Driver Identification based on GPS trajectories

Sasan Jafarnejad, German Castignani, and Thomas Engel,

***Abstract*—Connected vehicles and new paradigms in the mobility sector have recently pushed forward the need for accurately identifying who is behind the steering wheel on any driving situation. Driver Identification becomes part of a building block in the mobility area to enable new smart services for mobility like dynamic pricing for insurance, customization of driving features and pay-as-you-drive services. However, existing methods for driver identification depends on complex and high sample-rate vehicle data coming either from CAN-bus or from external devices. In this paper we propose to explore the potential for high accuracy driver identification with low-cost and low-sampled data, mainly GPS trajectories. Using a deep-learning based approach, we obtain overall error-rate of 1.9, 3.87, 5.71, 9.57, 13.5% for groups of 5, 10, 20, 50, 100 drivers. The results show outstanding accuracy and performance, enabling a fast and low-complex deployment.**

***Index Terms*—Driver identification, Deep Learning, Telematics, Driver profiling.**

I. Introduction

In the last years, driver Identification (ID) has gained popularity in the research community. This rise can be partially attributed to the ongoing transition of the mobility eco-system. People is now switching the traditional car-ownership model to a panoply of shared-mobility offers, including car-sharing, car-pooling, ride-hailing or short-term leasing, among others. In parallel, car manufacturers have been increasingly introducing car connectivity to even low-cost models, which has enabled substantial amount of car data available for research. This combination of facts enables new tailor-made mobility products with dynamic pricing, fraud-detection and automated claim management, for which driver ID becomes a crucial component. Also, on the big fleets area, in many countries, long haul truck drivers are by law required to take regular breaks and conform to maximum daily driving hours. However this is not being properly enforced, since traditional systems based on driver ID using smart-cards or another physical component is exposed to high level of fraud. Although research exists on this topic little is known whether stakeholder companies are actually deploying these solutions. Most recent research has dealt with the data from CAN-bus with high sampling rates, sometimes attained in unrealistic scenarios such as using external sensors. These solutions require specific hardware, powerful processors, and high data rate, which limits their deployability. There is then a concrete need for driver ID methods that are not hardware-dependant and that can guarantee high accuracy even with low-rate data, which minimizes communication and storage costs. In this context, we consider in this paper the scenario of having location-data only of vehicles, and we investigate whether this low-sample rate location data is sufficient for identifying drivers or not.

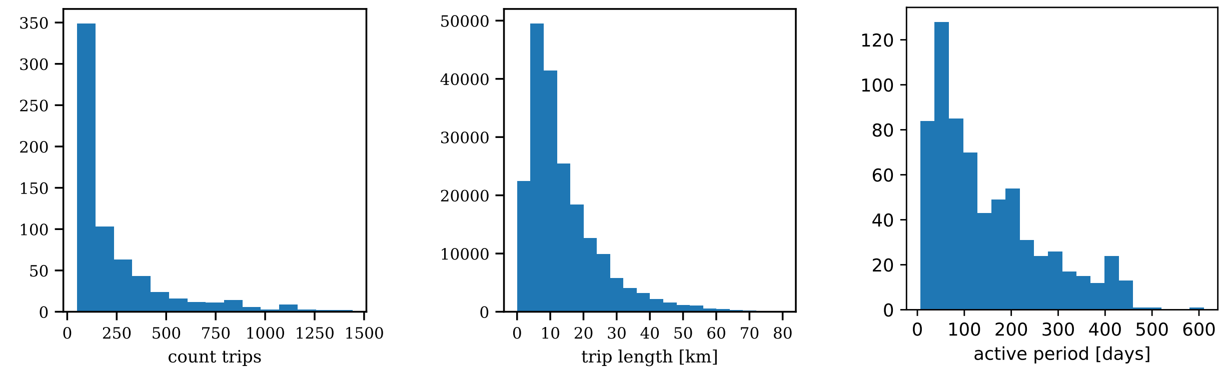
Driving data is inherently heterogeneous. Existing driver ID models in the literature only cover one aspect of driving data, i.e., the variability of vehicle dynamics (including speed, acceleration, etc.). In this paper, we provide with a method that relies only on GPS data and succeeds in accurately identifying the driver. Although we use an external web service to extract contextual metrics from the locations, our method stands as an end-to-end driver ID solution. In particular, we present an efficient way of encoding location coordinates and road-network links using embeddings to be used in deep neural network (DNN) models. We evaluate our proposed method using a large set of driving data covering thousands of different drivers being active for up to more than one year in the Greater Region of Luxembourg. Results presented outperform state-of-the-art driver ID methods.

The remainder of this paper is organized as follows. In Section II we present a complete taxonomy of the related work in the topic. Section III introduces and characterizes the data set used in this study. Section IV goes into the details of the proposed methodology. Section V describes the experimental setup and the baseline methods used for comparison. Then in Section VI we introduce the experimental setup and results. Finally in Section VII we conclude the paper and present the perspective of the future work.

II. Related work

There is a large body of research on driver ID, showing different goals and technical approaches. Traditionally biometric identifiers are the first to come to mind when we talk about popular techniques for identifying individuals. These are systems that employ personal traits, such as fingerprints, retina scans, voice or facial features for identification. Biometric methods are mature technologies, but they come with some limitations. As an example, a facial recognition system requires installation of a camera pointing to the driver, which will raise privacy concerns. Similarly, fingerprint scanners has been shown susceptible to copy. In [1] authors propose to install sensors inside the driver seat to ID drivers based on their posture and body structure, a part from the high costs, drivers find these sensors extremely intrusive. Even if we accept the risks and discomfort, wide deployment of such systems is in general costly and cumbersome to maintain. Therefore we think the focus should be on driver ID based on driver behavior.

Driver behavior refers to anything driver does during the act of driving which is measurable; Whether it is direct driver input, such as pedal operation or steering, or inferred data, like vehicle dynamics, such as acceleration patterns or speed profiles. Or even higher level features such as the ratio by which a driver exceeds the speed limit ($\frac{speed}{speed\_limit}$).



(a) Histogram of number of trips per driver. (b) Histogram of trip lengths. (c) Histogram of driver activity period.

Figure 1. Data set plots

Table I
FEATURES

	Metric	Continuous	Categorical	Sequential	Description
Temporal	Day of week	-	1 (7)	-	
	Hour	-	1 (24)	-	
	Minute	-	1 (4)	-	binned into 4 quarters
	Temporal exposure	3	-	-	Slight, Serious, Fatal
	Peak-hour	-	1 (2)	-	If trip took place at peak hour.
	Total time (s)	1	-	-	Duration of the trip
	daytimeproportion	1	-	-	Daytime Proportion of trip
Behavioral	RPA	1	-	-	Relative positive acceleration
	Steering	1	-	-	High, Low
	Brake & Acceleration	4	-	-	High, low
	Acc./decelleration g-force	6	-	-	Max, min and average of acceleration or decelleration g-force
	Count illegal actions	3	-	-	turn, u-turn, direction
	Average overspeed	3	-	-	-, curves, traffic
Spatial	Lat./Lon. of trip start/end	2×2	2×2 ($\sim$ 4k)	-	Latitude and longitude of start and end of trip
	Link IDs	-	-	varies	Sequence of links traversed during a trip
	Bearing	1	-	-	
	Distance	2	-	-	Haversine & Manhattan distance between start and end of trip
	Distance (km)	4	-	-	Distance travelled in motorway, rural, urban areas and their total
	Count of road features	4	-	-	Bridges, Tunnels, Urban Areas, signals, POIs
	Proportion (distance)	3	-	-	Proportion of distance traveled through motorway, urban and rural areas
	Proportion	5	-	-	FC1, FC2, FC3, FC4, FC5
	Proportion	6	-	-	express-highway, motorway, highway, urban, interurban, unclassified
	Border crossing	1	-	-	Number of times driver crosses the border
	Number of traffic signs	6	-	-	Priority, yield, stop, lane-merge, pedestrian, overtaking
	Count of road object	3	-	-	Bridges, roundabouts, tunnels
	Count of location data points	2	-	-	Total count, valid count
	Speed in traffic	1	-	-	Driving speed in relation to traffic
	Time in traffic	1	-	-	Time spent in congested traffic

In the literature, methods are mainly based on techniques relying in two categories of data, 1) *Car data*, often with high sample-rate and large number of features. This kind of data is usually collected from CAN-bus, using OBD-dongles, black-boxes or external sensors. 2) *Location data* provided by global navigation satellite systems (GNSSs). Usually lower sample-rate and noisy data which can be provided by smartphones, car-navigation system or external receivers. Automotive manufacturers have access to both these data categories, some even provide Application programming interfaces (APIs) to third-parties to access such data. Although Original equipment manufacturers (OEMs) can use the high sample-rate data internally, third parties can only access low sample-rate data. This is mainly due to data transmission and storage costs. In the near future, third party providers may get access to *car data* for applications that are deployed on car-infotainment systems, perhaps through Android Auto or Apple CarPlay. As an example, Daimler has recently launched a platform going into this direction[1].

A large portion of the driver ID literature is focused on *car data*. In [2] authors explore use of physical driving models, and Gaussian Mixture Model (GMM) for driver ID. They apply these methods to behavioral signals collected from a "car following" task. Signals such as gas and brake pedal position, vehicle speed and the following distance (FD) (distance to the leading vehicle). They observe that GMM performs better than the physical models. [3] propose use of GMM to model distribution of cepstral features extracted from pressure sensors retrofitted under the pedals. Cepstral features, especially Mel-frequency cepstral coefficient (MFCC)s are widely used in speech/speaker recognition field. [4] uses similar features but instead use extreme learning machine (ELM) model.
Although these works achieve fair results, they are not practical, mainly because production vehicles are not equipped with pressure sensors under the pedals. To get closer to a practical solution [5] use signals from standard vehicle sensors, such as

[1]https://developer.mercedes-benz.com/apis

Percentage Gas Pedal (GP) and Steering Wheel Angle (SW). They show that cepstral coefficients of SW are also important discriminators and can be used for driver ID, this effect is even more evident when the driver ID is being performed on a larger group of subjects. In [6], the authors obtain high accuracy on driver ID by taking advantage of three factors: 1) A wide set of signals with high sampling rate (60Hz), 2) a comprehensive set of spectral and statistical features, and 3) a set of Machine learning (ML) models. However they also show that brake pedal is the best discriminator signal and solely using it one can accurately identify unique drivers in a population of up to 15 drivers.

More recent works [5], [6] often use ML methods such as Support Vector Machine (SVM), Random Forest (RF) or Boosting trees. Such methods are often trained in a one-versus-all fashion. This means that for each class (driver) a model that discriminates that driver from the rest is fitted. The approach has a big disadvantages and presents poor scalability, because every-time a driver is added to the system all the models are needed to be re-trained. In [7], the authors show that GMM fitted on spectral features of both gas pedal and steering wheel can perform well, with as little as five minutes of training data per driver. It is also scalable, since the addition of new drivers only requires fitting a GMM to the new driver's driving data. There are other works that look at more particular situations, for example [8], which creates a profile for each driver based on the sequence of the actions they take before they begin to drive. They test this method on two groups of people, and achieve good accuracy. While manufacturers have access to sensors needed to implement such approach, its solution is highly vehicle-specific and difficult to deploy by third parties. Moreover, it is not difficult to behave similarly than someone so as to imitate the sequence of actions before driving. In [9] the authors focus on the possibility of driver ID using only single turn maneuvers. Although they do not achieve high accuracy they show that even in one single turn event there is enough information to discriminate between the drivers (even with limited confidence).

There are a few works that focus on *location data*. [10] uses location and accelerometer data to identify family members. They use location data to construct clusters of locations, and used the origin and destination clusters as a feature, among other features such as excessive manoeuvres extracted from accelerometer, weekday, departure time and trip duration. Then they used conventional ML algorithms to identify the driver, and achieved accuracy of about 70%. In [11], although the authors' focus is behavior change from riskiness aspects, they informally perform driver ID. They use a large dataset of 3000 drivers provided by an insurance company, that contains location and accelerometer data. They train a classifier to classify drivers as themselves, a risky driver or a safer driver. In the training process they sample 40 safe and risky drivers (to represent each class) and use their trip as proxy for risky or safe driving. From location data, they only use over-speeding events and the rest of the features are obtained by binning (thresholding) the accelerometer data. Chowdhury et al. [12] focus only on location data collected from smartphone, they estimate many secondary signals from speed and heading (e.g. jerk, lateral acceleration, and longitudinal acceleration etc.). They compute 137 statistical features per trip and use a RF classifier to identify the drivers. For groups of 4 to 5 they obtain accuracy of 82.3% on average.

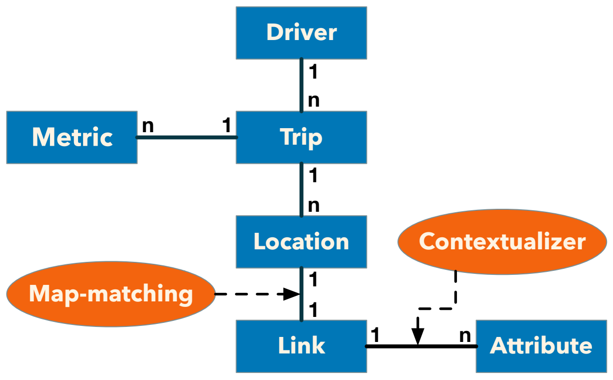


Figure 2. Data set model

III. Dataset

In this work we use a dataset provided by Motion-S SA. During a data collection campaign, more than ten thousand participants downloaded a gamified smartphone application to provide driving behavior tips, in exchange of receiving feedback on their driving behavior (riskiness factors) and a chance to win a prize. The participants have been collecting driving data in the background by detecting car start and stop using Bluetooth connection events with their car infotainment system. We refer to such a recording session as a *trip*. Conceptually a trip could be a commute to work-place or back, visiting the doctor's office, or going to the mall for shopping. Additionally there are times that the participant either disables, or chooses not to record certain trips, for any possible reason. During a trip, the app records timestamped locations updates every second. At the end of each trip or whenever the Internet connection is available the data gets uploaded to a remote server. A location update, is comprised of latitude, longitude, altitude, speed, bearing and estimated accuracy. The dataset used for this study is free of all personally identifiable and quasi-identifiers, we only use a pseudo-identifier to keep track of trips that belong to a *lambda* individual.

Each trip is a sequence of timestamped location data. After collection, location data are contextualized using Here.com[2] maps layer for map-matching and data augmentation. Map-matching is a process that associates a latitude, longitude coordinate to a segment of road network, which we call a *link* and is represented by a unique integer identifier. The data augmentation process is able to provide a wide spectrum of attributes that characterize a particular link, e.g., slope, speed limit, administrative region, country, road roughness, points of interest, traffic signals or whether it is a tunnel or bridge. After obtaining link attributes we compute additional features (*metrics*), e.g., speeding ratio, acceleration and steering. An overview of these features are listed in Table I.

[2]Here.com https://api.here.com

The dataset used in this paper contains heterogeneous variables describing many aspects of each trip. For that reason, in order to improve their understanding, we propose to categorize the features from two perspectives: *semantic* and *technical*. The semantic perspective is related to the nature of the metric while the technical perspective is related to the expected type of output for that particular feature.

A. Semantic categorization of features

The act of driving implies that an individual is driving in a certain place, at a certain point of time and behaving in a certain manner.Then, semantically categorize the features as follows:

Temporal: Features that are derived from the date and time of the collected locations of trip.

Spatial: Features that are merely derived from specific location coordinate and the route taken by the driver.

Behavioral: All the other features, these are features that are to some extents a function of driver, such as steering and acceleration patterns, etc..

Note that this categorization is somewhat arbitrary. One could argue that the hour trips are performed should be considered behavioral because it is driver's choice. Since our focus is driver ID and its applications, there are use-cases that cannot depend on every feature categories. For example in case of long haul truck drivers, if the truck is driven by an illegitimate driver, since the destination and likely the route does not change, we cannot rely on *spatial* or even *temporal* features, therefore *behavioral features* are our only chance to be able to discriminate the two drivers. Semantic categorization of the features is indicated on the left side of the Table I.

B. Technical categorization of features

The technical categorization refers to the expected output type, because every data type requires an appropriate model. There are three types of features that we are concerned with: 1) continuous 2) categorical 3) sequential.

Continuous features: As presented earlier, for every trip, a number of metrics is computed. We can represent real valued feature as $\boldsymbol{X} \in \mathbb{R}^{M \times N}$, where M is number of trips and N is number of features. The continuous features are scaled as below:

$$\mathbf{X}^n = \frac{\mathbf{X}^n - mean(\mathbf{X}^n)}{std(\mathbf{X}^n)} \tag{1}$$

$mean(\boldsymbol{X}^n)$ is the mean of the n^{th} dimension of the $\mathbf{X}$ across all trips of all drivers.
$std(\boldsymbol{X}^n)$ is the standard deviation of the n^{th} dimension of the $\mathbf{X}$ across all trips of all drivers. Number of continuous features per metric is indicated in *continuous* column of Table I.

Categorical: these features, even though often represented as an integer type, their numerical value carries little or no meaning. For example we treat *hour* as a categorical feature, because the hours 23 and 0 even though they are farthest apart among hours of day, they both represent midnight hours. Other

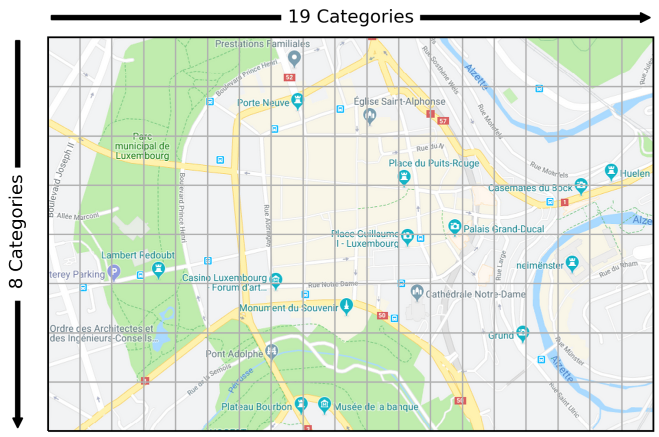


Figure 3. Location cross feature

examples are *day of week* consisting of 7 categories or *peak hour* having only two categories.
In particular, we propose to represent *location coordinates* of origin and destination of a trip as a categorical feature. A location coordinate consists of continuous values for longitude and latitude, $\mathbf{X} = (lon, lat)$, where $X \in \mathbb{R}^2$. However, we can also treat them as a cross feature, by binning the latitude and longitude in a grid on the map (see Figure 3). In this work we bin the coordinates to 0.003°. To do so, we assign a unique number to each cell in the grid $n \in \mathbb{N}$. Trip origin an destination coordinates are then mapped to the correspondent cell identifier $X \mapsto \mathbb{N}$, therefore every coordinate is represented by an integer. . The number of categorical features are indicated in *categorical* column of Table I, the number of categories are indicated between parentheses.

Sequential: The only sequential feature we consider is the sequence of *links* taken by the driver. As described earlier in this Section each location is mapped to a *link*. In other words this is a sequence of categorical features. We drop the repeated consecutive *links* but still the length of the sequence varies and is trip dependent.

Table II
DATASET STATISTICS

metric	pre-processing	
	before	after
Number of drivers	1385	678
Average trips per driver	261.51	297.06
Total trips	362198	201410

C. Pre-processing

To ensure quality, we filter the dataset according to the following heuristics. Short trips are often faulty and do not contain enough information to be useful therefore the trips shorter that five minutes are removed. Moreover since we calculate the trip-length as the difference between trip-start and trip-end timestamps. There are times that although the length is over five minutes, the trace is faulty and contains few location data points, to filter such instances we also drop

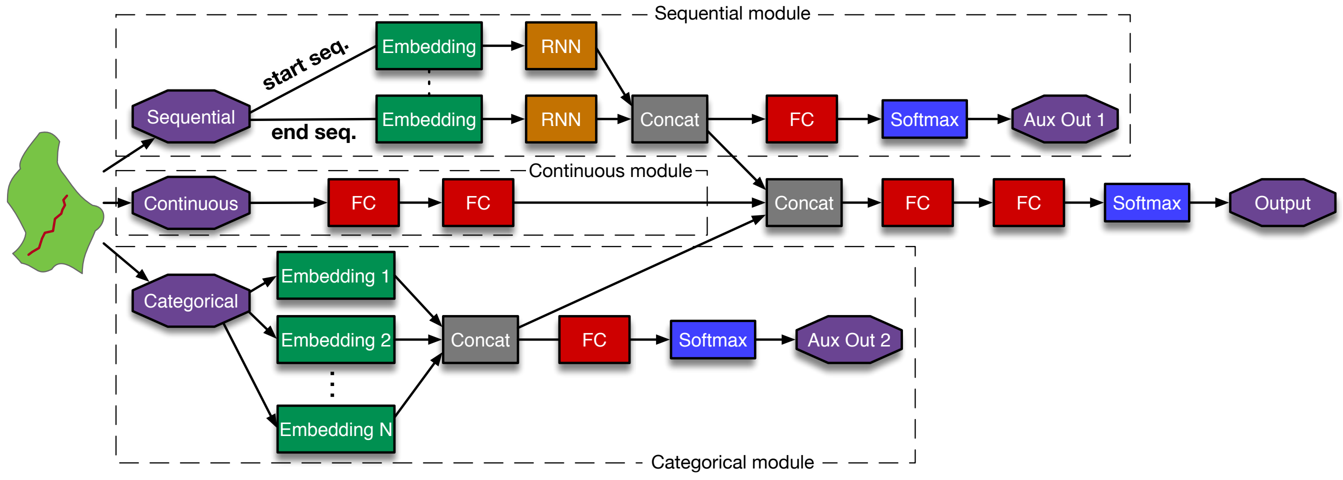


Figure 4. Architecture diagram

trips with 100 or fewer data points. Trips having constant speed is another pattern observed among faulty trips, so we decide to remove them from the dataset. We filter out cross-border or trips taking place outside Luxembourg territory. Lastly, we exclude drivers that have fewer than 50 trips in their records. Table II shows a summary of dataset statistics before and after pre-processing. There are initially 1385 drivers in our dataset with each from 13 to 5066 trips they total at 362198 trips, after pre-processing the number of distinct drivers reduces to 678 and a total of 201410 trips.

IV. Methodology

The goal of driver ID is to associate a driving trip or its representation $\mathbf{x}$ to its actual driver c from a known set of drivers (C). We propose in this paper to model the driver ID task as a supervised classification problem. Therefore we assume information on all possible targets, which is also known as a closed-world scenario, assuming that all examples are from classes in our training data set. We choose a Deep learning (DL) approach to tackle the problem. The motivation behind this decision is the ability of simultaneous handling of multiple data types. Moreover due to its modular design, more inputs of any kind can be easily added to the model.

The main challenge of this scenario is to handle sparse and high dimensional categorical data. We have two instances of such data, *links* and location cross features. For instance, only in Luxembourg (where main part of our data is coming from) there are about 60 thousand *links*. Categorical data is generally encoded using one-hot scheme, which for a feature with n values creates a n-dimensional vector, when having features with categories in orders of thousands, one-hot encoding is not effective. To tackle the issue of high dimensionality we use a similar technique as *word embedding* [13]. Word embedding is a technique used in natural language processing (NLP) that maps vocabulary to vectors of real numbers. We apply this technique to every categorical feature. As an example, we consider *links* to describe the method more in detail.

We can represent a trip as a sequence of links traversed, $U = \{\mathbf{u}_1, \dots, \mathbf{u}_L\}$, where $\mathbf{u}_l$ is a one-hot vector in space Δ^D representing *link*, D is total number of links in our covered region and L is the number of links in the trip. Since D is large, learning in Δ^D space is computationally expensive. Therefore we map links into an embedding space. We call this link embedding, which is semantically similar to word embedding, $\Delta^D \mapsto \mathbb{R}^P$, where P is the dimension of embedding space that can be between 32 to 128 depending on the hyper-parameters. After applying the map $f_{link\_emb} : U \mapsto V$, becomes $V = \{\boldsymbol{\nu}_1, \dots, \boldsymbol{\nu}_L\}$, where $\boldsymbol{\nu}_l \in \mathbb{R}^P$. In practice this mapping is either pre-trained (like word vectors used in NLP) or learned jointly with the model, we chose the latter because due to relatively small amount of training data, learning an embedding jointly with model allows us to learn an embedding specialized to discriminate between drivers. On the other hand we hypothesize learning a pre-trained embedding on a large corpus (millions of trips) that is then fine-tuned for target drivers should improve the generalization and even reduce the amount of data needed for training, due to lack of sufficient data we leave this for future work [14].

We can use a similar mapping for any other categorical data, with the difference that P should be chosen proportional to the number of categories of the feature.

A. Neural network architecture

We propose a DNN architecture that consists of three modules, each with a different feature vector, *continuous*, *categorical*, *sequential*. Figure 4 illustrates the proposed architecture.

Sequential module: The upper part of Figure 4 models the *link* sequence, which is the only sequential feature we consider in this work. Since the origin and destination of a trip has the highest importance, and also to avoid having to handle long sequences of variable length we decided to model the beginning and end of a link sequence separately. We introduce a hyper-parameter ρ that determines the length of the origin and destination sub-sequences. Denoting L as length of link sequence, if $L < 2\rho$ these sub-sequences may overlap and if $L < \rho$ the rest of the sub-sequence is filled with zeros. Each sub-sequence is passed through an embedding layer,

Table III
SUMMARY OF MODEL PARAMETERS FOR AN EXAMPLE EXPERIMENT WITH 50 DRIVERS. AUXILIARY OUTPUTS ARE OMITTED.

Module	Layer	Kernel, Bias	# params
Sequential	recurrent_gru_orig	(3,128),(128,)	98688
Sequential	recurrent_gru_dest	(3,128),(128,)	98688
Sequential	embed_link	(29166, 128)	3733248
Continuous	cont_dense_1	(72, 256),(256,)	18688
Continuous	cont_dense_2	(256, 128),(128,)	32896
Categorical	embed_location	(8201, 96)	787296
Categorical	embed_minute	(4, 6)	24
Categorical	embed_hour	(24, 12)	288
Categorical	embed_dayofweek	(7, 8)	56
out_branch	out_dense_1	(602, 512),(512,)	308736
out_branch	out_dense_2	(512, 128),(128,)	65664
out_branch	out_dense_3	(128, 50),(50,)	6450

these layers share their weights. The embedding vectors are initialized randomly and then trained to minimize the loss function. We feed the embeddings to the recurrent neural network (RNN). We experiment with long short-term memory (LSTM) and gated recurrent unit (GRU) cells [15], [16]. Then the output from two RNN units is concatenated and merged back to the main network. We also branch out an auxiliary output (*aux_out1*), which consists of a fully-connected layer with Softmax activation function.

Categorical module: For each categorical variable we create an embedding, the dimensions of this embedding is proportional to the number of categories in the variable. It is calculated using the Equation 2 for each categorical feature:

$$embed\_dim = min((\lceil \log_2(vocab\_size) \rceil + 1) * \kappa, 50) \quad (2)$$

where $vocab\_size$ is the number of categories and κ is a hyper-parameter. The resulting embeddings are then concatenated and passed through a fully-connected layer with rectified linear unit (ReLU) activations and merged back to the main branch. Similar to *sequential branch* to facilitate learning, we also branch out an auxiliary output using a fully-connected layer and a softmax.

Continuous module: The continuous branch is the main branch. All the continuous features are fed into two fully-connected layers and then the other two branches are merged (concatenated) in it. Then they are followed with two other fully-connected layers, a softmax activation. We concatenate outputs from these modules and feed them into two dense layers with ReLU activation functions, followed by a drop-out layer. Softmax function is applied to the output of the last drop-out layer.

Since we jointly train all modules to facilitate learning process we branch out auxiliary outputs from sequential and categorical modules, indicated as *aux_out_1* and *2* in Figure 4. Our goal is to minimize the error:

$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} J\left(y_n, f(X_n)\right) \quad (3)$$

where y is the target (correct driver) and $\mathbf{x}$ is the input feature vector and J measures the loss, when having one output.

$$J = -\frac{1}{N} \sum_{i=1}^{N} \log p_{model}[y_i \in C_{y_i}] \quad (4)$$

We adjust J to accommodate the auxiliary outputs.

$$J = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k \in outs} \omega_k (\log p_k[y_i \in C_{y_i}]) \quad (5)$$

Where ω is the output weight and $outs$ consists of all outputs $\{main\_out, aux\_out1, aux\_out2\}$, we assign weight 1 to the main output and 0.2 to the auxiliary outputs.

V. EXPERIMENTAL SETUP

We evaluate the performance of the proposed model with real-life applications in mind. We consider scenarios with $|C| \in \{5, 10, 20, 50, 100\}$ drivers. Our data set contains 678 drivers, however each driver has different number of trips and their trip composition, driving style varies. To cover a representative sample of drivers we repeat experiments for every $c \in C$, 30 times, each time with another random (no replacement) sample of drivers. This will also account for the case that drivers in a certain sample may be easy or difficult to discriminate.

Moreover in order to tackle the class imbalance in our data set, for every sampled driver, regardless of their number of trips, we randomly sample 200 trips (with replacement). This will lead to up or down sampling depending on how many trips a driver has in the data set. We split the 200 trips into training and validation sets in a 90%-10% ratio. Since the examples sampled for each experiment are balanced we use the accuracy as the evaluation metric:

$$acc = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i) \quad (6)$$

where $\mathbb{1}$ is the indicator function, $\hat{y}_i$ and y_i are respectively predictions and true driver for the i^{th} trip. Since for most experiments accuracy is close to 1 to improve readability we use the error-rate $(1 - acc)$. For each experiment we report the average and standard deviation of the error-rate across repetitions.

The neural network model is implemented using the functional API of Keras [17] with Tensorflow backend [18]. For training we used, batch size of 64 per GPU and optimized the model with Adam optimizer [19] for 50 epochs. The model often reaches the lowest error-rate in 20 epochs. We stored the model parameters at each epoch, and later picked the parameters with the best score on the validation set.

A. Hyper-parameter search

DL architectures have large number of hyper-parameters and the search space to find the optimal architecture is computationally expensive to cover. To find hyper-parameters that perform well we break the network down to its modules. For each module we perform a separate hyper-parameter search then we use the optimal parameters for the full module. Clearly with this approach we will not obtain the optimal parameters, but it should be give us a close approximate. The full set of

Table IV
HYPER PARAMETERS OF THE MODEL

Module	hyper-parameter	candidate values
Sequential	link embeddings	{32, 48, 64, 96, **128**}
	recurrent unit	{**GRU**, LSTM}
	recurrent hidden units	{64, 96, **128**}
	recurrent dropout rate	{0.0, 0.3, **0.5**}
	ρ (sequence length)	{10, **15**, 20, 30}
	fully connected	{**[256, 128]**, [128, 64], [256], [128]}
Continuous	fully connected	{**[512, 256]**, [256, 128], [128, 128], [128, 64]}
	dropout rate	{0.1, 0.3, **0.5**}
Categorical	location embedding dims.	{32, 48, 64, **96**, 128}
	κ (embedding coeff.)	{1, **2**, 3}

hyper-parameters we experimented with is given in Table V-A and the best performing parameters are indicated in boldface. We performed hyper-parameter search only over one set of randomly selected 50 drivers.

B. Baseline algorithms

Since there are no other works that we could directly compare our approach with, we introduce two baseline algorithms. The first model is LightGBM and a hidden Markov model (HMM) based model, to have a reference for the sequential module of our architecture. We also consider a meta model consisting of a combination of the two baselines as a comparable contender for our proposed DNN architecture.

1) LightGBM: LightGBM is a variation of gradient boosting decision tree (GBDT) and the state-of-the-art algorithm for structured data [20]. Although LightGBM can process categorical data it cannot process sequential data, therefore we do not use the sequence of *links* with this classifier. We use the following hyper-parameters:

- *learning_rate*=0.1
- *max_depth*=30
- *min_data_in_leaf*=20
- *num_leaves*=50
- *num˙round*=500

the hyper-paremeters were selected after running an extensive hyper-parameter search.

2) Hidden Markov model: To set a baseline for the recurrent module of our model, we use a sequence model based on HMM. We adapt the approach used in [21] to our driver ID application. In that work the assumption is that the trips are clustered in advance according to trip destination. Their goal was to predict the destination by identifying the cluster (hidden state). We consider the hidden state to be the driver, therefore our equivalent of the cluster is the driver. The model makes use of *link*-driver co-occurrence matrix F. $F \in \mathbb{N}^{m\times n}$ where $F_{i,j}$ is count of times driver $j \in C$ in traversed link $i \in L$.
Having constructed F, we can compute $p(l|C=c)$ and $p(C=c|l)$ for any given driver (c) and *link* (l) by employing the Equations 7 and 8.

$$p(l|C=c) = \frac{\#trips\ traversing\ l\ by\ driver\ c}{\#\ trips\ by\ driver\ c} = \frac{F_{l,c}}{\sum_{i=1}^{m} F_{i,c}} \tag{7}$$

$$p(C=c|l) = \frac{\#trips\ traversing\ l\ by\ driver\ c}{\#\ trips\ passing\ via\ l} = \frac{F_{l,c}}{\sum_{j=1}^{n} F_{l,j}} \tag{8}$$

Algorithm 1 shows how we perform the prediction. P_i represents likelihood of driver i being the actual driver. *Normalize* is a routine that normalizes every P_i to be a valid probability. T is total number of *links* in trip. It is possible that driver crosses a link that she never has or there could be a mistake in map-matching, to avoid getting zero probability for correct driver, the constant r is introduced alleviate this problem.

Algorithm 1 Driver prediction with drivers as hidden states

```
1: $n \leftarrow |C|$
2: $k \leftarrow 1$
3: $r \leftarrow 0.1$
4: for $i \in C$ do
5:     $P_i \leftarrow p(C=i|l_1)$
6: end for
7: while $k \leq T$ do
8:     for $i \in C$ do
9:         $P_i \leftarrow P_i \cdot p(l_{k+1}|C=i)$
10:    end for
11:    $normalize(P_i)_{i\in C}$
12:    for $i \in C$ do
13:        $P_i \leftarrow \frac{r}{n} + (1-r)P_i$
14:    end for
15:    $k \leftarrow k+1$
16: end while
17: return $\arg\max_{i\in C} P_i$
```

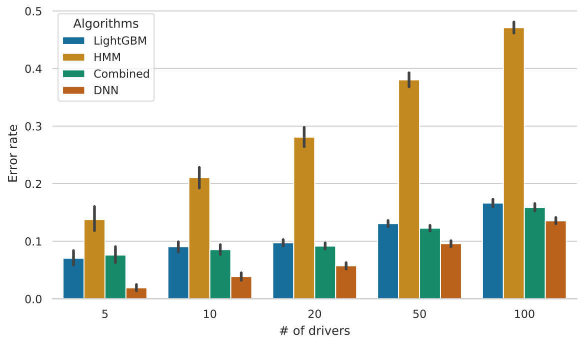
3) Combined model: We also evaluate the performance of a combined model of RF and the HMM model. We take a linear combination of their output probabilities and use the resulting probabilities for prediction. We introduce a hyper-parameter α to be able to adjust the contribution of each model. Equation 9 demonstrates this:

$$\hat{y} = \arg\max_{c\in C}\{\alpha h_{LightGBM}(\boldsymbol{X}) + (1-\alpha)h_{HMM}(\boldsymbol{X})\} \tag{9}$$

where $h_{LightGBM}$ and h_{HMM} are the LightGBM and HMM models and $\boldsymbol{X}$ is trip feature vector. Through experimentation we find the optimal value of α to be 0.7, and this is a setup that we used in our evaluations.

VI. Experimental Results

In this section presents the experiments performed and their results. We compare the proposed DNN architecture with baseline methods. Then we evaluate the effect of various combinations of semantic feature categories to simulate how



(a)

# drivers	Baselines			DNN
	LightGBM	HMM	Combined	
5	0.0704 (0.035)	0.138 (0.059)	0.0759 (0.04)	0.019 (0.016)
10	0.0905 (0.027)	0.211 (0.051)	0.0854 (0.024)	0.0387 (0.018)
20	0.0972 (0.015)	0.281 (0.048)	0.0915 (0.015)	0.0571 (0.016)
50	0.13 (0.016)	0.38 (0.035)	0.123 (0.015)	0.0957 (0.015)
100	0.166 (0.02)	0.471 (0.028)	0.159 (0.019)	0.135 (0.016)

(b)

Figure 5. Algorithm error-rate in graphical (a) and tabular form (b)

the model would perform for various use-cases. We investigate the effectiveness of using the cross features. And lastly we investigate the effect of amount of training data on prediction performance.

A. *Comparison with baselines*

In this experiment every algorithms is fed the same data sets and the only difference is in the features which is an artifact of inherent differences between the models. Particularly LightGBM model lacks the *link sequence* feature while HMM model only has the *link sequence* as its input feature. Figure 5a shows the error rates of the proposed model in compare to baselines. LightGBM and HMM models perform similarly for groups of 5 drivers, but as the number of drivers increase the HMM model significantly deteriorates in performance. The combined model as expected has lower error-rate for every scenario than LightGBM or HMM models. We can also see that the DNN model outperforms the baseline models in every scenario.

B. *Semantic feature categories*

In this experiment we evaluate the importance the three semantic feature categories that we introduced in Subsection III-A. We compared all possible combinations of feature categories. To depict the model performance under various circumstances and applications. The corresponding results are presented in Figure 7a both in graphical and numerical form. We can clearly see that *temporal* features alone, perform the worst with about 40% error rate for 5 drivers that increases to 79.3% for 100 drivers, this shows that merely knowing the time-of-day and day-of-week are not enough to discriminate between drivers. Second best feature category is the *behavioral*, it performs reasonably well for small number of drivers (19% error-rate for 5 drivers) but as the number of drivers increase the error rate also increases. Since behavioral features are not directly route dependant, this shows promise for applications such as such as theft detection or driver verification, because error-rate would even decrease further when trained used for smaller set of drivers. *Spatial* features by far are the best category, they provide error-rate of orders of magnitude lower than other feature categories, this is justified because each person only travels to a limited number of point of interests (POIs) and it is not difficult to learn them. It is important to note that combinations of feature categories always results in improved accuracy, this shows that all the feature categories contribute to the predictions, but their contribution is not equal. Since the contribution of temporal feature category is small we could remove them from the system without significant increase in error-rate, with nothing to lose in terms of error-rate we gain in being more privacy friendly.

Table V
EFFECTIVENESS OF LOCATION CROSS FEATURES

# drivers	Experiment error-rate - mean (std.)		
	Full model	Excl. cross features	Excl. coordinates
5	0.019 (0.016)	0.0211 (0.02)	0.0218 (0.023)
10	0.0387 (0.018)	0.0456 (0.02)	0.0422 (0.02)
20	0.0571 (0.016)	0.0663 (0.017)	0.0594 (0.015)
50	0.0957 (0.015)	0.115 (0.017)	0.0998 (0.017)
100	0.135 (0.016)	0.149 (0.017)	0.139 (0.015)

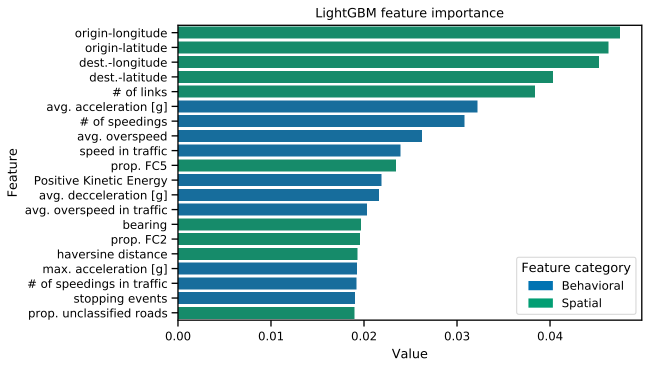
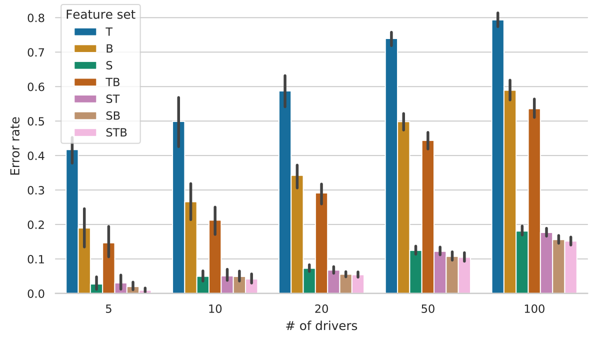


Figure 6. Top 20 most important features obtained from LightGBM classifier.

Figure 6 shows the top 20 features obtained from LightGBM model corresponding to a randomly selected experiment. Although the order may not completely correspond to the contribution of each feature to our DNN model it will help to get an idea of the most important features. We can see latitude and longitude of origin and destination constitute the top 4 features. Then there are a number of behavioral features such as a number of *speeding* related features and features derived from acceleration patterns. The other group of features are those reflecting the composition of the route taken by the drivers, the distance and the number of *links*.

C. *Effectiveness of cross features*

We introduced *location cross features* in Section III-B however knowing that in this work accurate origin and destination coordinates are available, this approach may seem



(a) Feature Category Error rate plot

Feature set / # drivers	T	B	S	TB	ST	SB	STB
5	0.417	0.190	0.027	0.147	0.030	0.020	0.009
10	0.499	0.266	0.050	0.213	0.051	0.049	0.042
20	0.587	0.342	0.073	0.291	0.068	0.056	0.054
50	0.739	0.498	0.125	0.444	0.122	0.107	0.105
100	0.793	0.589	0.181	0.536	0.177	0.156	0.152

(error rate)

(b) Feature Category Error rate table

Figure 7. Feature Category Error rate

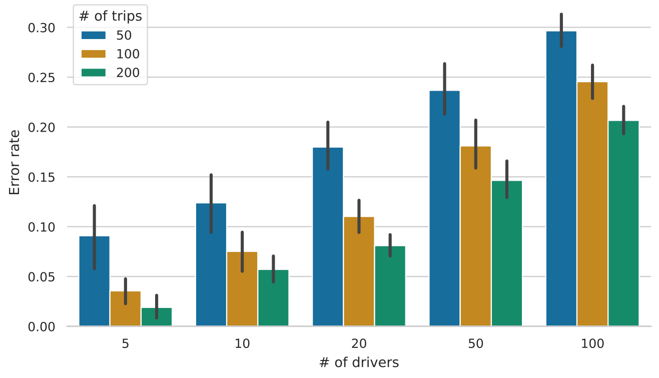


Figure 8. Trip count error-rate

unnecessary. Consider the case accurate coordinates are not available. This could be either due to lack of accurate data or suppose due to privacy considerations a portion of beginning and end of a trip is trimmed before upload to the server, or perhaps differential privacy mechanisms are applied to the data. Therefore in cases that we cannot rely on accuracy of location coordinates binning may provide a better performance because we no longer assume that coordinates are accurate measurements. To investigate effectiveness of these features we compare the model performance under three circumstances: 1) full model including both origin, destination coordinates and cross features 2) full model excluding cross features 3) full model excluding coordinates. The second experiment will show whether addition of cross features in presence accurate coordinates is unnecessary, and the third case will examine whether the model can perform well in absence of accurate coordinates.

Table V shows the results of the above-mentioned experiments. We can see that for any number of drivers removing the cross features will increase the error rate. This confirms that *cross features* contribute toward better driver ID (about 10% reduction of error-rate in average for 100 drivers). In absence of coordinates (3rd case) the model performs slightly worse than the full model but better than the case without the *cross features*, we believe this is due to superior generalization capabilities of the *embeddings*.

D. Amount of training data

We investigate how the amount of training data affects the model error-rate. We evaluate the model with $\mathcal{L} \in \{50, 100, 200\}$ trips. In these experiments each time we sample drivers, for each driver we randomly sample $\mathcal{L}$ trips, and use this data set for training and validation. The error-rate and its standard deviation decreases inversely with $\mathcal{L}$, however this effect dampens as $\mathcal{L}$ increases. This has two explanations, 1) There is only so much to learn, over that there are information that the current model cannot learn. 2) We do not have equal number of trips from each driver, it is possible the fact that we sample the trips with replacement, negatively affects the performance for the larger values of $\mathcal{L}$.

VII. Conclusion

In this work we presented an end-to-end system for driver ID. The input to the system is location coordinates corresponding to a journey, sampled every second. This data is easy to obtain, either through a Smartphone or from the car itself. Recently more and more car manufacturers are providing always on connection for their cars, that means they can send data to their servers. More recently many companies such as Mercedes Benz or BMW have provided API to access to such data. The input is then augmented and contextualized using an external service. The augmented features have various data types, they include continuous and categorical values as well as variable length sequential data. Our proposed system can accept all the above mentioned features and ID the drivers.

We compare our system with three strong baselines and outperform them. We argue that based on application use-case it may not be desirable to use for example spatial features, therefore we evaluate our method using different feature sets; even without spatial features that are the best performing category we obtain desirable performance only using behavioural features (19% for 5 drivers). As an example ride-sharing providers such as Uber or Lyft, can prevent fraud and ensure passenger safety by using this system only with behavioral features to verify driver's identify and trigger more reliable identification procedures. We also showed that our method improves as more data is collected, which is an ideal property for a system that is constantly collecting more data as it is being used. We obtain overall error-rate of 1.9, 3.87, 5.71, 9.57, 13.5% for groups of 5, 10, 20, 50, 100 drivers which shows a great promise and enables other applications to be built on top of this system.

There are further steps that can be taken after this work. We mentioned privacy issues and how cross features could help hiding the accurate information on origin and destination of trips, a follow up would be trimming for example the first and last mile of each trip and observe its impact on the system. To decrease the amount of data needed for training the neural network one can explore pre-training link embeddings on a large corpus or use techniques such as node2vec [22] that can be applied to the road network. It is also worth exploring if it is possible to learn directly from the location data for example by applying 1d convolutional neural netowrks or RNNs to longitudinal and lateral acceleration estimated from speed and bearing.

References

[1] A. Riener and A. Ferscha, "Supporting Implicit Human-to-Vehicle Interaction: Driver Identification from Sitting Postures," in *Proceedings of the First Annual International Symposium on Vehicular Computing Systems*. Dublin, Ireland: ICST, 2008.

[2] T. Wakita, K. Ozawa, C. Miyajima, K. Igarashi, K. Itou, K. Takeda, and F. Itakura, "Driver identification using driving behavior signals," *IEICE Trans. Inf. Syst.*, vol. E89-D, no. 3, pp. 1188–1194, mar 2006.

[3] C. Miyajima, Y. Nishiwaki, K. Ozawa, T. Wakita, K. Itou, K. Takeda, and F. Itakura, "Driver modeling based on driving behavior and its evaluation in driver identification," *Proc. IEEE*, vol. 95, no. 2, pp. 427–437, feb 2007.

[4] M. V. Martínez, J. Echanobe, I. Campo, M. Martinez, J. Echanobe, and I. del Campo, "Driver Identification and Impostor Detection based on Driving Behavior Signals *," *2016 IEEE 19th Int. Conf. Intell. Transp. Syst.*, pp. 372–378, nov 2016.

[5] S. Jafarnejad, G. Castignani, and T. Engel, "Towards a Real-Time Driver Identification Mechanism Based on Driving Sensing Data," *20th Int. Conf. Intell. Transp. Syst.*, no. October, p. 7, 2017.

[6] M. Enev, A. Takakuwa, K. Koscher, and T. Kohno, "Automobile Driver Fingerprinting," *Proc. Priv. Enhancing Technol.*, vol. 2016, no. 1, jan 2016.

[7] S. Jafarnejad, G. Castignani, and T. Engel, "Non-intrusive distracted driving detection based on driving sensing data," in *Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems - Volume 1: VEHITS,*, INSTICC. SciTePress, 2018, pp. 178–186.

[8] G. Kar, S. Jain, M. Gruteser, J. Chen, F. Bai, and R. Govindan, "PredriveID: pre-trip driver identification from in-vehicle data," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing - SEC '17*. San Jose, California: ACM Press, 2017, pp. 1–12.

[9] D. Hallac, A. Sharang, R. Stahlmann, A. Lamprecht, M. Huber, M. Roehder, R. Sosič, and J. Leskovec, "Driver identification using automobile sensor data from a single turn," *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, pp. 953–958, 2016.

[10] L. Moreira-Matias and H. Farah, "On Developing a Driver Identification Methodology Using In-Vehicle Data Recorders," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 9, pp. 2387–2396, 2017.

[11] J. S. Wijnands, J. Thompson, G. D. Aschwanden, and M. Stevenson, "Identifying behavioural change among drivers using Long Short-Term Memory recurrent neural networks," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 53, pp. 34–49, Feb. 2018.

[12] A. Chowdhury, T. Chakravarty, A. Ghose, T. Banerjee, and P. Balamuralidhar, "Investigations on Driver Unique Identification from Smartphones GPS Data Alone," *Journal of Advanced Transportation*, vol. 2018, pp. 1–11, 2018.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv:1301.3781 [cs]*, Jan. 2013, arXiv: 1301.3781. [Online]. Available: http://arxiv.org/abs/1301.3781

[14] Y. Bengio, "Deep Learning of Representations for Unsupervised and Transfer Learning," in *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, ser. UTLW'11. JMLR.org, 2011, pp. 17–37, event-place: Washington, USA. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045796.3045800

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[16] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv:1406.1078 [cs, stat]*, Jun. 2014, arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078

[17] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[19] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Dec. 2014, arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980

[20] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3146–3154. [Online]. Available: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

[21] Y. Lassoued, J. Monteil, Y. Gu, G. Russo, R. Shorten, and M. Mevissen, "A Hidden Markov Model for Route and Destination Prediction," *arXiv:1804.03504 [physics]*, Mar. 2018, arXiv: 1804.03504.

[22] A. Grover and J. Leskovec, "node2vec: Scalable Feature Learning for Networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. San Francisco, California, USA: ACM Press, 2016, pp. 855–864. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2939672.2939754