Inaccurate Prediction Is Not Always Bad: Open-World Driver Recognition via Error Analysis

1st Jianfeng Li
Department of Computing
Hong Kong Polytechnic University
Hong Kong, China
jfli.xjtu@gmail.com

2nd Kaifa Zhao
Department of Computing
Hong Kong Polytechnic University
Hong Kong, China
kaifa.zhao@connect.polyu.hk

3rd Yajuan Tang
Department of Electronic and Information Engineering
Shantou University
Shantou, China
yjtang@stu.edu.cn

4th Xiapu Luo
Department of Computing
Hong Kong Polytechnic University
Hong Kong, China
csxluo@comp.polyu.edu.hk

5th Xiaobo Ma
Department of Computer Science and Technology
Xi'an Jiaotong University
Xi'an, China
xma.cs@xjtu.edu.cn

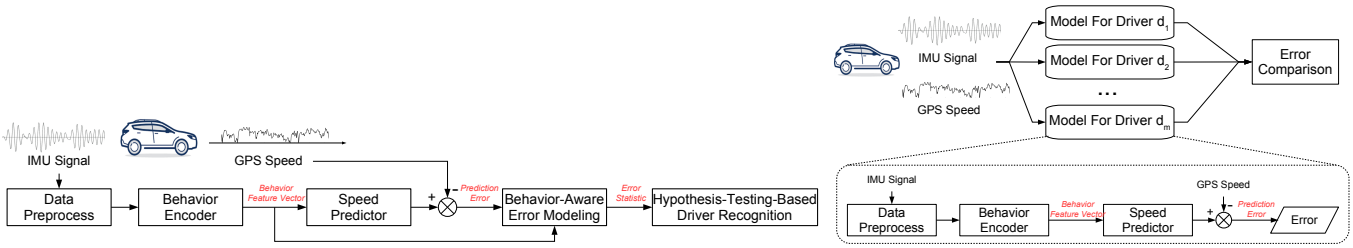
***Abstract*—Driver identification is of fundamental importance in many vehicle-related applications, such as fleet monitoring and anti-theft system. The vast majority of existing methods work under the closed-world assumption, which may be unrealistic in practice. In this paper, we consider a more practical but challenging scenario, i.e., open-world driver recognition, and propose a systematic method dubbed DRIVERPRINT. To recognize the driver of interest, DRIVERPRINT takes advantage of the behavioral predictability of the driver himself, thereby no need to collect data from other drivers for model training. Specifically, DRIVERPRINT predicts the behavior-related traveling speed with a driver-specific predictor, compares the prediction error with a pre-trained error model and finally recognizes drivers via error analysis. Besides open-world setting, our method is also compatible with closed-world driver classification. Real-world experiments demonstrate our method achieves reasonable accuracy. The average F1-score for open-world driver recognition is up to 0.91, while that for closed-world driver classification is up to 0.973.**

I. INTRODUCTION

The vehicle telematics techniques have bloomed unprecedentedly in last decades. As a salient example, modern automobiles are not only mechanic systems but also information systems equipped with hundreds of sensors [1] in favor of a higher safety, security, and economical efficiency. In addition to tracking vehicle state for vehicular automation, these sensors also provide good vantage points that facilitate driver behavior analysis, such as driving behavior classification [2]–[4], driving anomaly detection [5]–[7], and eco-driving evaluation [8], [9]. Among these research problems, driver identification is of fundamental importance in a wide spectrum of applications like fleet monitoring [10], anti-theft system [11], and usage-based insurance [12], thereby attracting a lot of attention from both academy and industry [13]–[22].

Driver identification is commonly formulated as a classification problem. Despite the success, the vast majority of existing methods (e.g., [15]–[17], [19]–[22]) only work under the *closed-world* assumption. That is, all drivers to be identified in the testing stage should also be present during model training stage. Consequently, these methods may be faced with substantial limitations when being applied in practice. For example, in an anti-theft system, it is impossible to guarantee the impostor will be present in the training dataset. In addition to the closed-world assumption, to carry out an accurate driver identification, existing methods [13]–[20] generally collect controller area network (CAN) data (e.g., steering-wheel angle, pedal position, and brake position), which are comprehensive and reliable to characterize driving behaviors. However, CAN protocols are often proprietary, and thus these methods need considerable effort for protocol reverse-engineering, potentially undermining their scalability. Moreover, collecting CAN data exposes a possible attack surface to the cyberspace because the vulnerabilities of aftermarket vehicle diagnostic tools may be exploited by the adversary to launch attacks [23]–[25], which seriously endangers the security of in-vehicle networks and even the driving safety.

In this paper, we aim to tackle the above limitations faced by existing methods. First, we consider a more practical but challenging scenario, i.e., open-world driver recognition, where drivers in the testing stage can be absent during model training stage, thereby obviating the need of closed-world assumption. To achieve this goal, a straightforward solution is formulating driver recognition as a binary classification problem, where the driver of interest is deemed to be the positive class and the rest as a whole the negative class. In other words, to recognize the driver of interest, one needs to collect driving data from a plethora of drivers with sufficient diversity to construct negative samples for model training, which is often of high cost. Different from such a solution, we are intended to achieve open-world driver recognition *without* collecting driving data from drivers other than the driver of interest. As such, it not only greatly reduces the cost in data collection and computational overhead but also mitigate privacy concern caused by accessing driving data from other



(a) Open-world driver recognition.

(b) Closed-world driver classification.

Fig. 1: The framework of DRIVERPRINT.

drivers. Second, we recognize drivers only using data collected from an onboard sensor consisting of an inertial measurement unit (IMU) and a GPS module, thereby no need to access CAN data. More specifically, we only make use of y-axis acceleration and z-axis gyroscope reading from the IMU and the GPS speed from the GPS module, which can be easily collected even using a smartphone.

Despite the promise, achieving the above goals is non-trivial. The major challenge is how to effectively extract exclusive features of the driver of interest to distinguish him/her from other drivers, whose driving characteristics are unknown a prior. To address this challenge, we propose to recognize a driver by taking advantage of the behavioral predictability when he/she is driving a vehicle. Specifically, we predict the traveling speed of the vehicle using a model trained based on his/her historical driving data. At the first glance, it is hard to find a direct connection between driver recognition and speed prediction, which is essentially a regression problem. Nevertheless, we are able to establish the underlying connection between them if we consider speed prediction for multiple drivers instead of individual ones. That is speed prediction for different drivers can be regarded as a series of similar machine learning tasks. One may predict the traveling speed of one driver using the model trained based on the historical driving data of *another* driver. If these two drivers behave more similarly, the speed prediction is expected to be more accurate. Therefore, prediction error is informative to characterize the similarity of drivers and thus analyzing prediction error potentially facilitates driver recognition.

We advance a systematic method dubbed DRIVERPRINT to achieve open-world driver recognition via error analysis. Such a task is non-trivial because i) traveling speed may dramatically vary across different driving behaviors (e.g., driving straight and turning) and ii) error model also changes due to different predictabilities associated to different driving behaviors. To handle these problems, DRIVERPRINT maps driving data to low-dimensional behavior feature vectors that reflect different driving behaviors with a behavior encoder. We then predict traveling speed and construct an error model in light of these behavior feature vectors. Finally, we transform the problem of open-world driver recognition into a hypothesis testing and recognize the driver of interest according to the testing results. Our contribution is three-fold.

- We propose DRIVERPRINT to carry out open-world driver recognition, which is capable of accurately recognizing the driver of interest i) without the aid of driving data from other drivers and ii) only using data from an onboard sensor without accessing CAN data.
- DRIVERPRINT is also compatible with closed-world assumptions. When applied in closed-world driver classification, it is flexible in handling varying driver set, e.g., adding a new driver, because models are trained in an incremental fashion, thereby no need to re-train them from the scratch.
- Real-world experiments demonstrate DRIVERPRINT is reasonably accurate. The average F1-score for open-world driver recognition is up to 0.91, while that for closed-world driver classification is up to 0.973.

Roadmap. We describe the overall framework of DRIVERPRINT in Section II and elaborate the system design in Section III. Section IV reports experimental results and Section V discusses related works. We finally conclude in Section VI.

II. SYSTEM OVERVIEW

DRIVERPRINT aims at an open-world driver recognition. More specifically, given some driving data (i.e., IMU readings and GPS speed records), we want to recognize whether or not they are from the driver of interest, say driver d.

DRIVERPRINT achieves this goal by predicting traveling speed associated to different driving behaviors and analyzing the prediction errors to identify if they deviate from the pre-trained error model. Fig. 1(a) illustrates the framework for open-world driver recognition. After data preprocessing, IMU readings are fed to the behavior encoder, which automatically extracts driving behaviors from IMU readings. Driving behaviors are represented as behavior feature vectors and they are used for predicting traveling speed and error modeling. By comparing the predicted traveling speed with GPS speed measured by GPS module, we obtain the prediction error. Assume that the model in question is trained based on the historical driving data from d. It is expected to be more accurate in predicting traveling speed if new driving data are from d, compared with other drivers. Therefore, prediction errors are informative in distinguishing drivers. Based on this observation, we train an error model using d's historical driving data and make use of it to analyze whether the prediction errors deviate from this model. Finally, we transform this problem into a hypothesis testing and recognize driver according to the testing result.

DRIVERPRINT is also compatible with closed-world setting. Fig. 1(b) illustrates the framework for closed-world driver classification. It is essentially a simplified version of the previous framework for open-world driver recognition. Without modeling prediction error and hypothesis testing, it directly compares prediction errors of models trained for different drivers. Given new driving data, if the model for driver d gives the most accurate prediction of traveling speed (i.e., the smallest prediction error), we identify they are from d.

III. SYSTEM DESIGN

In this section, we elaborate the design of different components of DRIVERPRINT and how they work together to carry out open-world driver recognition and closed-world driver classification.

A. Data Preprocessing

To carry out an effective and efficient driver recognition, DRIVERPRINT preprocesses IMU readings in four steps.

- ***Coordinate Calibration:*** it focuses on adjusting the IMU coordinate to be consistent with vehicle coordinate. To this end, we derive a transform matrix based on IMU reading when the vehicle stops (resp. that when the vehicle starts to move) to determine z-axis (resp. y-axis) of vehicle coordinate. By leveraging this transform matrix, we project IMU readings to the coordinate system of vehicle.
- ***Denoising IMU Signal:*** We filter out the noise of IMU readings caused by device vibration and sensor sensitivity based on Kalman filtering [26], [27].
- ***Eliminating Biase:*** We eliminate the bias of IMU reading by subtracting the average value of each reading measured when the vehicle stops.
- ***Data Slicing:*** We divide IMU readings and GPS speed records into sliding time windows in favor of efficient speed prediction and error analysis. Assume that the duration of each time window is τ and the period when we collect IMU readings and GPS speed records is T. We totally obtain $n = T/\tau$ time windows $\{w^{(i)}\}_{i=1}^{n}$. For the time window $w^{(i)}$, we denote by $\mathbf{x}^{(i)} = (\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \ldots, \mathbf{x}_N^{(i)})$ (resp. $v^{(i)}$) the IMU readings (resp. the average GPS speed) within it. In here, N is the number of IMU reading samples in $w^{(i)}$. In our experiment, the sampling rate of IMU is set to be 20 Hz and τ is set to be 10 seconds. Therefore, we obtain $N = 200$.

Generally, IMU outputs instantaneous acceleration and gyroscope readings at x-axis, y-axis, and z-axis (totally 6 different readings). To reduce computational overhead and mitigate the risk of overfitting during model training, we focus on y-axis acceleration and z-axis gyroscope reading[1] because they are the most informative to reflect driver's behavior. For example, y-axis acceleration can be used for characterizing the operation of pedal and brake, while z-axis gyroscope reading can be used for characterizing the operation of steering wheel. As such, each $\mathbf{x}_t^{(i)}$ for $1 \leq t \leq N$ contains a y-axis acceleration reading and a z-axis gyroscope reading.

[1]X-axis, y-axis, and z-axis correspond to pitch axis, roll axis, and yaw axis respectively.

B. Behavior Encoder

To facilitate speed prediction and error modeling, we devise a behavior encoder to map $\mathbf{x}^{(i)}$ into a low-dimensional behavior feature vector $\mathbf{z}^{(i)} = f(\mathbf{x}^{(i)})$ for each time window. We have implemented three different behavior encoders, i.e., multilayer perception (MLP) encoder, gated recurrent unit (GRU) encoder, and one dimensional convolutional neural network (1D-CNN) encoder. The comparison of their performance will be presented in Section IV.

*1) **MLP Encoder**:* As shown in Fig. 2(a), MLP encoder first flattens $\mathbf{x}^{(i)}$ and feeds it to a MLP, which is comprised of an input layer (400 neurons), a hidden layer (50 neurons), and an output layer (3 neurons). MLP finally outputs a $n_{\mathbf{z}}$-dimensional behavior feature vector $\mathbf{z}^{(i)}$. In this paper, $n_{\mathbf{z}}$ is set to be 3 by default.

*2) **GRU Encoder**:* In machine translation, the well known model Seq2Seq [28] encodes the meaning of a sentence with a recurrent neural network (RNN) or its variants. Borrowed from Seq2Seq, we also devise a behavior encoder based on GRU [29], a variant of RNN, to encode driving behaviors. GRU overcomes the gradient vanishing faced by the vanilla RNN by taking advantage of an update gate $u_t^{(i)}$ and a reset gate $r_t^{(i)}$ as shown in Fig. 2(b). In the GRU encoder, $\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \ldots, \mathbf{x}_N^{(i)}$ are sequentially fed into the GRU. The hidden state $\mathbf{h}_t^{(i)}$ is a $n_{\mathbf{z}}$-dimensional vector. $\mathbf{h}_0^{(i)}$ is set to be $\mathbf{0}$ and $\mathbf{h}_t^{(i)}$ $(t > 0)$ is updated through an iterative process:

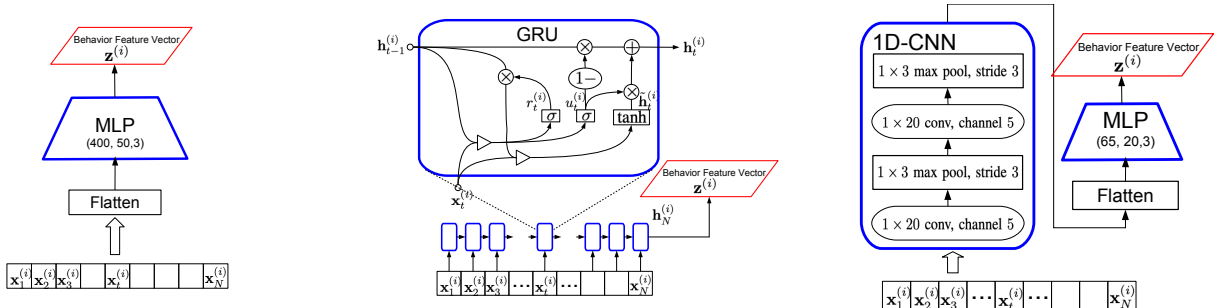
$$\begin{aligned} u_t^{(i)} &= \sigma(\mathbf{W}_u \mathbf{x}_t^{(i)} + \mathbf{U}_u \mathbf{h}_{t-1}^{(i)} + \mathbf{b}_u), \\ r_t^{(i)} &= \sigma(\mathbf{W}_r \mathbf{x}_t^{(i)} + \mathbf{U}_r \mathbf{h}_{t-1}^{(i)} + \mathbf{b}_r), \\ \tilde{\mathbf{h}}_t^{(i)} &= \tanh(\mathbf{W}_h \mathbf{x}_t^{(i)} + \mathbf{U}_h (r_t^{(i)} \odot \mathbf{h}_{t-1}^{(i)}) + \mathbf{b}_h), \\ \mathbf{h}_t^{(i)} &= (1 - u_t^{(i)}) \odot \mathbf{h}_{t-1}^{(i)} + u_t^{(i)} \odot \tilde{\mathbf{h}}_t^{(i)}, \end{aligned} \tag{1}$$

where $\mathbf{W}_u$, $\mathbf{W}_r$, $\mathbf{W}_h$, $\mathbf{U}_u$, $\mathbf{U}_r$, $\mathbf{U}_h$, $\mathbf{b}_u$, $\mathbf{b}_r$, and $\mathbf{b}_h$ are parameters learned during model training. The final hidden state will be regarded as the behavior feature vector, i.e., $\mathbf{z}^{(i)} = \mathbf{h}_N^{(i)}$.

*3) **ID-CNN Encoder**:* The third behavior encoder is based on 1D-CNN [30]. Fig. 2(c) shows the detail of network structure. 1D-CNN is comprised of two convolutional layers and two max pooling layers. It is worth noting that the input channel number of the first convolutional layer is 2, corresponding to y-axis acceleration and z-axis gyroscope. Besides, the input channel number of the second convolutional layer and output channel number of both convolutional layers are all set to be 5. Similar to the conventional pipeline in image classification, the output of 1D-CNN will be fattened and fed into a MLP and the MLP outputs a $n_{\mathbf{z}}$-dimensional behavior feature vector $\mathbf{z}^{(i)}$

C. Speed Predictor

The speed predictor is a function with respect to the behavior feature vector. Formally, we denote by $g(\cdot)$ the



(a) MLP encoder. (b) GRU encoder. (c) 1D-CNN encoder.

Fig. 2: Architecture of different behavior encoders.

speed predictor. The average speed within $w^{(i)}$ is predicted as $\hat{v}^{(i)} = g(\mathbf{z}^{(i)})$. We approximate g using a MLP.

Parameters of MLP and the previous behavior encoders are jointly learned based on the historical data from driver d. For simplicity, we denote by $\mathcal{M}_d = (f_d, g_d)$ the model trained based d's historical driving data. In here, f_d (resp. g_d) is the behavior encoder (resp. speed predictor) for d. We train $\mathcal{M}_d$ by minimizing the MSE loss

$$\min \sum_{(\mathbf{x}^{(j)}_{train}, v^{(j)}_{train}) \in \mathcal{D}_d} \| g_d(f_d(\mathbf{x}^{(j)}_{train})) - v^{(j)}_{train} \|_2, \qquad (2)$$

where $\mathcal{D}_d$ is a set consisting of d's historical driving data.

D. *Behavior-Aware Error Modeling*

We establish the error model for speed prediction based on d's historical driving data. If prediction error of new driving data greatly deviates from this model, it is highly suspicious that they are from the driver other than d. Our error model is based on Gaussian process (GP) regression [31] because it is capable of estimating not only the expectation of prediction error but also the standard deviation of it, which is used for error normalization in the downstream hypothesis testing. As shown in Fig. 1(a), the input of error modeling depends on the output of behavior encoder and speed predictor. To avoid potential overfitting during model training, we decouple the training of GP regressor and that of behavior encoder and speed predictor. Therefore, instead of $\mathcal{D}_d$, we train the GP regressor using another set of d's historical driving data denoted by $\mathcal{D}'_d$.

Formally, the absolute error when predicting $v^{(j)}_{train}$ is computed by $e^{(j)}_{train} = |g_d(f_d(\mathbf{x}^{(j)}_{train})) - v^{(j)}_{train}|$, where $(\mathbf{x}^{(j)}_{train}, v^{(j)}_{train}) \in \mathcal{D}'_d$. Combining all $e^{(j)}_{train}$ yields an error vector $\mathbf{e}_{train}$. Given a set of new driving data $\mathcal{D}_{test}$, we compute the absolute error of speed prediction in the ith time window by $e^{(i)} = |g_d(f_d(\mathbf{x}^{(i)})) - v^{(i)}|$, where $(\mathbf{x}^{(i)}, v^{(i)}) \in \mathcal{D}_{test}$. If $\mathcal{D}_{test}$ is from d, $e^{(i)}$ can be estimated using a GP regressor. Specifically, the joint distribution of $e^{(i)}$ and $\mathbf{e}_{train}$ follows a multivariate Gaussian distribution:

$$\begin{bmatrix} \mathbf{e}_{train} \\ e^{(i)} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{train} & \mathbf{K}_{i,train} \\ \mathbf{K}^\top_{i,train} & k^{(i)} \end{bmatrix} \right), \qquad (3)$$

where $\mathbf{K}_{train}$ and $\mathbf{K}_{i,train}$ are kernel matrices that characterize the correlation between error samples generated from training dataset $\mathcal{D}'_d$ and that between them and $e^{(i)}$. To compute these kernel matrices, we choose dot product kernel instead of another commonly used radial basis function (RBF) kernel because the former one performs more robust than the latter in our problem. Specifically, we obtain $\mathbf{K}_{train}(j,k) = \sigma_0^2 + \mathbf{z}^{(j)\top}_{train} \mathbf{z}^{(k)}_{train}$ and $\mathbf{K}_{i,train}(j) = \sigma_0^2 + \mathbf{z}^{(i)\top} \mathbf{z}^{(j)}_{train}$, where $\mathbf{z}^{(i)} = f_d(\mathbf{x}^{(i)})$ and $\mathbf{z}^{(j)}_{train} = f_d(\mathbf{x}^{(j)}_{train})$. Likewise, we obtain $k^{(i)} = \sigma_0^2 + \mathbf{z}^{(i)\top} \mathbf{z}^{(i)}$. In this paper, σ_0 is set to be 1 by default. From (3), we obtain the marginal distribution of $e^{(i)}$ is

$$e^{(i)} \sim \mathcal{N}\left(\mu^{(i)}, \sigma^{(i)}\right), \qquad (4)$$

where $\mu^{(i)} = \mathbf{K}^\top_{i,train} \mathbf{K}^{-1}_{train} \mathbf{e}_{train}$ (resp. $\sigma^{(i)} = -\mathbf{K}^\top_{i,train} \mathbf{K}^{-1}_{train} \mathbf{K}_{i,train} + k^{(i)}$) is the mean (resp. standard deviation) of $e^{(i)}$.

E. *Open-World Driver Recognition*

We identify whether or not new driving data are from d by checking if speed prediction errors deviate from his error model. Formally, We denote by $\tilde{e}^{(i)} = (e^{(i)} - \mu^{(i)})/\sigma^{(i)}$ the normalized absolute error when predicting $v^{(i)}$ and $\bar{e} = \sum_{i=1}^{n} \tilde{e}^{(i)}/\sqrt{n}$ the error statistic over all time windows of $\mathcal{D}_{test}$. Under the assumption of GP regression, i.e., $e^{(i)}$ following normal distribution, it is easy to prove that $\bar{e}$ is a random variable drawn from a standard normal distribution, i.e., $\bar{e} \sim \mathcal{N}(0,1)$, since it is a linear combination of random variables drawn from normal distribution (aka. Gaussian distribution). In fact, such a result still holds without GP regression assumption because it can also be derived according to the central limit theorem [32].

Hypothesis Testing. To recognize driver in a open-world setting, we conduct hypothesis testing by leveraging the error statistic $\bar{e}$. Recall that the model $\mathcal{M}_d$ is expected to be more accurate in predicting d's traveling speed compared with that of other drivers. It implies that if the driving data are from other drivers, $\bar{e}$ will be larger. Therefore, we identify wether or not the given driving data are from other drivers through a right tailed test.

- Null hypothesis H_0: $\mathcal{D}_{test}$ is from driver d.
- Alternative hypothesis H_1: $\mathcal{D}_{test}$ is not from driver d.

Given a confidence level α, the recognition threshold is computed by $\epsilon = \Phi^{-1}(\alpha)$, where $\Phi(x) = 1/\sqrt{2\pi}\int_{-\infty}^{x} e^{-t^2/2}\mathrm{d}t$ is the cumulative distribution function (CDF) of the standard normal distribution. If $\bar{e} < \epsilon$, we accept H_0 and otherwise we reject H_0.

F. *Closed-World Driver Classification*

In addition to open-world driver recognition, DriverPrint is also compatible with closed-world assumption to carry out driver classification. Such a task is much easier than open-world driver recognition because the information from different drivers (instead of only one driver) can be taken advantage of for accuracy improvement. As shown in Fig. 1(b), the model used for closed-world driver classification is essentially a simplified version for open-world driver recognition. Without modeling prediction error, it directly compares prediction errors derived by models for different drivers. Formally, let $\mathscr{D} = \{d_1, d_2, \ldots, d_m\}$ be a driver set comprised of drivers in question. We denote by $\mathcal{M}_k = (f_k, g_k)$ the model trained based on historical driving data from d_k. Given a set of new driving data $\mathcal{D}_{test} = \{(\mathbf{x}^{(i)}, v^{(i)})\}_{i=1}^{n}$, we infer it is from driver $\hat{d}$ whose model is able to provide the most accurate prediction

$$\hat{d} = \arg\min_{d_k \in \mathscr{D}} \sum_{i=1}^{n} \left| g_k(f_k(\mathbf{x}^{(i)})) - v^{(i)} \right|. \tag{5}$$

IV. Experiment

To demonstrate the effectiveness of DriverPrint when applied in practice, our experiments are based on real-world data. We evaluate DriverPrint in both open-world driver recognition and closed-world driver classification.

A. *Data Preparation*

TABLE I: Statistic analysis of different driving behaviors in our dataset.

Behavior	#Behavior Instance	Cumulative Duration (second)
Driving Straight	1,333	14,988
Turning Left	498	3,123
Turning Right	284	2,046
Left Lane Change	176	1,136
Right Lane Change	248	1,573
Left U-Turn	141	1,150
Right U-Turn	20	94
Left Curve	61	1,052
Right Curve	51	1,020
Back Up	27	166
Total	2,839	26,348

To collect driving data, we recruit three skillful drivers. To eliminate behavioral difference caused by different vehicles instead of driver themselves, they are required to drive the same car, i.e., a Skoda Octavia. In this car, we deploy a customized onboard sensor that integrates a MPU6050 IMU and a GPS module. Additionally, we also capture video using a front-view camera for behavior analysis. Specifically, we manually label 10 typical driving behaviors with the aid of video replay. Although these behavior labels are not necessary in the training and testing of DriverPrint, they provide a perspective that reveals why behavior-aware error modeling helps driver recognition. Table I details the number of different behavior instances and the cumulative duration corresponding to different behaviors. The total number of behavior instances is 2,839 and the total duration is more than 7 hours (26,348 seconds).

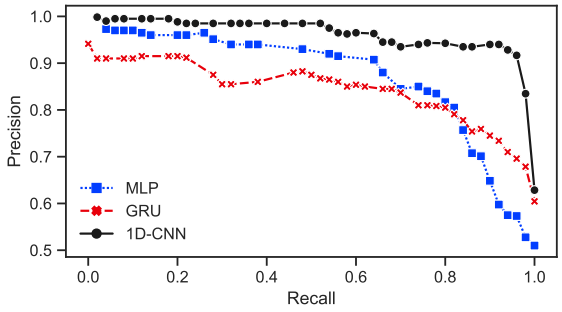


Fig. 3: Precision and recall for different encoders in open-world driver recognition.

B. *Experimental Setup*

For each driver, we equally split his/her driving data into 100 time slots and randomly choose 80 (resp. 20) time slots for training (resp. testing) in each experiment. Among the training time slots, we randomly choose 40 time slots to train behavior encoder and speed predictor, while the remaining 40 time slots are used for training error model (i.e., GP regressor). In each experiment, one driver, say a, is randomly chosen as the positive class and another driver, say b, is randomly chosen as the negative class. In the setting of open-world driver recognition, we only train the model of a, i.e., $\mathcal{M}_a$ while the model of b is unknown. In the setting of closed-world driver classification, models for both drivers are trained. To evaluate the performance, we choose precision, recall, and F1-score as accuracy metrics. We implement behavior encoders and speed predictor with Pytorch 1.4.0 and prediction error model (i.e., GP regressor) with scikit-learn 0.23.1.

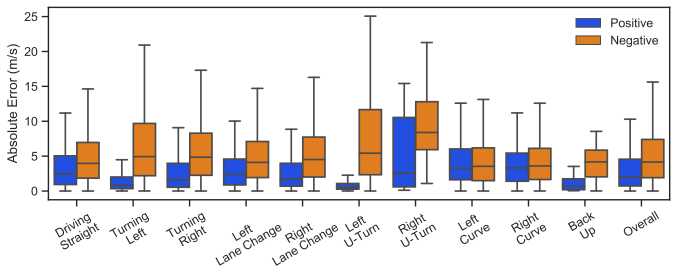
Statistical soundness. We run both open-world driver recognition experiments and closed-world driver classification experiments 100 times and find the mean and standard deviation to report the final performance.

C. *Open-World Driver Recognition*

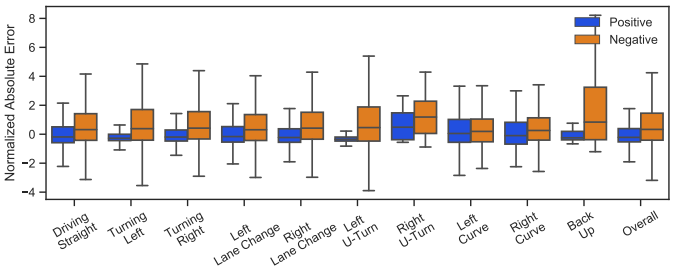
TABLE II: Evaluating open-world driver recognition (mean±standard deviation).

Encoder	Precision	Recall	F1-Score
MLP	0.800 ± 0.245	0.840 ± 0.367	0.707 ± 0.344
GRU	0.790 ± 0.247	0.940 ± 0.238	0.800 ± 0.258
1D-CNN	$\mathbf{0.895 \pm 0.204}$	$\mathbf{0.980 \pm 0.140}$	$\mathbf{0.910 \pm 0.188}$

We first report the experimental result in open-world driver recognition. Table II compares the performance of different behavior encoders. For each behavior encoder, the confidence



(a) Absolute error for speed prediction.



(b) Normalized absolute error for speed prediction.

Fig. 4: Error analysis in open-world driver recognition.

levels α for hypothesis testing have been tuned to maximize F1-score. It is easy to find DriverPrint achieves the most accurate driver recognition when 1D-CNN encoder is applied. The average F1-score is 0.91. GRU encoder achieves a worse performance with an average F1-score 0.8, while MLP encoder performs the worst as its average F1-score is 0.707. 1D-CNN encoder achieve the highest accuracy because it is able to effectively capture behavioral features at different timescales to achieve a good representation of driving behaviors. Fig. 3 shows how precision and recall change when varying confidence level α, which again demonstrates the advantage of 1D-CNN encoder. 1D-CNN encoder consistently outperforms the other two encoders in terms of both precision and recall.

We further analyze how behavior-aware error modeling facilitates driver recognition. Fig. 4 shows error distribution of speed prediction associated with different driving behaviors. Although DriverPrint is not designed for identifying these semantic driving behaviors, it implicitly encodes driving behaviors for purpose of more accurate speed prediction and error modeling. As shown in Fig. 4(a), turning left and left U-turn appear to be two most discriminative behaviors to recognize drivers due to the most significant difference between positive class and negative class. Another insight we can obtain from Fig. 4(a) is prediction error associated with different driving behaviors may follow different distributions (e.g., different means and variances), indicating it is difficult to find a globally optimal threshold to conduct driver recognition. The behavior-aware error modeling is devised to tackle this challenge. Fig. 4(b) shows the distribution of prediction errors that have been normalized by leveraging our error model (i.e, GP regressor). We can find normalized absolute errors associated with various driving behaviors are all around 0. Additionally, error variance of positive class tends to be more consistent across different behaviors, compared with that before normalization in Fig. 4(a). Benefit from such a normalization, it becomes much easier to recognize drivers based on hypothesis testing.

D. *Closed-World Driver Classification*

TABLE III: Evaluating closed-world driver classification (mean±standard deviation).

Encoder	Precision	Recall	F1-Score
MLP	0.955 ± 0.143	0.910 ± 0.286	0.880 ± 0.293
GRU	0.980 ± 0.098	0.960 ± 0.196	0.947 ± 0.204
1D-CNN	$\mathbf{0.990 \pm 0.070}$	$\mathbf{0.980 \pm 0.140}$	$\mathbf{0.973 \pm 0.147}$

We also evaluate DriverPrint in closed-world driver classification. Generally, the performance of all encoders is improved compared with that in open-world driver recognition. The underlying reason is more information (not only the model of positive class but also that of negative class) have been considered to carry out a more accurate classification. Among all encoders, 1D-CNN encoder again achieves the highest accuracy with average F1-score 0.973.

V. Related Work

Driver identification plays a fundamental role in many vehicle-related applications, such as feet monitoring [10], anti-theft system [11], and usage-based insurance [12]. Despite biology-based methods, such as facial recognition [33] and retina scans [34], may achieve impressive accuracy, they also raise serious privacy concerns and incur extra cost, e.g., installation of camera, limiting their practicality.

Compared with biology-based methods, identifying drivers based on driving behavior features that are reflected by various sensors is less privacy-intrusive and low-cost due to off-the-shelf in-vehicle sensors, thereby attracting lots of attention from both academy and industry [13]–[22]. Many existing works identified drivers from CAN data. For example, Hallac et al. took advantage of CAN data from a single turn to carry out driver identification [17]; Girma et al. proposed an LSTM-based deep learning model to identify driver identity based on the individual's unique driving pattern involved in CAN data [16]; Remeli et al. identified drivers based on the analysis of CAN logs [15]. Despite high utility and reliability, analysis based on CAN data is i) less scalable because CAN protocols are often proprietary and need substantial effort for reverse-engineering and ii) risky since it potentially exposes attack surface to the cyberspace. There are some works that identify drivers without accessing CAN data. For example, Sánchez et al. identified drivers based on tri-axial accelerometer signals from the smartphone [21]; Chowdhury et al. identified drivers based on GPS data [22]. No matter whether they access CAN data, the vast majority of existing methods (e.g., [15]–[17], [19]–[22]) work under the closed-world assumption, which is often unrealistic in practice.

We aim at open-world driver recognition without accessing CAN data in this paper to obviate the need of closed-world

assumption. Similar to our goal, there are a few works also conduct open-world driver recognition. For example, Dang and Fürnkranz recognized drivers by embedding driver behaviors into a low-dimensional space with Siamese LSTM networks [14]; Martínez et al. constructed an open-set classification model for impostor detection [13]. However, both works identified drivers with the aid of CAN data. Additionally, our method only make use of driving data from the driver of interest, whereas driving data from other drivers are also necessary in both works to either train the Siamese LSTM networks [14] or construct negative samples in training dataset [13].

VI. Conclusion

In this paper, we proposed a systematic method dubbed DriverPrint to carry out open-world driver recognition. Our method features several advantages compared with existing methods. First of all, DriverPrint obviates the closed-world assumption that is commonly involved in existing methods and thus is more practical. Second, it only makes use of driving data collected from an onboard sensor without accessing CAN data, thereby i) no need to reverse-engineer proprietary CAN protocols and ii) avoiding to expose attack surface to the cyberspace. Third, our method trains models only using driving data from the driver of interest without the aid of driving data from other drivers, which makes our method is low-cost in collecting training data and flexible for incremental model training. Last but not least, DriverPrint is not only competent in open-world driver recognition but also compatible with closed-world driver classification. We have demonstrated it is reasonably accurate in both scenarios based on real-world experiments. In the future, we plan to further validate the effectiveness and efficiency of DriverPrint through large-scale experiments.

References

[1] S. N. Narayanan, S. Mittal, and A. Joshi, "Obd_securealert: An anomaly detection system for vehicles," in *Proc. IEEE SMARTCOMP*, 2016.

[2] F. Martinelli, F. Mercaldo, A. Orlando, V. Nardone, A. Santone, and A. K. Sangaiah, "Human behavior characterization for driving style recognition in vehicle system," *Computers & Electrical Engineering*, vol. 83, p. 102504, 2020.

[3] A. Jaafer, G. Nilsson, and G. Como, "Data augmentation of imu signals and evaluation via a semi-supervised classification of driving behavior," *arXiv preprint arXiv:2006.09267*, 2020.

[4] E. Carvalho, B. V. Ferreira, J. Ferreira, C. De Souza, H. V. Carvalho, Y. Suhara, A. S. Pentland, and G. Pessin, "Exploiting the use of recurrent neural networks for driver behavior profiling," in *Proc. IJCNN*. IEEE, 2017, pp. 3016–3021.

[5] J. Yu, Z. Chen, Y. Zhu, Y. Chen, L. Kong, and M. Li, "Fine-grained abnormal driving behaviors detection and identification with smartphones," *IEEE transactions on mobile computing*, vol. 16, no. 8, pp. 2198–2212, 2016.

[6] M. Zhang, C. Chen, T. Wo, T. Xie, M. Z. A. Bhuiyan, and X. Lin, "Safedrive: online driving anomaly detection from large-scale vehicle data," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 2087–2096, 2017.

[7] M. R. Carlos, L. C. González, J. Wahlström, G. Ramírez, F. Martínez, and G. Runger, "How smartphone accelerometers reveal aggressive driving behavior?–the key is the representation," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[8] C. Chen, X. Zhao, Y. Yao, Y. Zhang, J. Rong, and X. Liu, "Driver's eco-driving behavior evaluation modeling based on driving events," *Journal of Advanced Transportation*, vol. 2018, 2018.

[9] V. C. Magaña and M. Muñoz-Organero, "Artemisa: A personal driving assistant for fuel saving," *IEEE Transactions on Mobile Computing*, vol. 15, no. 10, pp. 2437–2451, 2015.

[10] "Engineering safety with uber's real-time id check," https://eng.uber.com/real-time-id-check/, 2017.

[11] B. I. Kwak, J. Woo, and H. K. Kim, "Know your master: Driver profiling-based anti-theft method," in *Proc. PST*. IEEE, 2016, pp. 211–218.

[12] L. Guan, J. Xu, S. Wang, X. Xing, L. Lin, H. Huang, P. Liu, and W. Lee, "From physical to cyber: Escalating protection for personalized auto insurance," in *Proc. SenSys*, 2016, pp. 42–55.

[13] M. Martínez, J. Echanobe, and I. del Campo, "Driver identification and impostor detection based on driving behavior signals," in *Proc. ITSC*. IEEE, 2016, pp. 372–378.

[14] H. Dang and J. Fürnkranz, "Driver information embedding with siamese lstm networks," in *Proc. IV*. IEEE, 2019, pp. 935–940.

[15] M. Remeli, S. Lestyán, G. Acs, and G. Biczók, "Automatic driver identification from in-vehicle network logs," in *Proc. ITSC*. IEEE, 2019, pp. 1150–1157.

[16] A. Girma, X. Yan, and A. Homaifar, "Driver identification based on vehicle telematics data using lstm-recurrent neural network," in *Proc. ICTAI*. IEEE, 2019, pp. 894–902.

[17] D. Hallac, A. Sharang, R. Stahlmann, A. Lamprecht, M. Huber, M. Roehder, J. Leskovec *et al.*, "Driver identification using automobile sensor data from a single turn," in *Proc. ITSC*. IEEE, 2016, pp. 953–958.

[18] T. Wakita, K. Ozawa, C. Miyajima, K. Igarashi, K. Itou, K. Takeda, and F. Itakura, "Driver identification using driving behavior signals," *IEICE TRANSACTIONS on Information and Systems*, vol. 89, no. 3, pp. 1188–1194, 2006.

[19] 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," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 427–437, 2007.

[20] S. Lestyan, G. Acs, G. Biczók, and Z. Szalay, "Extracting vehicle sensor signals from can logs for driver re-identification," *arXiv preprint arXiv:1902.08956*, 2019.

[21] S. H. Sánchez, R. F. Pozo, and L. A. H. Gómez, "Deep neural networks for driver identification using accelerometer signals from smartphones," in *International Conference on Business Information Systems*. Springer, 2019, pp. 206–220.

[22] A. Chowdhury, T. Chakravarty, A. Ghose, T. Banerjee, and P. Balamuralidhar, "Investigations on driver unique identification from smartphone's gps data alone," *Journal of Advanced Transportation*, vol. 2018, 2018.

[23] D. Lyu and L. Xue, "Remote attacks on vehicles by exploiting vulnerable telematics," https://www.yumpu.com/en/document/read/56559249/remote-attacks-on-vehicles-by-exploiting-vulnerable-telematics, 2016.

[24] "Keen lab hackers managed to take control of tesla vehicles again," https://electrek.co/2017/07/28/tesla-hack-keen-lab/, 2019.

[25] "Hackers remotely kill a jeep on the highway," https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, 2019.

[26] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995.

[27] T. K. Chan, C. S. Chin, H. Chen, and X. Zhong, "A comprehensive review of driver behavior analysis utilizing smartphones," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[28] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[29] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[30] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *arXiv preprint arXiv:1905.03554*, 2019.

[31] C. K. Williams, "Prediction with gaussian processes: From linear regression to linear prediction and beyond," in *Learning in graphical models*. Springer, 1998, pp. 599–621.

[32] D. C. Montgomery and G. C. Runger, *Applied statistics and probability for engineers*. John Wiley and Sons, 2014.

[33] I. Masi, Y. Wu, T. Hassner, and P. Natarajan, "Deep face recognition: A survey," in *Proc. SIBGRAPI*. IEEE, 2018, pp. 471–478.

[34] M. L. Andersen, T. J. Stephens, and T. Lovell, "Wearable retina/iris scan authentication system," Aug. 29 2017, uS Patent 9,747,500.