# Driver Identification only using GPS trajectories: A Deep Learning Approach

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 depend 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 sample-rate data, mainly GPS trajectories obtained from smartphone. In this approach we contextualize each location data-points in a trip. Then we construct a set of continuous, categorical, and sequential features to represent the whole trip. For driver identification, we propose a Deep Learning (DL) architecture composed of embedding and recurrent neural networks (RNNs) layers. Our proposed approach, outperforms, LightGBM and HMM-based baselines. 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

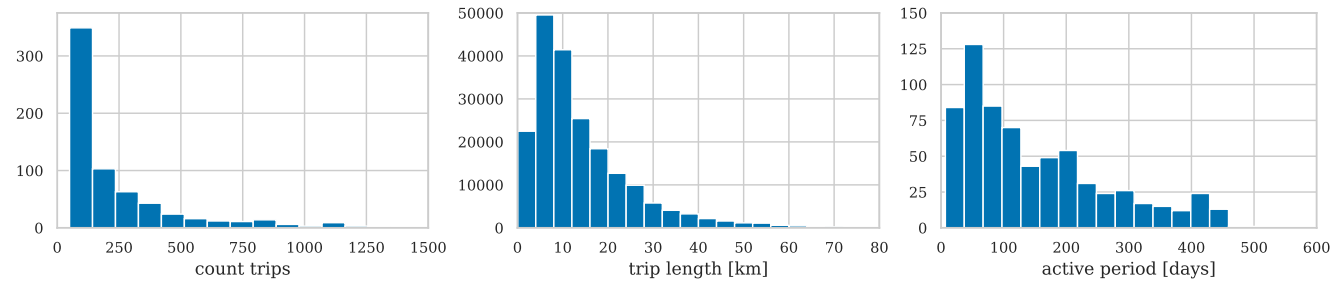
In recent years, the mobility landscape has transformed. People are now switching from the traditional car-ownership model to a panoply of shared-mobility offers, including car-sharing, car-pooling, ride-hailing or short-term leasing, among others. In parallel, car manufacturers have been increasingly introducing car connectivity (telematics) to even low-cost models. Even for cars without connectivity third-party aftermarket data collection devices (such as insurance dongles) exist that add additional connectivity. Availability of substantial amount of car data, enables new services such as fraud-detection, tailor-made mobility products with dynamic pricing, and automated claim management, for which driver identification (ID) becomes a crucial component. Driver ID refers to inferring driver's identity based on *driving behavior*. Anything driver does during driving can be considered *driving behavior*. This encompasses pedal operation and steering, vehicle dynamics or even higher level decisions such as speeding.

Driver ID also has indirect safety applications. Fatigue and drowsy driving—which are a major contributory factors to crash—are common among heavy vehicle long-haul drivers. In many countries, regulations limit the working hours of these drivers, however even though measures are in place to enforce such limitations, drivers resort to workarounds. Driver ID can add an extra level of confidence and prevent them. Regardless of the regulations, it is valuable for large fleets—heavy or otherwise—to be able to verify driver's identity without the need to install and maintain traditional authentication methods such as smart-cards or RFID tags.

Ride-hailing apps such as Uber, Lyft, and Bolt have revolutionized transportation by providing a level of comfort that once deemed unimaginable, but they have issues of their own; There have been many reports of assault and various other crimes by ride-hailing drivers [1], in addition there are cases that drivers commit fraud or game the system. They create fake global positioning system (GPS) traces, create ride requests from stolen accounts [2]. Introduction of background checks by the ride-hailing companies—which is now mandatory in some jurisdictions—has reduced the crime rates, but new workarounds emerge. Some drivers use stolen accounts, or share the their accounts. Similar challenge exists for on-demand delivery services such as Deliveroo, and even traditional fleets. This is another case that driver ID can prove useful, the major difference is that fleets and heavy vehicles are already equipped with data collection devices (in most cases) but in the case of ride-hailing apps, the only data collection device is the smartphone, which does not have access the rich *car data* that can be obtained from vehicle itself.

We can categorise the methods in the literature by the data they use: 1) *Car data*, often with high sample-rate, this kind of data is usually collected from car's internal network (most often 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. Most current research has dealt with the *car data*, these solutions require custom hardware, and high-sample data, which limits their deployment. There is need for driver ID methods that are not hardware-dependant and can guarantee high accuracy even with low sample-rate data, which minimizes communication and storage costs.

Therefore, in this paper we consider a scenario in which, only *location data* (GPS coordinates) of vehicles is available, and we investigate efficacy of this low-sample location data for identifying drivers. We provide a method that relies only on GPS data obtained from a smartphone and succeeds in accurately identifying the driver. We use and external service to contextualize the input, and extract features. These features have various data types, including continuous, categorical, as well as variable length sequential data. Our proposed system can accept all the above mentioned features and identify the drivers. 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




(a) (b) (c)

Figure 1. *(a)* Histogram of number of trips per driver. *(b)* Histogram of trip lengths in km. There were few trips longer than 80km that are excluded. *(c)* Histogram of active days, which is the interval between the first and the last trip made by each driver. Some outliers with active period longer than 600 days are excluded.

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.

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 VIII we conclude.

## II. Related work

The main focus of research in driver ID has been on *car data*, high sample-rate data obtained from the car itself. The general approach is to extract a set of features over sliding windows. Treat features from each window as a separate example, train a classifier and make predictions. In terms of features, a variety of methods is explored by the research community, in addition to simple statistics, some works use complex features such as cepstral coefficients [3], [4], [5], Discrete Wavelet transform (DWT) [6], and spectral features [7], [8]. In terms of the classification method, ensemble methods such as Random Forest (RF) [7], [6], [3], density estimation methods such as Gaussian mixture model (GMM) [5], [8] or artificial neural network (ANN) based method such as extreme learning machine (ELM) are used in the literature [4].

There are a few works that focus on *location data*. [9] 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 Machine learning (ML) algorithms to identify the driver, and achieved accuracy of about 70%. In [10], 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. [11] 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.

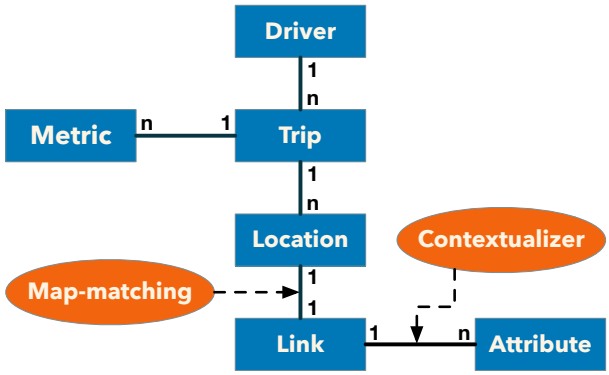



Figure 2. The relationship between different entities in the dataset. Each driver makes multiple *trips*, each *trip* is composed of multiple *location* data-points. A *map-matching* service maps a *location* data-point to one *link*, then a *contextualizer* service assigns multiple attributes to each *link*. For each *trip* we extract multiple metrics which take into account every *location*, their corresponding *links* and their *attributes*.

## 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,

Table I
METRICS, AND DIMENSION OF THEIR CORRESPONDING FEATURE VECTORS.

| | Metric | Continuous | Categorical | Sequential | Description |
|---|---|---|---|---|---|
| Temporal | Day of week | - | 1 (7) | - | Day of week |
| | Hour | - | 1 (24) | - | Hour |
| | Minute | - | 1 (4) | - | Minutes 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 | $2 \times 2$ | $2 \times 2$ ($\sim$ 4k) | - | Latitude and longitude of start and end of trip |
| | Link IDs | - | - | varies | Sequence of links traversed during a trip |
| | Bearing | 1 | - | - | Bearing |
| | 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 |

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[1] 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.

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:

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

*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.

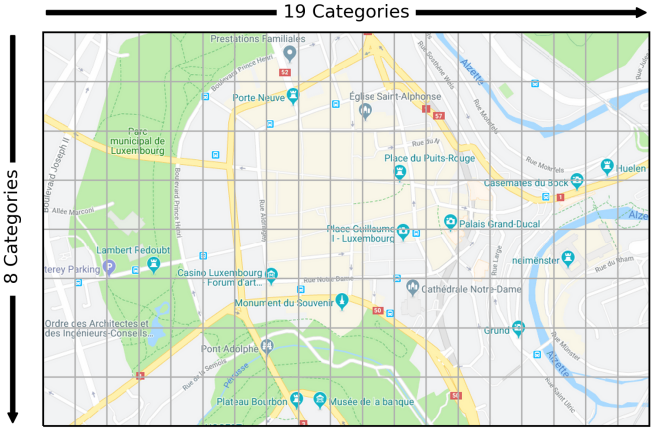



Figure 3. A demonstration of *location cross-features*. A grid is laid over the Luxembourg city's centre district. Each grid-cell can be thought of as a 2-d bucket that is assigned an integer identifier. Then any location data-point anywhere in a grid-cell is mapped to the grid-cell's identifier.

$std(\boldsymbol{X}^n)$ 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 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 | Preprocessing before | Preprocessing after |
|---|---|---|
| Number of drivers | 1385 | 678 |
| Average trips per driver | 261.51 | 297.06 |
| Total trips | 362198 | 201410 |

### C. Preprocessing

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 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 $\mathcal{C}_k$ from a known set of $K$ drivers. We propose to model the driver ID task as a supervised classification problem. Therefore we assume information on all possible targets, which is also known as *closed-world* scenario, assuming that all examples are from classes in the 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 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. However, 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* [12]. 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.

Let us take the example of *links* to describe the method more in detail. We can represent a trip as a sequence of links traversed, $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_L\}$, where $\mathbf{u}_l$ is a one-hot vector in space $\Delta^D$ 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 $\Delta^D$ 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 hyperparameters. After applying the map $f_{link\_emb} : U \mapsto V$, becomes $V = \{\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_L\}$, where $\boldsymbol{\nu}_l \in \mathbb{R}^P$. In

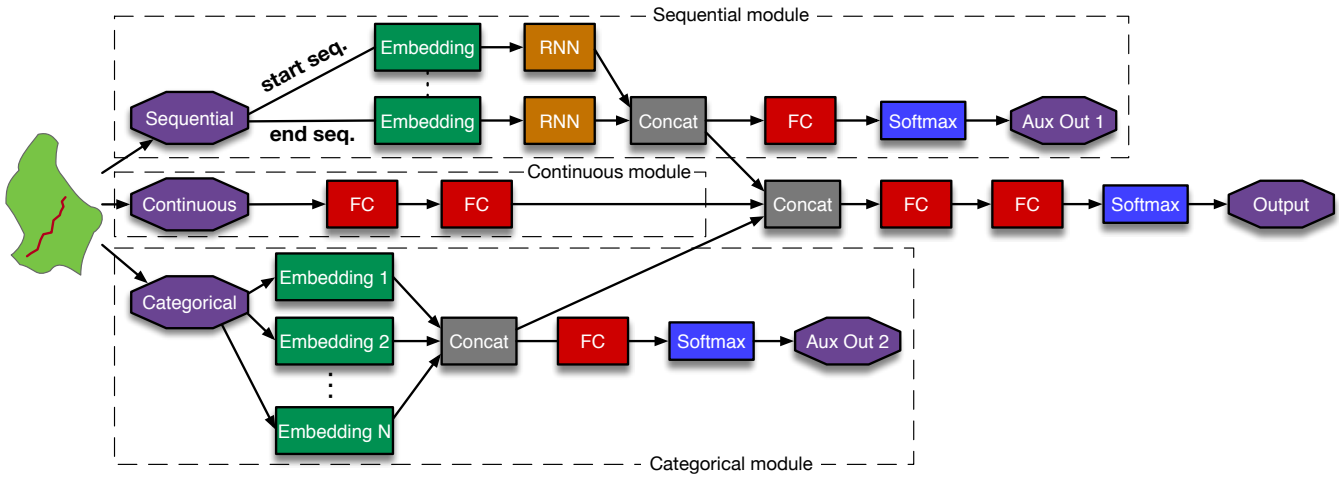



Figure 4. Architecture diagram

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 |

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 that is specialized in discriminating 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, which 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 [13].

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* and *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. To avoid having to handle long sequences of variable length, we separately model the beginning and end of a link sequence. We introduce a hyperparameter $\rho$ 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, 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 [14], [15]. 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 $\kappa$ is a hyperparameter. The resulting embeddings are then concatenated and passed through a fully-connected layer with rectified linear unit (ReLU) non-linearity [16] and merge 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:

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

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

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

We adjust $J$ to accommodate the auxiliary outputs.

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

Where $\omega$ 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:

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

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.

Table IV
Hyperparameters of the model

| Module | hyperparameter | 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**} |
| | $\rho$ (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} |
| | $\kappa$ (embedding coeff.) | {1, **2**, 3} |

### A. Hyperparameter search

DL architectures have large number of hyperparameters and the search space to find the optimal architecture is computationally expensive to cover. To find hyperparameters that perform well we break the network down to its modules. For each module we perform a separate hyperparameter 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 hyperparameters we experimented with is given in Table V-A and the best performing parameters are indicated in boldface. We performed hyperparameter 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]. It is optimized for faster training and efficiency. LightGBM is generally used for large amounts of data, to make sure it performs well with this amount of data we ran some initial experiments and compared its performance with Random Forest [21] and Gradient Boosting [22].

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 hyperparameters:

- *learning_rate*=0.1
- *max_depth*=30
- *min_data_in_leaf*=20
- *num_leaves*=50
- *num_round*=500

the hyperparemeters are chosen after running a hyperparameter 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 [23] 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.

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

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

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.

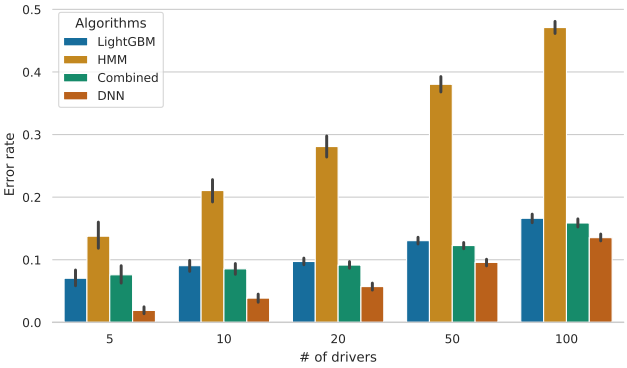
**Algorithm 1** Driver prediction with drivers as hidden states

$n \leftarrow |C|$
$k \leftarrow 1$
$r \leftarrow 0.1$
**for** $i \in C$ **do**
  $P_i \leftarrow p(C = i|l_1)$
**end for**
**while** $k \leq T$ **do**
  **for** $i \in C$ **do**
    $P_i \leftarrow P_i \cdot p(l_{k+1}|C = i)$
  **end for**
  $normalize(P_i)_{i \in C}$
  **for** $i \in C$ **do**
    $P_i \leftarrow \frac{r}{n} + (1 - r)P_i$
  **end for**
  $k \leftarrow k + 1$
**end while**
**return** $\arg\max_{i \in C} P_i$

*3) Combined model:* We also construct a model by combining LightGBM and the HMM model. Note that the proposed DNN takes advantage of *continuous*, *categorical*, and *sequential* features, but LightGBM only uses the first two, and HMM only the last one. The combined model therefore is expected to perform better than each single baseline methods. We take a linear combination of their output probabilities and use the resulting probabilities for prediction. We introduce a hyperparameter $\alpha$ 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. Empirically find the optimal value of $\alpha$ to be 0.7, and this is the value that we used in our evaluations.




(a)

| # drivers | Error-rate - mean (std.) Baselines | | | |
|---|---|---|---|---|
| | LightGBM | HMM | Combined | DNN |
| 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. Error-rate comparison between baseline and DNN. *(a)* and *(b)* are different representation of the same results. HMM has the highest error-rate, LightGBM performs much better than HMM. LightGBM/HMM combined model consistently provides slightly better performance than LightGBM. It is clear that DNN approach outperforms other approaches. We also observe that for 5 drivers, there is a large performance gap between DNN and combined model, however this gap decreases as the number of drivers increase to 100.

## 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 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

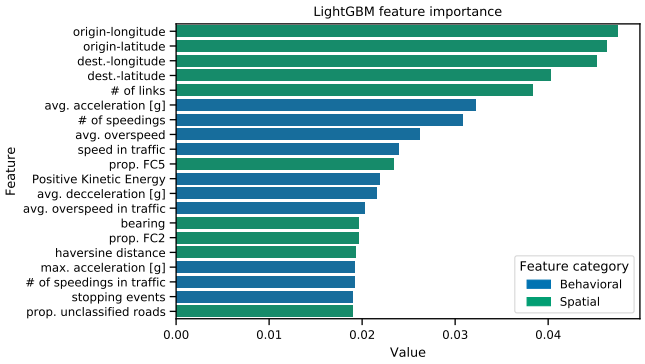



Figure 6. Top 20 most important features obtained from LightGBM. Spatial features related to origin and destination dominate the top spots, followed by even mix of spatial and behavioral features.

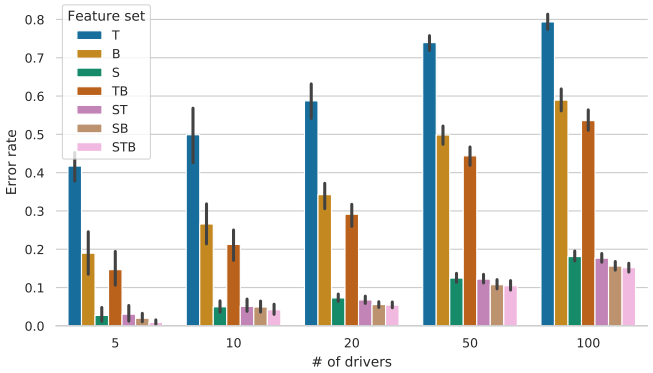



Figure 7. Error-rate per feature-set. Using all features (*STB*) results in low error-rate of 0.009 for 5 drivers and 0.152 for 100 drivers. Clearly the best single category is spatial (*S*), followed by behavioral (*B*).

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 than LightGBM or HMM for every scenario. 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 7. 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.

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*.

Table V
EFFECTIVENESS OF LOCATION CROSS-FEATURES

| # drivers | 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) |

### 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 unnecessary. Consider the case accurate coordinates are not available. This could be either due to lack of accurate data or because of privacy considerations 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 (perturbing the origin, destination coordinates) [24], [25]. 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*. In future work, in order to further asses the robustness of the system we suggest investigating how trimming the start and end of the trips would affect the performance.

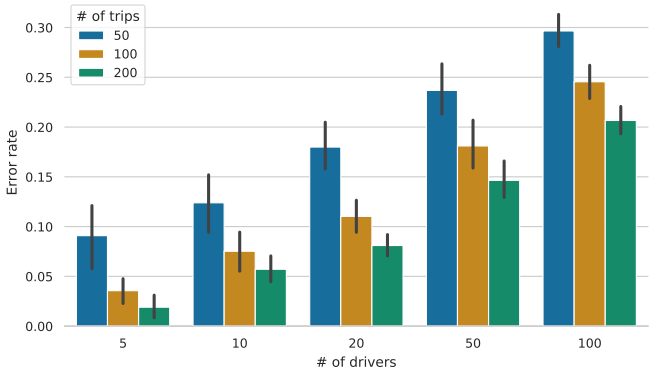



Figure 8. Error-rate for 3 experiments with 50, 100, 200 trips for training. We can clearly see that increase in amount of training data greatly reduced the error-rate. We can expect that even larger amount of training data would further decrease the error-rate.

### D. Amount of training data

We investigate how the amount of training data affects the model error-rate by experimenting with $\mathcal{L} \in \{50, 100, 200\}$ trips. In these experiments at each iteration, 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 proportional with $\mathcal{L}$, however this effect dampens as $\mathcal{L}$ increases. There are two explanations, 1) There is only so much to learn, over that there is little information that the the model can learn from the extra training data. 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 $\mathcal{L}$, perhaps too many repeated samples.

## VII. Discussion

We motivated this work as if driver identification is a desired functionality. However one can look at it from adversarial perspective. A system such as what we discussed here can be used by an adversary as part of a surveillance system or for data de-anonymization. Assume a dataset of anonymized GPS trajectories is published online. Now if the adversary somehow gets access to identities for a subset of the dataset or even a disjoint set, as long as there are individuals that have participated in both. In such cases, our proposed DNN can be used to de-anonymize the dataset. We see the biggest potential in driver identification only using *behavioral* features. Ride-sharing services such as Uber, Lyft need to verify driver's identity to prevent fraud and ensure rider's safety. In such case, we cannot rely on *spatial* or *temporal*, unless we encounter an obvious anomaly, for example a driver that always works between 9-17h, now starts taking rides at midnight, or imagine a change in city or country. Other than such extreme examples, the focus must be on *behavioral* features. Our approach achieved 81% accuracy for 5 drivers, better results can be achieved for smaller number of drivers, or reformulate the problem as a verification task, which is a simpler problem. For the spacial case of ride-sharing drivers, the cost of false positives is not high, a reasonable implementation can operate as follows. The DNN fails to verify the driver, then it triggers a more reliable authentication method such as facial recognition—Uber appears to be already using this approach this although it is unclear what triggers it[26]—verify driver's identity, and based on that decide to escalate or resolve the issue.

Furthermore to decrease the amount of training data one can explore pre-training link embeddings on a large corpus or use techniques such as node2vec [27] that can be applied to the road network. In node2vec, instead of using real driving data, we can construct the graph representing the road network and use random walks on the graph to learn representations (embeddings) for the nodes—which we would refer to as *links* in our problem—of the graph. Lastly, it is worth exploring if it is possible to learn directly from the location data for example by application of 1-dimensional convolutional neural networks or RNNs to longitudinal and lateral acceleration estimated from speed and bearing, which is available in location data obtained from smartphones.

## VIII. Conclusion

In this work we presented an end-to-end driver ID framework. The input to the system is location coordinates corresponding to a trip, sampled every second. This data is easy to obtain, either through a Smartphone or from the car itself. This is especially important because, more and more car manufacturers are providing always on connection for their cars, and upload car operation data to their servers. Some companies such as Mercedes Benz and BMW provide Application programming interfaces (APIs) for third-parties to access to such data. The input is then augmented and contextualized using an external service. The augmented features have various data types, including continuous, categorical, as well as variable length sequential data. Our proposed system can accept all the above mentioned features and identify the drivers. We compare our system with three strong baselines and outperform them. Since based on use-case it may not be desirable to use every feature categories, therefore we evaluated our method using different feature sets; We conclude that *spatial* features are most effective, then the *behavioral* features and the *temporal* features are the least effective. We also investigated the effectiveness of location cross-features

and showed that while using numerical value of latitude, longitude of the origin, destination and their cross-features is the most effective. When using either one of cross-features or numerical values, the former performs slightly better (up to one percentage point). We also showed that our method improves as more data is collected, which is ideal for a system that is constantly collecting more data per its usage. We obtain average 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.

## ACKNOWLEDGMENT

The authors would like to thank Motion-S SA for providing the dataset used in this study.

## REFERENCES

[1] "Who's Driving You? | Promoting for-hire vehicle safety and highlighting the risks of Uber and Lyft." [Online]. Available: http://www.whosdrivingyou.org/

[2] A. Ng, "Uber fights off scammers every day. Here's how it learned the tricks." [Online]. Available: https://www.cnet.com/news/uber-fights-off-scammers-every-day-heres-how-it-learned-the-tricks/

[3] 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.

[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] 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.

[6] 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.

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

[8] S. Jafarnejad, G. Castignani, and T. Engel, "Revisiting Gaussian Mixture Models for Driver Identification," in *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. Madrid: IEEE, Sep. 2018, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/8519588/

[9] 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.

[10] 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.

[11] 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.

[12] 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

[13] 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

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

[15] 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

[16] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. USA: Omnipress, 2010, pp. 807–814. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104425

[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] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[22] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. [Online]. Available: http://www.jstor.org/stable/2699986

[23] 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.

[24] C. Dwork, "Differential Privacy," in *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, ser. Lecture Notes in Computer Science, vol. 4052. Springer Verlag, Jul. 2006, pp. 1–12. [Online]. Available: https://www.microsoft.com/en-us/research/publication/differential-privacy/

[25] M. E. Andrs, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: differential privacy for location-based systems," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*. Berlin, Germany: ACM Press, 2013, pp. 901–914. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2508859.2516735

[26] "Engineering Safety with Uber's Real-Time ID Check," Mar. 2017. [Online]. Available: https://eng.uber.com/real-time-id-check/

[27] 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