TurnsMap: Enhancing Driving Safety at Intersections with Mobile Crowdsensing and Deep Learning

DONGYAO CHEN, University of Michigan, Ann Arbor, USA
KANG G. SHIN, University of Michigan, Ann Arbor, USA

Left turns are known to be one of the most dangerous driving maneuvers.[1] An effective way to mitigate this safety risk is to install a left-turn enforcement — e.g., a protected left-turn signal or all-way stop signs — at every turn that preserves a *traffic phase* exclusively for left turns. Although this protection scheme can significantly increase the driving safety, information on whether or not a road segment (e.g., intersection) has such a setting is not yet available to the public and navigation systems.

This paper presents a system, called `TurnsMap`, that exploits mobile crowdsensing and deep learning to classify the protection settings of left turns. One of our key findings is that crowdsensed IMU sensor (i.e., gyroscope and accelerometer) data from onboard mobile devices can be used to recognize different types of left-turn protection. `TurnsMap` first collects IMU sensor data from mobile devices or smartphones carried by the driver/passenger(s) in a moving car. It then feeds the data to an analytics engine powered by (1) a *data mining engine* for extracting and clustering left turns by processing raw IMU data, and (2) a *deep-learning pipeline* for learning the model from the IMU data to identify the protection type of each left turn.

We have built and used a large-scale real-world driving dataset to evaluate `TurnsMap`, demonstrating its capability of identifying different left-turn enforcements with 90.3% accuracy. A wide range of automotive apps can benefit (e.g., enhancing traffic safety) from the left-turns information unearthed by `TurnsMap`.

CCS Concepts: • **Human-centered computing** → *Ubiquitous and mobile computing systems and tools*;

Additional Key Words and Phrases: Transportation Safety, Mobile Sensing, Machine Learning

ACM Reference Format:
Dongyao Chen and Kang G. Shin. 2019. TurnsMap: Enhancing Driving Safety at Intersections with Mobile Crowdsensing and Deep Learning. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 3, Article 78 (September 2019), 22 pages. https://doi.org/10.1145/3351236

1 INTRODUCTION

Left turns are one of the most dangerous driving maneuvers that account for a large percentage of fatal traffic accidents [1]. For example, among the different driving maneuvers at intersections, left-turns are reported to be the deadliest due to oncoming (interrupting) traffic and pedestrians crossing the street the car is turning onto. According to a survey report by the U.S. Department of Transportation, left-turns-related accidents alone constitute *53%* of all intersection-related crashes in the U.S. [1]. In fact, to mitigate left-turns-related accidents, some companies like UPS are enforcing their delivery trucks to avoid, if possible, all left turns [2].

[1] In this paper, we focus on right-hand traffic countries. Our approach can be applied for right turns in left-hand traffic countries.

Authors' addresses: Dongyao Chen, University of Michigan, Ann Arbor, USA, chendy@umich.edu; Kang G. Shin, University of Michigan, Ann Arbor, USA, kgshin@umich.edu.

2474-9567/2019/9-ART78 $15.00
https://doi.org/10.1145/3351236

(a) Protected (b) Unprotected

Fig. 1. Different left-turn settings.

There have been extensive transportation engineering efforts to enhance the safety of left turns by deploying different left-turn regulations at intersections with high-frequency left turns. In general, these regulations can be categorized as protected and unprotected settings.

Protected settings. At a protected left-turn, when the left-turn signal (e.g., a green left arrow as shown in Fig. 1(a)) is turned on, only left-turning vehicles are permitted. This is the *most* effective way to avoid/reduce left-turn-related accidents. According to the NHTSA report [3], an intersection that has a protected left-turn signal can effectively reduce the left-turns-related accidents by **87%**, making the intersection safer for all traffic. Based on the traffic regulation [4], a left turn protection (e.g., a green leftward arrow in the U.S.) reserves a *traffic phase* — a time period assigned to certain traffic movements — exclusively for left turns.

Unprotected settings. At an unprotected left-turn, when vehicles are permitted to make left turns, it is the drivers' responsibility to stay alert and avoid interrupting traffic and/or pedestrians. For example, in Fig. 1(b), circular traffic lights (i.e., traffic lights that only show round-shaped light colors) and stop signs (as shown in the first three figures in Fig. 1(b)) provide less restrictive protection by halting the crossing traffic (i.e., perpendicular to the ego-vehicle's initial heading). The rightmost figure in Fig. 1(b) shows the scenario in which no sign/signal is installed to regulate traffic. Drivers of left-turning vehicles have to be extremely careful of, and avoid interrupting traffic/pedestrians from all possible directions.

The information of left-turn settings is essential for drivers to enhance safety by using it *before* making dangerous left turns; for example, drivers can choose to avoid less-protected left turns when planning a trip route with Google maps.

Unfortunately, such information is not yet available to the public/drivers for two reasons.

- *Lacking public datasets.* Although traffic signals and signs are installed by the local transportation department, the corresponding specification is fragmented or not even digitized due to the diverse development levels for different regions. For example, Waze's left-turn avoidance function [5] is available only in major cities of California.
- *High cost and low update-rate of road surveys.* Information of traffic lights/signs can be extracted from the images of intersections captured by a limited number of heavily instrumented road survey vehicles (e.g., mapping cars for Google StreetView). However, updating the road information is hampered by the prohibitive road-survey cost. As a result, updating the database of on-road images may take years even in well-developed regions. These limitations prevent a large coverage of road survey services and timely reflection of changes of traffic lights/signs.

To narrow this gap and evaluate left-turns safety by discovering and sharing the left-turn setting information on road segments (e.g., intersections), we have developed `TurnsMap`, a novel system that can effectively differentiate

protected/unprotected left turns by using the crowdsensed data (i.e., gyroscope, accelerometer, and GPS data) collected from mobile devices carried by drivers and passengers. In particular, TurnsMap is empowered by mobile crowdsensing and a novel data analytics pipeline. It does not require any additional hardware from the users but harvests the sensor data which are already being generated by various apps (e.g., fitness and health apps) on users' mobile devices. The low overhead/cost of TurnsMap facilitates its timely and economical deployment.

TurnsMap's data analytics scheme dives deep into the real-world driving behavior. Specifically, given a large number of instances (e.g., extracted from crowdsensed data) of left turns, for *less* protected left-turns (i.e., the protected setting is stricter than the unprotected ones), the interruptions caused by on-coming vehicles and/or pedestrians are more likely to occur. To quantify and analyze this characteristic, TurnsMap is powered by two key components. The first component is a *data mining engine*; after collecting the crowdsensed data, TurnsMap extracts left turns by analyzing the raw sensor data. TurnsMap then clusters left turns to find left-turn *hotspots* (road segments that have high-frequency left turns. This concept will be elaborated in Sec. 5). The second component is a *deep learning pipeline* (Sec. 6) that can learn the features of the underlying intersection information from the crowdsensed data. We use a novel data augmentation to enhance the size and representativeness of the dataset, which can be used for training with a recurrent neural network based on using long-short term memory (LSTM) units.

The evaluation of TurnsMap is also challenging due to the lack of mobile sensing-based dataset collected from drivers and ground truths of left-turns information. The mobile driving data is collected from smartphones of 18 different drivers (14 males and 4 females), building a 1.6GB dataset over a cumulated travel distance of more than 3,589km. To collect ground truths, we have designed an interactive traffic-scene annotation system (Sec. 6). We then recruited 231 participants on Amazon Mechanical Turk for the annotation task. Finally, we have aggregated the annotations from the participants to construct a new dataset which is comprised of 965 labeled left-turn hotspots. Our evaluation results show that TurnsMap is able to differentiate left-turn settings with 90.3% accuracy.

This paper makes the following main contributions:

- Building and demonstration of TurnsMap, a mobile crowdsensing system for classifying protected/unprotected left turns;
- Design, implementation and evaluation of a novel data analytics pipeline for large-scale time-series data classifications.

2 MOTIVATION

We motivate this work by answering two questions: (1) how does TurnsMap's revelation of left-turns information actually help the automotive ecosystem? and (2) what would be the advantage(s) of using mobile sensing?

2.1 How Does Classifying Left-Turn Settings Enhance Driving Safety?

As discussed in Sec. 1, the protected left-turn setting are the most effective way to mitigate the safety risk at intersections. However, it is not feasible or even desirable to install the protection at every left-turn hotspot, because strict protection of left turns (Fig. 1(a)) degrades the overall traffic throughput. For example, since a protected left-turn signal only allows left-turning vehicles to go through the intersection, NHTSA suggests such a signal to be installed only at those intersections where all intersecting roads have a similar traffic volume [6]. Since drivers (especially novice drivers with limited driving experience) may choose only protected turn lefts, the information of protection settings of left turns, if publicly available, will help them and automotive apps to enhance safety.

Navigation Systems. As we will elaborate in Sec. 8, planning a route with a minimum number of high-risk (i.e., un-/less-protected) left turns in navigation systems (e.g., Google maps , Waze [5]) by utilizing left-turns enforcement information would be highly desirable. Let us consider a typical usage scenario. If the user enables this functionality on his/her navigation app, then it can exploit the left-turns information on all possible routes to

generate the route with the least number of risky left-turns while ensuring an acceptable *estimated arrival time* (ETA) at the destination. Note that by providing its users with safer travel paths, a navigation app can also efficiently collect data from its users to help expand/update TurnsMap, achieving a *win-win* collaboration between the app and its users.

Self-driving Cars. As highlighted by self-driving car companies (e.g., Waymo [7, 8] and Zoox [9]), handling unprotected left turns is a challenging task for this rapidly emerging technology since it requires not only human-level driving skills but also even psychological understanding [10]. For example, some pedestrians may let left-turning cars go first, while others may ignore or fail to see left-turning cars and cross the street without paying attention to the approaching cars and pedestrians. In such a case, both human drivers and pedestrians could use a body language or even eye contact to resolve this conflict, but it is very challenging for autonomous cars to understand and resolve this type of conflict.

Hence, autonomous cars can avoid such intersections by using TurnsMap to enhance safety for both cars and pedestrians by selecting paths with a minimum number of unprotected left turns in the path planning process [6].

2.2 Why Mobile Sensing for Classifying Protected/Unprotected Left Turns?

TurnsMap outsources the data collection task to the crowd and only processes the IMU sensor data, thus can enable a much larger coverage and faster update-rate than the existing road-survey (e.g., mapping cars of Google Street View) and camera-based approaches, which collect road images by using cameras on either dedicated mapping car or users' smartphones. Specifically, the performance of understanding traffic scenes (i.e., classifying left-turn enforcements) from image data — one of the most common data sources for legacy scene perception tasks, could be hampered by both diverse real-world settings and limited resources:

- *Poor visibility of traffic signals/signs:* The visibility of traffic signals/signs may be distorted due to the weather, lighting condition and even local regulations. Moreover, the location of traffic lights/signs may be different in different regions — they could be located at the middle of an intersection or above a waiting line (e.g., in European countries). In fact, determining a region-of-interest (ROI) based on computer vision is still a challenging open problem [11]. The complexity of accounting for real-world environments makes it even harder for a computer vision task to function correctly on a large and diverse image dataset.
- *Limited coverage:* Due to the different level of development, a vast majority of regions in the world cannot be covered by dedicated road survey services. For example, as of 2017, only a small number of areas are fully covered by Google StreetView [12]. In fact, many countries and regions in Asia and Africa are not yet covered, nor planned to be covered in the near future.
- *High cost/overhead of collecting road image from smartphones:* Processing image data requires high computational power and is thus expensive to run locally on the user's smartphone. Even if the developer chooses to upload the image(s) to the cloud service for processing, transmission of the bulky image and/or video data may exhaust mobile network bandwidth. Moreover, to have a clear view of an intersection, the user has to fix the phone on the windshield, which is a stringent and often undesirable requirement for various use-cases.

The design of TurnsMap aims to collect and publicize the safety-critical left-turns information at low overhead/cost and eventually enhance traffic safety at scale. TurnsMap will be detailed in Secs. 5 and 6.

3 OVERVIEW OF TURNSMAP

As shown in Fig. 2, TurnsMap is comprised of three key building blocks: the *data collection module* for collecting mobile data (i.e., IMU + GPS) from users, *data mining module* for constructing the database of left turns, and *machine learning pipeline* for differentiating left-turn protection schemes. First, TurnsMap collects IMU sensor and GPS data from the mobile devices carried by drivers and passengers (Sec. 4). Next, the data mining module (Sec. 5)

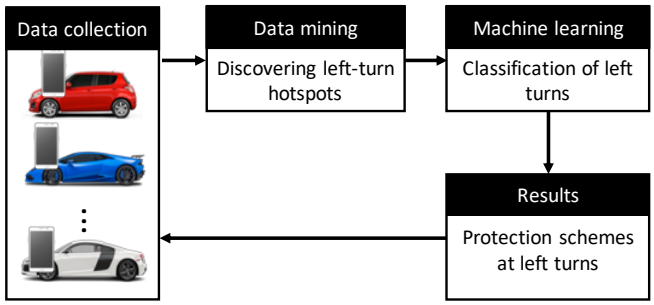


Fig. 2. An overview of TurnsMap

extracts left turns from these data and identifies the road segments, called *left-turn hotspots*, with high-density left-turn driving data traces. To classify protected/unprotected left turns based on people's driving behavior, we devise a machine learning pipeline. For fast and accurate collection of ground truths for the machine learning task, we take a human-in-the-loop approach by outsourcing annotation (i.e., ground truth labeling) tasks on Amazon Mechanical Turk. These efforts together help construct a dataset for traffic scene understanding at intersections. Finally, we propose a recurrent neural network-based model to learn the classifier for differentiating left-turn protection schemes. Note that TurnsMap can help enrich navigation app's functionality (Sec. 8) and enhance users' driving safety. Allowing the TurnsMap framework to intrinsically incentivizes users of automotive apps (e.g., navigation apps on smartphones) to participate in data collection for building a continuously-expanding database. This feature helps formulate a win-win collaboration between the user and TurnsMap.

4 COLLECTION OF DRIVING DATA

TurnsMap classifies left turns using smartphone IMU sensor and GPS data (i.e., the longitude and latitude pair). Specifically, TurnsMap needs a time series of sensor data collected from the user's phone while driving. A ground truth dataset of left-turn protection settings is also needed for building and testing the machine learning algorithm. Since no datasets are available for developing and testing TurnsMap, we collected and constructed a dataset. Our driving data collection methodology is introduced in this section and our approach for collecting the ground truth is highlighted in Sec. 6.

Collection of Real-world Driving Data from Smartphones. There are three key rationales behind the design of our mobile app for collecting the natural driving data. First, different phone postures (e.g., sitting on a phone mount or cupholder) can affect the IMU sensor readings. For example, if the smartphone's screen is facing down, its yaw axis angular speed will be the inverse of the readings with the screen facing up. Second, since smartphones have limited resources (e.g., CPU, battery, and cellular network bandwidth), the data collection should incur low overhead on the users' devices — a high overhead will likely discourage the users in adopting and/or contributing to TurnsMap. Third, a comprehensive database should cover heterogeneous intersection setups, i.e., the dataset should cover different drivers since different driving habits may incur diverse driving maneuvers at left turns. This is an essential requirement for building a comprehensive classifier that is resilient to the changing environment and users.

To handle different phone postures inside a car and achieve consistency in data analytics, Our data collection app aligns the phone coordinate with the earth coordinate by using the coordinate transformation as described in [13]. This capability enhances TurnsMap's usability by eliminating the restriction of the device's posture, i.e., a phone can be placed at any stable position, such as on a cupholder/phone mount, inside the car.

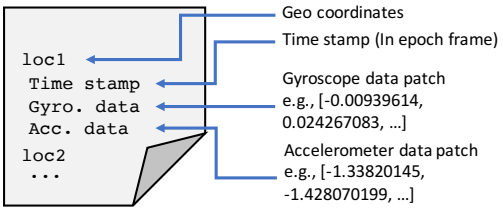


Fig. 3. The snapshot of the data the app collected.

Fig. 4. A suburban area of the collected data.

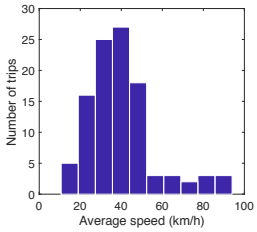


Fig. 5. Histogram of the averaged driving velocity.

Lowering the overhead on the user's smartphone is another key objective of our data collection app. Specifically, the app samples the thus-aligned IMU sensor data and GPS (i.e., (longitude, latitude) pair) data at 100Hz and 1Hz,[2] respectively. The snapshot in Fig. 3 shows the data format — one geolocation sample includes a timestamp vector, a gyroscope data vector, and an accelerometer data vector, where each vector consists of 100 data samples. Because of the light-weight feature of IMU and GPS data, our collection process incurs low overhead on users' mobile devices; our on-road data collection is shown to generate only 6.2MB data per hour. To enhance the flexibility of using their data plan, the participants can choose to upload the data in real time, or when a free Wi-Fi connection is available. Finally, the data collection app is shown to incur only marginal CPU and energy overheads (Sec. 7.2).

We also aim to enrich the diversity (e.g., different driving behaviors and car models) of our dataset from the participants' perspective. To this end, we recruited 18 drivers (14 males and 4 females of age ranging from 23 to 70) and installed our data collection app on their smartphones. The participants are students, university professors, and company employees. We used 10 different car models, including coupe, sedan, SUV, and sporty cars. To collect natural driving data, no restriction was imposed on driving route, vehicle type, smartphone model and location, etc. That is, the participants were told to drive their own cars for daily commute as normally as possible. To secure the participants' privacy and data safety, we applied and received our university's IRB approval (registration number: IRB00100245).

The data collection app has been collected data in a region that includes both urban and suburban environments in Ann Arbor and Detroit metropolitan area in the U.S. Our data-collection process has thus far yielded 1.6GB mobile IMU data, covering an area of approximately 300km^2, with the accumulated travel distance of more than 3,589 km. The data collection was conducted between 7:30 am to 6:30 pm. Each driver Here we show a suburban area of driving data we collected in Fig. 4. The red traces are the driving trajectories. The dataset includes 105 trips contributed by all participants, with the accumulated driving time 78.3 hours. The histogram of the average driving velocity is shown in Fig. 5. The majority of the trips has average speed between 20 – 45 km/h, reflect the normal average driving speed in cities. There are a few trips have high averaged speeds, e.g., exceed 80 km/h. These trips were collected from driving on highways.

5 MINING THE DRIVING DATA

`TurnsMap` infers left-turn protection settings by using the mobile sensor data. We need to answer the following questions for this inferencing. How to differentiate intersection settings by using the mobile sensory data and how to organize the crowdsourced data for this purpose? To answer the first question, we elaborate how IMU sensor readings can capture the interruptions between left-turning cars and the crossing traffic and/or pedestrians. These interruptions are then used to differentiate left turn setups. To answer the second question, we use data mining to extract and cluster the left-turn data snippets to discover left-turn *hotspots*, road segments with many left turns.

[2] 1Hz is the default sampling rate on the current location chipset embedded in smartphones.

Note that a hotspot could be an intersection and even an entrance of a popular plaza. These answers are essential for constructing a comprehensive dataset of left turns for the subsequent machine learning.

5.1 Differentiating Left-Turn Enforcements from IMU Sensor Readings

The key for distinguishing protected and unprotected left turns is the fact that left turns are likely to be interrupted on road segments without strict left-turn enforcements, such as left-turn signals or stop signs. Based on this observation, we uncover the root causes of interruptions which are identified by utilizing phone IMU sensor data.

The root causes of these interruptions are (i) the enforcement-free intersection setup to increase the traffic throughput[3] and (ii) pedestrians crossing the street the car is left-turning to. Due to the low priority given to left-turning cars, their drivers need to pay full attention to any sudden situation change and prepare to pause and/or yield to the oncoming traffic and pedestrians crossing the street they are turning onto.

Fig. 6 (a) shows possible interruptions that a car may experience while turning left at an intersection with a unprotected enforcement (i.e., circular traffic light in Fig. 6 (a)). The most common (as stated in the driver manual [14]) left-turn maneuver works as follows. A left-turning car first enters the intersection after the green light comes on and then suspends its turn if there is an oncoming car from the opposite direction. Upon resuming the left turn, the car needs to pay attention to, and prepare to pause for, pedestrians crossing the street/road it is turning onto. Finally, the left-turning car completes the turn by exiting the intersection.

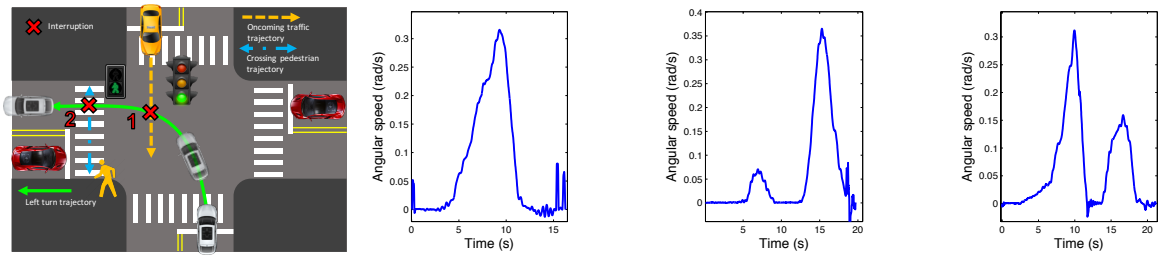
Similar interruptions also occur at the intersections with stop signs. After making a full stop at the sign, the left-turning car needs to proceed until the driver can see the approaching traffic. In summary, interruptions are likely to happen while making unprotected left turns at intersections.

In contrast, interruptions occur rarely at intersections with protected enforcements, because the protected traffic signal (i.e., the left-turn traffic arrow) grants left-turning cars an exclusive traffic phase, hence preventing pedestrians and oncoming vehicles from crossing the intersection. In fact, as stated in the NHTSA intersection engineering handbook [6], one of the main purposes of enforcing left-turns protection is to reduce the left-turning cars' conflicts with the oncoming traffic and crossing pedestrians.

`TurnsMap` uses the smartphone's gyroscope to detect the interruptions. Since an interruption is essentially a pause of steering maneuver, the coordinate-aligned gyroscope data (i.e., angular speed) can capture the differences between normal and interrupted left-turns. Fig. 6 (b)-(d) shows the gyroscope readings of three scenarios: no interruption, interruption at the middle of an intersection (interruption 1 in Fig. 6 (a)), and a left turn with an interruption at the crosswalk (interruption 2 in Fig. 6 (a)). When a car makes a left turn, the gyroscope reading captures the car's angular speed (ω). As depicted in Fig. 6(b), when the driver makes a (normal) left turn without interruption, s/he first turns the steering wheel counter-clockwise, creating a monotonic increase of the vehicle's angular speed ω. The driver then makes a clockwise turn of steering wheel until the car's orientation returns to straight, or ω returns to approximately 0. Hence, a smooth left turn generates a single *bump* in the gyroscope reading. When an interruption occurs, the rise of ω stops and returns to 0, thus creating the first bump in the gyroscope reading; after the driver resumes the left turn, ω will first rise and then drop until the driver exits the intersection, creating the second bump in the gyroscope reading. Figs. 6(c) and (d) show this unique pattern of the gyroscope reading.

The above finding/observation is the cornerstone of `TurnsMap`, showing that IMU data collected during each left-turn contains information useful to capture the left-turn enforcement information.

[3]Oncoming cars from the opposite direction share the same green light phase with left-turning cars.



(a) Interruptions in a left-turn. (b) No interruption (c) Interruption 1 (d) Interruption 2

Fig. 6. (a) Possible conflicts at an unprotected left-turn. The gyroscope readings shows (b) no interruption, (c) an interruption occurs at the middle of intersection, and (d) an interruption occurs at a crosswalk, respectively.

5.2 Extraction of Left Turns from Mobile Sensing Data

The size of the crowdsourced time-series data can be massive, and some portions of the data may be redundant and not useful for TurnsMap, i.e., driving on straight roads. Hence, it is necessary to extract data snippets that contain the targeted driving maneuver (i.e., left turn).

To achieve this goal, TurnsMap extracts left turns from the IMU and GPS sensor data of the driver's smartphone. Specifically, we propose a novel left-turn detection scheme based on the VSense algorithm [13] as detailed in Sec. 5.1.

Let us first review how the VSense algorithm detects left turns from the mobile sensing data. VSense detects vehicle steering maneuvers by harvesting the morphological patterns of the sequential[4] gyroscope data. First, VSense analyzes the gyroscope data to detect the start and end points of a "bump"-shaped curve by using a predetermined threshold derived from the common driving behavior. Next, to validate if the current bump is incurred by a driving maneuver or noise (e.g., system glitch), VSense examines the statistical features (e.g., duration, magnitude) of the bump. VSense also handles false detections. Specifically, since some vehicle movements (e.g., driving on a curvy road, or up/down hill) may also regard a bump-shaped curve as a turning maneuver. To detect these false detections, VSense calculates the moving vehicle's displacement and uses it to validate the detection result. By performing the above steps, a turning maneuver can be detected with *linear* time complexity. We refer the interested readers to Sec.3.2 of [13]. Compared to other sequential data analytics schemes, this method has the following advantages:

- **Resilience to the changing environment:** although approaches based on location information (e.g., GPS data) can detect the vehicle's heading and thus inferring turns, their performance depends on the received satellite signal strength (as shown in [15, 16]). So, they could result in varying performance due to the changing real-world environment.
- **Linear time complexity:** VSense [13] uses the statistical feature in time-series gyroscope data to detect the "bump" (as shown in Fig. 6) that reflects turns, thus only incurring linear time complexity. Compared to legacy time-series classification algorithms such as k-nearest-neighbors dynamic-time-warping (knn-DTW [17]), which has quadratic time complexity, VSense is more efficient in processing large-scale crowdsourced time-series data;
- **Easy parameter setting:** Unlike existing time-series data classifications, VSense does not require any pre-defined pattern but uses the thresholds derived from a natural driving model.

However, TurnsMap faces a unique challenge — one bump may not contain a complete left turn since the turn can be interrupted (e.g., by crossing pedestrian/cars), thus generating two or more bumps in a left turn. To overcome

[4]In this paper, we use the terms "sequential data" and " time-series data" interchangably.

this challenge and accurately crop the sensor data trace that contains a complete left turn, we examine whether or not the neighboring bumps are geo-located adjacently. Specifically, if neighboring bumps on the same IMU trace are adjacent in geo-distance, i.e., great-circle distance between two points on the surface of the earth, then they together represent a single left-turn maneuver. In this line, we propose a two-stage bump detection process. Upon detecting a bump by the VSense algorithm, whether the bump represents a complete left turn is not yet determined. To determine if the pending bump's geolocation is close to the neighboring bump(s), we check if the geo-distance between the centroids of geolocations of those bumps (let loc_{cur} and loc_{pend} denote the centroid of the geolocation of the current bump and the pending bump, respectively) is within a threshold θ_{geo}.[5] The centroid is derived by using $FindCentroid(\cdot)$, which is the mean of latitude and longitude readings. Note that this averaging step can also mitigate the fluctuation of GPS readings. The geo-distance is calculated by using the haversine formula [18]. Thus, if the current and pending bumps are from the same left turn, we merge the current bump with the pending bump(s). Otherwise, the current bump is a complete left turn. Finally, the start and end timestamps (i.e., $t_{cur,start}$, and $t_{cur,end}$) of the current bump can be used for cropping the left-turn data snippets from all necessary time-series data traces — gyroscope, accelerometer, and geolocation data. For example, a framed The left-turn detection algorithm is summarized in Algo. 1.

Algorithm 1 Left-turn detection algorithm of `TurnsMap`

Input gyroscope (Ω), accelerometer (Φ), and location (Γ) data traces
for ω in Ω **do**
  Detect a valid bump based on VSense algorithm
  Find the timestamp range of the current bump $[t_{cur,start}, t_{cur,end}]$
  $loc_{cur} = FindCentroid(\Gamma_{t_{cur,start}}^{t_{cur,end}})$. Here, $\Gamma_{t_{cur,start}}^{t_{cur,end}}$ is the cropped location data trace based on timestamp
  if $Haversine(loc_{cur}, loc_{pend}) \leq \theta_{geo}$ **then**
    /*Update the bump information*/
    $t_{cur,start} = t_{pend,start}$
    $loc_{cur} = FindCentroid(\Gamma_{t_{cur,start}}^{t_{cur,end}})$
  else
    /*$Range[t_{cur,start}, t_{cur,end}]$ includes a complete left turn*/
    Extract IMU and location data snippets (i.e., $\boldsymbol{gyro}$, $\boldsymbol{acc}$, and $\boldsymbol{loc}$) respectively.
    Calculate the centroid of this left turn: $c_k = FindCentroid(\boldsymbol{loc})$
  end if
  Update the timestamp range: $t_{pend,start} = t_{cur,start}$, $t_{pend,end} = t_{cur,end}$, $loc_{pend} = loc_{cur}$
end for

We can form the output of left-turn detection as follows. To organize the thus-extracted heterogeneous data (e.g., IMU sensor and geolocation traces), for *each* left-turn maneuver, we store its data as a tuple called *left-turn tuple* $\boldsymbol{l}$:

$$\begin{aligned} \boldsymbol{l}_k &= \{c_k, \boldsymbol{gyro}_k, \boldsymbol{acc}_k, \boldsymbol{loc}_k\}, \\ \boldsymbol{L} &= \{\boldsymbol{l}_1, \cdots, \boldsymbol{l}_k, \cdots, \boldsymbol{l}_K\} \end{aligned} \tag{1}$$

where $k \in \{1, \ldots, K\}$, K is the total number of left turns extracted by Algo. 1; c_k is the centroid of the k-th left turn; $\boldsymbol{gyro}$ and $\boldsymbol{acc}$ are IMU sensor vectors cropped from IMU sensor traces; $\boldsymbol{loc}$ is the geo-location data trace. $\boldsymbol{L}$ is the list that stores all left-turn tuples.

[5] $theta_{geo}$ is derived from the diameter of road intersections. We use θ_{geo} = 30m.

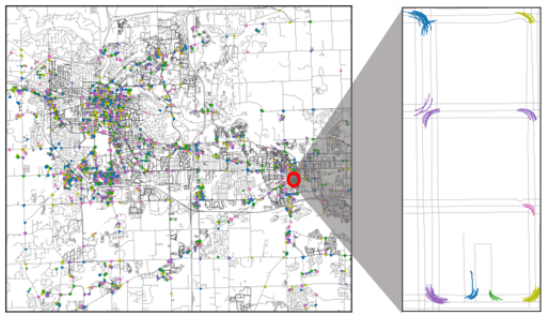


Fig. 7. Clustering result of DBSCAN. Each cluster of left-turn traces is a left-turn hotspot.

5.3 Construction of Left-turn Hotspots

It is not possible to capture the left-turn setting of an intersection based only on discrete left-turn traces. There could also be false detections of left turns in the left-turn extraction step; for example, a swerving maneuver at a parking lot could be classified as a left-turn maneuver. To address this problem, we cluster multiple left-turn traces to form *left-turn hotspots* — road segments with left turn maneuvers.

We propose a DBSCAN-based clustering [19] method to discover left-turn hotspots from $\boldsymbol{L}$. DBSCAN is a density-based algorithm that does not require a pre-defined number of clusters and is proven to perform well in clustering spatial data. In our case, $DBSCAN(\cdot)$ uses the centroid of each left-turn instance as the clustering criterion. The output (i.e., C) of $DBSCAN(\cdot)$ is a collection of clusters, which is comprised of left-turn tuples ($\boldsymbol{l}$s). We only keep those clusters with "enough" left turns — any cluster with less than 5 left-turn tuples are discarded owing to the insufficient number of traces. Finally, for each cluster, we group its gyroscope, accelerometer and location traces, respectively, to derive left-turn hotspots. Our clustering algorithm is summarized in Algo. 2.

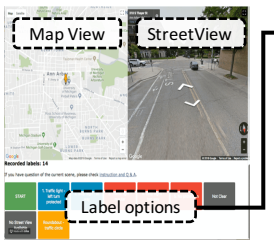
Each of the thus-constructed left-turn hotspot can be represented by a data tuple $\boldsymbol{x}$:

$$\begin{aligned} \boldsymbol{x}_i &= \{c_i, \boldsymbol{Gyro}_i, \boldsymbol{Acc}_i, \boldsymbol{Loc}_i\}, \\ \boldsymbol{X} &= \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_N\} \end{aligned} \tag{2}$$

where $i \in \{1, \ldots, N\}$, N is the number of left-turn clusters. $\boldsymbol{Gyro}$, $\boldsymbol{Acc}$, $\boldsymbol{Loc}$ denote the thus-aggregated gyroscope, accelerometer, and location traces, respectively. $\boldsymbol{X}$ is the list of left-turn hotspot tuples.

Algorithm 2 Cluster Left Turn Traces

```
Input:
C = DBSCAN(L)
for each cluster in C do
    if Number of left-turn instance in this cluster is less than 5 then
        /*Discard this cluster*/
    else
        c =FindCentroid(centroids of left-turn tuples in this cluster)
        Aggregate gyro, acc, and loc of each left-turn tuple into set Gyro, Acc, and Loc, respectively.
        The data tuple of this left-turn hotspot is {c, Gyro, Acc, Loc}.
    end if
end for
```



Option	Description
1	Traffic light - protected
2	Stop sign – all-way
3	Traffic light - regular
4	Stop sign – two-way
5	Unprotected
6	None of the above

Fig. 8. Online annotating system of TurnsMap. The left figure is a screenshot of its interface (the instruction section is omitted due to space limit). The right table shows the options available to the annotators.

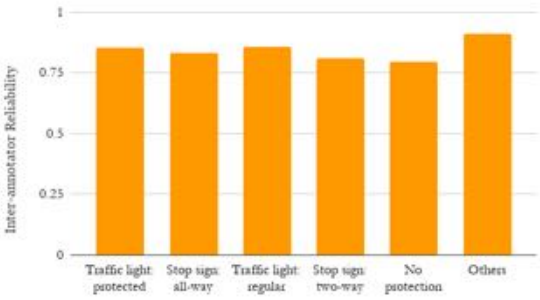


Fig. 9. Inter-annotator reliability of each type of left-turn protection.

Fig. 7 shows part of our clustering result. The location traces are interpolated to show the trajectory of a left-turning car. In the zoom-in view, one intersection is not covered due to lack of driving data, but the other intersections are clustered by DBSCAN. We evaluate the performance of our hotspot extraction in Sec. 6.1.

6 DEEP LEARNING FRAMEWORK

Thus far, we have constructed a dataset $\boldsymbol{X}$ of time-series traces at left-turn hotspots. To enable classification of left-turn settings by using the time-series data, we first collect the ground truth of each left-turn hotspot in $\boldsymbol{X}$, and then use a deep learning pipeline to train the classifier.

6.1 Collection of the Ground Truth

Due to the lack of the ground truth of each hotspot's left-turn setting, we need to collect the ground truth from sratch. A human-in-the-loop method is used to collect this information efficiently. Specifically, we outsource annotation tasks to the crowd on Amazon Mechanical Turk (AMT) and use Google StreetView — a large-scale image database that includes traffic scene information collected by mapping cars — as the accessible reference for the annotators to identify the left-turn protection setting of each hotspot.

An easy-to-follow annotation process is essential to minimize the participants' confusion for good quality of annotation. To meet this requirement, we design an interactive labeling/annotation webpage as the participants' working platform. Described below is how to annotate each left-turn hotspot on this webpage. Before starting the annotation, each AMT participant will be asked to read our instruction carefully to understand the annotation process. Next, our backend system (a php script running on the server for hosting the webpage) will randomly pick a hotspot $\boldsymbol{x}$ from $\boldsymbol{X}$ and then display the corresponding Google StreetView based on the centroid of $\boldsymbol{x}$. The annotator then needs to inspect the StreetView (i.e., by zooming in/out and changing the viewing angle) to get a clear view of the left-turn protection setting. The annotator will then be asked to choose one of the six options in the table of Fig. 8. Option 1 represents the protected left-turn signal; options 2–5 the unprotected left turns; option 6 none of the five aforementioned options according to the participant's perception/judgement, e.g., parking lots and/or road segments without any StreetView image. Finally, our backend system will record the annotator's input, and a new hotspot will be displayed on the webpage.

After publishing the annotation task on AMT, we recruited 231 annotators, and asked each of them to *independently* annotate/label 30 randomly-selected hotspots from $\boldsymbol{X}$. Each annotator spent an average of 26.5min to complete the task, which is reasonable for a new annotator to understand and annotate hotspots.

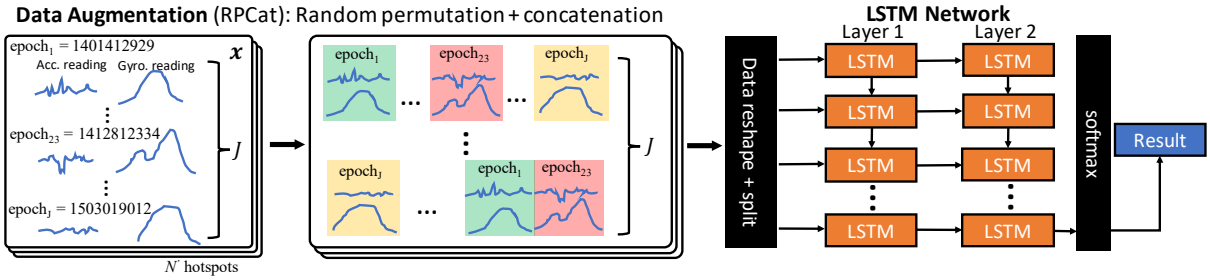


Fig. 10. The machine learning pipeline. The left frame shows the gyroscope and accelerometer traces in a left-turn hotspot $\boldsymbol{x}$; the middle frame shows RPCat permutation and concatenation of IMU data in $\boldsymbol{x}$; the right-hand side network shows the LSTM-based network architecture.

Although crowdsourcing the collection of ground truths via AMT can significantly speed up the annotation, erroneous annotations are inevitable since the AMT participants are not domain experts and may have different levels of perception. Thus, we try to enhance the quality of the collected annotations as follows.

- Recruiting proficient annotators. We ensure the annotators' competence by only recruiting "master" workers [20] — elite workers who have demonstrated superior performance in their previous tasks and have been acknowledged by the corresponding requesters. We must also pay the master workers more to incentivize their participation.
- Ensuring participants' clear understanding of the annotation task. A thorough and clear understanding of the annotation task can improve the participant's performance. To meet this goal, we only recruited drivers who reside in the U.S., since our driving data is collected from a US metropolitan city. Our easy-to-follow instruction helps the annotators understand different left-turn protection mechanisms and the annotation process.
- Aggregating the collected annotations via majority voting. Annotated data need to be filtered and refined for accuracy. We first ensure each left-turn hotspot received multiple annotations from the participants. Specifically, we discard any left-turn hotspot with less than 3 annotations. Then, for each remaining hotspot, we apply simple majority voting to determine its annotation. That is, we select the annotation with the largest number of repetitions. Note that a similar criterion is also commonly used in other large-scale ground truth collections, e.g., ImageNet [21] and DeepDrive [22].

We randomly select 1,000 hotspots from $\boldsymbol{X}$ and collected 6,016 annotations — each hotspot has an average of 5.47 annotations. We assess the quality [23] of our dataset by inspecting the inter-annotator reliability of each left-turn protection setting, which measures the consistency among different annotators. It is a commonly used evaluation metric for assessing the quality of annotations collected from the crowd [24]. As shown in Fig. 9, the inter-annotator reliability of each category is around 85%.

Moreover, the annotation result can also help us understand the performance of our left-turn hotspot extraction (as stated in Sec. 5). Since false detections of a hotspot may be due to cars' swerving in open areas (e.g., parking lots), we can use the annotators' feedback on each hotspot to inspect if the hotspot is indeed a road segment that can facilitate left turns. According to the aggregated annotation results, 96.5% (965 out of 1,000) of the annotated hotspots are road segments that can be categorized into one of the five scenarios as shown in Fig. 1. Our left-turn hotspot extractions shown to be able to accurately identify road segments that can facilitate left turns.

Finally, we construct our dataset by using the annotated hotspots. Specifically, for hotspot i, we denote its ground truth as y_i if it has an annotation that passed our quality control test. Thus, we can have

$$\begin{aligned} \boldsymbol{X} &= \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_{N'}\}, \\ \boldsymbol{Y} &= \{y_1, \ldots, y_i, \ldots, y_{N'}\} \end{aligned} \tag{3}$$

where $y \in \{1, 2, 3, 4, 5\}$ as shown in Fig. 8; $i \in \{1, \ldots, N'\}$, N' is the number of valid hotspots with valid annotations. In the current dataset, $N' = 965$. Note that by associating $\boldsymbol{X}$ with $\boldsymbol{Y}$, we can have $(\boldsymbol{x}, y)$ pairs, which can be applied in supervised machine learning.

6.2 Formulation of the Learning Problem

Classification of protected/unprotected left turns based on mobile sensing data at hotspots ($\boldsymbol{X}$) can be cast as a supervised learning problem for binary classification of time-series data. We modify $\boldsymbol{Y}$ to form a binary classification task: label 1 (as shown in Fig. 8) is categorized as a protected left turn while labels 2–5 are categorized as unprotected left turns. That is, y_i ={Protected, Unprotected}, $i \in \{1, \ldots, N'\}$, $N' = 965$. As a result, we have 206 protected and 759 unprotected hotspots. The supervised machine learning problem can be formally state as follows.

Problem statement. Given N' pairs of data of the form $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{N'}, y_{N'})\}$, the objective is to learn a model $\boldsymbol{X} \rightarrow \boldsymbol{Y}$, where y is an annotation, $\boldsymbol{x}$ is the measurement represented as a tuple of several time-series traces.

The key challenge in this learning task is how to form the input data by using the observation tuples (shown in Eq. (2)). The input data formation is different from many existing time-series classification tasks, such as pattern matching [25], and DeepSense [26] — a recent light-weighted deep learning framework which is optimized for running on mobile devices. Specifically, in the legacy time-series classification, each observation $\boldsymbol{x}$ is a single data trace (vector). For example, to train a classifier for the GaitID task (identifying a user based on his/her gait traces), each observation in the training set is a 2-dimensional vector of IMU sensor readings. However, in `TurnsMap`, each observation is a *tuple* of data traces. As shown in Eq. (2), the data of each observation $\boldsymbol{x}$ is comprised of $\boldsymbol{gyro}$ and $\boldsymbol{acc}$, which are sets of gyroscope and accelerometer data vectors of different lengths. Without such data formation, it is not possible to distinguish left-turn settings based on one time-series data vector. For example, the left side frame in Fig. 10 illustrats the accelerometer and gyroscope data traces collected from an unprotected left turn. Note that not all traces exhibit the interruption pattern, i.e., some of the collected time-series traces are as smooth as if they were collected from a protected left turn. Therefore, a machine learning pipeline needs to pre-process the time-series data tuple to form data that can be used as input for the machine learning algorithm.

6.3 Deep Learning Pipeline

To address these challenges, we propose a deep learning pipeline that is comprised by a novel data augmentation process and a deep Recurrent Neural Network (RNN) based on Long-Short Term Memory (LSTM) cells.

6.3.1 Data Augmentation. The goal of the data augmentation [27] is to construct and expand the training dataset without undermining the underlying pattern. For this purpose, we design *randomly permuted concatenation* (RPCat) for each hotspot data, i.e., $\boldsymbol{X}$. As shown in the second frame on the left of Fig. 10, RPCat has two steps: (i) expand the set of observations by permuting the sequence within $\boldsymbol{x}$; (ii) concatenate the data traces of each permuted observation. This approach is inspired by [28] in the area of natural language processing (NLP), where data snippets (e.g., phrases in NLP) are concatenated together to form an overview of the overall dataset.

Let us illustrate RPCat using an example shown in Fig. 10. Suppose a hotspot's data $\boldsymbol{x}$ is comprised of J gyroscope and J accelerometer traces (i.e., J is the number of the clustered left-turn maneuvers at this hotspot). The sequence of data traces in $\boldsymbol{x}$ is originally organized according to the timestamp (we use epoch time in `TurnsMap`) of the collected data. Note that different left turns at this hotspot are independent of each other. That is,

$$P(T_1, \ldots, T_j, \ldots, T_J) = P(T_1) \cdots P(T_j) \cdots P(T_J), \tag{4}$$

where T_j is the j^{th} left-turn event and $P(T_j)$ is the probability of T_j's occurrence. Due to the independence of different left turns, the arrival sequence of these data traces should *not* affect the classification result.

RPCat reflects the this insight and reconstructs $\boldsymbol{x}$ as shown in the second frame in Fig. 10. Specifically, RPCat firstly *permutes* the sequence of data traces in $\boldsymbol{x}$ and then *concatenates* the data. After each permutation, RPCat generates a new sequence of data traces. Finally, to unify the data size for machine learning, we resize each trace to length $FixLen$. We repeat RPCat J times to obtain a $J \times FixLen \times 2$ tensor for $\boldsymbol{x}$. We execute the above process for all N' hotspots. Finally, after reshaping and splitting the data based on batch size, we transfer the data into the input tensor for the LSTM network.

The parameters of the current RPCat are set as follows:

- **Repetition of permutation.** This parameter is to balance the dataset. For example, unprotected hotspots constitute the majority of the current dataset due to their prevalence in the real world. To avoid an imbalanced dataset, if $\boldsymbol{x}$ is an unprotected hotspot, we repeat the permutation J times, whereas for each protected hotspot, we permute $2J$ times. This allows us to have a more balanced dataset, which is necessary for better performance in training the LSTM network. We do not repeat RPCat full permutation times because permutation of J unique elements ($Perm(J,J)$) can be very large, e.g., $Perm(10,10) = 3,628,800$. Such a large dataset would be imbalanced and can take prohibitive long time for training.
- **Length of data traces.** Our current implementation uses $FixLen = 800$. The optimization of the unified data length is left as our future work.

It is importation to note that the permutation does *not* undermine the sequential pattern of the data for `TurnsMap`. As highlighted in Sec. 5, the sequential pattern (e.g., interruptions) of left-turn data resides in each left-turn maneuver. Since the permutation only shuffles the order of left turns, it does not change the sequential pattern of each left-turn data.

Now, the remaining question is how to learn the pattern based on the augmented training data.

6.3.2 RNN Framework. We use a deep RNN framework with LSTM cells to learn the underlying pattern(s) from the augmented data. As a specific type of RNN architecture, LSTM has proven advantages [29] over other machine learning algorithms, such as SVM, random forest, convolutional neural network, etc., for classifying sequential data. Specifically, each LSTM cell uses a forget gate to adjust the extent of passing the state variable to the next unit — an important feature for capturing dependencies through time [30]. Moreover, the forget gate design is shown to have a desirable effect on mitigating the vanishing gradient problem [31], a common limitation for the classical RNN architecture that may restrict the network from learning the pattern from sequential observations. We used stacked LSTM since a deeper hierarchical structure enables the network to gain more accurate understanding of the intrinsic pattern by disseminating the learning task to each layer [29, 32].

The hyper-parameter settings of our network are: 2 LSTM layers, 32 LSTM cells per layer, batch size of 300, and the training rate of 0.001. We use the tanh function for updating the cell and hidden state, whereas the sigmoid function is used for the gates in LSTM cells. To mitigate overfitting, we applied a 0.5 dropout rate in each LSTM layer. The Adam optimizer [33] is used for iteratively updating network weights. Finally, the network feeds the thus-learnt feature vector (1×32) into a softmax layer for generating the classification result.

Now, we first test the efficacy of our RPCat + deep LSTM pipeline by evaluating the training loss. To obtain the test data, we randomly selected 20% of hotspots from $\boldsymbol{X}$ and then applied RPCat to build an augmented testing set. We have implemented our learning pipeline with TensorFlow on our lab server equipped with one Titan Xp GPU. Fig. 11 shows the efficacy of RPCat and also the training and testing loss (training/testing loss is the penalty of misclassification of training/testing set [31]) by using our learning pipeline. Without applying RPCat, both training and testing losses converged to high values, indicating underfitting — the LSTM network does not have enough data for learning the underlying pattern. With RPCat, despite the fluctuation caused by the convergence feature of Adam optimizer, both training and testing losses decrease with the increasing number of epoch (in neural

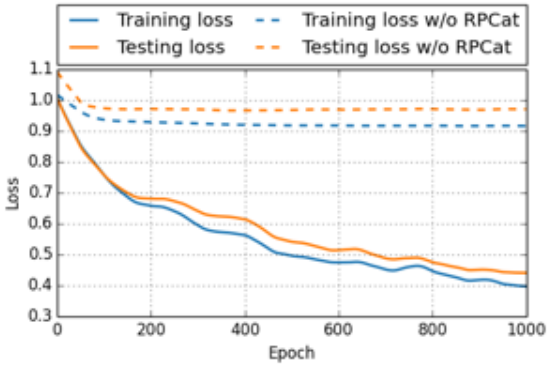


Fig. 11. Efficacy of RPCat on training with LSTM.

network epoch, one epoch means the number of times the entire dataset is processed by the algorithm). Specifically, training and testing losses are stabilized at 0.4 and 0.45, respectively, after about 900 epochs. We will evaluate the performance of our machine learning pipeline in Sec. 7.

7 EVALUATION

We first evaluate the overhead of TurnsMap, then the performance of our deep learning pipeline, and finally present the insights gained from the evaluation.

7.1 Performance Metrics of TurnsMap

7.1.1 Evaluation Metrics of the Deep Learning Pipeline. We evaluate our machine learning pipeline's performance by using the cross-validated performance metrics. Fig. 12 shows the confusion matrix where each row represents the actual class; each column represents the predicted class. Fig. 13 shows the precision, recall and F-1 score, where precision represents the ratio of true positive detections to the total positive detections, and recall represents the classifier's sensitivity with the ratio of true positive detections to the total *actual* positive observations. Accordingly, our classifier can identify different left-turn settings with high performance. Note that the number of instances of *unprotected* settings takes a large portion in our dataset due to its popularity in real-world, our classifier is not biased as evidenced by the performance metrics in Fig. 13. This shows the model trained on the augmented (and thus balanced) dataset is not biased and our LSTM framework successfully learns the feature(s) for differentiating left-turn protection schemes.

TurnsMap errs on the safe side of misclassifications to mitigate the impact of erroneous classifications. Specifically, it has a *lower* rate of misclassifying unprotected left turns as protected ones than that of classifying protected left turns as unprotected ones. This means that while identifying protected left turns with good accuracy, TurnsMap achieves better performance in identifying the unprotected left turns.

The performance of TurnsMap is essential for many applications, such as finding the safest left turns for novice drivers and route planning for self-driving cars, selecting a road with the minimum number of unprotected left turns using TurnsMap.

Now, we examine our classification performance based on the unbalanced dataset. To evaluate the classifier's performance while varying the threshold (i.e., cut-off probability), we first tested the performance and report the Receiver Operator Characteristic (ROC). Then, to evaluate whether performance of protected left turns detection

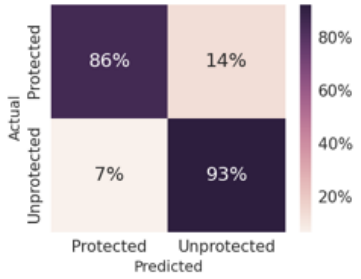


Fig. 12. The normalized onfusion matrix for analyzing TurnsMap's performance.

Category	Precision	Recall	F-1
Protected	0.90	0.86	0.88
Unprotected	0.91	0.93	0.92
Average	0.90	0.90	0.90

Fig. 13. Performance metrics of the LSTM-based pipeline.

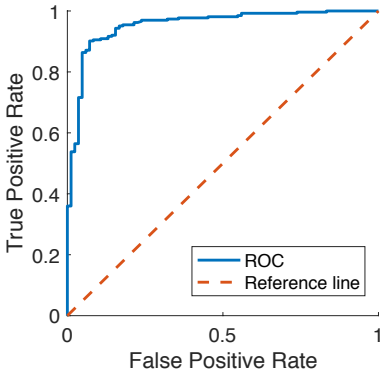


Fig. 14. ROC curve.

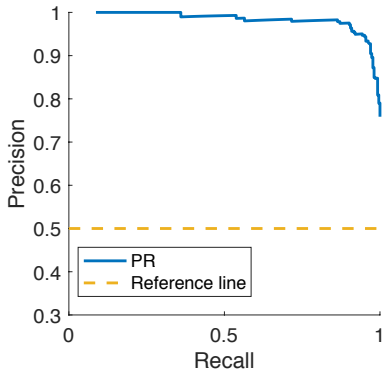


Fig. 15. Precision-recall (PR) curve.

is not dwarfed due to the imbalance of the current dataset, we also reported the Precision-Recall (PR) curve to examine if precision captures the number of *false positives*.

As shown in Figs. 14 and 15, both the ROC and PR curves show the efficacy of our classifier. Specifically, both curves have significantly better performance than their corresponding reference lines (i.e., random guess). Moreover, the area under curve (AUC) for ROC and PR curves is 0.954 and 0.926, respectively.

7.1.2 Comparison with Other Machine Learning Algorithms. Our model's performance is compared with the models trained by other learning algorithms based on the augmented dataset, and the *averaged* performance metrics are plotted in Fig. 16 where we tested a standard RNN, which is known to have good performance for sequential data [34]. One of its key differences from LSTM is that the standard RNN cell lacks gates (e.g., forget gate) for learning the sequential dependencies that may have varying (e.g., short- or long-term) timespans. We also compared with kernelized support vector machine (SVM), and random forest (RF). Specifically, the RNN uses the same hyper-parameter setting as our LSTM network. We used 200 decision trees for the RF algorithm. Note that we used the same training data setting for testing different algorithms. That is, the performance of RF and SVM with feature vector optimization is not the focus of this paper.

For TurnsMap, LSTM outperforms the others in all metrics considered. As discussed in Sec. 6, the advantages of LSTM over others is its capability of capturing dependencies through time. For example, RNN cells are susceptible

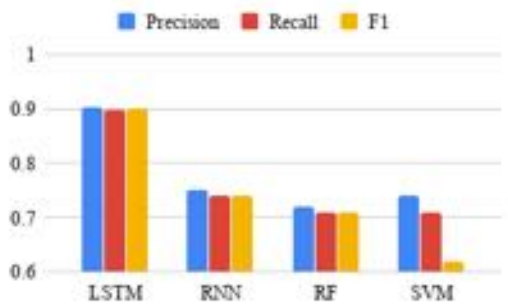


Fig. 16. Comparison between LSTM and other machine learning algorithms.

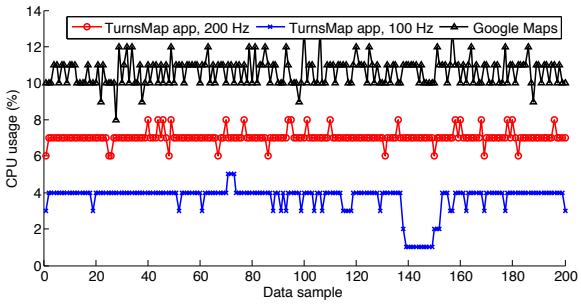


Fig. 17. CPU usage of TurnsMap app.

to the vanishing gradient decent problem, making it very hard to capture the dependencies in time-series data. SVM has a low average F-1 score, because it misclassified many unprotected hotspots as protected ones while showing good performance in detecting protected hotspots.

7.2 Overhead of the Data-collection App

Mobile devices are resource-limited and susceptible to power and/or computation-hungry apps. A high overhead of the data-collection app could consume lots of the mobile device's resources, thus discouraging use of TurnsMap. We have highlighted the data-collection app's generation of low network traffic in Sec. 4. Here we test its additional CPU usage and energy consumption on the users' mobile devices.

We installed our data-collection app on a Google Pixel phone that runs Android 7.1.2 Nougat OS. We experimented with two sampling rates of IMU sensor: 200Hz and 100Hz. We used the 100Hz sampling rate for most of time and also tested the 200Hz sample rate to study the maximum overhead incurred to users' smartphones.

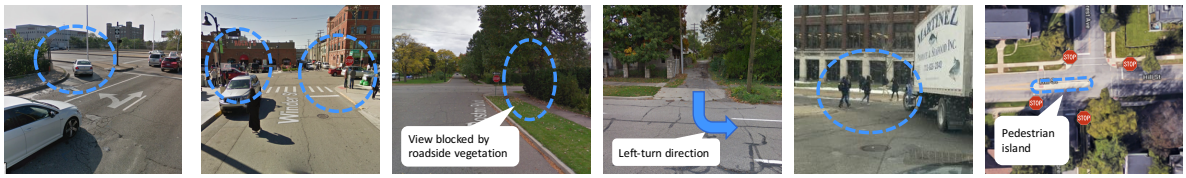
Since TurnsMap can be used as a collaborative add-on functionality for existing navigation apps (we will elaborate on this in Sec. 8), we also measured the overhead of a popular navigation app (i.e., Google Maps) and compared it with our data-collection app. Fig. 17 shows the CPU usage overhead of TurnsMap. It consumes an average of 7% and 3.7% CPU for 200Hz and 100Hz sampling rate, respectively, whereas Google Maps consumes 11% CPU. For the energy consumption, we measured the current drawn by the Pixel phone. It shows that the data collection app consumes only 45mA with IMU sensor sample rate of 100Hz, whereas Google Maps consumes 727mA. That is, the extra energy consumption is only 6.2% of the navigation app. This low overhead comes from the fact that the data collection activity does not use any power-hungry sensors (e.g., camera) and operates in background mode.

7.3 Real-world Insights from TurnsMap

TurnsMap can accurately differentiate the protection settings of different left turns. In the testing phase of TurnsMap (i.e., classifying testing data with the thus-learnt model), we have gained some interesting insights (Fig. 18).

7.3.1 Protected. One of the left turns with protected settings is shown in Fig. 18(a) — an isolated left-turn *road* in an urban area. The driver can use this dedicated road for left turns, a complete isolation from other traffic and pedestrians.

7.3.2 Unprotected. Fig. 18(b) shows a busy market in an urban area where left-turning cars are likely to be blocked by a large crowd of people and street stalls. Fig. 18(c) shows an intersection enforced by a 2-way stop sign where the



(a) Isolated left lane (b) Busy market (c) Blocked view (d) Exit w/o prot. (e) Crossing student (f) Pedestrian island

Fig. 18. Real-world insights from TurnsMap result.

view of right-hand-side road is blocked by lush vegetation. Fig. 18(d) shows an intersection without any protection, which is also a joint spot of a narrow road with a major road (with a speed limit of 45mph). Such a left turn is very risky because the left-turning driver needs to pick up the speed quickly while paying attention to fast-approaching cars from the right-hand side. Fig. 18(e) shows an intersection with all-way stop sign located inside of a campus. Especially during inter-class time (e.g., 10:30am on weekdays), many students travel through this intersection to get to other classrooms, thus increasing the possibility of causing frequent conflicts between left-turning cars and pedestrians. Fig. 18(f) shows a confusing layout in a suburban area, where each entrance of the intersection is enforced by a stop sign, but there is a pedestrian island in the middle of the street. This blockage in the middle of the road requires drivers to be extra careful when turning left.

8 USE CASE AND USER STUDY

Since left-turn enforcement is critical for driving safety, TurnsMap has potential for use in various app scenarios, such as navigation systems for both human- and self-driving cars, and reporting unprotected left-turn hotspots to the local transportation department. In this section, we focus on assessing TurnsMap's utility in navigation systems.

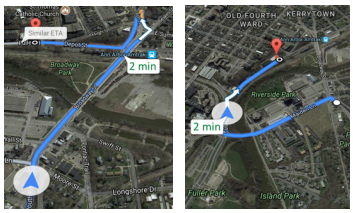
In existing navigation systems (e.g., Google map), one of the key limitations is the lack of safety assessment (as highlighted by Waze [5], finding dangerous left turns) of different routes. In fact, route/trip planning selects the route by minimizing the estimated time of arrival (ETA) [35]. As the result, the navigation system may generate a route with many risky unprotected left turns, as shown in Fig. 19(a).

One of TurnsMap's exemplary application is augmenting existing navigation app with left turn setting information at intersections. That is, with the left-turn information discovered by TurnsMap, TurnsMap can provide this information with the collaborating navigation app to plan a route that also considers left-turn safety at intersections — which is not yet achieved functionality by using state-of-the-art approaches. Specifically, users of a navigation app can enable unprotected left turn avoidance in the setting of the app. Thus, the app can suggest an alternative route, as shown in Fig. 19, which has safer turning maneuvers (e.g., right turns).

To fully analyze user's satisfaction on this functionality, we first conducted a survey of users' expectations of a navigation system that has the unprotected left-turn avoidance. Specifically, we recruited 564 participants from Amazon Mechanical Turk. Each participant needs to be a legitimate driver in the US to participate in our study. To assess people's expectation of this functionality, we asked the participants the following questions:

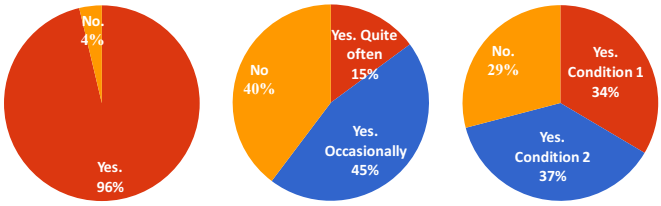
Q1. Can you differentiate left-turn enforcements?
Q2. Have you ever experienced dangerous/difficult unprotected left turns recommended by a navigation app?
Q3. If there is a navigation app that can help you avoid those risky left turns, would you use it?

Fig. 20 summarizes the survey results. For Q1, 96% of the participants can differentiate between protected and unprotected left-turns, indicating that most of our participants are aware of the protected/unprotected settings. For Q2, 60% of the participants experienced risky/difficult unprotected left turns recommended by the navigation app. Notice that 15% of the participants *often* encountered such left turns in the routes recommended by the navigation



(a) w/o TurnsMap (b) w/ TurnsMap

Fig. 19. Exemplary application. Here, (a) and (b) compares the usage of a navigation app w/ or w/o TurnsMap's capability.



(a) Question 1 (b) Question 2 (c) Question 3

Fig. 20. User study of the exemplary app. Here, (a), (b), and (c) show distributions of participants' responses to our survey questions respectively.

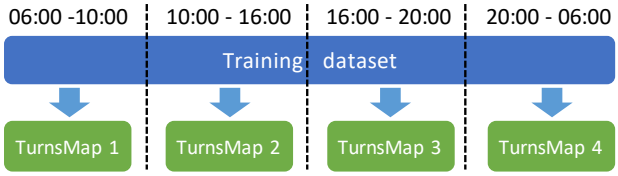


Fig. 21. Illustration of formulating TurnsMap for different time-of-day.

app. For Q3, 71% would like to use a navigation app that avoids unprotected left-turns while 34% prefer this functionality only if the alternative route has a similar ETA (Condition 1 in Fig. 20(c)). Surprisingly, *37%* indicated that if driving safety can be enhanced, they won't mind taking an alternative route with a prolonged ETA (Condition 2 in Fig. 20(c)), i.e., safety is more important to a large portion of navigation app users. In summary, the avoidance of unprotected left turns is highly desired for navigation systems.

9 LIMITATION AND FUTURE EXTENSION

Although TurnsMap can achieve high accuracy in differentiating protected and unprotected left-turn settings, there remains room for improvements, two of which are highlighted below.

9.1 Adapting TurnsMap to the Time-of-day

Currently, TurnsMap is trained based on driving data that is mostly collected during day-time (i.e., most of our training data is collected during 7:00 –18:00. Also, some intersections are equipped with advanced traffic regulation systems [36] that change the left-turn protection setting according to different times-of-day to accommodate the varying traffic conditions. The intersections that allow for flexible left-turn settings need both regular traffic light (i.e., circular light) and dedicated left-turn light (i.e., a green left-oriented arrow). For example, to maximize the traffic throughput during the morning rush hour, the intersection can use the regular traffic light to regulate left-turning traffic; during hours with moderate traffic, it uses the protected traffic light to enhance safety at the expense of traffic throughput. The detailed information (e.g., location and the specific left-turn setting) of these flexible traffic signals would be useful for various parties (drivers, navigation app developers). However, this information is unknown to the public.

TurnsMap can adapt to this sophisticated scenario by learning different classification models for different times-of-day. As shown in Fig. 21, 4 times-of-day can be 6:00 — 10:00, 10:00 — 16:00, 16:00 — 20:00, and 20:00 —

6:00 (the next day). Note that the first and third time segments denote the two usual daily rush hours. Based on the segmentation of time, one can divide the training data using the timestamp of data collection and train the corresponding TurnsMap for each time segment.

9.2 Scalability of TurnsMap

We test TurnsMap by using the driving data collected from the Detroit metropolitan region in the U.S. that covers various on-road environments (e.g., suburban and urban areas). Moreover, since TurnsMap harvests interruptions (as elaborated in Sec. 5), the root cause of risks at intersections, TurnsMap can be used to enhance driving safety at large scale. Specifically, TurnsMap can be used in cities/regions that have the same/similar traffic regulations. For cities/regions with different traffic regulations, the current trained model of TurnsMap can help facilitate data collection during the training phase. In particular, the app developer or local transportation department could apply the transfer learning method [37] to use the existing model as a pre-trained model.

10 RELATED WORK

10.1 Detecting the Driving Behavior

One of the major tasks of TurnsMap is to accurately detect the driver's maneuver based on mobile IMU sensor readings. Recently, there have been a growing number of research efforts to meet similar goals. For example, Yang *et al.* [38] used the smartphone accelerometer to detect if the phone is at the driver's seat. VSense [13] uses smartphones' built-in IMU sensors to detect various steering maneuvers. BigRoad [39] extracts the steering wheel angle from a smartphone's gyroscope. Researchers have also investigated the accuracy of using a mobile IMU sensor for detecting lane-change maneuvers [40]. One of TurnsMap's novelties is to detect interruptions (e.g., caused by crossing pedestrians and/or oncoming cars), a unique feature that can be used to derive the left-turns information at intersections.

10.2 Inferring the Road Information

Left-turns information is critical to driving safety, but the traffic databases maintained by local governments (e.g., data.gov [41], such as Open Data Portal [42]) lack this information. The collaborative mapping database, OpenStreetMap, has recently proposed a new feature to cover traffic light information [3] at intersections. However, this new feature is still in its infancy (i.e., under the community's review) and may take years to become practical. Even after receiving an approval, it will require a long time for the contributors (i.e., mappers) to collect the data due to the limited number of contributors.

To acquire traffic light/signal information at intersections, the corresponding industry has been devoted to the collection of left-turns information by taking images of intersections with heavily-instrumented mapping cars. For example, high-definition mapping services (Google Street View, here [43], TomTom [44]) use dedicated mapping cars to collect street images and derive intersection information automatically by using computer vision algorithms. However, this approach requires a significant investment of infrastructure, thus limiting its scalability, e.g., too expensive to use in under-developed or developing regions.

There have been proposals to detect intersection information with cameras in more efficient ways, but they suffer the natural limits of image data due to a variety of outdoor conditions [45]. Besides, placement of various traffic signals/lights make it hard for onboard perception systems to ensure that the intersection information is captured by using a limited number of cameras [46].

Aly *et al.* proposed Map++ [47], a crowdsourcing platform for mining traffic sign information from sensor data collected from mobile devices. However, Map++ detects only coarse-grained road information, e.g., existence/absence of traffic light. TurnsMap extracts and uses the spatio-temporal features from IMU sensor data to infer the specific protection type at left-turn hotspots — a safety-critical information unachievable with Map ++.

Recently, a traffic pattern analysis with mobile crowdsensing is gaining popularity due to its potential for benefiting smart cities. SmartRoad [48] uses mobile GPS data to determine whether or not traffic lights and/or stop signs are present. However, none of these existing studies explored left-turns information at intersections, albeit its importance to driving safety.

Unlike existing schemes, we focused on the left-turns information, a safety-critical yet unavailable data which can be efficiently collected by TurnsMap's crowdsensing. By crowdsensing mobile IMU sensor readings, TurnsMap can collect this information in an efficient, scalable manner.

11 CONCLUSION

TurnsMap is a novel way of utilizing crowdsensing mobile IMU data and deep learning for enhancing driving safety. It can infer the left-turn protection settings on road segments with high accuracy. To train and evaluate TurnsMap, we constructed a real-world dataset. TurnsMap is empowered by two key enablers: a novel data mining engine for extracting left-turn hotspots and a deep learning-based pipeline for learning the model for classifying left-turn enforcements. The left-turns information uncovered by TurnsMap is essential for driving safety and can benefit many vibrant apps in the automotive and transportation ecosystem.

REFERENCES

[1] Safety issues of left turns at intersections. https://www.washingtonpost.com/news/innovations/wp/2014/04/09/the-case-for-almost-never-turning-left-while-driving/.
[2] Why ups trucks never turn left. https://www.cnn.com/2017/02/16/world/ups-trucks-no-left-turns/index.html.
[3] Proposed features/turn signals. http://wiki.openstreetmap.org/wiki/Proposed_features/turn_signals#Traffic_signal_only_for_traffic_that_turns_left.
[4] Intersections and right of way. https://www.dmv.org/how-to-guides/intersections-and-right-of-way.php.
[5] Google's waze is helping drivers avoid left-hand turns. http://fortune.com/2016/06/18/google-waze-difficult-intersection/.
[6] Manual on uniform traffic control devices. https://mutcd.fhwa.dot.gov/pdfs/2009r1r2/mutcd2009r1r2edition.pdf.
[7] Waymo's big ambitions slowed by tech trouble. https://www.theinformation.com/articles/waymos-big-ambitions-slowed-by-tech-trouble.
[8] Accelerating the pace of learning. https://medium.com/waymo/accelerating-the-pace-of-learning-36f6bc2ee1d5.
[9] The first look inside zoox's mysterious robo-taxi. https://www.bloomberg.com/news/articles/2017-11-29/the-first-look-inside-zoox-s-mysterious-robo-taxi.
[10] Waymo tests its self-driving cars in my town. here are the odd things i've seen. https://www.forbes.com/sites/rrapier/2017/11/12/waymo-tests-its-self-driving-cars-in-my-town-here-are-the-odd-things-ive-seen/#57a5eb724e4a.
[11] Manato Hirabayashi, Adi Sujiwo, Abraham Monrroy, Shinpei Kato, and Masato Edahiro. Traffic light recognition using high-definition map features. *Robotics and Autonomous Systems*, 111:62–72, 2019.
[12] Coverage of google streetview. https://en.wikipedia.org/wiki/Coverage_of_Google_Street_View.
[13] Dongyao Chen, Kyong-Tak Cho, Sihui Han, Zhizhuo Jin, and Kang G. Shin. Invisible sensing of vehicle steering with smartphones. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, 2015.
[14] Turn left at a traffic light safely. http://drivinginstructorblog.com/turn-left-traffic-lights/.
[15] Cheng Bo, Xiang-Yang Li, Taeho Jung, Xufei Mao, Yue Tao, and Lan Yao. Smartloc: Push the limit of the inertial sensor based metropolitan localization using smartphone. In *Proceedings of the 19th Annual International Conference on Mobile Computing and Networking*, MobiCom'13, 2013.
[16] Kenneth Gade. The seven ways to find heading. *The Journal of Navigation*, 69(5):955–970, 2016.
[17] k nearest neighbor dynamic time warping. http://www.cs.unm.edu/~mueen/DTW.pdf.
[18] Haversine formula. https://en.wikipedia.org/wiki/Haversine_formula.
[19] Martin Ester, Hans peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.
[20] Amazon mechanical turk requester ui guide. https://docs.aws.amazon.com/AWSMechTurk/latest/RequesterUI/amt-ui.pdf.
[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the 22th IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'09, 2009.
[22] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

[23] Stefanie Nowak and Stefan Rüger. How reliable are annotations via crowdsourcing: a study about inter-annotator agreement for multi-label image annotation. In *Proceedings of the international conference on Multimedia information retrieval*, pages 557–566. ACM, 2010.
[24] Kevin A Hallgren. Computing inter-rater reliability for observational data: an overview and tutorial. *Tutorials in quantitative methods for psychology*, 8(1):23, 2012.
[25] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, May 2017.
[26] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*, WWW'17, 2017.
[27] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.
[28] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP'14.
[29] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.
[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
[31] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
[32] Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 190–198. Curran Associates, Inc., 2013.
[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 2nd International Conference on Learning Representations*, ICLR'14, 2014.
[34] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.
[35] How does google maps calculate your eta? https://www.forbes.com/sites/quora/2013/07/31/how-does-google-maps-calculate-your-eta/#e1dc5a3466e2.
[36] L. Kanth Nandam and T. Douglas Hess. Dynamic change of left turn phase sequence between time-of-day patterns-operational and safety impacts. *Institute of Transportation Engineers*, 2000.
[37] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Proceedings of the 28th Conference on Neural Information Processing Systems*, 2014.
[38] J. Yang, S. Sidhom, G. Chandrasekaran, T. Vu, H. Liu, N. Cecan, Y. Chen, M. Gruteser, and R. P. Martin. Sensing driver phone use with acoustic ranging through car speakers. *IEEE Transactions on Mobile Computing*, 11(9):1426–1440, Sept 2012.
[39] Luyang Liu, Hongyu Li, Jian Liu, Cagdas Karatas, Yan Wang, Marco Gruteser, Yingying Chen, and Richard P. Martin. Bigroad: Scaling road data acquisition for dependable self-driving. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, 2017.
[40] Hang Qiu, Jinzhu Chen, Shubham Jain, Yurong Jiang, Matt McCartney, Gorkem Kar, Fan Bai, Donald K Grimm, Marco Gruteser, and Ramesh Govindan. Towards robust vehicular context sensing. *IEEE Transactions on Vehicular Technology*, 67(3):1909, 2018.
[41] Traffic signal data on data.gov. https://catalog.data.gov/dataset?q=traffic+lights&sort=none.
[42] Open data portal for traffic light at intersection. http://gisdata-arlgis.opendata.arcgis.com/datasets/af497e2747104622ac74f4457b3fb73f_2.
[43] Here hd map. https://www.here.com/en.
[44] Tomtom hd map. https://www.tomtom.com/en_us/.
[45] Moises Diaz-Cabrera, Pietro Cerri, and Paolo Medici. Robust real-time traffic light detection and distance estimation using a single camera. *Expert Systems with Applications*, 42(8):3911–3923, 2015.
[46] Nathaniel Fairfield and Chris Urmson. Traffic light mapping and detection. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2011*, ICRA'11.
[47] Heba Aly, Anas Basalamah, and Moustafa Youssef. Map++: A crowd-sensing system for automatic map semantics identification. In *Proceedings of the 11th Annual IEEE International Conference on Sensing, Communication, and Networking*, SECON'14, 2014.
[48] Shaohan Hu, Lu Su, Hengchang Liu, Hongyan Wang, and Tarek F. Abdelzaher. Smartroad: Smartphone-based crowd sensing for traffic regulator detection and identification. *ACM Transactions on Sensor Network*, 11(4), July 2015.