of them as a testing dataset and 5% (9 tiles) of them as a validation dataset. The remaining 80% (144 tiles) are used as training dataset.

**SpaceNet Roads Dataset.** Another dataset we used is the SpaceNet roads Dataset [96]. Because the ground truth of the testing data in the SpaceNet dataset is not public, we randomly split the 2549 tiles (non-empty) of the original training dataset into training(80%), testing(15%) and validating(5%) datasets. Each tile is a 0.4 km by 0.4 km square. Similar to the city-scale dataset, we resize the spatial resolution of the satellite imagery to one meter per pixel.

### 3.3.2 Baselines

We compare Sat2Graph with four different segmentation-based approaches and one graph-based approach.

**Segmentation-Based Approaches.** We use four different segmentation-based approaches as baselines.

1. *Seg-UNet:* Seg-UNet uses a simple U-Net [85] backbone to produce road segmentation from satellite imagery. The model is trained with cross-entropy loss. This scheme acts as the naive baseline as it is the most straightforward solution for road extraction.

2. *Seg-DRM [72](ICCV-17):* Seg-DRM uses a stronger CNN backbone which contains 55 ResNet [52] layers to improve the road extraction performance. Meanwhile, Seg-DRM proposes to train the road segmentation model with soft-IoU loss to achieve better performance. However, we find training the Seg-DRM model with cross-entropy loss yields much better performance in terms of topology correctness. Thus, in our evaluation, we train the Seg-DRM model with cross-entropy loss.

3. *Seg-Orientation [17](ICCV-19):* Seg-Orientation is a recent state-or-the-art approach which proposes to improve the road connectivity by joint learning of road

orientation and road segmentation. Similar to Seg-DRM, we show the results of Seg-Orientation trained with cross-entropy loss as we find it performs better compared with soft-IoU loss.

4. *Seg-DLA:* Seg-DLA is our enhanced segmentation-based approach which uses the same CNN backbone as our Sat2Graph model. Seg-DLA, together with Seg-UNet, act as the baselines of an ablation study of Sat2Graph.

**Graph-Based Approaches.** For graph-based approaches, we compare our Sat2Graph solution with RoadTracer [14](CVPR-18) by applying their code on our dataset. During inference, we use peaks in the segmentation output as starting locations for RoadTracer's iterative search.

### 3.3.3 Implementation Details

**Data Augmentation:** For all models in our evaluation, we augment the training dataset with random image brightness, hue and color temperature, random rotation of the tiles, and random masks on the satellite imagery.

**Training:** We implemented both Sat2Graph and baseline segmentation approaches using Tensorflow. We train the model on a V100 GPU for 300k iterations (about 120 epochs) with a learning rate starting from 0.001 and decreasing by 2x every 50k iterations. We train all models with the same receptive field, i.e., 352 by 352. We evaluate the performance on the validation dataset for each model every 5k iterations during training, and pick up the best model on the validation dataset as the converged model for each approach to avoid overfitting.

### 3.3.4 Evaluation Metrics

In the evaluation, we focus on the topology correctness of the inferred road graph rather than edge-wise correctness. This is because the topology correctness is often crucial in many real-world applications. For example, in navigation applications, a small missing road edge in the road graph could make two regions disconnected. This small missing road segment is a small error in terms of edge-wise correctness but a huge error in terms of topology correctness.

We evaluate the topology correctness of the inferred road graphs through two metrics, TOPO [20] and APLS [96]. Here, we describe the high level idea of these two metrics. Please refer to [20, 96] for more details about these two metrics.

**TOPO metric:** TOPO metric measures the similarity of sub-graphs sampled on the ground truth graph and the inferred graph from a seed location. The seed location is matched to the closest seed node on each graph. Here, given a seed node on a graph, the sub-graph contains all the nodes such that their distances (on the graph) to the seed node are less than a threshold, e.g., 300 meters. For each seed location, the similarity between two sampled sub-graphs is quantified as precision, recall and $F_1$-score. The metric reports the average precision, recall and $F_1$-score over randomly sampled seed locations over the entire region.

The TOPO metric has different implementations. We implement the TOPO metric in a very strict way following the description in [55]. This strict implementation allows the metric to penalize detailed topology errors.

**APLS metric:** APLS measures the quality of the shortest paths between two locations on the graph. For example, suppose the shortest path between two locations on the ground truth map is 200 meters, but the shortest path between the same two locations on the inferred map is 20 meters (a wrong shortcut), or 500 meters, or doesn't exist. In these cases, the APLS metric yields a very low score, even though there might be only one incorrect edge on the inferred graph.

### 3.3.5 Quantitative Evaluation

**Overall Quality.** Each of the approaches we evaluated has one major hyper-parameter, which is often a probability threshold, that allows us to make different precision-recall trade-offs. We change this parameter for each approach to plot an

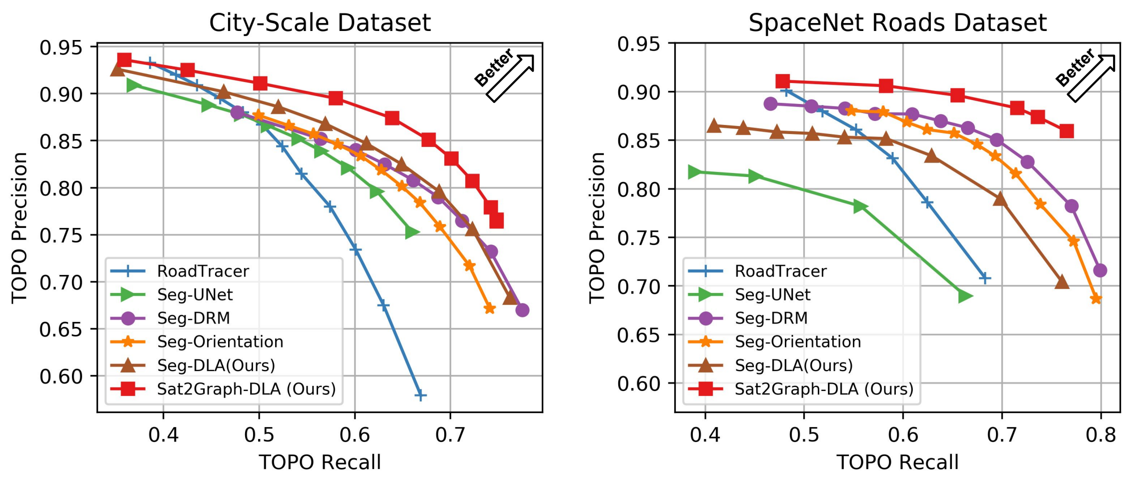


Figure 3-3: TOPO metric precision-recall trade-off curves

precision-recall curve. We show the precision-recall curves for different approaches in Figure 3-3. This precision-recall curve allows us to see the full picture of the capability of each approach. We also show the best achievable TOPO $F_1$-score and APLS score of each approach in Table 3.1 for reference.

From Figure 3-3, we find an approach may not always better than another approach at different precision-recall position (TOPO metric). For examples, the graph-based approach RoadTracer performs better than others when the precision is high, whereas the segmentation-based approach DeepRoadMapper performs better when the recall is high.

Meanwhile, we find an approach may not always better than another approach on both TOPO and APLS. For example, in Table 3.1, RoadTracer has the best APLS score but the worst TOPO $F_1$-score in the five baselines on the city-scale dataset. This is because RoadTracer is good at coarse-grained road connectivity and the precision of inferred road graphs rather than recall. For example, in Figure 3-4(a), RoadTracer is better compared with Seg-DRM and Seg-Orientation in terms of road connectivity when the satellite imagery is full of shadow.

In contrast, Sat2Graph surpasses all other approaches on APLS metric and at all TOPO precision-recall positions – for a given precision, Sat2Graph always has the best recall; and for a given recall, Sat2Graph always has the best precision. We think

| Method | City-Scale Dataset | | | |
|---|---|---|---|---|
| | Prec. | Rec. | $F_1$ | APLS |
| RoadTracer[14](CVPR-18) | 78.00 | 57.44 | 66.16 | 57.29 |
| Seg-UNet | 75.34 | 65.99 | 70.36 | 52.50 |
| Seg-DRM[72](ICCV-17) | 76.54 | 71.25 | 73.80 | 54.32 |
| Seg-Orientation[17](ICCV-19) | 75.83 | 68.90 | 72.20 | 55.34 |
| Seg-DLA(ours) | 75.59 | 72.26 | 73.89 | 57.22 |
| Sat2Graph-DLA(ours) | 80.70 | 72.28 | **76.26** | **63.14** |

| Method | SpaceNet Roads Dataset | | | |
|---|---|---|---|---|
| | Prec. | Rec. | $F_1$ | APLS |
| RoadTracer[14](CVPR-18) | 78.61 | 62.45 | 69.60 | 56.03 |
| Seg-UNet | 68.96 | 66.32 | 67.61 | 53.77 |
| Seg-DRM[72](ICCV-17) | 82.79 | 72.56 | 77.34 | 62.26 |
| Seg-Orientation[17](ICCV-19) | 81.56 | 71.38 | 76.13 | 58.82 |
| Seg-DLA(ours) | 78.99 | 69.80 | 74.11 | 56.36 |
| Sat2Graph-DLA(ours) | 85.93 | 76.55 | **80.97** | **64.43** |

Table 3.1: Comparison of the *best achievable* TOPO $F_1$-score and APLS score. We show the best TOPO $F_1$-score's corresponding precision and recall just for reference not for comparison. (All the values in this table are percentages)

this is because Sat2Graph's graph-tensor encoding takes advantages from both the segmentation-based approaches and graph-based approaches, and allows Sat2Graph to infer stacking roads that none of the other approaches can handle. As an ablation study, we compare Sat2Graph-DLA with Seg-DLA (Seg-DLA uses the same CNN backbone as Sat2Graph-DLA). We find the superiority of Sat2Graph comes from the graph-tensor encoding rather than the stronger CNN backbone.

**Benefit from GTE.** In addition to the results shown in Table 3.1, we show the results of using GTE with other backbones. On our city-wide dataset, we find GTE can improve the TOPO $F_1$-score from 70.36% to 76.40% with the U-Net backbone and from 73.80% to 74.66% with the Seg-DRM backbone. Here, the improvement on Seg-DRM backbone is minor because Seg-DRM backbone has a very shallow decoder.

**Sensitivity on $w$.** In our decoding algorithm, we have a hyper-parameter $w$ which is the weight of the cosine distance term in equation 3.1. In Table 3.2, we show how this parameter impacts the TOPO $F_1$-score on our city-wide dataset. We find the performance is robust to w - the $F_1$-scores are all greater than 76.2% with w in

| Value of $w$ | 1 | 5 | 10 | 25 | 75 | 100 | 150 |
|---|---|---|---|---|---|---|---|
| $F_1$-score | 75.87% | 76.28% | 76.62% | 76.72% | 76.55% | 76.26% | 75.68% |

Table 3.2: TOPO $F_1$ scores on our city-wide dataset with different $w$ values.

the range from 5 to 100.

**Vertex threshold and edge threshold.** In our basic setup, we set the vertex threshold and the edge threshold of Sat2Graph to the same value. However, we can also use independent probability thresholds for vertices and edges. We evaluate this by choosing a fixed point and vary one probability threshold at a time. We find the vertex threshold dominates the performance and using a higher edge probability threshold (compared with the vertex probability) is helpful to achieve better performance.

**Stacking Road.** We evaluate the quality of the stacking road by matching the overpass/underpass crossing points between the ground truth graphs and the proposed graphs. In this evaluation, we find our approach has a precision of 83.11% (number of correct crossing points over the number of all proposed crossing points) and a recall of 49.81% (number of correct crossing points over the number of all ground-truth crossing points) on stacked roads. In fact only 0.37% of intersections are incorrectly predicted as overpasses/underpasses (false-positive rate). We find some small roads under wide highway roads are missing entirely. We think this is the reason for the low recall.

### 3.3.6 Qualitative Evaluation

**Regular Urban Areas.** In the regular urban areas (Figure 3-4), we find the existing segmentation-based approach with a strong CNN backbone (Seg-DLA) and better data augmentation techniques has already been able to achieve decent results in terms of both precision and recall, even if the satellite imagery is full of shadows and occlusions. Compared with Sat2Graph, the most apparent remaining issue of the segmentation-based approach appears at parallel roads. We think the root cause of this issue is from the fundamental limitation of segmentation-based approaches — the road-segmentation intermediate representation. Sat2Graph eliminates this limitation through graph-tensor encoding, thereby, Sat2Graph is able to produce detailed road structures precisely even along closeby parallel roads.

**Stacked Roads.** We show the orthogonal superiority of Sat2Graph on stacked roads in Figure 3-5. None of the existing approaches can handle stacked roads, whereas Sat2Graph can naturally infer stacked roads thanks to the graph-tensor encoding. We find Sat2Graph may still fail to infer stacking roads in some complicated scenarios such as in Figure 3-5(d-e). We think this can be further improved in a future work, such as adding discriminative loss to regulate the inferred road structure.

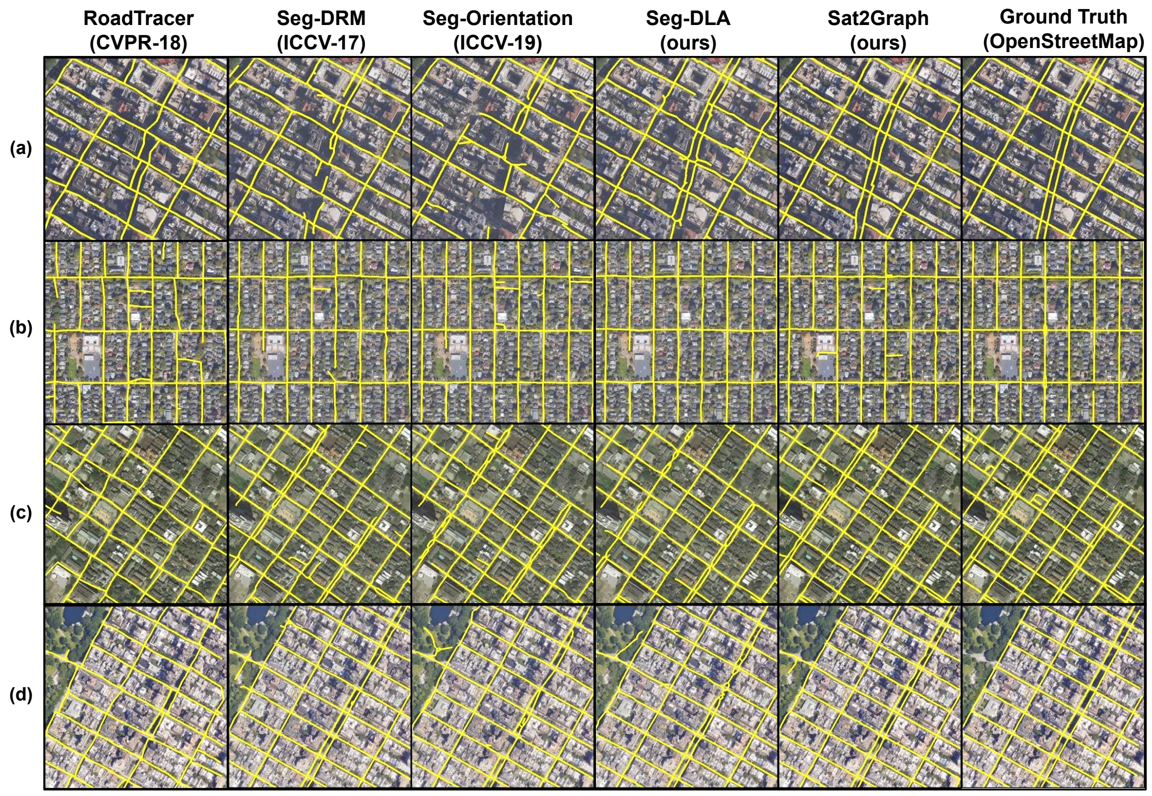


Figure 3-4: Qualitative Comparison in Regular Urban Areas. We use the models that yield the best TOPO $F_1$ scores to create this visualization.

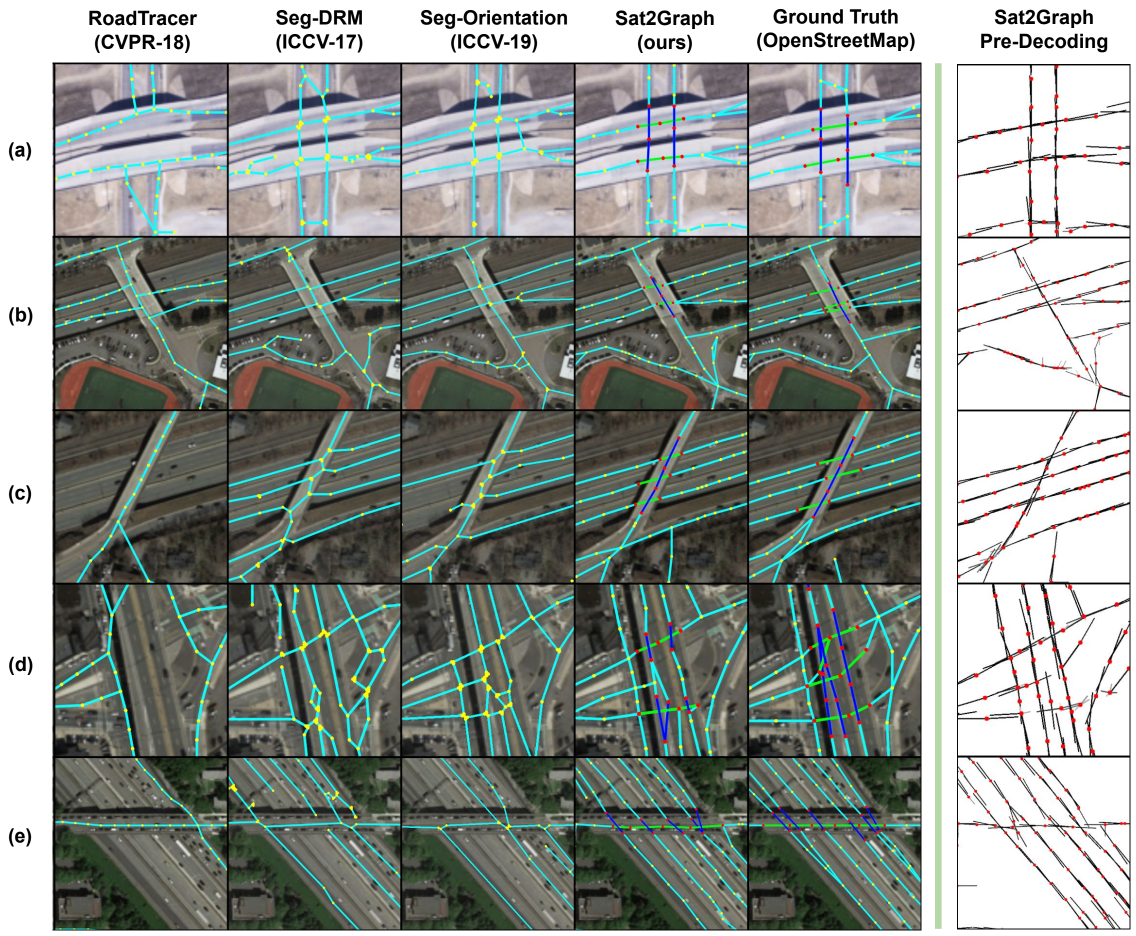


Figure 3-5: Qualitative Comparison for Stacked Roads. Sat2Graph robustly infers stacked roads in examples (a-c), but makes some errors in (d) and (e). Prior work infers only planar graphs and incorrectly captures road topology around stacked roads in all cases. We highlight edges that cross without connecting in green and blue.

## 3.4 Discussion

There are two concerns regarding Sat2Graph: (1) it seems that the heavily restricted and non-lossless graph-tensor encoding may not be able to correctly represent all different road network graphs, and (2) training a model to output GTE representation with supervised learning seems impossible because the GTE representation is not consistent.

**Concern about the encoding capability.** We think there are two reasons that make GTE able to encode almost all road network graphs.

| $D_{\max}$ | 3 | 4 | 5 | 6 | 8 |
|---|---|---|---|---|---|
| Fixed with *undirected* property | 8.62% | 2.81% | 1.18% | 0.92% | 0.59% |
| Fixed with *interpolatable* property | 0.013% | 0.0025% | 0.0015% | 0.0013% | 0.0013% |

Table 3.3: The ratios of edges fixed using the undirected and interpolatable properties.

First, the road network graph is *undirected.* Although roads have directions, the road directions can be added later as road attributes, after the road network extraction. In this case, for each edge $e = (v_a, v_b)$, we only need to encode one link from $v_a$ to $v_b$ or from $v_b$ to $v_a$, rather than encode both of the two links. Even though GTE has the $\frac{360}{D_{max}}$-degree sector restriction on outgoing edges from one vertex, this undirected-graph property makes it possible to encode very sharp branches such as the branch vertices between a highway and an exit ramp.

Second, the road network graph is *interpolatable.* There could be a case where none of the two links of an edge $e = (v_a, v_b)$ can be encoded into GTE because both $v_a$ and $v_b$ need to encode their other outgoing links. However, because the road network graph is interpolatable, we can always interpolate the edge $e$ into two edges $e_1 = (v_a, v')$ and $e_2 = (v', v_b)$. After the interpolation, the original geometry and topology remain the same but we can use the additional vertex $v'$ to encode the connectivity between $v_a$ and $v_b$.

In Table 3.3, we show the ratios of edges that need to be fixed using the *undirected* and *interpolatable* properties in our dataset with different $D_{\max}$ values.

**Concern about supervised learning.** Another concern with GTE is that for one input graph, there exist many different isomorphic encodings for it (e.g., there are many possible vertex interpolations on a long road segment.). These isomorphic encodings produce inconsistent ground truth labels. During training, this inconsistency of the ground truth can make it very hard to learn the right mapping through supervised learning.

However, counter-intuitively, we find Sat2Graph is able to learn through supervised learning and learn well. We find the key reason of this is because of the inconsistency of GTE representation doesn't equally impact the vertices and edges in a graph. For example, the locations of intersection vertices are always consistent in different isomorphic GTEs.

We find GTE has high label consistency for supervised learning at important places such as intersections and overpass/underpass roads. Often, these places are the locations where the challenges really come from. Although GTE has low consistency for long road segments, the topology of the long road segment is very simple and can still be corrected through GTE's decoding algorithm.

## 3.5 Conclusion

In this work, we have proposed a simple, unified road network extraction solution that combines the advantages from both segmentation-based approaches and graph-based approaches. Our key insight is a novel graph-tensor encoding scheme. Powered by this graph-tensor approach, Sat2Graph is able to surpass existing solutions in terms of topology-similarity metric at all precision-recall points in an evaluation over two large datasets. Additionally, Sat2Graph naturally infers stacked roads like highway overpasses that none of the existing approaches can handle.

# Chapter 4

# Lane-Level Street Map Extraction

Digital maps with lane-level details are the foundation of many applications, including lane-to-lane navigation, route planning for delivery fleets or autonomous vehicles, and lane-level localization. However, creating and maintaining digital street maps, especially with lane-level details, are time-consuming and labor-intensive. As a result, automatic mapping techniques have drawn much attention from both industry and academics, and they have proposed many solutions to automate the mapping process. Albeit great efforts devoted to this problem, given the complex nature of real-world scenarios, how to automatically produce digital maps at scale with low costs and an acceptable accuracy is still an open research question.

We can put automatic mapping techniques into two categories based on their data sources. (1) One type of mapping solution relies on the sensors mounted on vehicles, e.g., GPS receivers, IMUs, cameras, and lidars. This type of mapping solution can provide decent mapping accuracy. However, because it relies on sensor-equipped vehicles, the mapping cost is often high, especially when maintaining an up-to-date map covering a large region. (2) Another type of mapping solution relies on remote sensing data, i.e., imagery and radar data, collected from airplanes or satellites. This type of mapping solution can quickly scale up to a large region at a low cost. However, limited by the top-down view and the resolution of the data, prior works mainly focus on extracting maps at road level [72, 14, 118, 17, 106, 70, 37, 56, 92, 13, 16], extracting road curbs [103, 104, 105], segmenting visible lane markers [11, 46], and inferring lane count and lane position [58, 113]; none of them extract a complete and routable lane-level street map.

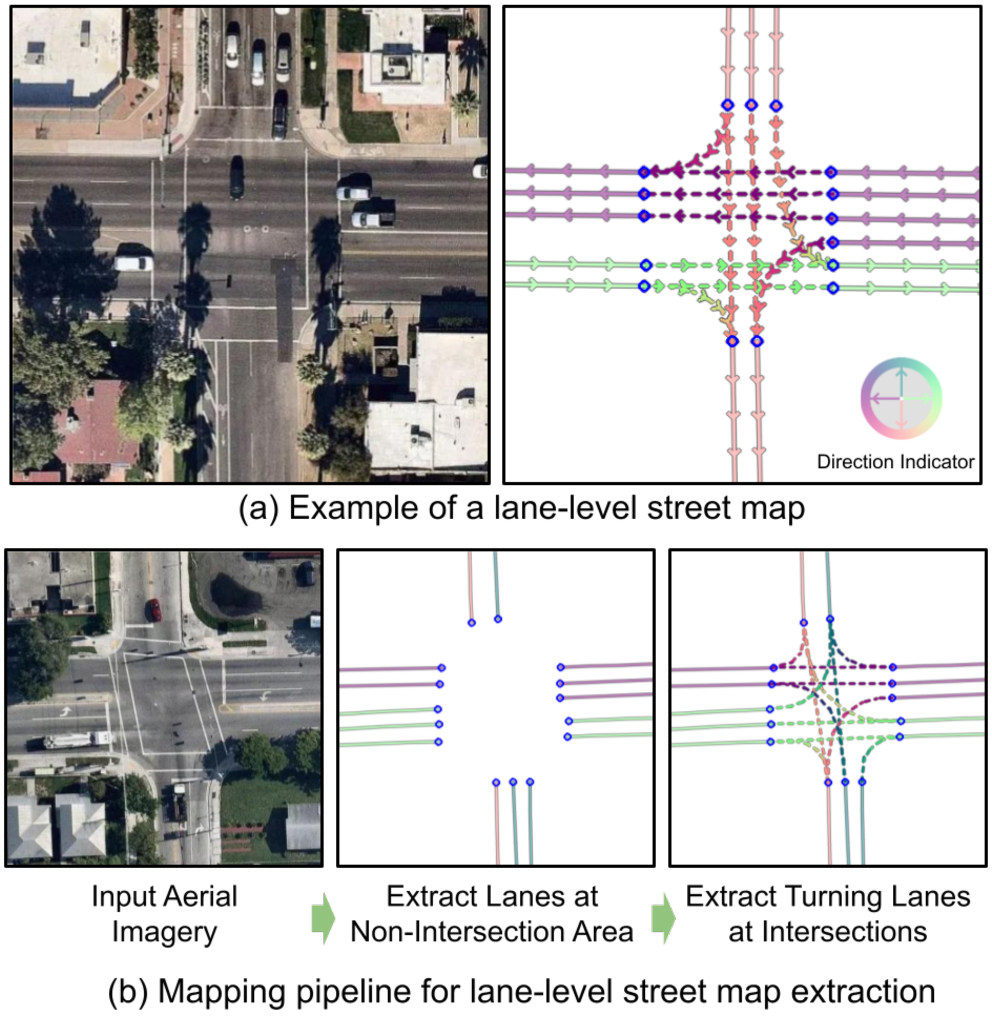

Direction Indicator
(a) Example of a lane-level street map
Input Aerial Imagery
Extract Lanes at Non-Intersection Area
Extract Turning Lanes at Intersections
(b) Mapping pipeline for lane-level street map extraction

Figure 4-1: Example of a lane-level street map and our proposed mapping pipeline. We show lanes at non-intersection areas with solid lines and show the turning lanes with dashed lines. The colors of lanes indicate their driving directions. We highlight the terminal nodes in blue.

This work takes one step forward toward a promising direction – extracting routable lane-level street maps from aerial imagery. In Figure 4-1 (a), we show an example of the lane-level street map at an intersection. Unlike road-level street maps, the lane-level street maps capture detailed road network features, including lane geometry such as lane merging and lane diverging, and rich semantic information such as driving directions, e.g., the one-way road in Figure 4-1 (a), and turning lane restrictions, e.g., the left-only and right-only lanes in Figure 4-1 (a,b). Because of the complexity of lane topology and semantic, extracting lane-level street maps from aerial imagery is a challenging vision task and existing road-level street map extraction solutions are incompetent to solve it.

In this work, to address the challenge, we propose a mapping pipeline for lane-level street map extraction. We have a key observation that, unlike road-level map extraction, extracting the entire lane-level maps in one shot can be very challenging. However, the lane extraction problem becomes solvable if we divide the task into sub-tasks and solve them separately. Hence, we split the lane extraction task into two sub-tasks. As shown in Figure 4-1(b), we first create lane-level maps at non-intersection areas with lane geometry and lane direction. Then, we enumerate all the possible turning lanes, i.e., a connection between two terminal nodes (we highlight the terminal nodes in blue in Figure 4-1), at the intersections. We check if the turning lanes are valid and extract the geometry of the valid turning lanes to complete the map. Here, we consider the lanes that go straight at intersections also turning lanes.

We evaluate our mapping pipeline on a dataset containing 400 km of lanes in four US cities, Boston, Seattle, Phoenix, and Miami. In the evaluation, we compare different neural network designs at each stage of the mapping pipeline, showing the effectiveness of our mapping pipeline and the potential of aerial-imagery-based lane-level street map extraction.

In this work, we make the following contributions,

- We proposed an automatic mapping pipeline to extract routable lane-level street maps from aerial imagery. To the best of our knowledge, this is the first work that extracts a fully-routable lane-level street maps from aerial imagery.
- Our evaluation demonstrates the effectiveness of our mapping pipeline and unveils challenges in the task, which can inspire future research work.

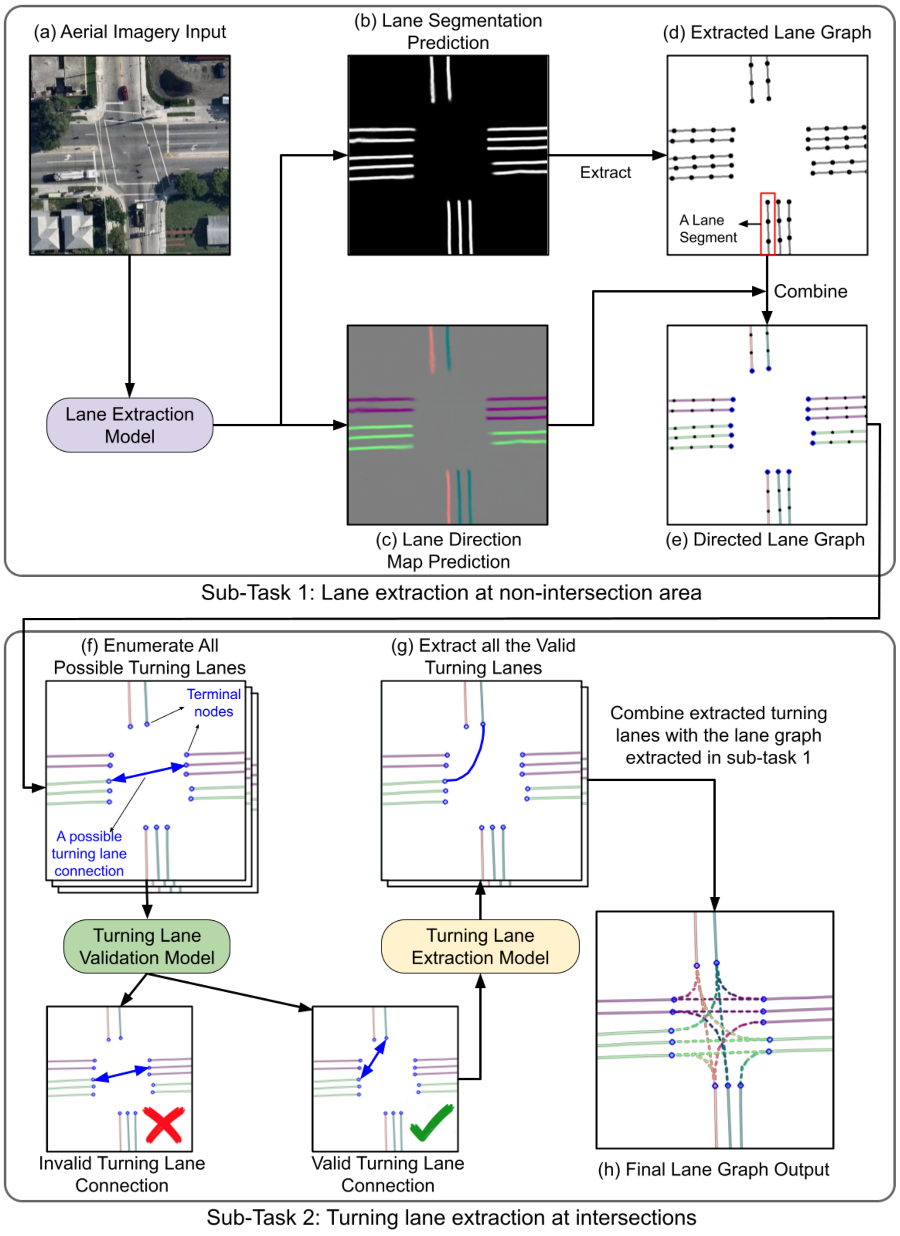
## 4.1 Background

**Mapping with aerial or satellite imagery.** Extracting digital maps from aerial or satellite imagery has been extensively studied. Most of the prior works focus on extracting road-level maps. A widely-used strategy to solve this problem is to turn the road extraction problem into a road segmentation problem, for examples, DeepRoadMapper [72], D-LinkNet [118], joint orientation learning [17], and many other works [106, 24, 95, 45] – they all adopt a segmentation-based map extraction strategy. Besides the segmentation-based approaches, RoadTracer [14] first proposes to extract the road network using an iterative graph construction approach that does not rely on the road segmentation but constructs the road network directly. After RoadTracer, several follow-up works [70, 92, 37] have proposed improvements upon the graph construction strategy. Recently, Sat2Graph [56] proposes the graph-tensor encoding to unify the segmentation strategy and the graph construction strategy, and shows promising results. However, all of those works focus on map extraction at road levels, and we cannot directly apply them to lane-level map extraction given the complexity of lane-level maps in terms of both topology and semantic.

Besides road topology extraction, many other works extract different map features from aerial or satellite imagery. For example, lane marker extractions [11, 46], lane curb extraction [103, 104, 105], lane count and position extraction [58, 113], road attribute inference [58], road safety assessment [59, 78], etc. These works extract important features about the road networks; however, they cannot produce a complete routable lane graph that is directly useful for downstream applications.

**Mapping with vehicle sensors.** Creating maps using sensor data collected on vehicles, including dedicated survey vehicles and other vehicles like taxis, has drawn great attention in industry and academics. Prior works have explored mapping solutions with different sensor data sources including GPS [6, 21, 28, 41, 44, 55, 91, 10], cameras [64, 63], lidars [62, 65, 83, 116], and the combinations of different data sources [119]. In this work, we focus on mapping with remote sensing data, i.e., aerial imagery, which complements the mapping solutions based on vehicle sensors.

## 4.2 Mapping Pipeline



(a) Aerial Imagery Input
(b) Lane Segmentation Prediction
(d) Extracted Lane Graph
Extract
A Lane Segment
Combine
Lane Extraction Model
(c) Lane Direction Map Prediction
(e) Directed Lane Graph
Sub-Task 1: Lane extraction at non-intersection area
(f) Enumerate All Possible Turning Lanes
Terminal nodes
A possible turning lane connection
(g) Extract all the Valid Turning Lanes
Combine extracted turning lanes with the lane graph extracted in sub-task 1
Turning Lane Validation Model
Turning Lane Extraction Model
Invalid Turning Lane Connection
Valid Turning Lane Connection
(h) Final Lane Graph Output
Sub-Task 2: Turning lane extraction at intersections

Figure 4-2: Our proposed lane-level street map extraction pipeline.

In this work, we represent the lane-level street map as a directed graph $G = \{V, E\}$, where the vertices represent locations in a plane and the edges represent lane segments (centerlines). This graph captures both non-intersection lanes and the turning lanes (virtual lanes) at intersections. The direction of the edge encodes the driving direction of the lane. We call this directed graph a lane graph when we use it to describe a lane-level street map. While a lane-level street map contains many features, the lane graph is a basic but essential map representation capturing the core features that make the lane-level street map routable. In this work, we propose a mapping pipeline to extract the lane graph from aerial imagery.

Though prior works have proposed many solutions on road network extraction, we cannot directly use them to extract lane graphs because most of the existing works extract the road network as a planar graph (no intersecting edges). However, at road intersections, the lane graph is not planar – it needs to represent turning lanes that intersect but are not connected. Nevertheless, we need to extract the driving directions of the lanes, whereas this is not necessary for road-level map extraction. Therefore, we have to design a new solution to extract the lane graph.

We propose a mapping pipeline to extract lane graphs from aerial imagery. As shown in Figure 4-2, we split the lane graph extraction task into two sub-tasks. In the first sub-task, the goal is to extract lanes at non-intersection areas, where the lane graph is planar; therefore, we can adapt existing road extraction algorithms to extract the lanes. We show the details of this sub-task in Section 4.2.1.

In the second sub-task, the goal is to extract turning lanes at intersections. At intersections, the lane graph is no longer planar, and it contains many intersecting edges, making it very challenging to extract the lane graph. To overcome this challenge, we extract each turning lane separately instead of extracting the entire lane graph in one shot. As we have already extracted the lane graph at non-intersection areas, we can locate all the terminal vertices (the endpoints of lanes at intersections, highlighted in blue circles in Figure 4-2 (f)). Because each turning lane has to start from one terminal vertex and end at another terminal vertex, we can enumerate all the possible pairs of terminal vertices that may have valid turning lanes connecting them. In our mapping pipeline, we enumerate all the terminal vertex pairs whose distance (distance between two vertices in a pair) is below a threshold. Then, for each pair, we validate its connectivity using a classifier (a neural network). If the classifier indicates a turning lane connecting the pair of terminal vertices, we extract the vectors of the turning lanes (Figure 4-2 (g)). Finally, we merge the extracted turning lanes with the lane graph extracted in sub-task 1 to create a complete lane graph.

### 4.2.1 Lane Extraction at Non-Intersection Areas

In the first sub-task of the mapping pipeline, we extract the lane graph at non-intersection areas using a segmentation-based approach. In this approach, we represent the lane graph as a lane segmentation, e.g., Figure 4-2 (b), where each location in the segmentation has a value of either one or zero , indicating if that location has a lane or not. To extract the lane graph from aerial imagery, we first use a semantic segmentation model to extract the lane segmentation from aerial imagery and then extract the lane graph (Figure 4-2 (d)) from the lane segmentation. Different from road extraction tasks, we also need to extract the direction of the lanes. To do so, we extract a direction map (Figure 4-2 (c)) from the input aerial imagery and combine it with the extracted lane graph to create the final directed lane graph (Figure 4-2 (e)). Next, we discuss the details of the model architecture, and the training/inference processes.

**Lane extraction model.** Suppose the input aerial image has a spatial dimension of $N \times N$ with three color channels. We use a convolutional neural network to extract the lane segmentation $s \in \mathbb{R}^{N \times N}, 0 \leq s_{i,j} \leq 1$ and the direction map $d \in \mathbb{R}^{N \times N \times 2}, -1 \leq d_{i,j,k} \leq 1$. In the direction map $d$, if the location $(i, j)$ overlaps with a lane, we encode the lane direction as a normalized 2D vector in $d_{i,j}$; otherwise, we set $d_{i,j}$ to a zero vector.

In Figure 4-3, we show the architecture of our lane extraction model. Inspired by [118, 45], we adapts an UNet [85] structure with a ResNet [52] encoder for better feature extraction and dilated convolutional layers [108] for larger receptive field. To output the lane segmentation and the lane direction map, we use two branches after the last decoder block so that these two tasks can share most of the neural weights in the model. We use *softmax* as the activation function for the lane segmentation branch and use linear activation function for the direction map branch.

**Ground truth.** We create the ground truth lane segmentation $\hat{s}$ and the direction

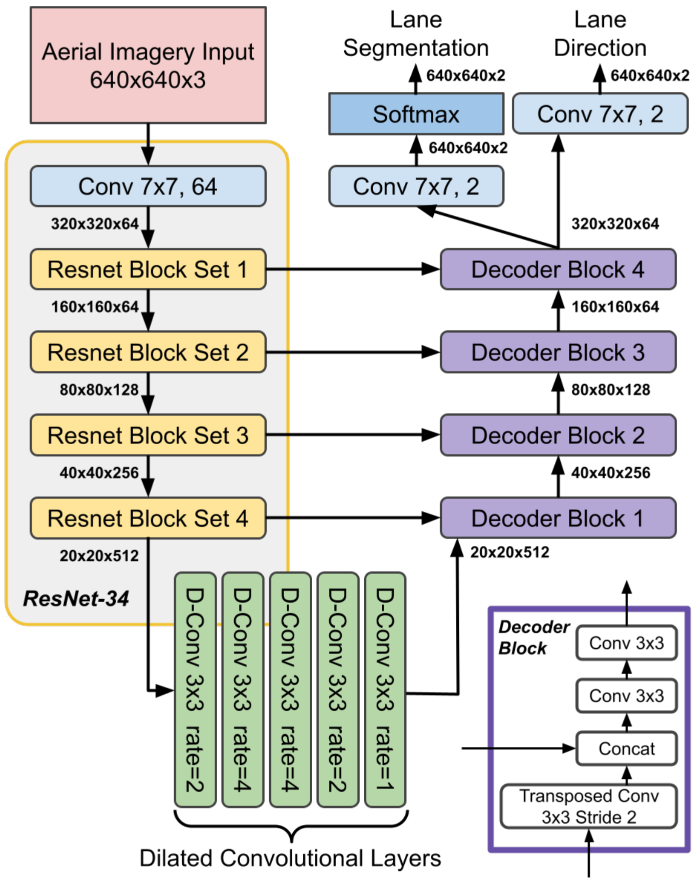


Figure 4-3: Lane Extraction Model

map $\hat{d}$ by rendering edges of the lane graph with a width of 5 pixels, which is 0.625-meter wide as the imagery in our evaluation dataset has a ground sampling distance (GSD) of 0.125 meters/pixel.

**Training.** We set the input window size to $640 \times 640$ during training. We intend to use this large window size so that the model can learn to use global information. For example, if trees occlude a short lane segment, the model should still extract it by using nearby information.

Inspired by the joint orientation learning work [17], we train the lane segmentation branch and the lane direction branch jointly as Batra et al. [17] have shown that learning road extraction jointly with road orientation can achieve better connectivity.

For the lane segmentation branch, inspired by the SpaceNet challenge [45], we use a linear combination of the cross-entropy loss ($\mathcal{L}_{ce}$) and the soft dice loss ($\mathcal{L}_{dice}$) as the loss function. For the lane direction branch, we use $\mathcal{L}_2$ loss. Therefore, the overall loss function is,

$$\mathcal{L} = \mathcal{L}_2(d, \hat{d}) + \frac{1}{2}(\mathcal{L}_{ce}(s, \hat{s}) + \mathcal{L}_{dice}(s, \hat{s})) \tag{4.1}$$

**Inference.** We need to apply our model to large aerial images whose size is often over thousands of pixels during inference. To run the lane extraction model on such large input images, we use a 2-dimension sliding window approach where the sliding window size is $640 \times 640$. Each time we move this sliding window by 256 pixels either vertically or horizontally to cover the entire input image. Because we move the sliding window 256 pixels each time, the inference results from different sliding window instances may overlap on each other. When this happens, we use the average result from different sliding window instances.

**Lane graph extraction.** After we extract the lane segmentation and the lane direction map from the input aerial image, we first extract the lane graph (undirected) from the lane segmentation. Similar to other segmentation-based road extraction work, we binarize the output lane segmentation with a threshold (e.g., 0.5), create a skeleton from this binary segmentation mask using morphology thining, and turn the skeleton into a graph.

Next, we use the direction map output to assign directions to the lane graph. As shown in Figure 4-2 (d), we first decompose the lane graph into individual lane segments. Each lane segment consists of a sequence of edges $[e_1, ..., e_m]$ ($e_i$ connects to $e_{i+1}$). Because the direction map output can be very noisy, we use the entire lane segment to decide its direction. Formally, we compute the following value for each lane segment.

$$c = \sum_{i=1}^{m} \sum_{(x,y) \in P(e_i)} \langle D(e_i), d_{x,y} \rangle \tag{4.2}$$

, where $P(e) = \{(x, y) | \text{Edge e intersects location (x,y)}\}$, $D(e)$ is the normalized direction vector of edge $e$, and $\langle \, , \rangle$ is the vector inner product operator. If $c$ is greater than zero for a lane segment $[e_1, ..., e_m]$, that indicates the direction of the lane segment should be from $e_1$ to $e_m$; otherwise, the direction should be the opposite — from $e_m$ to $e_1$.

### 4.2.2 Turning Lane Extraction

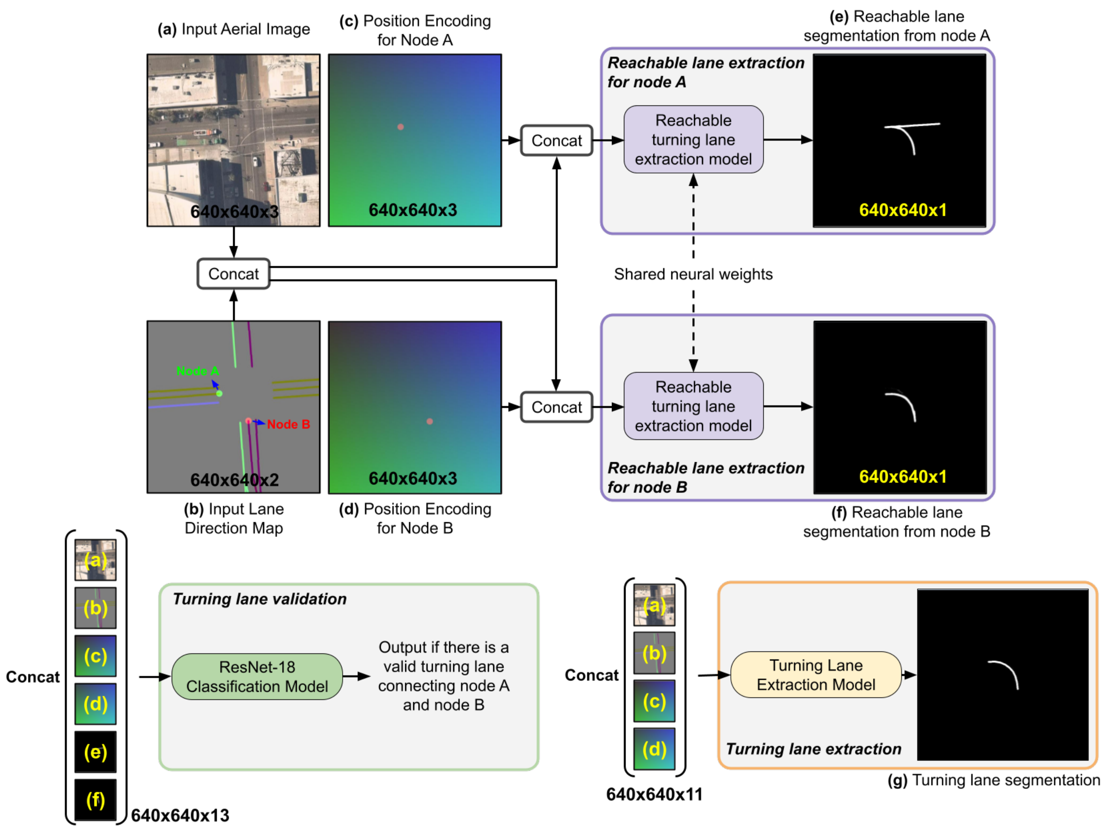

(a) Input Aerial Image
640x640x3
(c) Position Encoding for Node A
640x640x3
(e) Reachable lane segmentation from node A
Reachable lane extraction for node A
Reachable turning lane extraction model
640x640x1
Concat
Concat
Shared neural weights
Node A
Node B
640x640x2
(b) Input Lane Direction Map
640x640x3
(d) Position Encoding for Node B
Concat
Reachable turning lane extraction model
Reachable lane extraction for node B
640x640x1
(f) Reachable lane segmentation from node B
Concat
(a)
(b)
(c)
(d)
(e)
(f)
640x640x13
Turning lane validation
ResNet-18 Classification Model
Output if there is a valid turning lane connecting node A and node B
Concat
640x640x11
Turning Lane Extraction Model
Turning lane extraction
(g) Turning lane segmentation

Figure 4-4: Given a pair of terminal nodes A and B, we use the above architecture to infer if there is a valid turning lane connecting node A and node B. If so, we extract the turning lane vectors using a segmentation approach.

After the first stage of the mapping pipeline, we get a lane graph covering all the non-intersection areas, and this lane graph has many terminal nodes at intersections. In the second stage, we extract the turning lanes at intersections by examining all the terminal node pairs whose distances are below a threshold, i.e., 70 meters. We consider those terminal node pairs as candidate turning lanes. For each terminal node pair $(v_A, v_B)$, we use neural network models to validate the turning lane going from node A to node B and extract the turning lane vectors. In Figure 4-4, we show the detailed workflow.

**Input.** For each terminal node pair $(v_A, v_B)$, we consider a window of size $N \times N$ centered at the midpoint between node A and node B. We use the aerial image, the direction map (extracted in stage-1) in this window as inputs. Meanwhile, for each node, we create an auxiliary input $p \in \mathbb{R}^{N \times N \times 3}$ whose first channel is a binary mask indicating the position of the node, and the second and third channels encodes the offsets from the node. Formally, suppose the position of the node is at $(n_x, n_y)$, we define $p$ as,

$$\begin{aligned} p_{x,y,1} &= 1 \text{ if } x = n_x, y = n_y, \text{ otherwise } 0 \\ p_{x,y,2} &= \frac{1}{N}|x - n_x| \quad p_{x,y,3} = \frac{1}{N}|y - n_y| \end{aligned} \tag{4.3}$$

We use $p$ to provide auxiliary position information for the neural network models so that the model can reason the relative distances effectively.

**Turning lane validation.** To validate a candidate turning lane $(v_A, v_B)$, we first extract the segmentation of the reachable lanes from node A and node B. Here the reachable lanes from a node are the lanes that can be reached on the lane graph through only turning lanes. As examples, we show the corresponding reachable lanes from node A and node B in Figure 4-4 (e) and (f), respectively. We extract the reachable turning lane segmentation using a neural network that has the same architecture (and the same loss function) as Figure 4-3 except the direction prediction branch.

After we extract the reachable lane segmentation, we concatenate it with all other input data and feed them into a binary classification model that predicts if the candidate turning lane is valid or not.

*Alternative-1:* An alternative design is to get rid of the reachable lane extraction modules and directly feed the input data to a binary classification model. Our evaluation finds that this design yields poor accuracy on the testing dataset because the turning lane validation task depends on spatial information such as the relative positions of lanes and lane markers. However, the encoder-only structure in the binary classification model makes it difficult to reason the spatial correlation effectively. In contrast, our design uses the reachable lane extraction model that adapts a UNet (encoder-decoder) structure to help reason the spatial correlation. Therefore, our design can achieve much higher accuracy on unseen data.

**Turning lane extraction.** To extract the turning lane, we first adopt the same neural network model as the reachable lane extraction model to extract the segmentation of the turning lane, and then extract the lane vector from the segmentation (similar to sub-task 1). In our evaluation, we find that this solution is sufficient to achieve very high accuracy.

*Alternative-2:* With this turning lane extraction model, we can derive an alternative design for turning lane validation. In this alternative design, we train a lane extraction model to produce the turning lane segmentation if the input node pair is valid and produce an empty segmentation map otherwise. We check if a path connects the two nodes in the segmentation output to validate the turning lane during testing. This alternative design is much simpler than ours. However, this alternative design also performs poorly because it often produces noisy and disconnected segmentation for valid turning lanes, making it hard to reach a clear decision by only checking the segmentation output.

**Training.** During training, we randomly sample terminal node pairs whose distances are shorter than 70 meters in the ground truth lane graph and use the ground truth labels to generate the corresponding direction maps. For the turning lane validation model, we train it together with the reachable lane extraction model end-to-end. For the turning lane extraction model, we train it with *only* valid terminal node pairs so that the model can always produce a sharp turning lane segmentation.

## 4.3 Evaluation

### 4.3.1 Dataset

We find there are very limited public datasets or resources for lane-level map extraction. One of the related datasets is the Argoverse dataset [31] where they provide two high-definition maps, but they do not provide the corresponding aerial imagery. Hence, to evaluate our proposed mapping pipeline, we create our dataset.

Our dataset covers about 400 km of lanes in four cities, Miami, Boston, Seattle, and Phoenix. For the aerial imagery, we collect them from MapBox [1] and resize the images to 0.125 meters per pixel. For the lane graph labels, we adapt the lane graph labels in the Argoverse dataset for Miami. We tweak the lane graph labels so that they match our aerial imagery. To improve the diversity of the dataset, we manually annotate the lane graphs in three more cities, Boston, Seattle, and Phoenix. We organize the dataset as 35 tiles. The dimension of each tile is 4096 by 4096 (pixels). We use 24 tiles as the training dataset and 11 tiles as the testing dataset. Our dataset is available on GitHub.

### 4.3.2 Metrics

To evaluate the quality of the extracted lane graphs, we adapt two widely used metrics in road extraction problems, the GEO metric and TOPO metric.

**GEO metric.** In the GEO metric, we interpolate (densify) the ground truth lane graph and the extracted lane graph so that the distances between any two connected vertices are 0.25 meters. After the interpolation, we got the ground truth lane graph $\hat{G} = \{\hat{V}, \hat{E}\}$, and the extracted lane graph $G = \{V, E\}$, where $\hat{V}, V$ are the sets of vertices and $\hat{E}, E$ are the sets of edges. We consider a pair of vertices $(v \in V, \hat{v} \in \hat{V})$ as a valid match if the distance between the two vertices is less than $r$ meters. Then, we compute a maximal one-to-one matching between $\hat{V}$ and $V$ and denote the matched vertices in the extracted lane graph as $V_{match}$. Finally, we report the precision $\frac{|V_{match}|}{|V|}$, recall $\frac{|V_{match}|}{|\hat{V}|}$ and $F_1$ score based on the matching result.

The threshold $r$ determines the error tolerance of the metric. In our evaluation, we set it to 1 meter so that the metric can penalize minor errors.

**TOPO metric.** The GEO metric focuses on local correctness, but it does not take connectivity into account. For example, if there is a small missing gap in an extracted lane, the GEO metric will still report a very high recall even though the small missing gap makes the entire lane disconnected. In contrast, the TOPO metric takes connectivity into account. We implement the TOPO metric on top of the GEO metric. For each matched vertex pair $(v, \hat{v})$ in the GEO metric, we consider the subgraphs $S_v$ and $\hat{S}_{\hat{v}}$ on $G$ and $\hat{G}$ where all the vertices in $S_v$ and $\hat{S}_{\hat{v}}$ can be reached from $v$ and $\hat{v}$ by walking on the graph for less than 50 meters, respectively. We then compute the GEO metric between the two sub-graphs $S_v$ and $\hat{S}_{\hat{v}}$, denoted as $\text{Precision}_{\text{GEO}}(S_v, \hat{S}_{\hat{v}})$ and $\text{Recall}_{\text{GEO}}(S_v, \hat{S}_{\hat{v}})$. Finally, we report the TOPO precision and recall defined as,

$$\begin{aligned}
\text{Precision}_{\text{TOPO}} &= \frac{\sum_{\text{matched } (v,\hat{v})} \text{Precision}_{\text{GEO}}(S_v, \hat{S}_{\hat{v}})}{|V|} \\
\text{Recall}_{\text{TOPO}} &= \frac{\sum_{\text{matched } (v,\hat{v})} \text{Recall}_{\text{GEO}}(S_v, \hat{S}_{\hat{v}})}{|\hat{V}|}
\end{aligned} \tag{4.4}$$

Here, we can consider the TOPO metric as a weighted version of the GEO metric where each matched vertex pair only contributes a fraction to the precision and recall based on the correctness of the local connectivity.

**Directed versions.** To evaluate the directed lane graphs, we derive the directed versions of the GEO and TOPO metric. In the directed version, we assign a direction to each vertex based on its edges (we ignore vertices that have more than two neighbors). During matching, we only consider the vertex pairs whose angle differences are less than 60 degrees. We use these directed versions to evaluate lane graphs with driving direction information.

### 4.3.3 Implementation Details

We implement our mapping pipeline in Tensorflow [2] and use Adam optimizer [66] for training. For all the models, we train them for 500 epochs on our training dataset. We start with a learning rate of 0.001 and decrease it by ten at the 350-th epoch and 450-th epoch during training. We augment the input images with random rotations, cropping, color balances, and brightness. Our implementation is available on GitHub.

### 4.3.4 Performance of each Component

In this section, we evaluate each component in our mapping pipeline separately. The components include,

(1) Undirected lane extraction in sub-task 1.

(2) Lane direction inference in sub-task 1.

(3) Turning lane validation in sub-task 2.

(4) Turning lane extraction in sub-task 2.

| Method | GEO metric | | | TOPO metric | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | $F_1$ Score | Precision | Recall | $F_1$ Score |
| Basic UNet | 0.811 | 0.762 | 0.786 | 0.747 | 0.622 | 0.679 |
| Add Dilated Layers | 0.818 | 0.792 | 0.805 | 0.753 | 0.680 | 0.715 |
| With ResNet18 Encoder | 0.828 | 0.812 | 0.820 | 0.768 | 0.708 | 0.737 |
| With ResNet34 Encoder | **0.835** | **0.821** | **0.828** | **0.774** | **0.724** | **0.748** |

Table 4.1: Lane extraction accuracy at non-intersection areas.

**Undirected lane graph extraction.** We report the evaluation result of the undirected lane graph extracted at non-intersection areas (see Figure 4-2(d)). We compare the extracted graph with the ground truth lane graph using GEO and TOPO metric (undirected). In Table 4.1, we report the precision, recall, and $F_1$-scores of our proposed model and several alternatives. As shown in Table 4.1, our proposed solution achieves decent $F_1$ scores in both the GEO metric and TOPO metric. Compared with the alternatives, adding dilated layers and using ResNet encoder can help improve the lane extraction accuracy. Overall, our model improves the GEO $F_1$-score by 4.2 points and improves the TOPO $F_1$-score by 6.9 points against the UNet baseline.

**Lane direction inference.** For each lane in the ground truth graph, we extract its direction from the lane direction map prediction (see Figure 4-2(c)) and compare the extracted direction with the actual direction. If the directions match, we consider the lane as a correct lane. In Table 4.2, we report the lane direction inference accuracy, which is the ratio between the total length of the correct lanes and the total length of all lanes.

As shown in Table 4.2, we find all the models can achieve high overall accuracy (around 94%). If we look at the accuracy on one-way and two-way roads separately, we can find that the accuracy on two-way roads is much higher than the accuracy on one-way roads because inferring the lane direction on one-way roads is very challenging. Usually, not many visible signs from aerial imagery (e.g., arrows) can explicitly tell the lane's direction. The neural network model often has to rely on weak indicators such as the heading of cars and the positions of stop lines to speculate the direction.

| Method | Lane direction inference accuracy | | |
|---|---|---|---|
| | One-way | Two-way | All |
| Basic UNet | 67.68% | 97.81% | 93.95% |
| +Dilated Layers | 66.73% | 98.22% | 94.19% |
| +ResNet18 | **71.40%** | 98.15% | **94.72%** |
| +ResNet34 | 69.61% | **98.23%** | 94.56% |

Table 4.2: In our testing dataset, about 14.7% of lanes (in terms of lane length) belong to one-way roads. This table reports the lane direction inference accuracy for one-way roads, two-way roads, and all roads separately.

**Turning lane validation.** To evaluate the turning lane validation model, we enumerate all the turning lane candidates (a pair of terminal vertices whose distance is below 70 meters) at the intersections in the ground truth lane graph. Suppose we have $N$ turning lane candidates. We check each of them using our lane validation model and produce a label indicating the valid turning lanes during the evaluation. Let $V_{GT}$ be the set of ground truth valid turning lanes, and $V_{Infer}$ be the set of predicted valid turning lanes. Because the counts of valid turning lanes and invalid turning lanes are imbalanced, we do not use accuracy as the metric. Instead, we report the precision, recall, $F_1$-score, and IoU defined as,

$$\begin{aligned} \text{Precision} &= \frac{|V_{GT} \cap V_{Infer}|}{|V_{Infer}|} \quad \text{Recall} = \frac{|V_{GT} \cap V_{Infer}|}{|V_{GT}|} \\ F_1 &= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad \text{IoU} = \frac{|V_{GT} \cap V_{Infer}|}{|V_{GT} \cup V_{Infer}|} \end{aligned} \tag{4.5}$$

| Method | Prec. | Rec. | $F_1$ | IoU |
|---|---|---|---|---|
| Classification only | 0.391 | 0.443 | 0.415 | 0.262 |
| Segmentation only | 0.586 | 0.892 | 0.707 | 0.547 |
| Ours full solution | **0.921** | **0.961** | **0.941** | **0.888** |

Table 4.3: Turning lane validation performance.

We show the results of our proposed solution and two alternatives in Table 4.3. Our solution achieves a high $F_1$-score of 94.1%, which is much higher than the $F_1$-scores of the two alternative solutions (Section 4.2.2) – 41.5% for the classification only alternative and 70.7% for the segmentation only alternative.

**Turning lane extraction.** We compare each extracted turning lane (in graph format) with the ground truth turning lane using the TOPO metric. We find our proposed method has very high accuracy in this task; it achieves an average TOPO precision of 94.2% and an average TOPO recall of 95.8%.

### 4.3.5 Overall Performance

We compare the complete lane graph extracted from our mapping pipeline with the ground truth lane graph using the GEO and TOPO metrics (directed). In Table 4.4, we report the precision, recall, and $F_1$-scores of the extracted lane graphs. We also show the results of the partial lane graphs from two earlier stages in the mapping pipeline. Overall, our mapping pipeline achieves a 76.5% $F_1$-score in the GEO metric and a 62.7% $F_1$-score in the TOPO metric. We consider this a decent result because the matching threshold in our GEO metric and TOPO metric is only one meter.

| Lane graphs at different pipeline stages | GEO metric | | | TOPO metric | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | $F_1$ Score | Precision | Recall | $F_1$ Score |
| Lane graph at non-intersection areas (undirected) | 0.835 | 0.821 | 0.828 | 0.774 | 0.724 | 0.748 |
| Lane graph at non-intersection areas (directed) | 0.800 | 0.787 | 0.793 | 0.745 | 0.697 | 0.720 |
| Final lane graph (directed) | 0.770 | 0.760 | 0.765 | 0.612 | 0.642 | 0.627 |

Table 4.4: Accuracy of the lane graphs at different pipeline stages.

### 4.3.6 Qualitative results

We show examples of the extracted lane graphs in Figure 4-5 (a-d). In those examples, our mapping pipeline correctly extracts the lane graphs in many challenging scenarios including lane diverging, left-only and right-only turning lanes in Figure 4-5 (a), four-lane four-way intersection in Figure 4-5 (b), two-lane four-way intersections in Figure 4-5 (c), and three-way intersections in Figure 4-5 (d).

**Failure example.** In Figure 4-5 (e), we show an example of the output lane graph that has several errors. As shown in Figure 4-5 (e), our mapping pipeline fails to extract the correct lane geometry and direction at non-intersection areas in the red rectangles c,d, and e, which in turn messes up the turning lane extraction in the right two intersections. We observe similar errors on small roads where the lane markers are missing, unclear, or the entire roads are occluded.

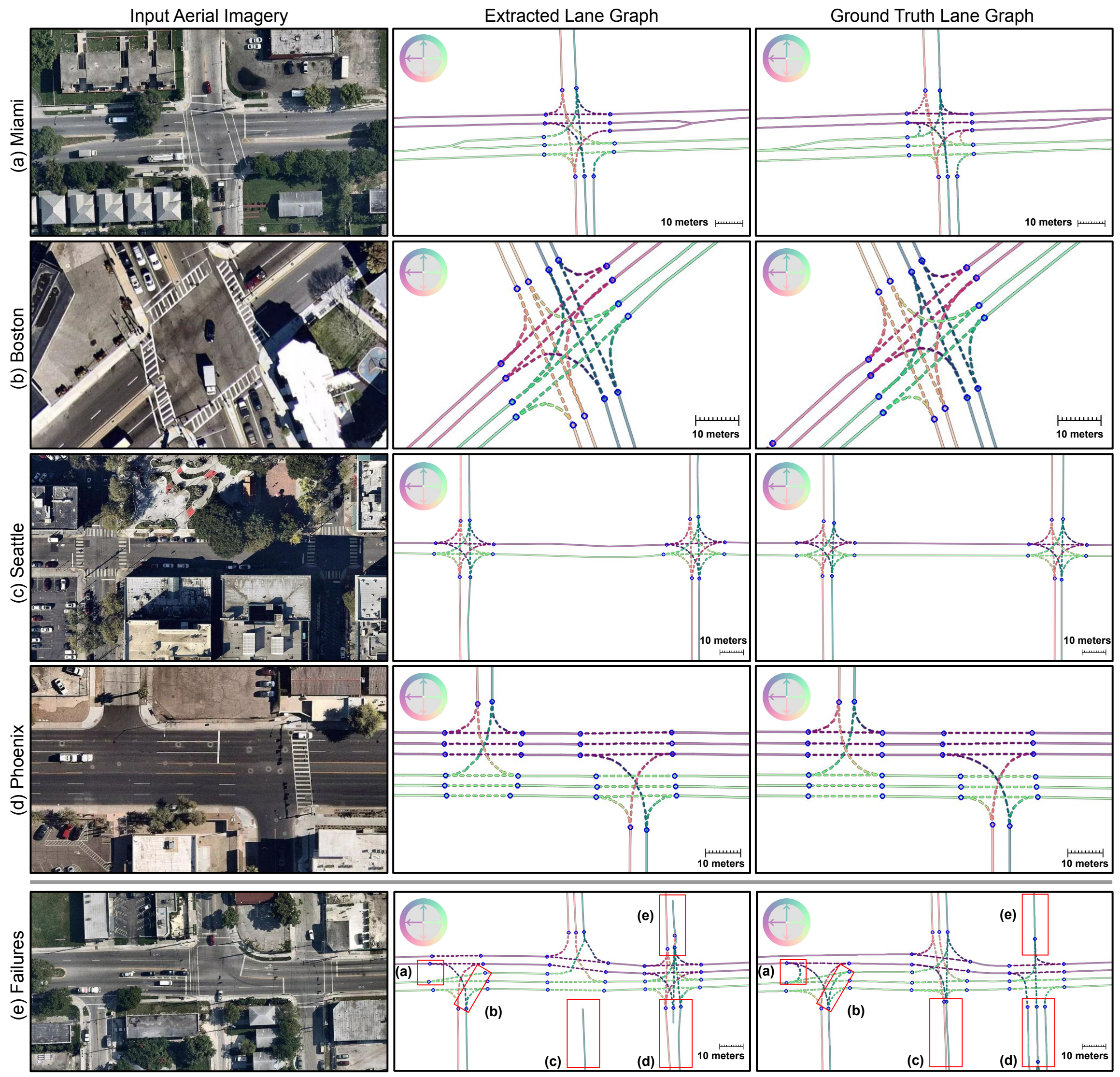


Figure 4-5: Examples of the extracted lane graph from our testing dataset.

Our model also fails to infer the u-turn lane (red rectangle a) and an *unusual* right turn lane (red rectangle b). These two failure cases are examples of a fundamental challenge in aerial-imagery-based lane graph extraction – it is often hard or even impossible to infer the correct turning lanes at some intersections without ground road sign information.

**Limitation.** The above failures unveil a major limitation of our mapping pipeline — the quality of the extracted lane graph depends on the visibility of important road features such as lane markers. This limitation motivates a future work to estimate a confidence score for the extracted lane graph based on the visibility of roads. With this confidence estimator, we can still use our mapping pipeline to extract lane graphs on high-confidence areas, e.g., the places in Figure 4-5 (a-d), and complement other mapping solutions.

## 4.4 Conclusion

In this work, we propose a mapping pipeline to extract routable lane-level street maps from aerial imagery. To the best of our knowledge, this is the first work that proposes a complete mapping pipeline for routable lane-level street map extraction from aerial imagery. We evaluate our solution on a dataset consisting of 400 km of lanes, showing the effectiveness of our solution and unveiling several challenges in the problem. Overall, we show that extracting lane-level street maps from aerial imagery is a promising direction toward scalable data-driven map making.

# Chapter 5

# Road Attributes Inference

Detailed road attributes enrich maps and enable numerous new applications. For example, mapping the number of lanes on each road makes lane-to-lane navigation possible, where a navigation system informs the driver which lanes will diverge to the correct branch at a junction. Similarly, maps that incorporate the presence of bicycle lanes along each road enable cyclists to make more informed decisions when choosing a route. Additionally, maps with up-to-date road conditions and road types improve the efficiency of road maintenance and disaster relief.

Unfortunately, producing and maintaining digital maps with road attributes is tedious and labor-intensive. In this paper, we show how to automate the inference of road attributes from satellite imagery.

Consider the problem of determining the number of lanes on the road from images. A natural approach would be to map this problem to an image classification problem. Because the number of lanes of one road may vary, we can scan the road with a sliding window and train a classifier to predict the number of lanes in each window along the road independently. After we have the classifier predictions in each window, we can apply a post-processing step to improve the prediction results; e.g., using the road network graph to remove inconsistent predictions along the road. Some prior map inference papers adopt such a strategy [25, 78].

This approach suffers from a fundamental limitation: the *limited effective-receptive-field of image classifiers.* Consider Figure 5-1(a), where lane markings are visible only

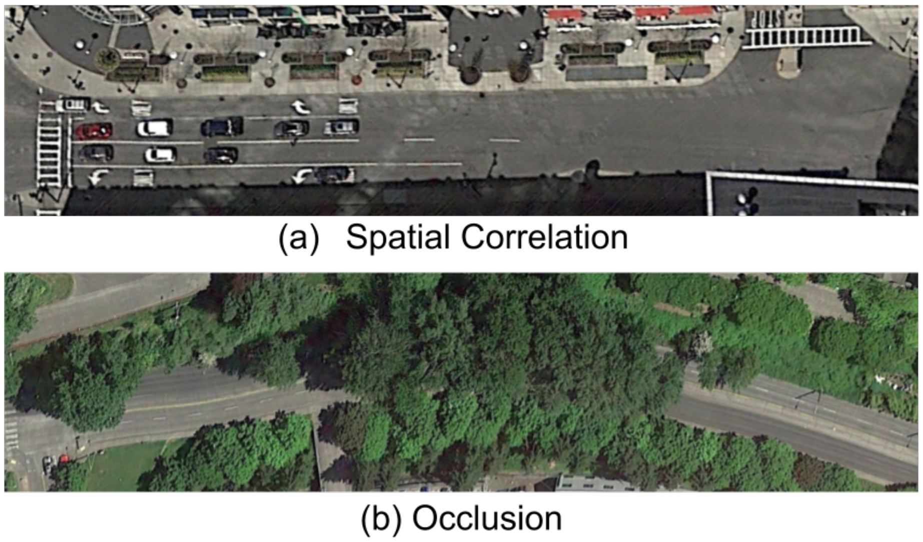


Figure 5-1: Challenges in road attribute inference. In (a) the lane markings are absent on one side of the road. In (b) the road is occluded by trees and the lane markings are also partially missing. To make correct predictions at all positions on the road, we need to incorporate both the local information and the global information along the road network graph.

on the left. Because the road width remains the same, these lane markings imply the remainder of the road on the right has the same number of lanes, despite not having explicit markings. However, because practical image classifiers can only be scalably trained with small windows of the city-wide satellite imagery as input, a window-based image classifier cannot capture this spatial correlation and would not correctly predict the number of lanes in the right portion of the road. Thus, the limited effective-receptive-field of the classifier does not capture the long-term spatial propagation of the image features needed to accurately infer road attributes.

To overcome this limitation, prior work [73] proposes adding a global inference phase to post-process the output from the local classifiers. We find that this fix is inadequate. For example, see Figure 5-2(a), where the lane count changes from 4 to 5 near an intersection. The image classifier outputs partially incorrect labels. However, the post-processing strategy (Markov Random Field) cannot fix this problem as the global inference phase only takes the predictions from the image classifier as input and it may not be able to tell whether the number of lanes indeed changes or it is an error

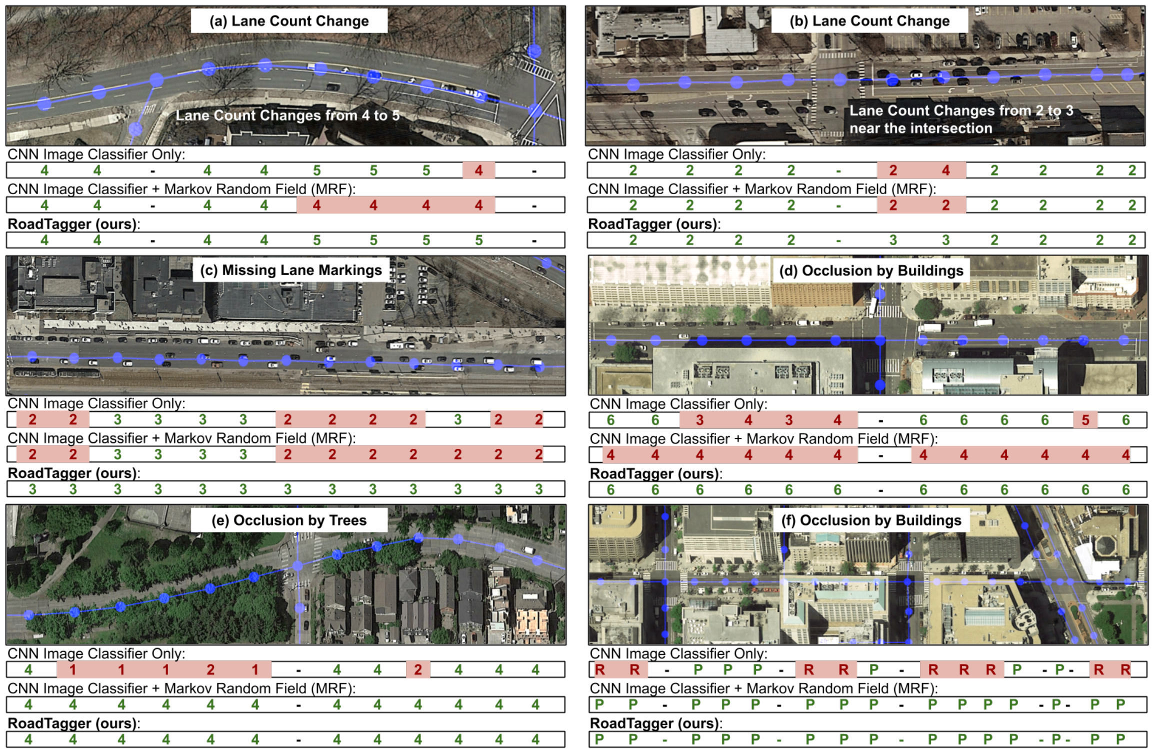

(a) Lane Count Change
Lane Count Changes from 4 to 5
CNN Image Classifier Only:
4 4 - 4 4 5 5 5 4 -
CNN Image Classifier + Markov Random Field (MRF):
4 4 - 4 4 4 4 4 4 -
RoadTagger (ours):
4 4 - 4 4 5 5 5 5 -
(b) Lane Count Change
Lane Count Changes from 2 to 3 near the intersection
CNN Image Classifier Only:
2 2 2 2 - 2 4 2 2 2 2
CNN Image Classifier + Markov Random Field (MRF):
2 2 2 2 - 2 2 2 2 2 2
RoadTagger (ours):
2 2 2 2 - 3 3 2 2 2 2
(c) Missing Lane Markings
CNN Image Classifier Only:
2 2 3 3 3 3 2 2 2 2 3 2 2
CNN Image Classifier + Markov Random Field (MRF):
2 2 3 3 3 3 2 2 2 2 2 2 2
RoadTagger (ours):
3 3 3 3 3 3 3 3 3 3 3 3 3
(d) Occlusion by Buildings
CNN Image Classifier Only:
6 6 3 4 3 4 - 6 6 6 6 5 6
CNN Image Classifier + Markov Random Field (MRF):
4 4 4 4 4 4 - 4 4 4 4 4 4
RoadTagger (ours):
6 6 6 6 6 6 - 6 6 6 6 6 6
(e) Occlusion by Trees
CNN Image Classifier Only:
4 1 1 1 2 1 - 4 4 2 4 4 4
CNN Image Classifier + Markov Random Field (MRF):
4 4 4 4 4 4 - 4 4 4 4 4 4
RoadTagger (ours):
4 4 4 4 4 4 - 4 4 4 4 4 4
(f) Occlusion by Buildings
CNN Image Classifier Only:
R R - P P P - R R P - R R R P - P - R R
CNN Image Classifier + Markov Random Field (MRF):
P P - P P P - P P P - P P P P - P - P P
RoadTagger (ours):
P P - P P P - P P P - P P P P - P - P P

Figure 5-2: Examples of lane inference (examples (a-e)) and road type inference (example (f)). In each image, blue lines show the road graph. The number of lanes or the type of the road predicted by the CNN Image Classifier (with and without MRF) and RoadTagger on each segment are shown along the bottom of each figure. For road type inference, we use capital P to represent primary roads and capital R to represent residential roads. We color the output numbers and letters green for correct predictions and red for incorrect predictions.

of the image classifier. This limitation is caused by the *information barrier* induced by the separation of local classification and global inference; the global inference phase can only use the image classifier's prediction as input, but not other important information such as whether trees occlude the road or whether the road width changes.

We propose RoadTagger, an end-to-end road attribute inference framework that eliminates this barrier using a novel combination of a Convolutional Neural Network (CNN) and a Graph Neural Network (GNN) [100]. It takes both the satellite imagery and the road network graph as input. For each vertex in the road network graph, RoadTagger uses a CNN to derive a feature vector from a window of satellite imagery around the vertex. Then, the information from each vertex is propagated along the road network graph using a GNN. Finally, it produces the road attribute prediction at each vertex. The GNN eliminates the effective-receptive-field limitation of local image classifiers by propagating information along the road network graph. The end-to-end training of the combined CNN and GNN model is the key to the success of the method: RoadTagger doesn't select features using only the CNN; instead, by backpropagating from the output of the GNN, the information barrier that limited previous post-processing methods is eliminated.

We evaluate the performance and robustness of RoadTagger with both a real-world dataset covering a 688 $\texttt{km}^2$ area in 20 U.S. cities and a synthesized dataset focused on different challenges in road attribute inference. We focus on two types of road attributes: the number of lanes and the type of road (e.g., primary or residential). In the real-world dataset, we show that RoadTagger surpasses a set of CNN-based image classifier baselines (with and without post-processing). Compared with the CNN image classifier baseline, RoadTagger improves the inference accuracy of the number of lanes from 71.8% to 77.2%, and of the road type from 89.1% to 93.1%. This improvement comes with a reduction of the absolute lane detection error of 22.2%. We show output examples in Figure 5-2. On the synthesized dataset, we found that RoadTagger is able to learn complicated inductive rules and is robust to different disruptions.

## 5.1 Related Work

Cadamuro et al. adapt CNN image classifiers to predict road quality from imagery [25]. Najjar et al. use satellite imagery to create a road safety map of a city [78] by adapting a CNN image classifier to assign a safety score to each input satellite image window. Azimi et al. apply a CNN to perform a semantic segmentation for lane markings [11]. However, because these schemes derive labels directly from the CNN, they are only able to infer attributes that pertain to small objects (e.g. a satellite image window or a lane marker), and not attributes over an entire road.

To address this issue, Máttyus et al. propose modeling the problem as an inference problem in a Markov Random Field (MRF) [73]. They develop an MRF model that encodes low-level image features such as edge, pixel intensity, and image homogeneity; high-level image features such as road detector results and car detector results; and domain knowledge such as the smoothness of the road and overlapping constraints. They show that this model can infer various road attributes, including road width, centerline position, and parking lane location. They further extend the approach to take ground images (dashcam) into account [74]. In contrast to this MRF approach, RoadTagger does not require specification of different features and domain knowledge for each road attribute. Instead, RoadTagger extracts the useful image features and domain knowledge through end-to-end learning, making it a powerful and easy-to-use framework for different road attributes.

Recent work has also explored using satellite imagery for fully automated road network inference and building footprint mapping. DeepRoadMapper [72] segments the satellite imagery to classify each pixel as road or not road. It then extracts the road network from the segmentation result and applies additional post-processing heuristics to enhance the road network. RoadTracer [14] trains a CNN model to predict the direction of the road. It starts from a known location on the road and traces the road network based on the direction prediction from the CNN model. Hamaguchi et al. [51] use an ensemble of size-specific CNN building detectors to produce accurate building footprints for a wide range of building sizes.

## 5.2 RoadTagger

RoadTagger uses both a CNN and a Gated Graph Neural Network (GGNN) [68] to infer road attributes from satellite imagery and the corresponding road network graph. We assume here the road network graph is already available and accurate, since prior work shows how to infer the graph from satellite imagery [14, 72] or GPS traces [21, 6, 44, 41, 28, 91, 55].

Figure 5-3 shows an overview of RoadTagger. The first step is to densify the road

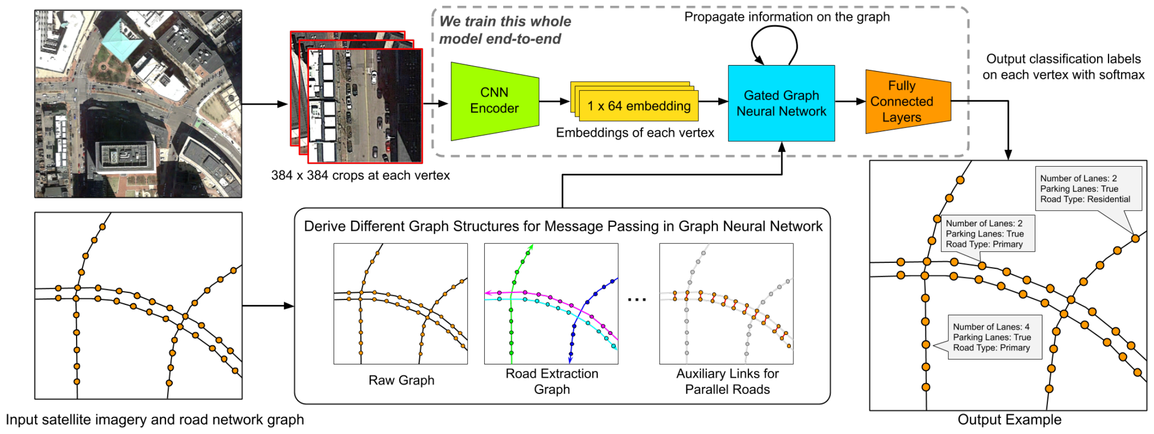

We train this whole model end-to-end
Propagate information on the graph
Output classification labels on each vertex with softmax
CNN Encoder
1 x 64 embedding
Embeddings of each vertex
Gated Graph Neural Network
Fully Connected Layers
384 x 384 crops at each vertex
Derive Different Graph Structures for Message Passing in Graph Neural Network
Raw Graph
Road Extraction Graph
Auxiliary Links for Parallel Roads
Input satellite imagery and road network graph
Number of Lanes: 2
Parking Lanes: True
Road Type: Residential
Number of Lanes: 2
Parking Lanes: True
Road Type: Primary
Number of Lanes: 4
Parking Lanes: True
Road Type: Primary
Output Example

Figure 5-3: The overview of RoadTagger road attribute inference framework.

network graph so that there is one vertex every 20 meters. Then for each vertex in the graph, get the satellite imagery in the area around the vertex to a CNN encoder, and rotate it so that the road direction is always vertical in the image. Each image is $384 \times 384$ pixels, corresponding to a $48 \times 48$ meter tile at 12.5 cm/pixel image resolution. This high resolution is needed to capture details on the road such as lane markings. The CNN encoder uses 12 convolutional layers and 3 fully-connected layers to extract a 64-dimension embedding for each vertex. The method then passes the embeddings of all the vertices to the GNN module. The GNN propagates local information around each vertex to neighbouring vertices on the road network graph. After a few steps of information exchanges, the GNN produces the final prediction of each vertex through three additional fully-connected layers and a soft-max layer.

The model can be expressed as

$$y_v = f_{\text{GNN}}(f_{\text{CNN}}(s_v), G) \tag{5.1}$$

where $s_v$ is the input satellite image tile at vertex $v$, $f_{CNN}()$ is the CNN encoder, $G$ is the densified road network graph, $f_{GNN}()$ is the graph neural network module and $y_v$ is the set of output road attribute labels (soft-max) at vertex $v$. We train the model end-to-end with cross-entropy loss using known ground-truth labels at each vertex $v$.

### 5.2.1 Graph Neural Network Module

A Gated GNN is an extension of a Recurrent Neural Network (RNN) on a graph, using a Gated Recurrent Unit (GRU) [38] to propagate information between adjacent vertices of the graph. We represent the embedding of the vertex $v$ as $x_v$. Before applying the GGNN, we extend the dimension of $x_v$ from 64 to 128 through two fully-connected layers (the $f_{\text{raise}}()$ function). Extending the dimension of the original embedding helps ensure that we don't induce an information bottleneck when the information is propagating on the graph. We represent the hidden state at propagation step $t$ of vertex $v$ as $h_v^t$. Then, the basic propagation model on the road network graph $\{V, E\}$ can be written as

$$\begin{aligned}
h_v^0 &= [f_{raise}(x_v)] \\
m_v^t &= f_1(h_v^{t-1}) \\
a_v^t &= \frac{1}{|N(v)|} \sum_{u \in N(v)} m_u^t \\
h_v^t &= f_{GRU}(h_v^{t-1}, a_v^t)
\end{aligned} \tag{5.2}$$

Here, $N(v)$ is a function representing the set of all logical neighbors of $v$ (explained below), $f_1$ is a fully-connected layer, and $f_{GRU}$ is a GRU. The algorithm uses this propagation function to propagate the information on the graph for $T$ steps, finally producing the prediction labels through three additional fully-connected layers and a soft-max layer.

#### Graph Structures

In the propagation model (5.2), the choice of edge placement defined by $N(v)$ is critical to the inference performance; because it controls how the vertices in the graph communicate with each other.

We investigate different *graph structures* derived from the original road network graph to define $N(v)$. These graph structures share the same set of vertices as the original road network graph but may have different edges defined by $N(v)$. Because the graph structure controls the propagation of information, we can use it to restrict communication to only a subset of the graph or enable communication between two vertices that were not connected in the original road network graph. A good graph structure can improve the performance of the graph neural network. We now discuss four example graph structures:

1. *Original Graph.* The structure enables communication between all connected vertices in the road network graph. It allows the GNN to learn the best way of communication without any restriction or preference. However, this freedom also makes it less efficient to learn certain types of road attributes; such as for road-specific attributes, when two roads with different road-specific attributes interact at an intersection, the graph neural network model need to make sure the information from one road won't mess up with the information from the other road.

2. *Road Extraction Graph.* This structure helps propagate messages only within the same road. To automate this process, extract road chains belonging to the same logical road from the original graph. Here, we call a sequence of connected edges a "road chain belonging to the same logical road" when the directional difference between any two consecutive edges is less than 60 degrees. We show examples of road chains with different colors in Figure 5-3. We find that this restriction is helpful for the propagation of road-specific information as it removes the ambiguity at intersections.

3. *Road Extraction Graph (Directional).* We can decompose one road chain into two chains with opposing directions. This decomposition yields two separate graph structures. By providing two graph structures with opposite edge direction to the GNN, we can explicitly specify the source of each message. This modification can help the GNN learn more efficiently.

4. *Auxiliary Graph for Parallel Road Pairs.* The original graph adapts two separate vertex chains to represent a parallel road pair. This representation prevents

communication between the two roads in a parallel road pair. We enable this communication by adding auxiliary edges between the two roads in a parallel road pair. We show an example of the auxiliary edges in Figure 5-3. The intuition here is that roads belonging to the same parallel road pair often share the same road attributes, e.g., road type. If one road in a pair of parallel roads is occluded by buildings or trees in the satellite image, RoadTagger can still infer the road attributes correctly by incorporating information from the other road through the auxiliary edges.

We find all these graph structures improve RoadTagger in different ways. Thus, we extend the propagation model (5.2) to support multiple graph structures. Instead of aggregating messages from different graph structures together, we treat them separately. To support $k$ different graph structures, we extend the dimension of the hidden state from $m$ to $k \times m$, where the messages from the $i$-th graph structure are stored in the $i$-th $m$-dimensional chunk in the hidden state vector. Although messages from different graph structures are stored separately, they can still interact with each other in the GRU ($f_{GRU}()$).

### 5.2.2 Training RoadTagger

Training a deep model with both a CNN and a GNN is not straightforward. We found that training RoadTagger with standard cross-entropy loss and common anti-overfitting techniques such as data augmentation and dropout was insufficient. This is because, compared with CNN-based image classifiers on the same training dataset, RoadTagger usually has more parameters, but perceives less diversity.

Moreover, nodes in RoadTagger have a larger set of inputs than the local information in CNNs due to the usage of graph neural networks. This extra information makes RoadTagger even easier to overfit. Unfortunately, there is no well-known regularization mechanism that can be applied during training procedure to prevent RoadTagger from overfitting to this extra information.

In order to prevent RoadTagger from overfitting, we explore two training techniques, *random vertex dropout* and *graph Laplace regularization* [89].

**Random Vertex Dropout.** We represent the embedding of vertex $v$ as $e_v$. During training, we randomly pick up 10% of vertices and set their embedding to a random vector $e_v^*$ where,

$$e_v^* = e_v \odot r, \quad r_i \sim \mathcal{U}([-1, 1]) \tag{5.3}$$

Here, we use $\odot$ to denote element-wise multiplication. We stop the gradient back-propagation for the dropped vertices. This random vertex dropout is an extension of the standard dropout. However, instead of setting the embeddings to all zeros, we set them to a random vector. This is because an all-zeros vector is too easy for the neural network to distinguish. We aim to use this random vertex dropout to simulate scenarios where the road is partially occluded by trees, buildings or bridges. This can increase the diversity of the training dataset and thus reduce over-fitting.

**Graph Laplace Regularization.** As we mentioned, there is no regulation mechanism in our training procedure to prevent RoadTagger from abusing the extra information. To overcome this limitation, we add a regularization term based on graph Laplace regularization to the final loss function. We represent the final soft-max output for vertex $v$ as a $n$-dimensional vector $\mathbf{y_v}$, where $n$ is the number of classes of the road attribute. Then, the regularization term for vertex $v$ can be written as,

$$\mathtt{L_{reg}}(v) = \lambda(v) |\mathbf{y_v} - \frac{1}{|N(v)|} \sum_{u \in N(v)} \mathbf{y_u}|^2 \tag{5.4}$$

where $\lambda$ is the weight of the regularization term at different vertices. We set $\lambda(v)$ to zero if vertex $v$ and its neighbours $N(v)$ have inconsistent ground truth labels. Otherwise, we set $\lambda(v)$ to a constant weight factor.

This additional term forces RoadTagger to generate consistent labels for neighbouring vertices regardless of their correctness. It acts as a regularization term for the cross-entropy loss, which only focuses on per-vertex correctness; thus, it reduces overfitting.

## 5.3 Evaluation

In this section, we evaluate the performance and robustness of RoadTagger. We compare RoadTagger against CNN image classifier based solutions. In the evaluation, we focus on the architecture comparison between RoadTagger's end-to-end CNN+GNN framework and the CNN only image classifier solution. We use the same configuration for all the convolutional layers and fully connected layers except the last one in both RoadTagger's CNN encoder and the CNN image classifier. We also use the same input satellite image size for both RoadTagger and the CNN image classifier.

In the evaluation, we demonstrate RoadTagger's performance improvement in road attribute inference in a large scale real world environment. We evaluated the performance of different variants of RoadTagger . In addition, we show when and why RoadTagger can yield better performance and analyze its limitations via a robustness study on a synthesized dataset.

### 5.3.1 Dataset

We conduct our evaluation on two datasets, one real-world dataset and one synthetic micro-benchmark. For the real-world dataset, we collect the road attributes (ground truth labels) from OpenStreetMap [50] and the corresponding satellite imagery through the Google static map API [48]. This dataset covers 688 $km^2$ area in 20 U.S. cities. We manually verified the labels of 16 $km^2$ of the dataset from four representative cities: Boston, Chicago, Washington D.C., and Seattle. We use one third of the verified dataset as validation dataset and two thirds of it as testing dataset. We use all the remaining dataset as training dataset.

We focus on inferring two types of road attribute: the number of lanes and the types of roads (residential roads or primary roads). We use these two types of road attributes as representatives because they both have spatial correlation such that nearby segments tend to have the same labels.

### 5.3.2 Implementation Details

We implemented both RoadTagger and the CNN image classifier using Tensorflow [3]. We use both input image augmentation and dropout to reduce over-fitting. For RoadTagger, we set the graph Laplace regularization weight to 3.0 and set the propagation step to 8 in the graph neural network. We train the model to predict both the number of lanes and the type of the road simultaneously. We train the model on a V100 GPU for 300k iterations with a learning rate starting from 0.0001 and decreasing by 3x every 30k iterations. For both models, we use a batch size of 128. In RoadTagger, at each iteration, we pick up a random vertex in the road network graph and start a DFS or BFS from it. We use the first 256 vertices in the search result to generate a sub-graph as input graph to RoadTagger. We only consider a random 128 vertices from the 256 vertices in the loss function. For both models, we use batch normalization to speed up training. We use the model that performs best on the validation set as the final model.

### 5.3.3 Baselines

We compare RoadTagger against four different baselines, including (1) using only CNN image classifier, (2) using CNN image classifier with smoothing post-processing, (3) using CNN image classifier with Markov Random Field (MRF) post-processing, and (4) using CNN image classifier with larger receptive fields (1.5x and 2.0x).

In the smoothing post-processing approach, we set the probability outputs of each vertex to be the average probability of itself and its neighbouring vertices in the road network graph. This simple post-processing step can remove scattered errors and make the output labels more consistent.

In the MRF post-processing approach, we use a pairwise term in the energy function of MRF to encourage the road segments, which are connected and *belonging to the same logical road*, to have the same label. The energy function of the post-processing MRF is,

$$E(x) = \sum_{i} - \log P(x_i) + \lambda \sum_{\text{connected } i,j} |x_i - x_j|^n \tag{5.5}$$

We find the best hyper-parameters of MRF ($n$ and $\lambda$) through brute-force search on the validation set. At the inference time, we use belief propagation to minimize the energy function $E(x)$.

We also evaluate the CNN image classifier with larger receptive fields. The original CNN receptive field at each vertex is a 48x48 meter tile. We derive new baseline approaches by enlarging the receptive field to a 72x72 meter tile (1.5x) and a 96x96 meter tile (2.0x).

### 5.3.4 Evaluation on Real-World Dataset

| Schemes | # of Lane Acc. | Gain | Road Type Acc. | Gain | ALE | Reduction |
|---|---|---|---|---|---|---|
| CNN Image Classifier **(naive baseline)** | 71.8% | - | 89.1% | - | 0.374 | - |
| - with smoothing post-processing | **74.1%** | **2.3%** | 90.6% | 1.5% | **0.337** | **9.8%** |
| - with MRF post-processing | 73.7% | 1.9% | 92.2% | 3.1% | 0.355 | 5.1% |
| CNN Image Classifier (1.5x receptive field) | 71.8% | 0.0% | 90.1% | 1.0% | 0.367 | 1.9% |
| - with smoothing post-processing | 74.0% | 2.2% | 91.1% | 2.0% | 0.340 | 9.1% |
| - with MRF post-processing | **74.1%** | **2.3%** | **92.9%** | **3.8%** | 0.340 | 9.1% |
| CNN Image Classifier (2.0x receptive field) | 68.8% | -2.0% | 89.1% | 0.0% | 0.393 | -5.1% |
| - with smoothing post-processing | 70.6% | -1.2% | 89.9% | 0.8% | 0.371 | 0.8% |
| - with MRF post-processing | 70.2% | -1.6% | 91.6% | 2.5% | 0.386 | -3.2% |
| **RoadTagger (ours)** | **77.2%** | **5.4%** | **93.1%** | **4.0%** | **0.291** | **22.2%** |

Table 5.1: Performance of RoadTagger and different CNN image classifier baselines. In the table, we highlight both the best and the second best results.

We use the overall accuracy as metrics for both the number of lane prediction and the road type prediction. For the number of lanes prediction, a two-lane road may be incorrectly recognized as a three-lane road or even a six-lane road. However, the overall accuracy metric doesn't penalize more for the wrong prediction with six lanes than the wrong prediction with three lanes. Thus, we use an additional metric, the *absolute lane error (ALE)*, to take the degree of error into account. We represent the output prediction of vertex $v$ as $y_v$ and the corresponding ground truth is $\hat{y_v}$, where both $y_v$ and $\hat{y_v}$ are integers between 1 and 6. Then, the ALE is defined as,

$$\text{ALE} = \frac{1}{|V|}\sum_{v\in V}|y_v - \hat{y_v}| \tag{5.6}$$

We use this absolute lane error (ALE) as a complement to the overall accuracy metric in our evaluation.

### Comparison against Baselines

We report the overall accuracy of the two types of road attributes and the absolute lane error for the CNN image classifier baselines and RoadTagger in table 5.1. We show the result of RoadTagger with its best configuration in this table. As shown in the table, RoadTagger surpasses all the CNN image classifier based baselines. Compared with the baseline using only CNN image classifier, RoadTagger improves the inference accuracy of the number of lanes from 71.8% to 77.2%, and of the road type from 89.1% to 93.1%. This improvement comes with a reduction of the absolute lane detection error of 22.2%. Compared with the best baseline (with MRF post-processing and 1.5x larger receptive field), RoadTagger still improves the accuracy of the lane count inference by 3.1 points, which comes with a reduction of the absolute lane detection error of 14.4%, and achieves similar accuracy in road type inference.

We show output examples of the number of lane prediction and the road type prediction in Figure 5-2. We find RoadTagger performs more robust in many challenging places than the CNN image classifier. This is because the usage of the graph neural network enables RoadTagger to transitively incorporate information from nearby road segments. Meanwhile, during training, unlike the CNN image classifier, RoadTagger treats all vertices on the sub-graph as a whole rather than treating each vertex independently. This doesn't force RoadTagger to learn how to map the image of a building into a two-lane road when the building occludes the road in the training dataset. Instead, RoadTagger can learn a more generic inductive rule to understand the spatial correlations and effect of different visual features (e.g., bridges, trees, intersections, etc) on road attributes. Although post-processing approach can be applied to fix some of the scattered errors, e.g., example (e) and (f) in Figure 5-2, it cannot fix errors which requires more context information such as the errors in examples (a-d) in Figure 5-2. This limitation is due to the *information barrier* induced by the separation of local classification and global inference.

| Scheme | # of Lane | Road Type | ALE |
|---|---|---|---|
| RoadTagger with | - | - | - |
| - Raw | 74.0% | 91.2% | 0.332 |
| - Road | 75.5% | 92.3% | 0.327 |
| - Road(D) | 75.6% | 92.0% | 0.324 |
| - Raw+Road(D)+Aux | 77.2% | 93.1% | 0.291 |

Table 5.2: Impact of different graph structures used in RoadTagger. Here, we use abbreviations to denote different graphs. We use **Raw** for the original road network graph, **Road** for the road extraction graph, **Road(D)** for the road extraction graph with directional decomposition and **Aux** for the auxiliary graph for parallel roads.

### Comparison within RoadTagger

Within RoadTagger, we first compare the performance of RoadTagger with different graph structures. We show results of RoadTagger with different graph structures in Table 5.2.

For a single graph structure, we find adding more restrictions into the graph structure can yield better performance, e.g., the performance of using road extraction graph is better than the performance of using the original raw road network graph. This is because in the road extraction graph, message propagation in the graph neural network is restricted to be within each logical road. This can remove the ambiguity of message propagation at intersections in the original road network graph, thus, improve performance.

RoadTagger supports using multiple graph structures. We find using the combination of the Raw graph, Road(D) graph and Aux graph can yield better performance compared with the performance of using a single graph structure. This is because using multiple graph structures allows our neural network model to learn the best message propagation graph(s) for different attributes end-to-end.

As we mentioned before, we adopt two training techniques to improve the performance of RoadTagger. We show the comparison results in Table 5.3. We find both of these two techniques are critical to the performance improvement of RoadTagger; the random vertex dropout has more impact on the number of lane inference and the graph Laplace regularization has more impact on the road type inference.

| Scheme | # of Lane | Road Type | ALE |
|---|---|---|---|
| RoadTagger | 77.2% | 93.1% | 0.291 |
| No Vertex Dropout | 74.7% | 92.7% | 0.325 |
| No Regularization. | 76.5% | 90.8% | 0.300 |

Table 5.3: Impact of random vertex dropout and graph Laplace regularization.

### 5.3.5 Evaluation on Synthesized Micro-Benchmark

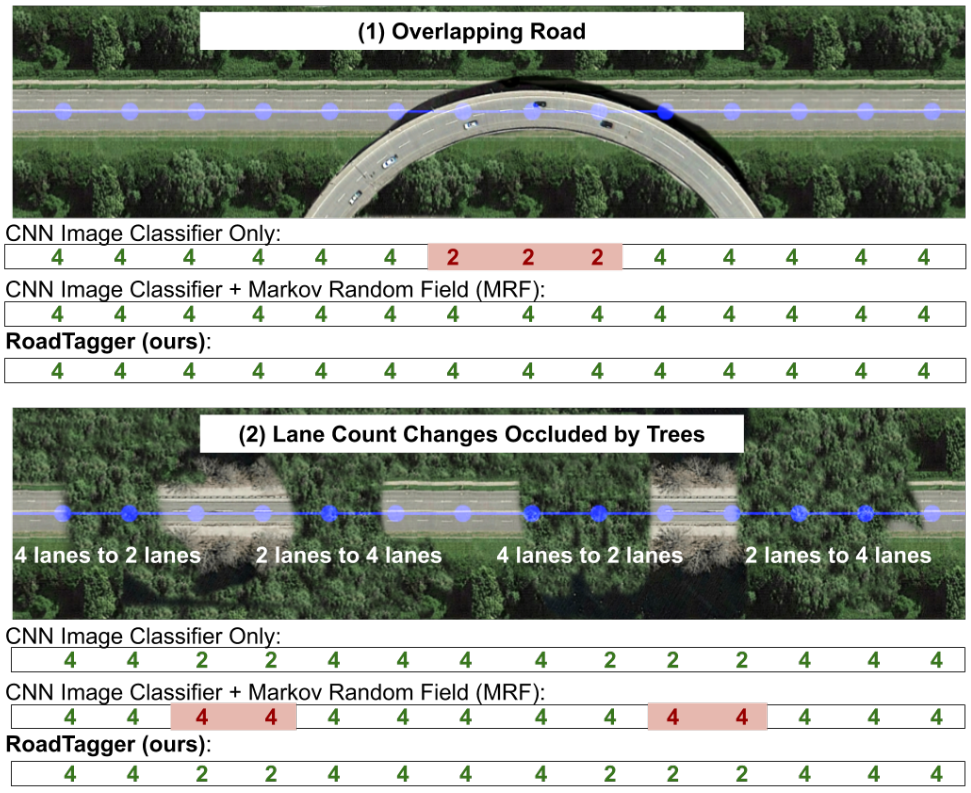


Figure 5-4: Two representative samples of the micro benchmark. RoadTagger predicts both of them correctly.

We conduct an extensive evaluation of RoadTagger on a micro benchmark. In this micro benchmark, we inject different types of challenges to the satellite imagery. We would like to study the impact of occlusions with different types and amounts as well as other challenges such as missing lane markings in a controlled way. In this micro benchmark, we find RoadTagger is robust to a wide range of different disruptions. These disruptions include removing all the lane markings on part of the roads, alternatively occluding the left and right side of the roads, and even occluding the target road with an overpass road.

We show two representative examples of this benchmark in Figure 5-4. Please refer to the Appendix A for the evaluation on the whole micro-benchmark.

We find the two examples shown in Figure 5-4 particularly interesting. In example (a), an overpass road occluded the target road. In example (b), the lane count changes when the road is occluded by trees. To correctly predict the number of lanes in both example (a) and (b), RoadTagger needs to know that when the starting point of an overpass road is detected, the visual features of the overpass should be ignored until the far edge of the overpass is detected. At the same time, RoadTagger needs to know if the road is temporarily occluded by trees, the following road segment still belongs to the same target road. We find RoadTagger's end-to-end architecture enables it to learn all this knowledge correctly without any additional labels or explicit features. This is perhaps the most attractive part of RoadTagger.

## 5.4 Discussion

**Can RoadTagger Generalize to City-Scale Graphs?** In our evaluation, we find RoadTagger can generalize well to city-scale graphs. During inference, RoadTagger labels the whole road network graph (with 3,000 to 4,000 vertices) for each 2km by 2km region in one shot (Use the entire road network graph as input). We find training RoadTagger with 256-node subgraphs can generalize well in larger graphs, e.g., graphs with 3,000 to 4,000 nodes.

**Errors Made by RoadTagger.** We observe two types of errors made by RoadTagger in our evaluation. (1) We find RoadTagger makes wrong predictions for invisible roads (occluded by trees or buildings) when the disruptions are longer than the GNN propagation step (we show examples of this type of failure in the supplementary material). We think this type of error can be eliminated through enlarging the propagation step and the subgraph size (i.e., 256) during training. (2) We find RoadTagger outputs road attributes in ABABA style along the road when the road attribute is ambiguous. We think this issue can be addressed by incorporating GAN into RoadTagger framework.

## 5.5 Conclusion

In this work, we propose RoadTagger. RoadTagger adapts a novel combination of CNN and graph neural network to enable end-to-end training for road attribute inference. This framework eliminates fundamental limitations of the current state-of-the-art that relies on a single CNN with post-processing. We conduct a comprehensive evaluation of the performance and robustness of RoadTagger; the evaluation result shows a significant improvement in both performance and robustness compared with the current state-of-the-art. The result also shows RoadTagger's strong inductive reasoning ability learned end-to-end. We believe RoadTagger framework is a fundamental improvement in road attributes inference and can be easily extended to other road attributes.

# Chapter 6

# Traffic Accident Risk Map Inference

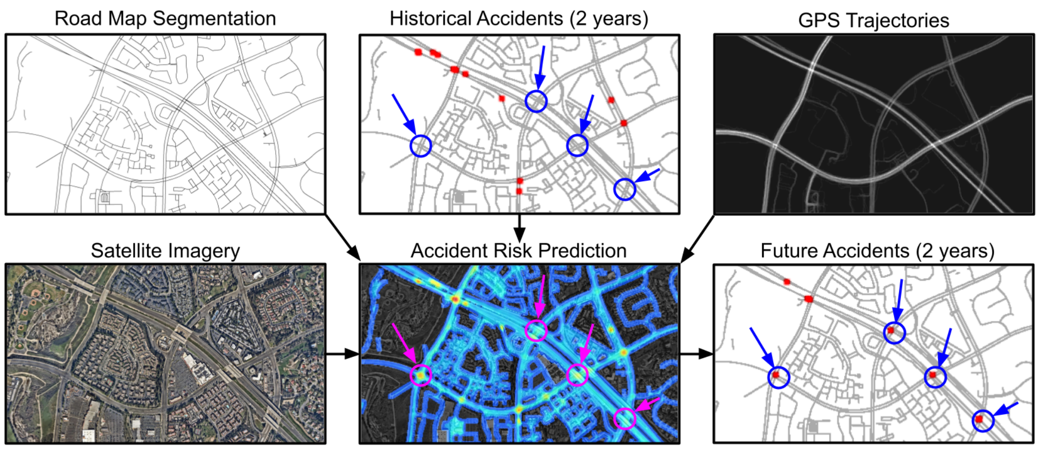


Figure 6-1: Our model inputs road maps, satellite imagery, GPS trajectories, and historical traffic accidents. It outputs accident probability distribution. Note that our model has identified a few locations as high-risk (highlighted with circles) even though they have no historical accidents. Locations that our model has identified as high-risk experienced accidents during the follow-up years.

According to WHO, each year 1.35 million people die and 20 to 50 million people sustain non-fatal injuries from traffic accidents [81]. In the US alone, traffic accidents cost $871 billion annually [23]. In most countries traffic accidents cost about 3% of the GDP [81]. By identifying high-risk locations on the map, many groups, including drivers, police departments, transportation departments and insurance companies can take actions to reduce this risk.

Accident risk maps assign an expected rate of accident over a given time period to each location on the map. Prior works predict accident maps with resolutions of a few hundred meters (Table 6.1.1). In this work we predict maps with 5m$\times$5m resolution because there are important details that are not captured in lower resolutions. At this resolution, sparsity causes a bias-variance trade-off in the estimation of the underlying risk. We explain this challenge in Section 6.1.

We improve this trade-off by incorporating context information from satellite imagery, GPS trajectories, and road maps. We use an end-to-end deep neural network to combine different data modalities. We discuss the details of our model in Section 6.2. Figure 6-1 shows our four input modalities, our prediction, and the accidents in the follow-up years.

At 5m$\times$5m resolution, evaluation is also challenging because the ground-truth is noisy (It is sampled from a hidden risk distribution.) In Section 6.3 we present a process to estimate the prediction error with respect to the true underlying risk distribution. Our maps outperform prior work in terms of resolution and prediction error.

## 6.1 Challenge of Sparsity

In the US, the average annual rate of reported accidents on a 5m$\times$5m block of road is about 1 in 1000. Our analysis on US traffic accidents dataset [76] shows that 31% of the accidents occur in places where no other accidents happened nearby (within 50 meters) within four years. Therefore, Monte Carlo probability estimation will miss some high-risk areas and misidentify low-risk areas as high-risk.

Our goal in accident risk prediction is not to identify exactly where new accidents will occur because this is impossible. Instead, our goal is to identify the underlying risk of accidents at each location, whether accidents occur or not. Ideally, we should use the underlying risk of accidents as ground-truth. However, the underlying risk of accidents is unknown. Therefore, we use a map of future accidents as an alternative ground-truth. The map of future accidents is a Monte Carlo estimation of the underlying rate of accidents, so it carries a large amount of estimation error. This error leads to challenges in both prediction and evaluation.

**Prediction**: Assume that a 5m×5m grid cell has a 1% annual rate of accidents. In one year, the number of accidents at this location would be either 0 or 1. Estimating the true risk of 1% given an input of 0 or 1 is challenging.

**Evaluation**: Since the true underlying rate of accidents is unknown, we can only use an observed rate of accident as a proxy to ground-truth. Therefore, our ground-truth itself carries error and this adversely affects the evaluation.

One way to deal with the challenges of sparsity is to reduce estimation resolution. However, this low resolution causes bias in estimation. As an example of this bias, assume that a dangerous intersection and a safe street are grouped into one single cell. An estimate for the risk in this cell will underestimate the risk of the dangerous intersection and will overestimate the risk of the safe street (Figure 6-2-c). This underestimation and overestimation repeats with varying input, therefore, it is a form of model bias.

## 6.1.1 Prior Work

| Year | Authors | Resolution | Method | Input data |
|---|---|---|---|---|
| 2005 | Chang et al. [30] | Entire highway | Decision Tree | Road map, average daily traffic (AADT), weather |
| 2005 | Chang et al. [29] | Entire highway | Neural Networks | Road map, average daily traffic (AADT), weather |
| 2007 | Caliendo et al. [26] | Entire highway | Max. Likelihood | Road map, AADT, slope and presence of junctions |
| 2016 | Chen et al. [35] | 500m × 500m | SdAE [18] | GPS trajectories, historical accidents |
| 2017 | Yuan et al. [111] | road segments | Deep networks | Historical Accidents, road map, weather |
| 2017 | Najjar et al. [78] | 150m × 150m | Pre-trained Alex-net | Satellite imagery, accident history |
| 2018 | Ren et al. [84] | 1km × 1km | LSTM | Historical accidents |
| 2018 | Chen et al. [32] | 500m × 500m | SdAE [18] | Traffic flow (from plate recognition system), accident history |
| 2018 | Yuan et al. [110] | 5km × 5km | ConvLSTM | Traffic volume, road condition, weather, satellite imagery |
| 2019 | Bao et al. [12] | > 360m | STCL-Net | Crash, GPS, road, land use, population and weather data |
| 2020 | Zhou et al. [120] | 1.5km × 1.5km | RiskSeq | Traffic flow, road network, weather and accident history |
| 2021 | This work | 5m × 5m | End-to-end deep net | Satellite imagery, GPS trajectories, road map, accident history |

Table 6.1: Overview of prior work on traffic and accident map prediction. We predict accident maps with one to two orders of magnitude higher resolution than prior work. We also use richer input data than prior works.

Most prior works use low resolutions to control sparsity (Table 6.1.1). Several works in transportation journals adopt a high resolution, but they are used for visualization purposes and do not evaluate their results.

The general problem of accident prediction is a frequent subject of study [107], but few studies produce accident maps. Most works focus on the effect of certain events (weather, calendar, lighting) on the frequency of accidents [60]. In this work, we study the spatial accident maps and only review major prior works that produce accident maps. These works either produce coarse resolution maps or use kernel density estimation.

**Coarse resolution**: Some works choose a coarse spatial granularity to predict accident risk. The most relevant works include the work by Najjar et al. [78] that uses satellite imagery, and the work by Chen et al. [35] that uses GPS trajectories. Other notable works are listed in Table 6.1.1. Coarse-resolution models miss accident hot-spots and misidentify low risk locations as high risk (Figure 6-2-c).

**Kernel Density Estimation**: Most works in transportation journals use Kernel Density Estimation. KDE applies a Gaussian kernel to historical measurements [93]. Xie et al. [101] use KDE along the roads rather than in the 2D domain. Most KDE works evaluate their performance only visually. The most notable exception is the work by Xie et al. [102] that calculates statistical significance levels. Anderson et al. [9] identify accident hot-spots but do not quantitatively evaluate the performance. Le et al. [67] identify hot-spots and evaluate their ranking. All of these KDE-based works only use historical accidents and a road map to visualize accidents [82, 22, 94, 12, 120]. These works use similar KDE techniques in different cities around the world. Okabe et al. [80], Netek et al. [79] and Shariat et al. [75] implemented KDE in GIS environment. We implemented KDE and compared against it (Figure 6-2).

### 6.1.2 Addressing the Challenge of Sparsity

Prior work based on historical accidents uses variants of Monte Carlo estimation. As such, it only works for places where there is sufficient historical accident data and well-maintained records. To overcome this challenge, we note that places with similar road structures, similar visual appearances, and similar traffic patterns are likely to have similar accident risk profiles. If one intersection experiences as accident, we can share some risks with similar intersections. In order to generalize from one intersection to another, we need some context information that can capture the similarity between

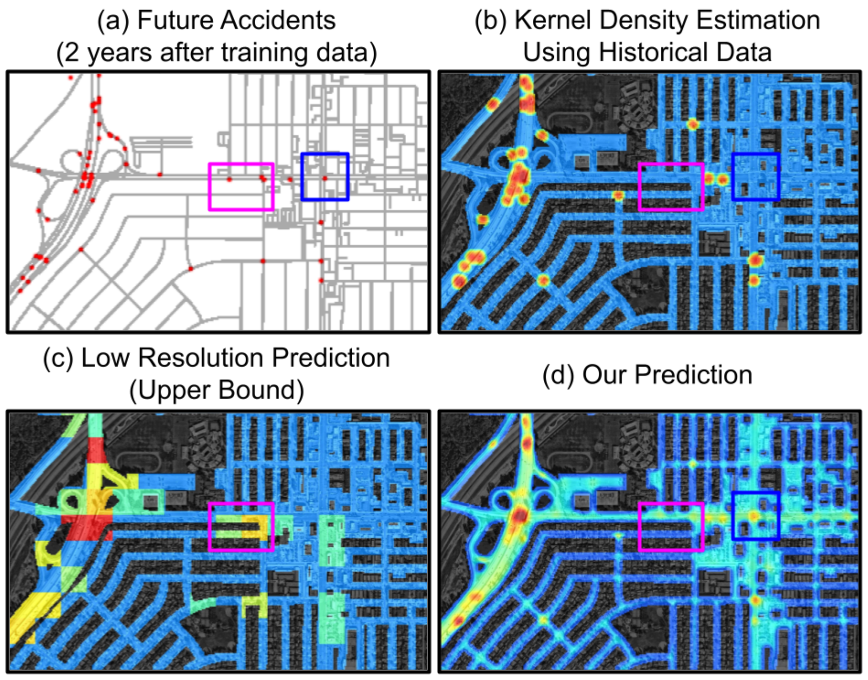

(a) Future Accidents
(2 years after training data)
(b) Kernel Density Estimation
Using Historical Data
(c) Low Resolution Prediction
(Upper Bound)
(d) Our Prediction

Figure 6-2: (b) Kernel density estimation (KDE) generally highlights areas with historical accidents as hot-spots. Therefore, it fails to predict new accidents (compare the blue box) and can only work for high-risk areas. (c) Low-resolution models have a high bias because they may assign the same risk score to a freeway and its neighboring residential road simultaneously (compare the purple box). (d) Our approach can identify high-risk locations that have not experienced accidents in historical data but are likely to experience accidents in the future. Note that the KDE predicts very specific risky locations, many of which do not have accidents in the future. In contrast, our method accurately highlights the roads where future accidents happen, properly attributing more risk to intersections and ramps. We did not visualize historical accidents as they look similar to KDE.

intersections.

In this work, we use context from satellite imagery, GPS trajectories, and road maps. We use a deep model that inputs context and learns to generate useful representations for each position. Our model learns an internal metric based on an accident-based similarity score.

The three data modalities that we used (in addition to historical accident data) provide complementary information. For example, GPS trajectories carry information about the density, speed, and flow of traffic. Satellite imagery carries information about the road, such as the number of lanes, whether there is a road shoulder, and whether there are many pedestrians.

### 6.1.3 Evaluation with Ground-Truth Error

We use future accidents as a proxy to the underlying rate of accidents. This proxy has an error that adversely affects evaluation, therefore we need to isolate it.

Assume a map has $n$ grid cells and there is an underlying rate of accident for each cell $R_i$ that we want to estimate. There is an observation of the historical rate of accidents at each location $H_i$. There is also an observation of the future rate of accident $F_i$. Since accidents are independent events, We can assume $H_i$ and $F_i$ are drawn from a Poisson distribution with rate $R_i$. We can write down the rate of all grid cells write them down in the following vector form:

$$|H - F|_2^2 = |H - R|_2^2 + |F - R|_2^2 + 2(H - R)(F - R). \tag{6.1}$$

Since at each location $i$, $H_i$ and $F_i$ are independent draws from the same Poisson distribution, in expectation, $F$ and $H$ have orthogonal deviations from the distribution mean $R$. Therefore, the last term in Equation 6.1 is negligible in practical settings. Also $H_i$ and $F_i$ have similar expected errors. Simplifying, we get:

$$|F - R|_2^2 = |H - R|_2^2 = \frac{1}{2}|H - F|_2^2. \tag{6.2}$$

Even though we don't know $R$, we can approximate the error of $F$ with respect to $R$.

Our goal is to estimate the underlying risk of accidents $\hat{R}$. Since $R$ is not given, we use $F$ as a proxy and calculate $|\hat{R} - F|_2^2$ as prediction error. There is a similar relation to equation 6.1 between $\hat{R}$ and $F$:

$$|\hat{R} - F|_2^2 = |\hat{R} - R|_2^2 + |F - R|_2^2 + 2(\hat{R} - R)(F - R). \tag{6.3}$$

Note that in equation 6.3 the last term multiplies two residuals from $R$. If these two residuals have any significant correlation, it means that our prediction shares some error with the test set. This is not possible because our model doesn't get feedback from the test set. Therefore, these two residuals should be uncorrelated in practical settings. Using Equations 6.2 and 6.3 we have:

$$|\hat{R} - R|_2^2 = |\hat{R} - F|_2^2 - \frac{1}{2}|H - F|_2^2. \tag{6.4}$$

This way we eliminate the error of $F$ from our evaluation.

If the residuals were I.I.D., the dot product between residuals would reach zero with a rate of $\Theta(\frac{\sqrt{n}}{n})$. Even though the residuals are not I.I.D., in practice, accident maps span a diverse area and accident rates are bounded; therefore we can argue that the last term in Equations 6.1 and 6.3 grow slower than $\Theta(1)$ and reach zero when $n$ is large.

We tried formulating the same logic using maximum-likelihood, KL-divergence and $L_1$-norm. However, the properties of orthogonality that make this analysis work are only available in $L_2$-norm.

## 6.2 Learning to Predict Risk Maps

We use a deep model to predict accident risk at every grid cell on the map. In the design of this model, we need to overcome the challenges caused by sparsity while making sure that our model can learn useful information from all input sources. We illustrate our model architecture in Figure 6-3. The model takes different data modalities as input and predicts a 2D risk map $y \in \mathbb{R}^{N \times N}$, where $N$ is the dimension of the target map grid, and we set it to 48. Next, we discuss the details of our model.

**Model Inputs**: Our model takes five different data sources as input to predict a risk map for an $N \times N$ map gird with a resolution of 5 meters. The first input is an RGB-channel satellite image. We use a higher-resolution ($8N \times 8N$) imagery to capture more visual information. In this case, the satellite image input is represented as an $8N \times 8N \times 3$ tensor. The second input is a segmentation mask of the road map in the target region. Similar to the satellite image input, we use a high resolution for the road mask ($8N \times 8N \times 1$).

Our model also takes GPS trajectories as input. We represent the GPS trajectories in two formats: One is a 2D histogram ($N \times N \times 1$) which encodes the density of the GPS trajectories (at log-scale) on each grid cell. The other format extends the 2D GPS histogram with 12 additional features, encoding the statistics (10-th, 50-th and 90-th percentiles) of the speeds, accelerations, turning angles, and the counts of left/right/no turns of all the GPS trajectories that pass through each grid cell. Combined with the original 2D GPS histogram, this input format contains 13 channels in total, and we represent it as an $N \times N \times 13$ tensor.

The last input source is the historical accident data. We use the rate of historical accidents in each grid cell.

During training, we randomly drop out each input source with a probability of 20%. We find that this strategy is helpful when the training dataset is small. Our model supports using a subset of the above five data sources as input. In this case, we can predict risk maps even if some data sources are not available in the region of study.

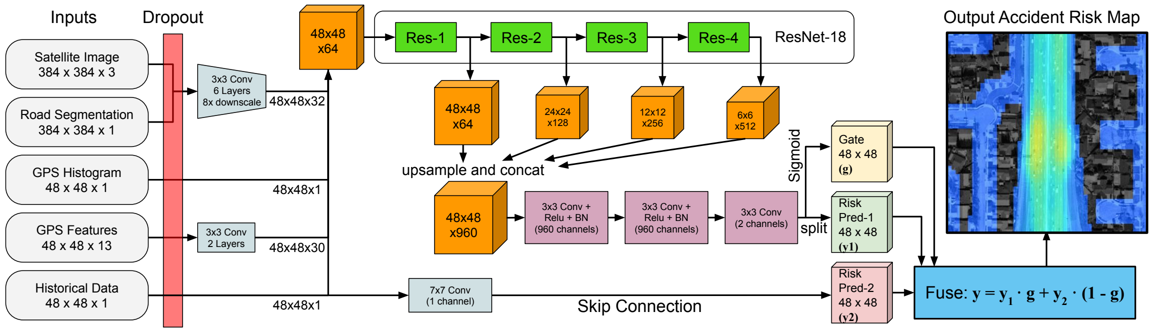


Figure 6-3: We use a deep model that takes four different data sources as input and predicts an accident risk map at 5-meter resolution – only at this high resolution can we distinguish the different risks in the output example where the freeway road has a higher risk than the nearby residential roads and the ramp merging and exiting area has an even higher risk than other places.

**Model architecture**: In our model, we first pre-process input data so that different sources of data all have the same spatial dimensions. We stack the satellite input and the road segmentation input into one tensor and pass it to a 6-layer CNN encoder which down-scales the input dimension and extends the channel width from 4

(3 RGB channels + 1 map channel) to 32. Meanwhile, we use a 2-layer CNN encoder to increase the dimensions of the GPS feature input from 13 to 30. Therefore, if we stack all the input sources together, the total number of channels add up to 64.

After pre-processing, we stack all the input sources into an $N \times N \times 64$ tensor and pass it to a ResNet-18 encoder. We take the feature maps after each residual block set and up-sample them so that they all have the same spatial dimensions. Then, we stack them into a $N \times N \times 960$ feature map and pass this feature map to a 3-layer CNN decoder.

**Skip Connections and Fusion**: We don't use this 3-layer CNN decoder to predict the final risk map directly; instead, we introduce a skip connection and a fusion module to produce the final risk map. We observed that the historical data is very similar to the training target in some high-risk regions. As a result, if we directly produce the risk map using the 3-layer CNN decoder, the model relies on the historical data and ignores other data sources, ending up in a low-performance local optima. To overcome this issue, we let the 3-layer CNN decoder predict two $N \times N$ tensors: a risk map denoted as $y_1$ and a gate $g$ where $g_{i,j} \in (0, 1)$. We use the weighted average of $y_1$ and another risk map prediction $y_2$, which only uses the historical data as the final output. Formally, we have $y = y_1 \cdot g + y_2 \cdot (1 - g)$. This allows our model to focus on learning the residual between the historical data and the target.

**Target and loss function**: We temporally partition accidents into two groups: historical accidents (happened before some time $t$) and future accidents (happened after $t$). A historical accident map is given as input to let the model understand the distribution of accidents. A future accident map is given as the prediction target. We use future accidents as a proxy for the true underlying risk distribution which is unknown. The future accident map is a sparse sample from the true underlying risk distribution. Therefore, it is noisy and not ideal. However, it is useful because the sampling error in the future accident map is not correlated with the sampling error in the historical accident map (because they are independent samples). Therefore, future accident map does not carry a systematic bias from historical accidents, so it

| | LA | NYC | Chicago | Boston |
|---|---|---|---|---|
| Tiles | 813 | 458 | 282 | 319 |
| Accidents | 351k | 88k | 45k | 33k |
| GPS (km) | 3.1M | 1.8M | 0.7M | 2.0M |

Table 6.2: Dataset details in each city. The fact that Boston has fewer accidents explains why more features don't always help.

is a useful proxy. Our loss function is the mean squared error between our prediction and the future accident map.

## 6.3 Evaluation

### 6.3.1 Dataset

We evaluate our model on a dataset covering an area of 7,488 km$^2$ from four metropolitan areas: Los Angeles, New York City, Chicago, and Boston. The dataset is organized as 1,872 2km$\times$2km tiles. For each tile, we collect satellite imagery from MapBox [1] and create the road segmentation mask using OpenStreetMap [49]. Our imagery has a resolution of 0.625 meters. We also construct the road segmentation mask with this resolution.

We use a proprietary GPS dataset collected from 2015 to 2017 in the four metropolitan areas as the source of GPS trajectories. This dataset contains a total of 7.6 million km of GPS trajectories with a 1-second sampling rate.

We use the US accidents dataset [76] that contains 4.2 million records for accidents that were occurred in the US from 2016 to 2020. Each record comes with coordinates, timestamps, and a few other fields of information. We split this accident dataset into two parts containing the data from the first two years and the data from the last two years. We use the first two years' data as historical data to feed into the model as input. We use the last two years' data as future accidents. Future accidents are used for training and evaluation. In table 6.3.1, we summarize the amount of available data in each city that helps to compare the results from the four different cities.

### 6.3.2 Training Details

In our evaluation, we split the dataset spatially into a training set (80%), a testing set (15%), and a validation set (5%). We train our models on the training set for 50 epochs, start with a learning rate of 0.0001 and decrease it by a factor of $\frac{1}{10}$ at the 20-th epoch and the 40-th epoch. The training took 6 days on one Nvidia V-100 GPU. After training, we use the validation set to find the best model and evaluate the model on the testing set. The training/evaluation code, the output accident maps, and the instruction to download the dataset are available on GitHub.

### 6.3.3 Evaluation Settings

We have two major evaluation settings: with history and without history. In the "with history" setting, we supply historical accidents to the model, while in the "without history" setting, we do not supply historical accidents to the model.

Depending on the use case, historical accident data may or may not be available to the model. If historical accident data is available and the goal is to produce an accurate accident map, then accident history data should be used as input. If historical accident data is unavailable, or if this model is being used as a recommender system or to compare hypothetical designs, then historical data cannot be supplied.

We evaluate a few variants for each of the "with history" and "without history" settings. These variants include six variants of our model, kernel density estimation, and theoretical upper-bounds for low-resolution techniques. We compare with the theoretical upper-bounds as a reference to show the effect of the resolution.

In the "with history" evaluation setting, we use two years or accident data as input, while in the "without history" evaluation setting, we do not use historical accidents as input.

### 6.3.4 Evaluation Metrics

In prior works, accident map prediction is formulated either as a binary classification problem or as a regression problem. Classification-based works assign a binary label (whether any accidents happened) to each cell within a time window of interest. Then they predict a score for each cell within the time window of interest. Finally, they compare their prediction scores with the binary ground truth and evaluate their performance using the precision-recall curve and average precision.

Regression-based techniques predict the number of accidents within each cell and the time window of interest. Regression-based techniques often have a low resolution; therefore, several accidents could occur within each cell. Regression-based techniques typically use RMSE between the ground-truth number of accidents in each cell and their estimation to evaluate their regression performance.

We evaluate our model with both AP and RMSE. Figure 6-4-a shows our precision-recall curve for the "with history" model. Figure 6-4-b shows our precision-recall curve for the "without history" model. Table 6.3.4 also compare average precision quantitatively.

Traffic accident estimation techniques that perform regression use RMSE to evaluate their performance. We compared our performance with RMSE in table 6.3.4. AP and RMSE have a few notable differences. First, AP puts a higher weight on high-risk locations than RMSE. Second, AP does not distinguish between one or many accidents in a cell. AP is useful for evaluating performance in high-risk areas. RMSE is useful to evaluate overall performance.

When reading these precision-recall curves, we should note that the prior probability of the prediction target has a large effect on AP statistics. A classification task on a 10m$\times$10m map has four times higher prior than a classification on a 5m$\times$5m map. Therefore, average precision numbers on different resolutions are different and should not be compared. Furthermore, since ground-truth itself is noisy, there is an upper limit on maximum AP.

### 6.3.5 Baselines and Prior Work

Unfortunately, the code for most of the prior work is not available. Furthermore, each prior work has studied one separate city with private data. We use the US accidents dataset [76] that is a large scale and publicly available dataset covering the entire US.

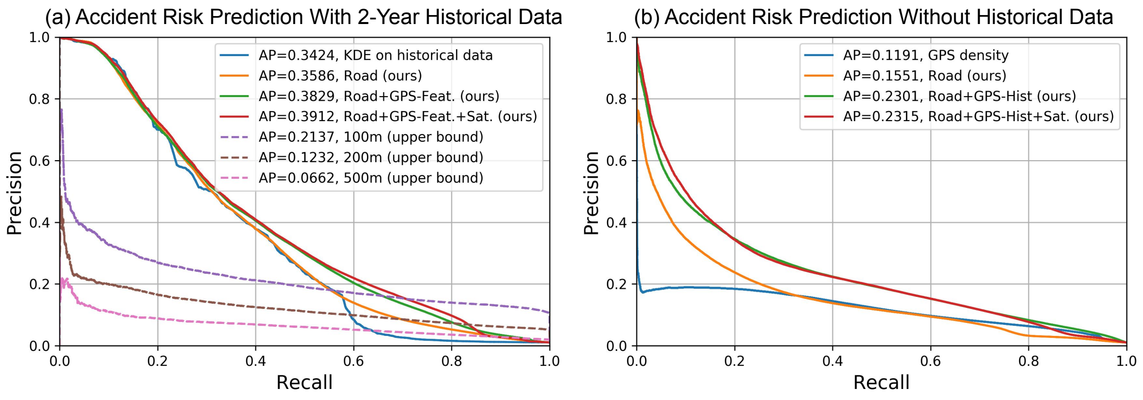


Figure 6-4: (a) Precision-Recall curves for "with history" setting. We present the results of three variants of our model, Kernel Density Estimation (KDE), and three upper-bounds on low-resolution techniques. The improvement from satellite imagery is on the lower-risk (right) side of the curve. KDE identifies the most high-risk places well but performs worse on low-risk places. (b) Precision-Recall curves for "without history" setting. In the absences of historical data, accident prediction accuracy is lower and the context information is more useful. In this case, GPS trajectories are effective in improving the performance.

| | Methods | Average Precision (%) | | | | | RMSE ($10^{-6}$) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | LA | NYC | CHI | BOS | Avg. | LA | NYC | CHI | BOS | Avg. |
| w/o historical data | GPS Density | 16.82 | 11.87 | 6.95 | 5.83 | 11.90 | 1.397 | 2.652 | 5.216 | 4.555 | 2.823 |
| | Road (ours) | 19.67 | 13.36 | 10.85 | 13.75 | 15.51 | 1.330 | 2.630 | 5.035 | 4.363 | 2.730 |
| | Road+Satellite (ours) | 24.61 | 14.13 | 13.98 | 11.74 | 19.81 | 1.282 | 2.574 | 4.869 | 4.309 | 2.662 |
| | Road+GPS-Hist (ours) | 28.59 | **17.56** | **22.71** | 16.72 | 23.01 | 1.239 | **2.552** | 4.690 | **4.203** | **2.594** |
| | Road+GPS-Hist+Sat. (ours) | **28.83** | 16.51 | 21.02 | **16.75** | **23.15** | **1.233** | 2.556 | 4.688 | 4.243 | 2.599 |
| | Road+GPS-Feat. (ours) | 27.28 | 15.11 | 19.66 | 15.12 | 21.71 | 1.271 | 2.616 | 4.744 | 4.296 | 2.648 |
| | Road+GPS-Feat.+Sat. (ours) | 28.29 | 16.41 | 22.26 | 14.49 | 22.15 | 1.242 | 2.608 | **4.591** | 4.338 | 2.618 |
| with historical data | 100m (upper bound) | 23.64 | 20.28 | 13.79 | 16.40 | 21.37 | 1.328 | 2.462 | 4.925 | **4.210** | 2.644 |
| | 200m (upper bound) | 14.28 | 10.68 | 6.20 | 8.54 | 12.32 | 1.404 | 2.657 | 5.163 | 4.462 | 2.804 |
| | 500m (upper bound) | 7.86 | 5.61 | 2.94 | 3.70 | 6.62 | 1.439 | 2.729 | 5.255 | 4.583 | 2.872 |
| | KDE on historical data | 42.60 | 22.11 | 25.68 | 20.06 | 34.24 | 0.945 | 2.562 | 4.060 | 5.073 | 2.529 |
| | Road (ours) | 43.48 | 25.41 | 27.26 | 23.16 | 35.86 | 0.901 | 2.459 | 3.938 | 4.758 | 2.412 |
| | Road+Satellite (ours) | 44.77 | 23.91 | 27.08 | 21.09 | 35.10 | 0.860 | 2.306 | 3.897 | 4.570 | 2.317 |
| | Road+GPS-Hist (ours) | 46.42 | 27.71 | 30.83 | **24.87** | 38.33 | 0.865 | 2.365 | 3.818 | **4.461** | **2.304** |
| | Road+GPS-Hist+Sat. (ours) | 46.27 | 26.96 | 30.01 | 23.93 | 37.79 | 0.859 | **2.278** | 3.888 | 4.573 | 2.308 |
| | Road+GPS-Feat. (ours) | 46.55 | 28.07 | **33.38** | 24.13 | 38.28 | **0.852** | 2.330 | **3.701** | 4.724 | 2.318 |
| | Road+GPS-Feat.+Sat. (ours) | **47.67** | **28.90** | 32.95 | 24.63 | **39.11** | 0.853 | 2.316 | 3.799 | 4.783 | 2.339 |

Table 6.3: Comparison of AP and RMSE for different methods. The first 7 rows compare the methods without using historical data as input, and the last 7 rows compare the methods that use the historical data as input. We also show the theoretical upper-bounds for the low-resolution risk maps at rows 8-10. In this comparison, all variants of our model consistently outperform other methods on both metrics.

In order to compare to the prior work we perform the following:

1. Since several prior works use KDE, we implemented and evaluated KDE as a baseline. The details of KDE is presented in [93]. We tuned the parameters of KDE so that it can achieve its highest average precision.
2. Since prior works use lower resolution than we do (Table 6.1.1), they cannot pinpoint accident hot-spots. This has a profound adverse effect on their performance. The effect of low resolution (100m×100m vs 5m×5m) is so significant that even if the prior works are allowed to optimize their output on the actual test-set, they still under-perform comparing to our model. To compare with the prior works, we use the best theoretical possible prediction (optimized on the test set with the knowledge of the future accidents) at their resolution. We refer to this as theoretical upper-bound for their accuracy. We show that our technique outperforms this theoretical upper-bound for prior works. We use theoretical upper-bound because the code for prior works is not available and they are evaluated on different cities than ours.
3. Different prior works use different sources of data as input (Table 6.1.1). We measure the effect of different sources of data on the performance of the model. This measures the effect of the extra data that we use.

### 6.3.6 Evaluation Results

We summarize our results in Table 6.3.4 and Figure 6-5. We show the APs and RMSEs of different approaches under two setups – with and without historical data. Next, we discuss a few insights we learned from this experiment.

**Prediction with historical data**: Our model uses other data sources to improve the risk map prediction when the historical data is available. As a result, our model performs better than the KDE-based approaches that only take the historical data input into account. As shown in Table 6.3.4, compared to the KDE-based approaches, our models improve the AP by 4.87 points and reduce the RMSE by 8.8%.

**Prediction without historical data**: When historical data is not available, our model can still use the other data sources to estimate the risk, achieving an AP

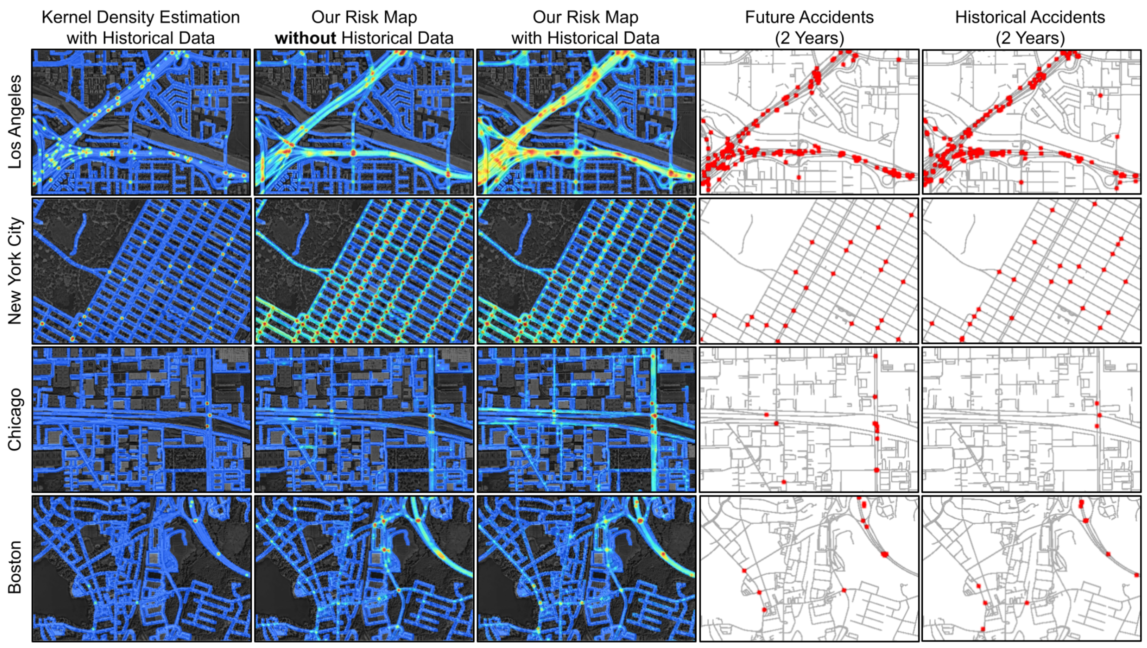

Kernel Density Estimation with Historical Data
Our Risk Map **without** Historical Data
Our Risk Map with Historical Data
Future Accidents (2 Years)
Historical Accidents (2 Years)
Los Angeles
New York City
Chicago
Boston

Figure 6-5: We show the risk maps produced by the KDE approach, and our approach (with and w/o historical data), along with the 2-year future accidents (used as the target in our evaluation) and the 2-year historical accidents in the four cities. We find that our risk maps can capture the underlying risk distribution that determines the probability of future accidents at all places and do so even without any historical data. In contrast, the KDE-based approach can only highlight places where there were accidents before and fail to assess the risks at other places. For example, in New York City, the accidents happened at random intersections in those two 2-year periods. Even though it looks like our risk map has low precision because there is no accident at many high-risk intersections in a 2-year period, our model captures the underlying risk distribution — accidents happen at those intersections with a similar chance.

of 23.15% and an RMSE of 2.594; this accuracy is significantly improved compared to a baseline method that only uses GPS density to estimate the traffic risk. More importantly, unlike most prior works that rely on historical accidents, we can use this model to create risk maps for places that do not have historical data, holding the potential to create broader impact.

**Low resolution prediction upper bounds**: We find that our models can outperform the 100-meter resolution upper bound by a large margin — 17.74 points on the AP metric and a 12.8%-reduction on the RMSE metric when the historical data is available. Even when the historical data is not available, our models can still outperform this upper bound thanks to the high resolution of our predicted risk maps.

**Impact of each data source**: Within our model, we evaluate six variants that use different combinations of data sources. In our experiments, 2 of the variants do not have GPS input. Comparing them with the other four variants, we can observe the benefit of the GPS data source. This benefit is due to two reasons, (1) the information carried by GPS data, such as the volume of the traffic, has a strong correlation with accident risk, (2) because information contained in the GPS data after aggregation (e.g., using a histogram) is relatively limited, overfitting becomes unlikely, and the prior learned from one place can be easily generalized to other places. This property is especially important in our scenario where the ground truth data is sparse.

Besides the benefit of the GPS data source, we also observe another important fact. Among the four cities, LA is the most unsafe city (has the highest accident density), followed by New York City, Chicago and Boston. The sparsity of our dataset in each city follows the reverse order. If we look at the average precision (AP) of each model (with historical data) in these four cities, we find that the models that take more information as input generally perform better in LA and New York City where accidents are less sparse. However, they perform worse in Boston, where the accidents are sparser. This fact verifies that the challenge caused by sparsity does indeed exist.

Because the GPS data contains important information about traffic patterns that can be helpful for accident risk prediction, instead of using the aggregated GPS histogram or hand-crafted statistics such as the median speed, we tried to use a Deep Set [112] to extract more information from the raw GPS trajectories end-to-end. However, we found that DeepSet actually harms the accuracy in our dataset because the provision of the extra rich features from the raw GPS trajectories greatly increases model variance and overfitting. We observe this fact even in LA, where it has the highest accident density.

**Cross-city evaluation**: We evaluate the generalization ability of our model in a cross-city evaluation setup where we train a model on three different cities and test it on an unseen city. As shown in Table 6.4, we find our model can generalize well on unseen testing cities.

| Training cfg. | LA | NYC | CHI | BOS |
|---|---|---|---|---|
| Same cities | 47.67% | 28.90% | 32.95% | 24.63% |
| Cross-city | 47.61% | 27.79% | 31.33% | 24.52% |

Table 6.4: Comparison of the average precision of city-specific models vs a cross-city model. City-specific models perform slightly better. We believe this is because each city has certain unique characteristics.

**Hyper-parameters**: Since the targets (accidents) are sparse, over-fitting is a major issue. We found that optimal hyper-parameters (including model size) highly depend on the size and sparsity of the dataset. Larger models generally perform better in larger cities. Hyper-parameters must be tuned to the input size. In this work, we focused on the logic behind model design rather than tuning hyper-parameters to one dataset.

**Insight**: In summary, we find that the sparsity of accidents is a major challenge in the design of an accident risk prediction model. On the one hand, we need to use more data sources and deeper architectures so that we can learn a good estimation of the accident risk. On the other hand, due to the sparsity of the accidents, using more input features and deeper models can lead to overfitting.

## 6.4 Conclusion

We presented an end-to-end deep model that predicts high-resolution traffic accident risk maps. Since accident data is sparse, sample efficiency is key for a successful accident risk estimation technique. To improve sample efficiency, we use a model that establishes the similarity between locations, not just based on proximity (as in KDE) but also on similarity in appearance. We developed a model to use satellite imagery, GPS trajectories, and road maps to achieve this. We extensively evaluated and showed that our model has state-of-the-art performance. Besides the improved performance and the useful maps we generated, our evaluations provide insights into how to achieve high performance in the face of accident data sparsity.

**Future work**: One potential extension of this work is to combine this work with temporal risk prediction techniques to establish a spatio-temporal accident risk model.

In the simplest form, there could be independent spatial and temporal components in the model. A comprehensive accident risk model could potentially input other factors, including weather patterns, driver characteristics, driving behavior, and vehicle condition.

**Accident related applications**: Our model is flexible in terms of what data sources are available. Once our model is trained, we can apply it to countries where detailed historical accident data is not published. Furthermore, this model can be potentially used to compare city layout designs before construction.

**Other Applications**: Even though this model has been developed for accident prediction, this fundamental technique can work for similar sparse location-based problems, including 911 emergency risk maps or taxi demand maps.

# Chapter 7

# Conclusion

This thesis presents five automation techniques that utilize aerial imagery, satellite imagery and GPS trajectory data to extract base maps and infer essential map features. These techniques advance the state-of-the-arts in map extraction field by improving the map extraction accuracy and enabling the inference of new map features.

Specifically, RoadRunner (§2) extracts accurate base maps from GPS trajectories. It exploits the long-term structure of GPS trajectories to generate accurate maps in dense urban areas and complex intersections where prior methods fail. Sat2Graph (§3) focuses on base map extraction from aerial imagery and satellite imagery. It proposes the graph-tensor encoding scheme that eliminates several limitations of the road segmentation encoding scheme, improving the base map extraction accuracy by a large margin. RoadTagger (§5) infers the road attributes from aerial imagery. It improves inference accuracy using a novel combination of convolutional neural networks and graph neural networks. Chapter 5 and chapter 6 present the first works to extract a fully-routable lane-level street map from only aerial imagery and to infer a 5-meter high-resolution traffic crash risk map, respectively.

Digital maps are ruling the world with their ubiquitous influence in almost all applications and services. Research work in digital map creation and maintenance holds the potential to reduce the mapping cost, enrich map features, and improve map coverage, making digital maps accessible to everyone and benefiting their daily lives and work. This thesis stretches the surface of map-making research. Looking ahead, there are still many open research problems and opportunities in this field. To conclude this thesis, we outline a few of them.

**Interactive map-making.** The map extraction algorithms proposed in this thesis target a specific workflow where we train the deep learning models with predefined datasets and apply the models to unseen places to extract the maps. However, this workflow has two drawbacks in practice. First, the state-of-the-art models still don't outperform humans, and the extracted maps still require human validation, which is also an expensive process. Second, the definitions in road maps have regional differences. Two visually similar roads may be considered a primary road in one place but a residential road in another. This regional difference makes it challenging to train one model that can work for all cases.

A more practical workflow is interactive map-making, where the human map makers can interact with the automation algorithm. For example, in one interaction iteration, the automation algorithm first extracts the map. Then the human map makers provide feedback to the automation algorithm, and the automation algorithm can adjust its output based on the feedback. It is an exciting research direction to design efficient interaction protocols and the core deep learning algorithms that should have the ability to react to human input in real-time.

**Extracting more digital map features.** This thesis proposes techniques that use aerial/satellite imagery and GPS data to extract four map features, road network base maps, road attributes, drivable lane maps, and traffic crash risk maps. Besides these features, many other map features can fulfill different needs and help improve map users' experience if we can create such a map efficiently. For example, we can build a safety map for pedestrians, which may take the existence of street lights, criminal rate, popularity, and traffic condition into account. This map can help pedestrians better plan their routes and mitigate potential risks. We can also build a surface-type map for runners to help them find running routes that are best for their knees.

**Specialized deep learning algorithms and platforms for geospatial data.** Deep learning has become a standard building block in many techniques for enriching digital maps. These techniques need to process a wide range of different geospatial data with different modalities. Geospatial data often comes with global positions, and different data points are related not only based on their data types but also their locations. Unlike imagery, video, and text data, there is limited support for geospatial data in existing machine learning algorithms and platforms, creating two exciting research problems. One is to design specialized neural network architectures that have the capability to express spatial correlations and work natively with multi-modalities geospatial data. The other is to build specialized machine learning platforms that can simplify the development of geospatial machine learning applications and automatically optimize the performance of model training and model deployment given the unique character of geospatial data layout.

# Appendix A

# RoadTagger Synthesized Micro-Benchmark

In the following pages, we show the whole micro-benchmark as well as the corresponding outputs from the CNN image classifier, CNN image classifier with MRF post-processing and our proposed RoadTagger. We use this synthesized micro-benchmark to study the robustness of RoadTagger in a controllable way. This micro-benchmark complements the large-scale real-world dataset we used in our paper.

As we mentioned in the paper, in this micro-benchmark, we find RoadTagger is robust to a wide range of different disruptions. These disruptions include removing all the lane markings on part of the roads, alternatively occluding the left and right side of the roads, and even occluding the target road with another overpass road.

**Highlight Examples.** We find the overpass example shown in Figure A-1(3) particularly interesting. In this example, an overpass road occluded the target road. Although the number of lanes of the overpass road is very clear on the satellite imagery, our RoadTagger learns to ignore this strong visual signal. This shows the strong ability of inductive reasoning learned by RoadTagger as it not only learns the number of lanes from the imagery but also learns how to extract other useful information and use this additional information to improve the prediction.

Although adding Markov Random Field (MRF) as post-processing to the CNN

Image Classifier approach can handle some of the disruptions, it cannot work well in all scenarios due to the *information-barrier* induced by the separation of CNN image classifier and the post-processing; the MRF post-processing may abuse its prior knowledge and make incorrect corrections such as what happened in example (1) and (2) in Figure A-1.

In fact, to correctly predict the number of lanes in both example (1) and example (3) in Figure A-1, RoadTagger needs to know that when the starting point of an overpass road is detected, the visual features of the overpass should be ignored until the far edge of the overpass is detected. At the same time, RoadTagger needs to know if the road is temporarily occluded by trees, the following road segment still belongs to the same target road. RoadTagger's end-to-end architecture enables it to learn all this knowledge without any additional labels or explicit features. This is perhaps the most attractive part of our RoadTagger framework.

**Failure Examples.** We also show two examples in Figure 6 where RoadTagger failed to predict the lane count correctly. In this two examples, the occlusion is longer than the propagation step (a configurable hyper-parameter of RoadTagger) of our graph neural network model (i.e., 8 in our current setup), which makes it impossible to propagate information between the left side and the right side of the road. Therefore, RoadTagger yields incorrect predictions in these two examples.

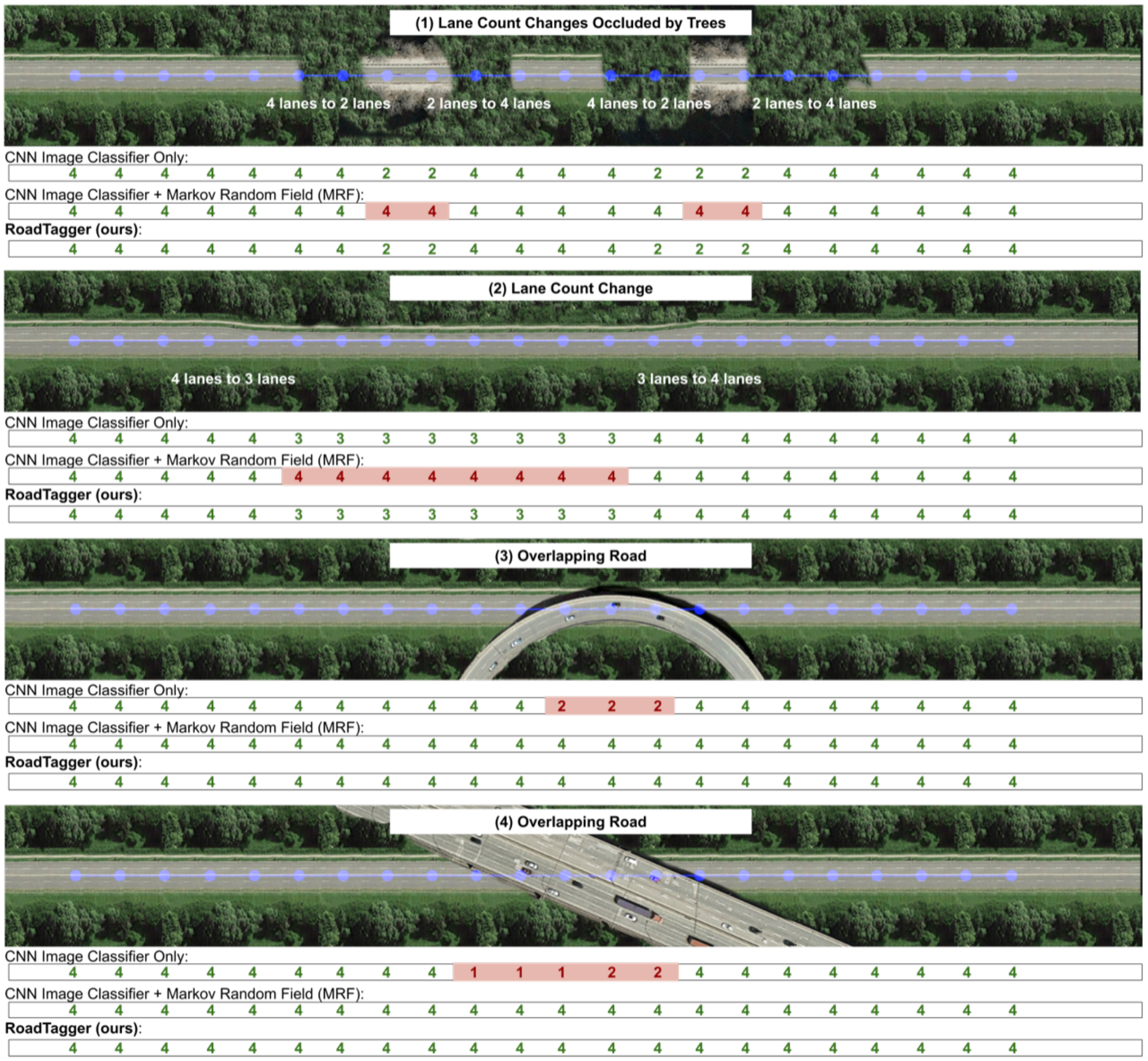


Figure A-1: Examples (1) to (4). In each image, blue lines show the road graph. The number of lanes predicted by the CNN Image Classifier, CNN image classifier with MRF post-processing and RoadAnnotator on each segment are shown along the bottom of each figure. We color the output numbers green for correct predictions and red for incorrect predictions.

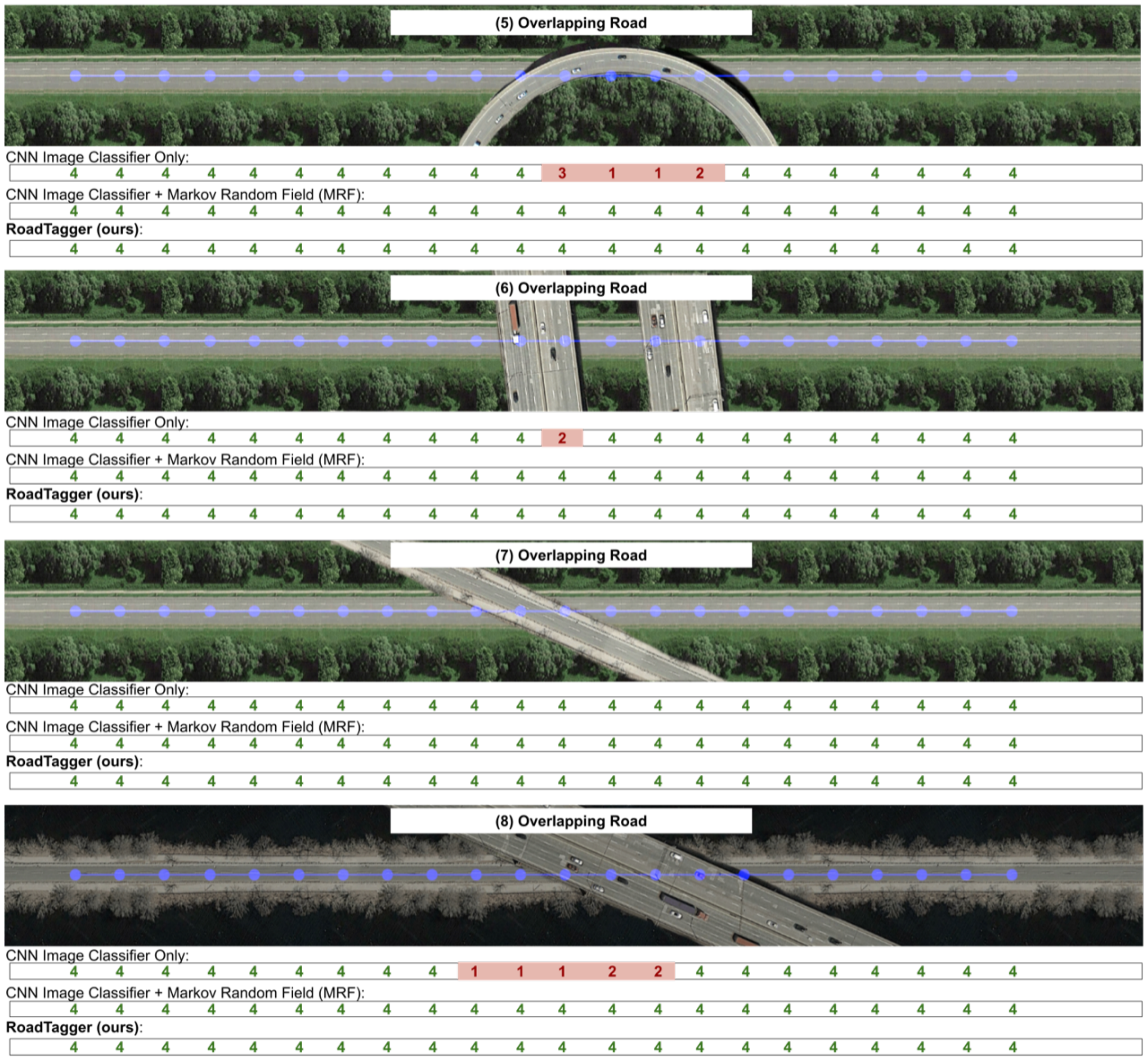


Figure A-2: Examples (5) to (8). In each image, blue lines show the road graph. The number of lanes predicted by the CNN Image Classifier, CNN image classifier with MRF post-processing and RoadAnnotator on each segment are shown along the bottom of each figure. We color the output numbers green for correct predictions and red for incorrect predictions.

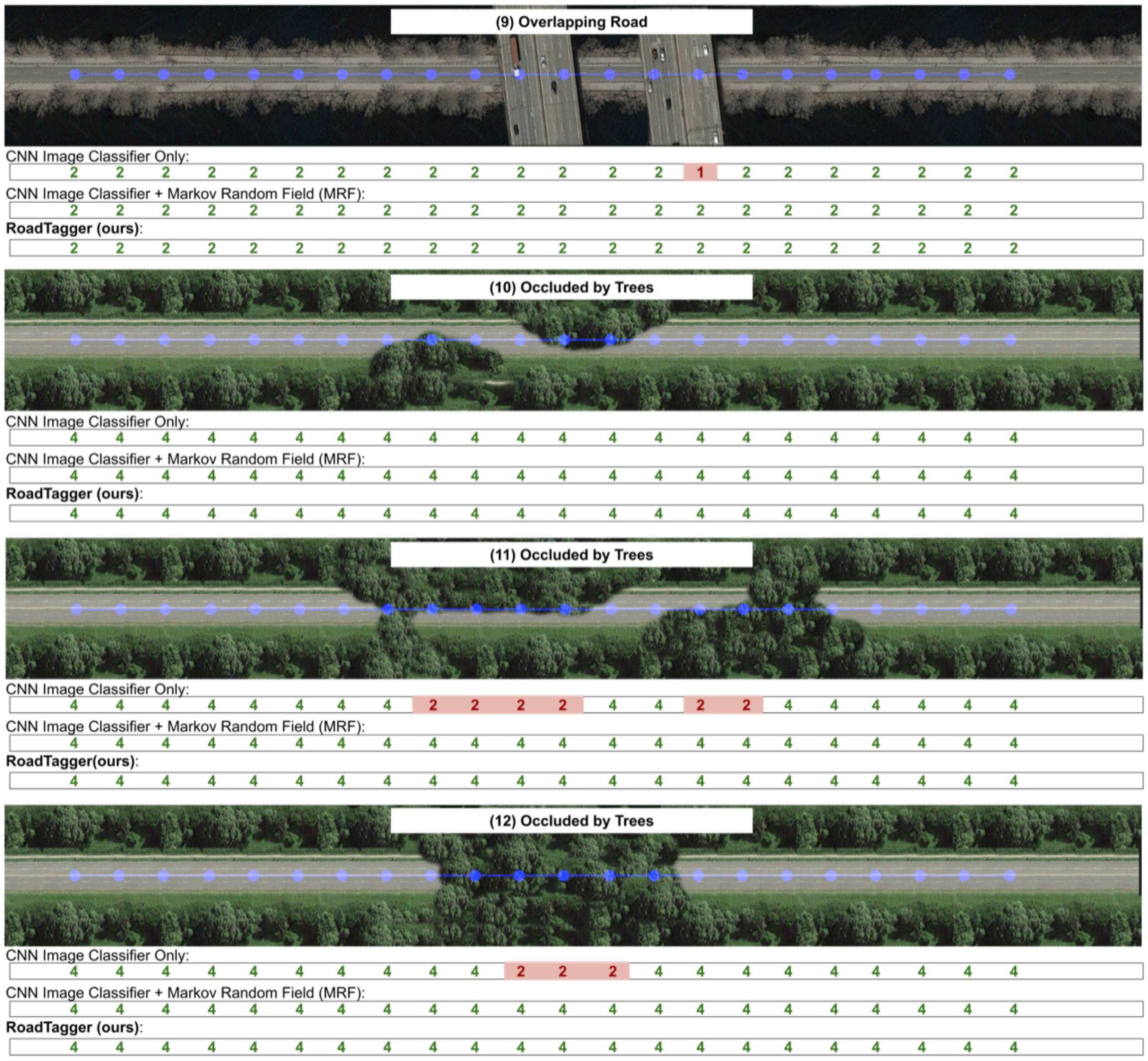


Figure A-3: Examples (9) to (12). In each image, blue lines show the road graph. The number of lanes predicted by the CNN Image Classifier, CNN image classifier with MRF post-processing and RoadAnnotator on each segment are shown along the bottom of each figure. We color the output numbers green for correct predictions and red for incorrect predictions.

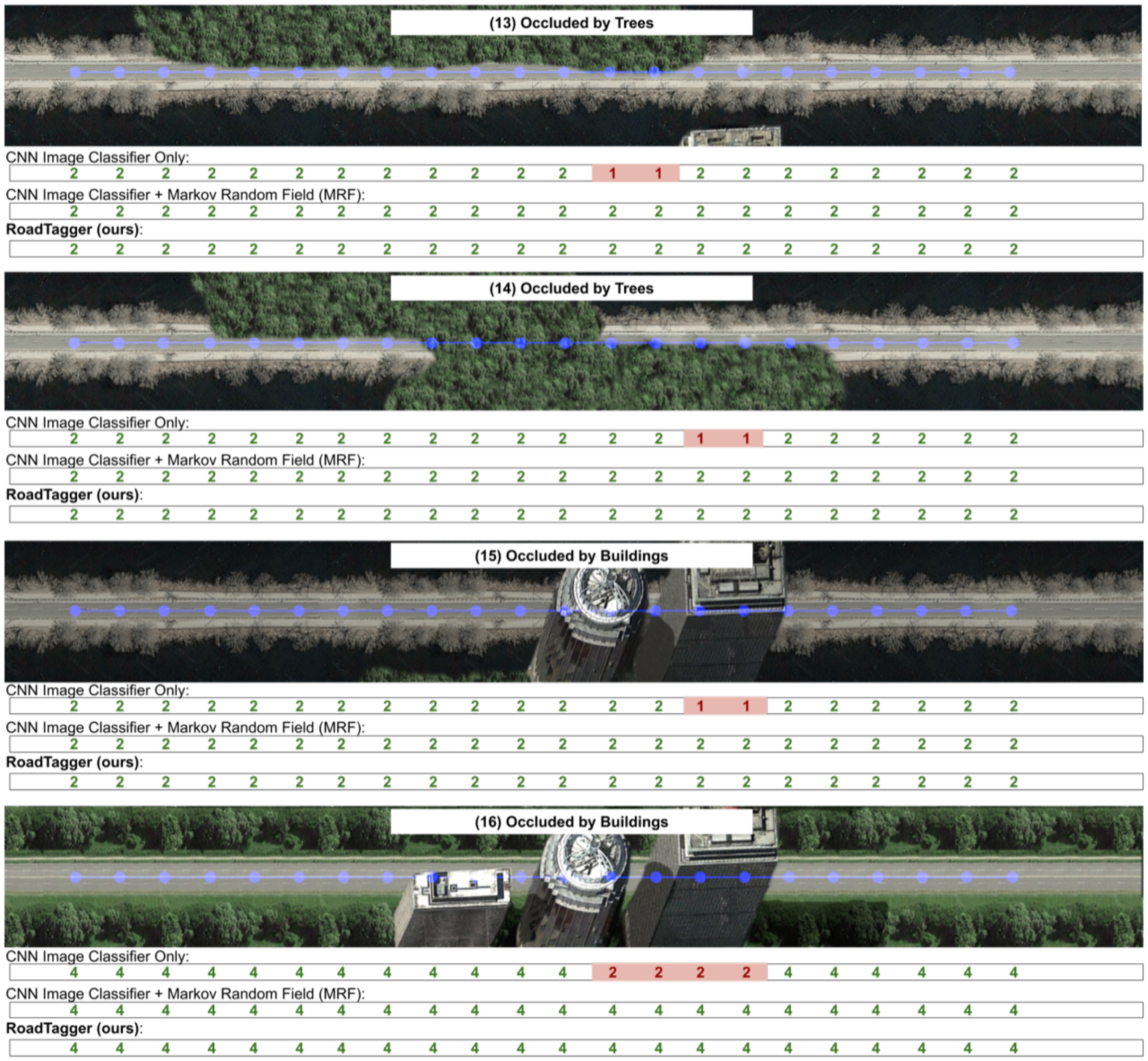


Figure A-4: Examples (13) to (16). In each image, blue lines show the road graph. The number of lanes predicted by the CNN Image Classifier, CNN image classifier with MRF post-processing and RoadAnnotator on each segment are shown along the bottom of each figure. We color the output numbers green for correct predictions and red for incorrect predictions.

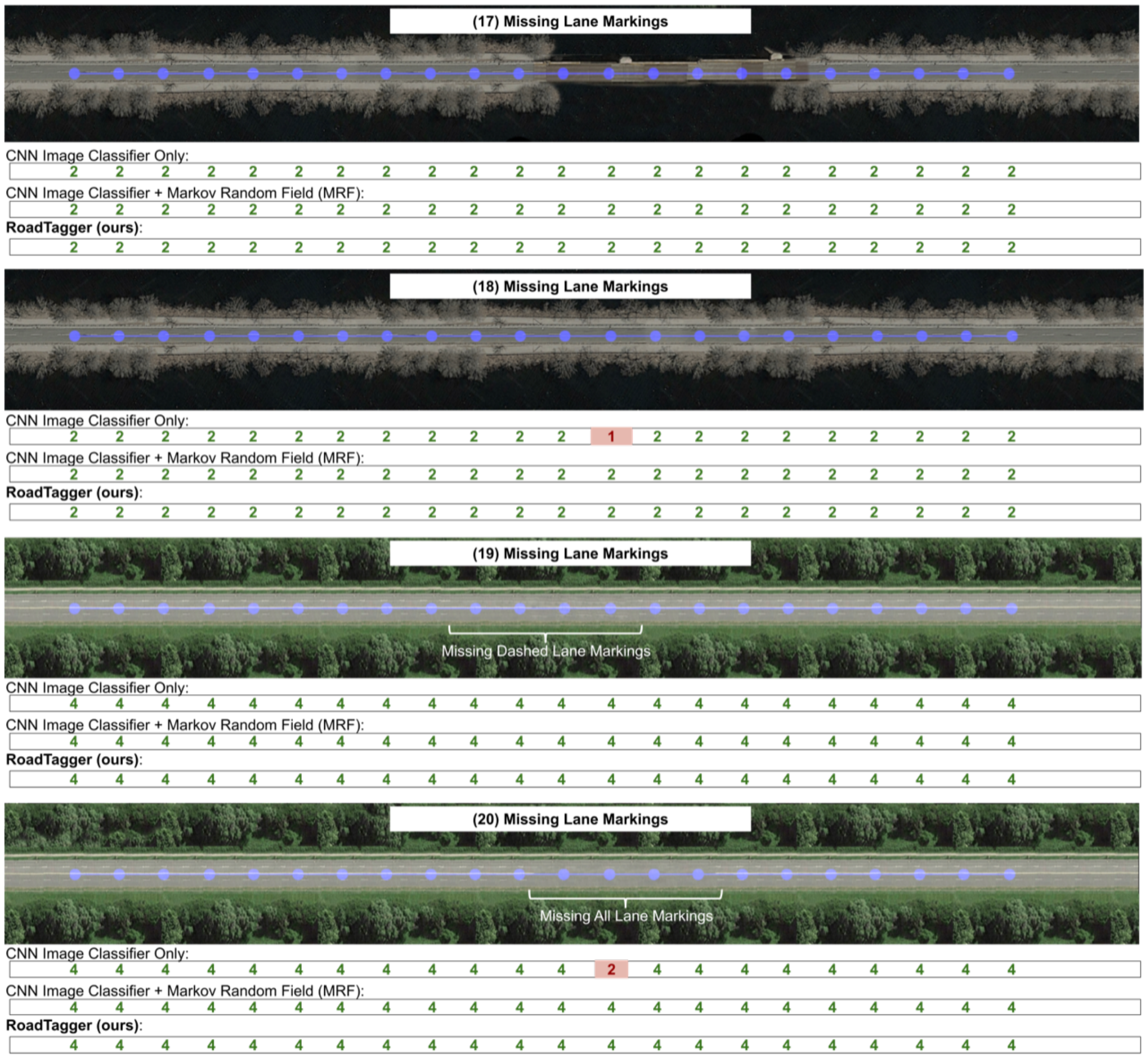


Figure A-5: Examples (17) to (20). In each image, blue lines show the road graph. The number of lanes predicted by the CNN Image Classifier, CNN image classifier with MRF post-processing and RoadAnnotator on each segment are shown along the bottom of each figure. We color the output numbers green for correct predictions and red for incorrect predictions.

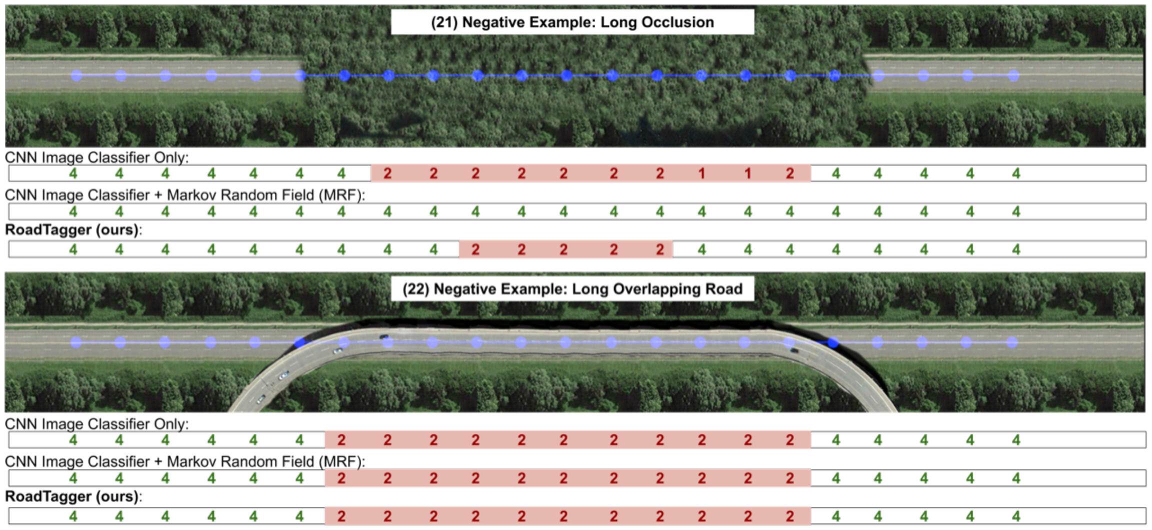


Figure A-6: Examples (21) to (22). In each image, blue lines show the road graph. The number of lanes predicted by the CNN Image Classifier, CNN image classifier with MRF post-processing and RoadAnnotator on each segment are shown along the bottom of each figure. We color the output numbers green for correct predictions and red for incorrect predictions.

# Appendix B

# Crash Risk Maps

## Model comparison using lift chart

Lift chart is a risk analysis tool used by insurance actuaries for risk assessment. Lift charts can compare a risk prediction model against a baseline. Figure B-1 shows a lift chart comparing different versions of our "with history" model against KDE as a baseline. In this figure, road grid cells are sorted in decreasing order according to their risk estimate. Grid cells with equal risk estimates are permuted randomly. Given any threshold, the cumulative percentage of recalled accidents are estimated both with the baseline and our model. Given any threshold, the ratio between our recalled accidents and the baseline's recalled accidents is reported as "lift".

All versions of our "with history" model outperform KDE. Please note that KDE's kernel is optimized in all experiments. Also KDE does not apply to "without history" models because KDE needs history.

In the top one percent of high-risk areas, our model does not significantly outperform KDE (the left side of the Figure B-1). This is because in the most high-risk areas, the frequency of accidents is high enough that KDE has low variance. In contrast, our model outperforms KDE in lower-risk areas as KDE gets affected by data sparsity in low-risk areas more seriously.

Between our models, road topology alone is useful because it can identify interchanges and intersections. However, among the top 5% of dangerous locations, road

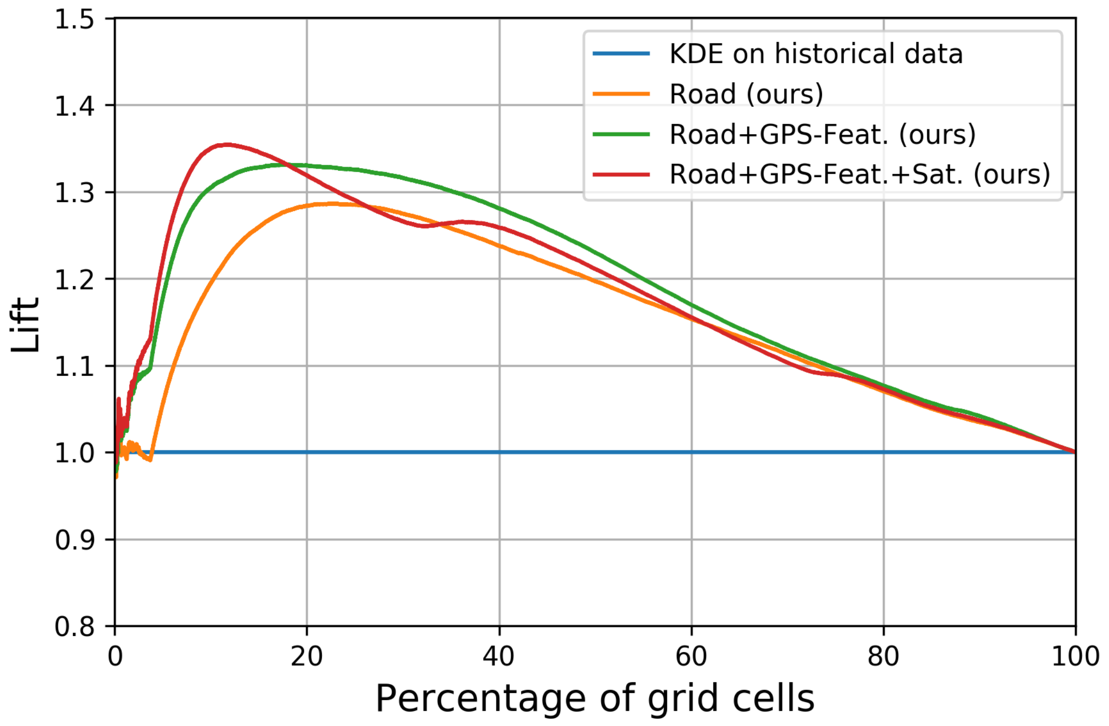


Figure B-1: Lift chart to compare our models with a KDE baseline. Note that different variants of our model outperform KDE.

topology alone cannot identify risk any better than KDE. GPS trajectories are helpful to distinguish between high-risk and very high-risk areas. Satellite imagery is also helpful in high-risk areas but does not help in low-risk areas in the presence of GPS trajectories.

Figure B-2 provides an explanation as to why KDE fails. Since KDE works based on historical frequency analysis, in places where no accidents have occurred, KDE estimates the frequency as zero. As shown in Figure B-2, KDE can recall about 58% of accidents in the top 4% of grid cells. However, after this, all other grid cells are seen to have a similar risk profile. A major advantage of our technique is that we can estimate accident risk in places that have not experienced any accidents previously.

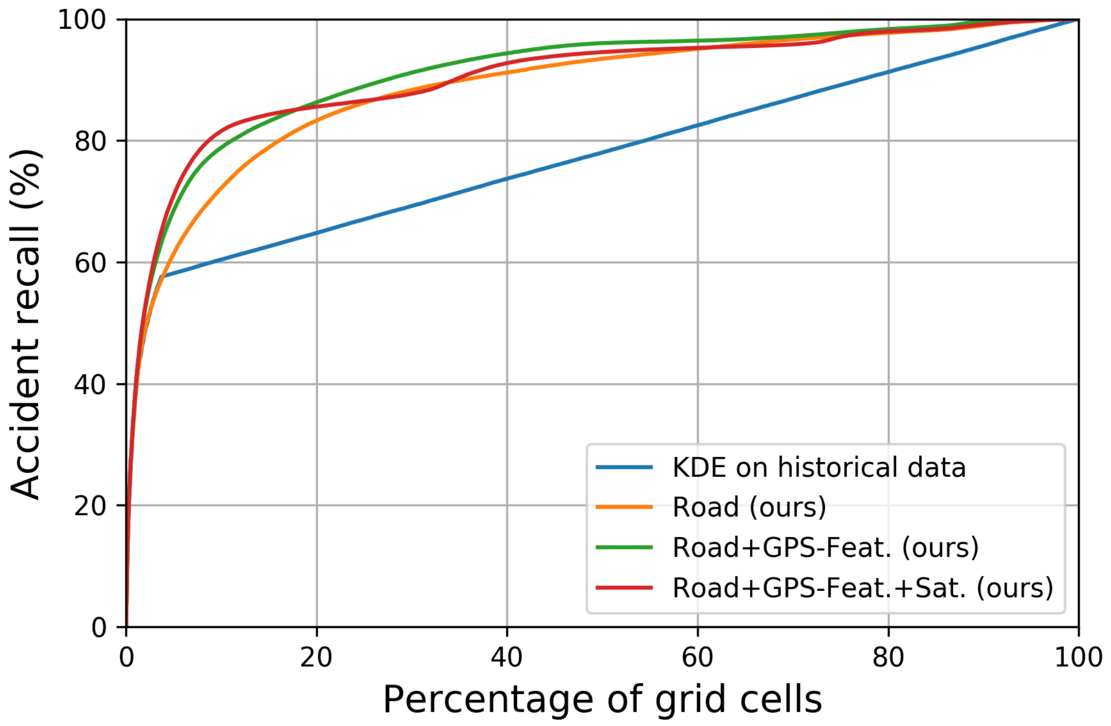


Figure B-2: Comparison of different models based on accident recall. Each model ranks grid cells according to accident risk. Since KDE heavily relies on historical accidents, except the top 5% of dangerous grid cells where historical accidents have occurred, it cannot distinguish high-risk and low-risk grid cells, therefore, its curve has a constant slope.

# Bibliography

[1] Mapbox. www.mapbox.com. Accessed: 2021-03-01.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[4] Gabriel Agamennoni, Juan I Nieto, and Eduardo M Nebot. Robust inference of principal road paths for intelligent transportation systems. *IEEE T-ITS*, 12(1):298–308, 2011.

[5] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19(3):601–632, 2015.

[6] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19(3):601–632, 2015.

[7] Mahmuda Ahmed and Carola Wenk. Constructing street networks from GPS trajectories. In *ESA*, 2012.

[8] Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995.

[9] Tessa K Anderson. Kernel density estimation and k-means clustering to profile road accident hotspots. *Accident Analysis & Prevention*, 41(3):359–364, 2009.

[10] Mohammad Ali Arman and Chris MJ Tampère. Lane-level routable digital map reconstruction for motorway networks using low-precision gps data. *Transportation Research Part C: Emerging Technologies*, 129:103234, 2021.

[11] Seyed Majid Azimi, Peter Fischer, Marco Körner, and Peter Reinartz. Aerial lanenet: Lane-marking semantic segmentation in aerial imagery using wavelet-enhanced cost-sensitive symmetric fully convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 57(5):2920–2938, 2018.

[12] Jie Bao, Pan Liu, and Satish V Ukkusuri. A spatiotemporal deep learning approach for citywide short-term crash risk prediction with multi-source data. *Accident Analysis & Prevention*, 122:239–254, 2019.

[13] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. Machine-assisted map editing. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 23–32, 2018.

[14] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4720–4728, 2018.

[15] Favyen Bastani, Songtao He, Mohammad Alizadeh, Hari Balakrishnan, Sam Madden, Sanjay Chawla, Sofiane Abbar, and David DeWitt. Unthule: An incremental graph construction process for robust road map extraction from aerial images. Submitted to CVPR 2018.

[16] Favyen Bastani, Songtao He, Satvat Jagwani, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and Mohammad Amin Sadeghi. Updating street maps using changes detected in satellite imagery. *arXiv preprint arXiv:2110.06456*, 2021.

[17] Anil Batra, Suriya Singh, Guan Pang, Saikat Basu, CV Jawahar, and Manohar Paluri. Improved road connectivity by joint learning of orientation and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10385–10393, 2019.

[18] Yoshua Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009.

[19] James Biagioni and Jakob Eriksson. Map inference in the face of noise and disparity. In *ACM SIGSPATIAL 2012*.

[20] James Biagioni and Jakob Eriksson. Inferring road maps from global positioning system traces: Survey and comparative evaluation. *Transportation research record*, 2291(1):61–71, 2012.

[21] James Biagioni and Jakob Eriksson. Map inference in the face of noise and disparity. In *ACM SIGSPATIAL 2012*, 2012.

[22] Michal Bíl, Richard Andrášik, and Zbyněk Janoška. Identification of hazardous road locations of traffic accidents by means of kernel density estimation and cluster significance evaluation. *Accident Analysis & Prevention*, 55:265–273, 2013.

[23] Lawrence Blincoe, Ted R Miller, Eduard Zaloshnja, and Bruce A Lawrence. The economic and societal impact of motor vehicle crashes, 2010 (revised). Technical report, 2015.

[24] Alexander Buslaev, Selim Seferbekov, Vladimir Iglovikov, and Alexey Shvets. Fully convolutional network for automatic road extraction from satellite imagery. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 207–210, 2018.

[25] Gabriel Cadamuro, Aggrey Muhebwa, and Jay Taneja. Assigning a grade: Accurate measurement of road quality using satellite imagery. *arXiv preprint arXiv:1812.01699*, 2018.

[26] Ciro Caliendo, Maurizio Guida, and Alessandra Parisi. A crash-prediction model for multilane roads. *Accident Analysis and Prevention*, 39(4):657–670, 2007.

[27] Lili Cao and John Krumm. From GPS traces to a routable road map. In *ACM SIGSPATIAL*, pages 3–12, 2009.

[28] Lili Cao and John Krumm. From GPS traces to a routable road map. In *ACM SIGSPATIAL*, pages 3–12, 2009.

[29] Li-Yen Chang. Analysis of freeway accident frequencies: Negative binomial regression versus artificial neural network. *Safety Science*, 43(8):541–557, 2005.

[30] Li-Yen Chang and Wen-Chieh Chen. Data mining of tree-based models to analyze freeway accident frequency. *Journal of Safety Research*, 36(4):365–375, 2005.

[31] Ming-Fang Chang, John W Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3d tracking and forecasting with rich maps. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[32] C. Chen, X. Fan, C. Zheng, L. Xiao, M. Cheng, and C. Wang. Sdcae: Stack denoising convolutional autoencoder model for accident risk prediction via traffic big data. In *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, pages 328–333, 2018.

[33] Chen Chen and Yinhang Cheng. Roads digital map generation with multi-track gps data. In *International Workshop on Geoscience and Remote Sensing*, 2008.

[34] Chen Chen, Cewu Lu, Qixing Huang, Qiang Yang, Dimitrios Gunopulos, and Leonidas J. Guibas. City-Scale Map Creation and Updating using GPS Collections. In *KDD*, 2016.

[35] Quanjun Chen, Xuan Song, Harutoshi Yamada, and Ryosuke Shibasaki. Learning deep representation from big and heterogeneous data for traffic accident inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[36] 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.

[37] Hang Chu, Daiqing Li, David Acuna, Amlan Kar, Maria Shugrina, Xinkai Wei, Ming-Yu Liu, Antonio Torralba, and Sanja Fidler. Neural turtle graphics for modeling city road layouts. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4522–4530, 2019.

[38] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pages 2067–2075, 2015.

[39] Dragos Costea, Alina Marcu, Emil Slusanschi, and Marius Leordeanu. Creating roadmaps in aerial images with generative adversarial networks and smoothing-based optimization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2100–2109, 2017.

[40] Jonathan J Davies, Alastair R Beresford, and Andy Hopper. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing*, 5(4), 2006.

[41] Jonathan J Davies, Alastair R Beresford, and Andy Hopper. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing*, 5(4), 2006.

[42] Ole Henry Dørum. Deriving double-digitized road network geometry from probe data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 15. ACM, 2017.

[43] Stefan Edelkamp and Stefan Schrödl. Route planning and map inference with global positioning traces. In *Computer Science in Perspective*. 2003.

[44] Stefan Edelkamp and Stefan Schrödl. Route planning and map inference with global positioning traces. In *Computer Science in Perspective*. 2003.

[45] Adam Van Etten. City-scale road extraction from satellite imagery v2: Road speeds and travel times. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1786–1795, 2020.

[46] Peter Fischer, Seyed Majid Azimi, Robert Roschlaub, and Thomas Krauß. Towards hd maps from aerial imagery: Robust lane marking segmentation using country-scale imagery. *ISPRS International Journal of Geo-Information*, 7(12):458, 2018.

[47] A Fortier, Djemel Ziou, Costas Armenakis, and S Wang. Survey of work on road extraction in aerial and satellite images. *Center for Topographic Information Geomatics, Ontario, Canada. Technical Report*, 241(3), 1999.

[48] Google. Google Static Maps API. https://developers.google.com/maps/documentation/maps-static/intro. Accessed: 2019-03-21.

[49] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4), 2008.

[50] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[51] Ryuhei Hamaguchi and Shuhei Hikosaka. Building detection from satellite imagery using ensemble of size-specific detectors. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 223–2234. IEEE, 2018.

[52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[53] Songtao He and Hari Balakrishnan. Lane-level street map extraction from aerial imagery. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2080–2089, 2022.

[54] 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 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 3–12. ACM, 2018.

[55] 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 *ACM SIGSPATIAL*, 2018.

[56] Songtao He, Favyen Bastani, Satvat Jagwani, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Mohamed M Elshrif, Samuel Madden, and Mohammad Amin Sadeghi. Sat2graph: road graph extraction through graph-tensor encoding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV 16*, pages 51–67. Springer, 2020.

[57] Songtao He, Favyen Bastani, Satvat Jagwani, Edward Park, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Samuel Madden, and Mohammad Amin Sadeghi. Roadtagger: Robust road attribute inference with graph neural networks. *arXiv preprint arXiv:1912.12408*, 2019.

[58] Songtao He, Favyen Bastani, Satvat Jagwani, Edward Park, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Samuel Madden, and Mohammad Amin Sadeghi. Roadtagger: Robust road attribute inference with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10965–10972, 2020.

[59] Songtao He, Mohammad Amin Sadeghi, Sanjay Chawla, Mohammad Alizadeh, Hari Balakrishnan, and Samuel Madden. Inferring high-resolution traffic accident risk maps based on satellite imagery and gps trajectories. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11977–11985, 2021.

[60] Antoine Hébert, Timothée Guédon, Tristan Glatard, and Brigitte Jaumard. High-resolution road vehicle collision prediction for the city of montreal. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1804–1813. IEEE, 2019.

[61] Stefan Hinz and Albert Baumgartner. Automatic extraction of urban road networks from multi-view aerial imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 58(1-2):83–98, 2003.

[62] Namdar Homayounfar, Wei-Chiu Ma, Justin Liang, Xinyu Wu, Jack Fan, and Raquel Urtasun. Dagmapper: Learning to map by discovering lane topology. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2911–2920, 2019.

[63] Wonje Jang, Jhonghyun An, Sangyun Lee, Minho Cho, Myungki Sun, and Euntai Kim. Road lane semantic segmentation for high definition map. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1001–1006. IEEE, 2018.

[64] Jinyong Jeong, Younggun Cho, and Ayoung Kim. Road-slam: Road marking based slam with lane-level accuracy. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1736–1473. IEEE, 2017.

[65] Soren Kammel and Benjamin Pitzer. Lidar-based lane marker detection and mapping. In *2008 IEEE Intelligent Vehicles Symposium*, pages 1137–1142. IEEE, 2008.

[66] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[67] Khanh Giang Le, Pei Liu, and Liang-Tay Lin. Traffic accident hotspot identification by integrating kernel density estimation and spatial autocorrelation analysis: a case study. *International Journal of Crashworthiness*, pages 1–11, 2020.

[68] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[69] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. PolyMapper: Extracting city maps using polygons. *arXiv preprint arXiv:1812.01497*, 2018.

[70] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. Topological map extraction from overhead images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1715–1724, 2019.

[71] Xuemei Liu, James Biagioni, Jakob Eriksson, Yin Wang, George Forman, and Yanmin Zhu. Mining Large-Scale, Sparse GPS Traces for Map Inference: Comparison of Approaches. In *KDD*, 2012.

[72] Gellért Máttyus, Wenjie Luo, and Raquel Urtasun. DeepRoadMapper: Extracting road topology from aerial images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3438–3446, 2017.

[73] Gellert Mattyus, Shenlong Wang, Sanja Fidler, and Raquel Urtasun. Enhancing road maps by parsing aerial images around the world. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1689–1697, 2015.

[74] Gellért Máttyus, Shenlong Wang, Sanja Fidler, and Raquel Urtasun. HD maps: Fine-grained road segmentation by parsing ground and aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3611–3619, 2016.

[75] Afshin Shariat Mohaymany, Matin Shahri, and Babak Mirbagheri. Gis-based method for detecting high-crash-risk road segments using network kernel density estimation. *Geo-spatial Information Science*, 16(2):113–119, 2013.

[76] Sobhan Moosavi, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, Radu Teodorescu, and Rajiv Ramnath. Accident risk prediction based on heterogeneous sparse data: New dataset and insights. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 33–42, 2019.

[77] Agata Mosinska, Pablo Márquez-Neila, Mateusz Koziński, and Pascal Fua. Beyond the pixel-wise loss for topology-aware delineation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[78] Alameen Najjar, Shun'ichi Kaneko, and Yoshikazu Miyanaga. Combining satellite imagery and open data to map road safety. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[79] Rostislav Netek, Tomas Pour, and Renata Slezakova. Implementation of heat maps in geographical information system–exploratory study on traffic accident data. *Open Geosciences*, 10(1):367–384, 2018.

[80] Atsuyuki Okabe, Toshiaki Satoh, and Kokichi Sugihara. A kernel density estimation method for networks, its computational method and a gis-based tool. *International Journal of Geographical Information Science*, 23(1):7–32, 2009.

[81] World Health Organization et al. Global status report on road safety 2018: Summary. Technical report, World Health Organization, 2018.

[82] V Prasannakumar, H Vijith, R Charutha, and N Geetha. Spatio-temporal clustering of road accidents: Gis based analysis and assessment. *Procedia-social and behavioral sciences*, 21:317–325, 2011.

[83] Baoxing Qin, ZJ Chong, Tirthankar Bandyopadhyay, Marcelo H Ang, Emilio Frazzoli, and Daniela Rus. Road detection and mapping using 3d rolling window. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 1239–1246. IEEE, 2013.

[84] Honglei Ren, You Song, Jingwen Wang, Yucheng Hu, and Jinzhi Lei. A deep learning approach to the citywide traffic accident risk prediction. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3346–3351. IEEE, 2018.

[85] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[86] Stefan Schroedl, Kiri Wagstaff, Seth Rogers, Pat Langley, and Christopher Wilson. Mining gps traces for map refinement. *Data Mining and Knowledge Discovery*, 9(1):59–87, Jul 2004.

[87] Qian Shi, Xiaoping Liu, and Xia Li. Road detection from remote sensing images by generative adversarial networks. *IEEE access*, 6:25486–25494, 2017.

[88] Wenhuan Shi, Shuhan Shen, and Yuncai Liu. Automatic generation of road network map from massive gps, vehicle trajectories. In *IEEE ITSC 2009*, 2009.

[89] Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning theory and kernel machines*, pages 144–158. Springer, 2003.

[90] 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 2018 SIAM International Conference on Data Mining*. SIAM, 2018.

[91] 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 2018 SIAM International Conference on Data Mining*. SIAM, 2018.

[92] Yong-Qiang Tan, Shang-Hua Gao, Xuan-Yi Li, Ming-Ming Cheng, and Bo Ren. Vecroad: Point-based iterative graph exploration for road graphs extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8910–8918, 2020.

[93] George R Terrell and David W Scott. Variable kernel density estimation. *The Annals of Statistics*, page 1236, 1992.

[94] Long Tien Truong and Sekhar VC Somenahalli. Using gis to identify pedestrian-vehicle crash hot spots and unsafe bus stops. *Journal of Public Transportation*, 14(1):6, 2011.

[95] Adam Van Etten. City-scale road extraction from satellite imagery. *arXiv preprint arXiv:1904.09901*, 2019.

[96] Adam Van Etten, Dave Lindenbaum, and Todd M Bacastow. Spacenet: A remote sensing dataset and challenge series. *arXiv preprint arXiv:1807.01232*, 2018.

[97] Weixing Wang, Nan Yang, Yi Zhang, Fengping Wang, Ting Cao, and Patrik Eklund. A review of road extraction from remote sensing images. *Journal of traffic and transportation engineering (english edition)*, 3(3):271–282, 2016.

[98] Jan D Wegner, Javier A Montoya-Zegarra, and Konrad Schindler. A higher-order CRF model for road network extraction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1698–1705, 2013.

[99] Jan Dirk Wegner, Javier Alexander Montoya-Zegarra, and Konrad Schindler. Road networks as collections of minimum cost paths. *ISPRS Journal of Photogrammetry and Remote Sensing*, 108:128–137, 2015.

[100] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[101] Zhixiao Xie and Jun Yan. Kernel density estimation of traffic accidents in a network space. *Computers, environment and urban systems*, 32(5):396–406, 2008.

[102] Zhixiao Xie and Jun Yan. Detecting traffic accident clusters with network kernel density estimation and local spatial statistics: an integrated approach. *Journal of transport geography*, 31:64–71, 2013.

[103] Zhenhua Xu, Yuxiang Sun, and Ming Liu. icurb: Imitation learning-based detection of road curbs using aerial images for autonomous driving. *IEEE Robotics and Automation Letters*, 6(2):1097–1104, 2021.

[104] Zhenhua Xu, Yuxiang Sun, and Ming Liu. Topo-boundary: A benchmark dataset on topological road-boundary detection using aerial images for autonomous driving. *arXiv preprint arXiv:2103.17119*, 2021.

[105] Zhenhua Xu, Yuxiang Sun, Lujia Wang, and Ming Liu. Cp-loss: Connectivity-preserving loss for road curb detection in autonomous driving with aerial images. *arXiv preprint arXiv:2107.11920*, 2021.

[106] Xiaofei Yang, Xutao Li, Yunming Ye, Raymond YK Lau, Xiaofeng Zhang, and Xiaohui Huang. Road detection and centerline extraction via deep recurrent convolutional neural network u-net. *IEEE Transactions on Geoscience and Remote Sensing*, 57(9):7209–7220, 2019.

[107] George Yannis, Anastasios Dragomanovits, Alexandra Laiou, Francesca La Torre, Lorenzo Domenichini, Thomas Richter, Stephan Ruhl, Daniel Graham, and Niovi Karathodorou. Road traffic accident prediction modelling: a literature review. In *Proceedings of the institution of civil engineers-transport*, volume 170, pages 245–254. Thomas Telford Ltd, 2017.

[108] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[109] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2403–2412, 2018.

[110] Zhuoning Yuan, Xun Zhou, and Tianbao Yang. Hetero-convlstm: A deep learning approach to traffic accident prediction on heterogeneous spatio-temporal data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 984–992, 2018.

[111] Zhuoning Yuan, Xun Zhou, Tianbao Yang, James Tamerius, and Ricardo Mantilla. Predicting traffic accidents through heterogeneous urban data: A case study. In *Proceedings of the 6th international workshop on urban computing (UrbComp 2017), Halifax, NS, Canada*, volume 14, page 10, 2017.

[112] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. *arXiv preprint arXiv:1703.06114*, 2017.

[113] Andi Zang, Runsheng Xu, Zichen Li, and David Doria. Lane boundary extraction from satellite imagery. In *Proceedings of the 1st ACM SIGSPATIAL Workshop on High-Precision Maps and Intelligent Applications for Autonomous Vehicles*, pages 1–8, 2017.

[114] Xiangrong Zhang, Xiao Han, Chen Li, Xu Tang, Huiyu Zhou, and Licheng Jiao. Aerial image road extraction based on an improved generative adversarial network. *Remote Sensing*, 11(8):930, 2019.

[115] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. Road extraction by deep residual U-Net. *IEEE Geoscience and Remote Sensing Letters*, 15(5):749–753, 2018.

[116] Junqiao Zhao, Xudong He, Jun Li, Tiantian Feng, Chen Ye, and Lu Xiong. Automatic vector-based road structure mapping using multibeam lidar. *Remote Sensing*, 11(14):1726, 2019.

[117] Ke Zheng and Dunyao Zhu. A novel clustering algorithm of extracting road network from low-frequency floating car data. *Cluster Computing*, pages 1–10, 2018.

[118] Lichen Zhou, Chuang Zhang, and Ming Wu. D-linknet: Linknet with pretrained encoder and dilated convolution for high resolution satellite imagery road extraction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 182–186, 2018.

[119] Yiyang Zhou, Yuichi Takeda, Masayoshi Tomizuka, and Wei Zhan. Automatic construction of lane-level hd maps for urban scenes. *arXiv preprint arXiv:2107.10972*, 2021.

[120] Zhengyang Zhou, Yang Wang, Xike Xie, Lianliang Chen, and Chaochao Zhu. Foresee urban sparse traffic accidents: A spatiotemporal multi-granularity perspective. *IEEE Transactions on Knowledge and Data Engineering*, 2020.