ATARAXIS: A Deep Learning Approach for Hardwareless In-Vehicle Presence Detection

Magnus Oplenskedal
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
magnukop@ntnu.no

Amir Taherkordi
University of Oslo
Oslo, Norway
amirhost@ifi.uio.no

Peter Herrmann
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
herrmann@ntnu.no

***Abstract*—Accurately detecting the mobile contexts, in which public transport vehicles and their passengers operate, is key for future intelligent context-aware services in transportation systems. A prominent example is *in-vehicle presence detection* that, for instance, can be used to provide services such as automated ticketing, dynamic vehicle distribution, and live route optimization. To use such services in practice, the *in-vehicle presence detection* needs to be close to infallible. However, most existing solutions in this field suffer from low spatiotemporal accuracy. To address this challenge, in this paper we introduce ATARAXIS, an approach to *hardwareless* in-vehicle presence detection. In particular, we develop a deep convolutional neural network that can be trained to detect if a user is inside a public transportation vehicle such as a tram, subway, or bus from the raw sensor events generated by the sensors in a single ordinary smartphone. We show that this information can be used to infer the in-vehicle presence of users over time when combined with other sources such as the GPS trace of the user and that of the public transport vehicles. ATARAXIS has the capability to distinguish between the four user modes *driving a car*, *riding a bike*, *walking*, and *using public transport* with an accuracy of 98.69%. That is higher than existing techniques for transport mode detection. Since we are interested in applying ATARAXIS in practice, we also made experiments on the battery consumption and CPU overhead to be expected. The results show that the use of ATARAXIS in smartphones incurs a negligible computational overhead and power consumption, even though we base our approach on sensor data collection and a deep learning model.**

***Index Terms*—Mobile Context, In-Vehicle Presence Detection, Transport Mode Detection, Sensor Event Streams Analysis, Deep Learning, Intelligent Transportation**

I. INTRODUCTION

Many public transportation providers offer smartphone applications that provide services such as route planning, path finding, live updates regarding the transportation infrastructure, and ticket sales. In particular, in Northern Europe, the providers move rapidly away from legacy systems and routines such as ticket sales by vehicle operators or ticket-machines, the manual control of tickets, and ticket validation machines. A goal of shifting services to this technology is the reduction of infrastructure- and personnel-related costs. Another advantage is the improvement of the user experience and the removal of unnecessary steps for the passengers which makes public transport more attractive and is therefore a small step towards combating climate change.

The rapid development of mobile technologies like the Internet of Things (IoT) and cellular network infrastructures (*e.g.*, 5G) will lead to unprecedented opportunities making the next generation of public transportation even more appealing. To support such improvements, however, *mobile context* information [1] about passengers and the vehicles transporting them, have to be extremely accurate. If we know with a probability bordering on certainty whether a person is inside a vehicle, what type of vehicle it is, and more specifically, *in which* vehicle the person is travelling, we can use this information to provide context-aware services for the providers such as route optimization, dynamic vehicle distribution, and live passenger flow analysis.

Another type of context-aware service is provided within the so-called Be-In/Be-Out (BIBO) systems [2] which can automatically issue tickets based on the exact duration/distance traveled by users. Such a system alleviates the user from having in-depth knowledge regarding the fare structure and the ticketing system used.

Like the aforementioned services, a BIBO system can only work in practice if an excellent in-vehicle presence detection accuracy can be achieved. Therefore, we proposed two deep learning-based frameworks, called DEEPMATCH [3] and DEEPMATCH2 [4], respectively, that provide a good accuracy of up to 98.51%. More details about these approaches and other BIBO technology are provided in Sec. II. An important disadvantage of these approaches, however, is that they require extra hardware (*e.g.*, reference devices as fixed equipment installed in the bus to transmit BLE signals to passengers' smartphone) which imposes additional costs associated with implementation and maintenance.

This calls for a solution that works without using additional hardware. In this paper, we address this need by introducing our new approach ATARAXIS. In particular, we propose a deep learning model enabling *hardwareless* in-vehicle presence detection. It depends only on the presence of two platforms:

- Smartphones carried by passengers that offer certain sets of sensors like accelerometers, barometers, magnetometers, gyroscopes, and GPS receivers.
- An external service providing the spatiotemporal information—the real-time and historical locations of a certain vehicle. In Norway, the government-funded orga-

nization Entur has set strict regulations and requirements to what kind of hardware all vehicles operated by public transportation providers have to be equipped with. This entails hardware supporting the submission of real-time data to a publicly available API [5] based on the SIRI 2.0 standard [6]. One of the requirements is the real-time location of the vehicle, such that all vehicles have to be equipped with GPS.

ATARAXIS is a deep convolutional neural network trained on our own dataset collected by volunteers performing four different activities: *walking*, *bike*, *car* and *public transport*. The goal of the learning model is to recognize the activity of a user based on the sensor event streams of their smartphones. From this, we can recognize whether the user is believed to be present inside a public transportation vehicle. If a user is assumed to be in a public transport vehicle, we can then compare traces of the positions of users and vehicles in their vicinity using the Entur API [5] or a similar service, to find out in which public vehicle they are traveling. Based on this information, a BIBO system to ticket the users automatically can then be realized.

The main contribution of this paper is the introduction of our transport mode selection model while we will explain the alignment of traveller position traces with those of vehicles using the Entur service elsewhere. The rest of the paper is organized as follows: Our previous work, *i.e.*, the approaches DEEPMATCH and DEEPMATCH2, are sketched in Sect. II. In Sect. III we provide an in-depth presentation of our proposed approach followed by a report on experimental evaluation results in Sect. IV. In Sect. V, we present related works, to which our approach is compared before we conclude the paper with a discussion of future plans in Sect. VI.

II. DEEPMATCH AND DEEPMATCH2

Various approaches realizing BIBO systems were developed. They can be loosely divided into the two categories described in the following.

The *first* category of approaches use communication technologies such as Radio Frequency Identification (RFID) and Bluetooth Low Energy (BLE). These build up temporary connections between the user's mobile device and certain fixed hardware installed in the vehicles. Examples of such systems are SEAT [7] and EasyRide [8].

The *second* category relies on analyzing event streams produced by the sensors embedded in the smartphones carried by passengers. Modern smartphones are provided with numerous sensors such as gyroscopes, magnetometers, accelerometers, barometers and GPS. HybridBaro [9] and RideSense [10] are prominent solutions for sensor-based solution used to provide in-vehicle presence detection.

In our previous work [3], [4], we argue that the accuracy of these approaches is not good enough to use them in practice. To alleviate this issue, we proposed two deep learning-based frameworks, called DEEPMATCH [3] and DEEPMATCH2 [4]. In these approaches, each vehicle needs to be equipped with a stationary *Reference Device* (RefDev), embedded with the same sensors that can be typically found in modern smartphones. The sensor event streams generated by the on-board reference device and that of the passenger phones believed to be inside the vehicle are compared with one another using a special deep learning model built and trained for this specific purpose. If the deep learning model predicts that the two sensor streams match, the devices are assumed to be sensing from within the same vehicle inferring that the smartphone and, in consequence, its user are effectively riding in the vehicle.

Both deep learning models used to perform the in-vehicle presence detection in DEEPMATCH and DEEPMATCH2 consist of several distributed modules. Copies of one module, *i.e.*, the encoder part of a Stacked Convolutional Autoencoder, are embedded in apps installed in the smartphone carried by users and in the RefDev in the vehicle, respectively. This encoder module is used for sensor data compression and feature extraction. The most significant difference between both approaches is that the compression factor of DEEPMATCH2 is four times as high as the one used in DEEPMATCH.

The other part of each of the two models is a separate convolutional neural network that matches the data generated by the encoders in the RefDev and the passenger phone. It is supposed to run on an external server, *e.g.*, in a cloud or within the vehicle realizing an Edge Computing solution [11]. In the experiments for DEEPMATCH presented in [3], we achieved an in-vehicle presence prediction accuracy of 97.81% which, as we elaborated in the article, is sufficient to carry out passenger trip inference with a negligible error rate. Thanks to some further changes in the extended model DEEPMATCH2 [4], we reached a slightly improved accuracy of 98.51% in spite of the drastically smaller data sets transferred between the devices and the external server.

The disadvantage of DEEPMATCH, DEEPMATCH2, and other approaches requiring extra hardware like the RefDevs installed in the vehicles, is the increased costs associated with implementation and maintenance. Moreover, Public Transport Authorities (PTA) are often reluctant to use approaches that require extra hardware. Besides the additional costs, a reason for this is that the PTAs often do not operate the vehicles themselves but outsource their operation to subcontractors. Many of the vehicle operators provide service for several PTAs and want to have the freedom to easily move their vehicles between different areas. When moving a vehicle between areas, however, they have to exchange all special built in devices which can be quite laborious. That explains the reluctance of the PTAs and their subcontractors to additional hardware like the RefDevs. Therefore, we decided, not to stop with DEEPMATCH and DEEPMATCH2 but also to look at hardware-less in-vehicle presence detection technology, which led us to creating ATARAXIS presented in the following.

III. ATARAXIS

In this section, we provide an overview of our approach. Then we elaborate on the hardware and system settings on which our approach is built, followed by a presentation of how

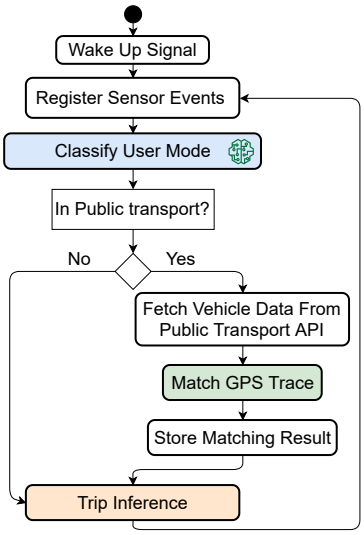

Fig. 1: Outline of the ATARAXIS algorithm

the data used to train our deep learning models were collected and transformed to suitable data sets. Thereafter, we introduce the design and architecture of our learning models, before the training regime of our experiments is described. Finally, we present the design rationale and experimental settings of ATARAXIS.

A. Overview

The core algorithm of our approach ATARAXIS is depicted in Fig. 1. A user carries a smartphone, on which an application featuring the ATARAXIS deep learning model is installed. In both Android and iOS, an application can be configured to run long-living background processes which can listen for certain *triggers* and then prompt other services in the application. In ATARAXIS, the background service listens for a *wake-up signal*, *e.g.*, a user movement over a longer distance like 100m. When the wake-up signal is sensed by the device, the application starts to register events from all relevant sensors. After a certain fixed period, the events sensed in this period are fed to the ATARAXIS deep learning model, which uses this *segment* of data to predict whether the user is travelling in a public transport vehicle, in a car, on a bike, or is walking.

If this model predicts that the phone and its user are, indeed, in a *public transport* vehicle, the application also fetches data from an external service like Entur [5] in Norway that provides real-time GPS data of all public transportation vehicles in the area. The application then filters out all transport vehicles which are further away than a certain threshold (*e.g.*, 20 meters) and tries to match the GPS traces of the remaining ones with the one of the user taken by the GPS receiver built into the smartphone. If one vehicle with a similar trace as that of the smartphone is detected, the user mode prediction is locally stored together with the GPS matching result for further processing. Thereafter, the algorithm starts a new loop.

The sequence of GPS matchings and user mode predictions generated over time are thereafter used in the *trip inference* step. Here, we can infer over several classifications in a sequence to find out, if the user was in a certain user mode over time. For that, we use a method introduced in [4] which allows this inference with a very high accuracy even if some classifications in the sequence are wrong.

In this method, we use the cumulative binomial probability described by the following formulas:

$$P(k \geq M) = \sum_{k=M}^{n} \binom{n}{k} p^k (1-p)^{n-k} \tag{1}$$

$$P(k \leq M) = \sum_{k=0}^{M} \binom{n}{k} p^k (1-p)^{n-k} \tag{2}$$

The variable n refers to the number of user mode classifications performed in a single sequence while k describes the number of classifications that are correctly predicted. Finally, in formula (1), M describes the minimum and in (2) the maximum number of user mode classifications, for a certain user mode um, that needs to be in the sequence in order to decide that the user is effectively in um for the whole sequence. M can be calculated using equation (3):

$$M = \left\lfloor (1 - fault\ \ tolerance) \cdot n + \frac{1}{2} \right\rfloor \tag{3}$$

Here, the *fault tolerance* describes the percentage of all predictions in the sequence that can be wrong whilst still correctly inferring that the user is in a particular user mode during this sequence.

To find an optimal value for M, where we achieve the least amount of false positives and false negative sequence inferences, we searched for a suitable fault tolerance value. Our tests described in [4] show that the most accurate results are achieved when a fault tolerance value of 40% is used.

In the rest of this section, we describe the ATARAXIS deep learning model, the central aspect of our approach, in detail.

B. Hardware Requirements and System Settings

As mentioned in the introduction, the ATARAXIS deep learning model requires events gathered by the typical sensors embedded in modern smartphones in order to be able to perform user mode classification. Through experiments, we found out that using events from the *accelerometer*, *magnetometer*, and *gyroscope* sensors provides the highest user mode detection accuracy, an accuracy of 98.69%. We chose to use the raw sensor output from these sensors instead of output from software-based sensors (such as linear acceleration), since software-based sensors are less commonly available in older phones.

To make correct predictions of the user activities, the classifier needs access to additional information about the users' movements. For instance, the raw hardware-based sensor events from the accelerometer embedded in smartphones outputs the acceleration for a coordinate system aligned with the chassis of the phone. In consequence, if users change the orientations of their devices, *e.g.*, by taking them out of their pocket, the acceleration values will change drastically. This

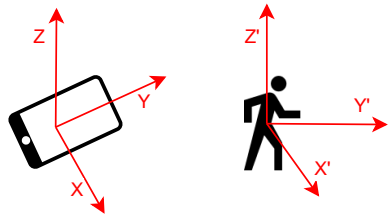

Fig. 2: The orientation of a smartphone vs. that of the carrying user

TABLE I: Example datapoints

Mode	Sensor	Value	Timestamp	ID	Device
Car	Acc. X	1.582	3366...	15	75i3...
Car	Acc. Y	0.285	3366...	15	75i3...
Car	Acc. Z	10.468	3366...	15	75i3...
Car	Mag. X	23.245	3366...	15	75i3...
Car	Mag. Y	9.875	3366...	15	75i3...
Car	Mag. Z	-4.875	3366...	15	75i3...
Car	Gyro.	0.019	3366...	15	75i3...

is quite disadvantageous, since the change in orientation is wrongly interpreted as a change in acceleration of the user.

To solve this issue, one typically transforms the collected accelerometer data represented along the X, Y, and Z axes of the phone's coordinate system to the X', Y', and Z' axes of the carrier's coordinate system utilizing the angular rotations performed around X, Y, and Z, see Fig. 2. In some works, this reorientation is performed by handcrafted algorithms, in contrast to learned by the model itself, before the data is fed to the classifier [12]. The reorientation is calculated from the accelerometer data collected on the smartphone's coordinates, in addition to the output from the *gravity* sensor, which is a software-based sensor derived from the accelerometer together with the magnetometer and the gyroscope. As previously mentioned, however, the accessibility of such software-based sensors varies. Therefore, we chose to use the raw, sensor outputs to train our deep learning model. This forces the model to learn to take the orientation of the device in relation to its user into consideration. As pointed out in our empirical studies discussed in Sect. IV, the learning models provided with input from these three sensors, *i.e.*, the accelerometer, magnetometer and gyroscope, are the ones providing the highest classification accuracy.

C. Data Collection and Dataset Creation

In the following, we describe first, how we collected and pre-processed sensor data. Thereafter, we discuss how the datasets, that we used to train and test the ATARAXIS model, were created from these sensor data.

1) Collection and Preprocessing: In previous work on in-vehicle presence detection, we developed the Android application *DataCollector* [4] which makes it easy for the carrier of a smartphone to collect, label, timestamp, and persist sensor events gathered by all the sensors embedded in this device. Each sensor event is stored as a datapoint in a local SQLite database. A datapoint contains the timestamp when the event was gathered as well as the type of sensor emitting the event, the value of the event, the device ID, and the collection ID, see Table I. The data acquired by the application can later be uploaded to a computer and digested by our custom made *Data Analysis Tools*. The user can start and stop the data collection anytime. Of course, that should happen when a certain travelling mode is entered and left, respectively. All events sensed during a data collection session are then marked with a unique ID for later processing.

TABLE II: An Example of interpolated data

Time	Mode	Acc.X.	Mag.X.	Gyro.	...
0 ms	Car	5.624	21.835	0.059	...
100 ms	Car	5.584	22.834	0.1356	...
200 ms	Car	5.530	24.547	0.077	...
300 ms	Car	5.673	25.125	0.080	...

When attaching listeners to sensors using the Android framework, the developer can configure a preferred data collection rate for each individual sensor. In reality, the rate at which the events are generated, deviates from the specified sampling rate with an average of one to two milliseconds due to limitations of the sensor control software. The consequence is that not all sensors emit their events exactly at the same time such that it is not possible to get the exact set of datapoints at a point in time. In our approach, however, we need one event for each sensor at the exact same timestamp. In order to achieve that in spite of this problem, we created a set of data analysis tools that is able to interpolate the gathered data. This interpolation is performed through the following four steps:

1) Define a global start time. For that, we use the timestamp of the first sensor event in a data collection.
2) Subtract the global start time from all timestamps of all data points to get a relative timestamp.
3) Interpolate the values for each sensor event with a fixed frequency.
4) Remove the original sensor events.

When this process is complete, we achieve a set of unified datapoints. For example, the datapoints from Table I are replaced by those in Table II.

2) Dataset Creation: The goal of dataset creation is to create suitable datasets to support the development of a *User Mode Classifier* able to distinguish between the four user activities walking, using public transport, driving in a car, and riding a bicycle, as depicted in Fig. 3.

By using our data analysis tools, we can create suitable datasets by configuring the following parameters:

- Number of datapoints in each sample,
- Sensors, the data of which shall be included in each sample,
- Number of samples in each dataset.

The tools guarantee that all datapoints included in a sample are from a sequential collection done in the same trip since all datapoints have to contain identical data collection IDs. Moreover, to find out which sensors contribute to good accuracy

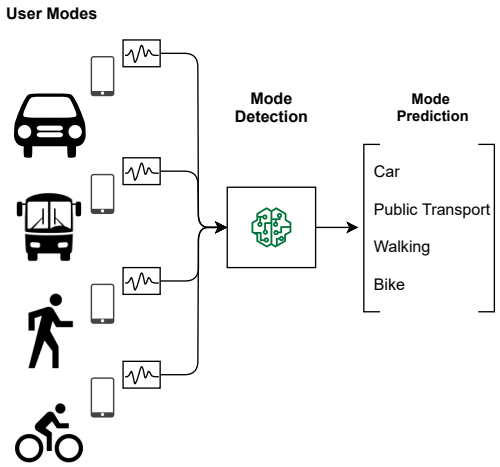

Fig. 3: The different user modes ATARAXIS is capable of recognizing

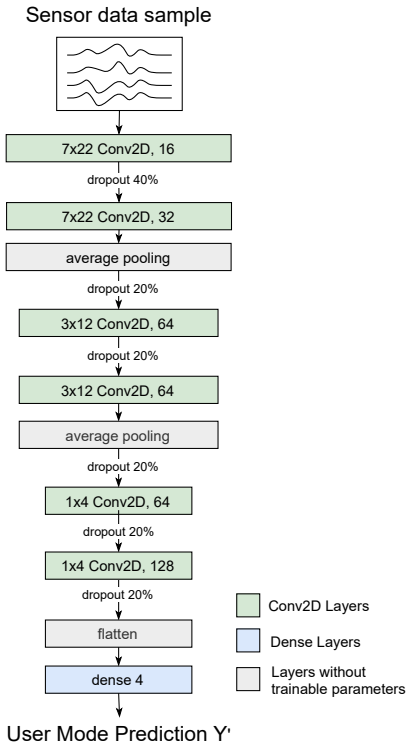

Fig. 4: ATARAXIS deep learning model architecture

results and which might be impedimental, we can also create samples containing events only from subsets of the sensors available. Additionally, to find the optimal length of the input to the model, we can build datasets consisting of samples of different lengths.

D. Design and Architecture of the Learning Model

Using various combinations of datasets created with the data analysis tools, we conducted a plethora of tests with different architectures. The deep learning model depicted in Fig. 4 rendered the best results. It is a Convolutional Neural Network (CNN, ConvNet). These networks are specially suited to detect and extract time-invariant features in sequential data, see [13], [14], [15], [16]. This is exactly what we need to solve our problem.

TABLE III: Example dataset sample used to train ATARAXIS

sensor	t0	t1	t2	t3
Acc. X	1.582	1.232	1.531	1.622
Acc. Y	0.285	0.182	0.335	0.233
Acc. Z	10.468	10.231	10.468	10.468
Mag. X	23.245	23.325	23.245	23.245
Mag. Y	9.875	9.575	9.999	9.234
Mag. Z	-4.875	-4.235	-4.536	-4.658
Gyro.	0.019	0.002	0.123	0.321

The green boxes in Fig. 4 represent two-dimensional convolutional layers. They contain filters that are trained to detect patterns in the input sensor event sequences unique to the various user modes contained in our datasets. Each green convolutional filter executes the three consecutive operations: convolution, rectified linear unit activation (ReLu), and batch normalization, see [17]. The inputs to the convolutional layers are two-dimensional. As pointed out by the example in Table III, the rows of the matrix refer to sensor inputs, while the columns point to the points of time for which the sensor values were interpolated. The texts in the green boxes describe the size of the convolutional filter matrices used in a layer. For instance, in the model's outermost layer, the number of rows of the convolutional matrix equals that of the input matrix and the number of columns is 22, *i.e.*, each filter processes 22 events from all sensors for each convolution. In each green box, the size of the filter for the corresponding layer is shown by the text on the left side, whilst the number at the right side refers to the number of filters, *e.g.*, 16 on the outermost layer.

The grey boxes represent layers without any trainable parameters. Two layers of this class are average pooling layers that perform dimensionality reduction by outputting the average for every two consecutive values of their input. Thus, the large number of dimensions of the input data can be gradually decreased. That is important to reduce the complexity of the overall machine learning model, *i.e.*, the number of trainable parameters. Reducing the complexity and size of the machine learning model also helps us to avoid overfitting the model to its training data.

Finally, the blue box at the bottom of the architecture represents a dense layer. It has four outputs equaling the number of classes, *i.e.*, user modes, that the overall model has to predict. Since this layer acts as the models output, the layer uses the softmax activation function to output a probability distribution over the four user modes.

As can be seen in Fig. 4, our model consists mainly of convolutional layers that are interspersed with two average pooling layers as well as one flatten layer and lastly a dense layer. We use a significant number of dropout layers, as shown in the figure, which helped the model in avoiding overfitting to the training data.

TABLE IV: ADAM optimizer settings

Parameter	Value
alpha (learning rate)	0.001
beta1	0.9
beta2	0.999
epsilon	1e-07

E. Model Training

In this subsection we will describe the configurations and hyperparameters used when training our ATARAXIS deep learning models. As described above, the datasets used as input to our machine learning model are formed as two-dimensional representations of the sensor events registered over a certain time period where the rows point at specific sensor events and the columns at points in time (see Table III). For each input sample in the training set, there is further a true label Y, representing the class, the sample belongs to, *i.e.*, one of the four user modes presented in Fig. 3. During training, the label Y of each sample is converted to a one-hot-encoding, *i.e.*, a vector of length equal to the number of classes in the training set. The output of the last layer of the deep learning model is a probability vector with the same length, produced by the softmax-activation function. The vector element with the largest value is the one predicted by the model as the correct user mode. In the following, we name the predicted output vector as Y'.

The goal of model training is to reduce the disagreement between the actual one-hot encoded label Y and the element of Y' with the largest value. We quantify this disagreement using *categorical cross-entropy*.

$$L = -\sum_{i=1}^{n} y_i \cdot log(y'_i)$$

Here, y'_i represents the i-th scalar value in the model output vector Y', while y_i is the corresponding target value in the one-hot encoded label Y.

Furthermore, Stochastic Gradient Descent in the form of the *ADAM optimizer* [18] is used to update the trainable parameters of the model using the disagreement found by the loss function described above. The ADAM optimizer was configured with setting depicted in Table IV.

F. Design Rationale behind the Deep Learning Model

When developing the deep learning model, we first started with a very small model architecture of only three layers with few small filters within each convolutional layer. We trained the iterations of our model on the created datasets following the description in Sect. III-C2 and evaluated the results using the well-known performance metrics precision, recall, and F1-score which are described more in-depth in Sect. IV-A.

Then we gradually and incrementally changed the hyperparameter configuration of the model and calculated the performance metrics for each new iteration. We also tried to use dense layers instead of conv layers as well as both, bigger and smaller filter sizes in addition to varying the number of filters within each conv layer. Furthermore, we gradually created a deeper neural network until we reached the architecture described in Fig. 4 that rendered better precision metrics values than all other iterations we tried.

All experiments were carried out on a desktop PC, equipped with an Intel i7 4.00GHz CPU, 32 GB memory and a Nvidia GTX 1080 GPU. The Data collector application was developed for the Android platform, while the Data Analysis tools were implemented in Python. The deep learning models were trained and evaluated using Google Tensorflow 2.0, version 2.0.0-rc0 [19].

IV. EVALUATION

In Sect. IV-A, we introduce the performance metrics, that we use to evaluate the ATARAXIS deep learning model in greater detail. Thereafter, we describe the data collected by our volunteers, how it was collected, and the various datasets created from it in Sect. IV-B. Finally, the battery consumption as well as the CPU usage and run-time overhead of our approach for the passenger smartphones are discussed in Sects. IV-C and IV-D, respectively.

A. Definitions and Metrics for Evaluation

In our evaluations, we use a number of definitions that are introduced as follows: A *positive sample* represents segments belonging to *Class 1*, *i.e.*, it covers a case in which a deep learning model predicts the correct user mode. In contrast, a *negative sample* is from *Class 0*, *i.e.*, it refers to a case when a wrong user mode is predicted. According to the common denominations in binary classification, we further define the following terms: *True Positive (TP)* as a correctly classified positive sample, *True Negative (TN)* as a correctly classified negative sample, *False Negative (FN)* as a positive sample that is wrongly classified as negative, and a *False Positive (FP)* as a negative sample which is falsely classified as positive.

Using these four terms, we can define the following three metrics that are helpful to evaluate the performance of a machine learning model:

- *Precision (PR)*: The ratio of correct positive predictions to the total number of predicted positive samples, *i.e.*, out of all samples classified as positive, how many belong to *Class 1*:

$$PR \triangleq \frac{TP}{TP + FP} \tag{4}$$

- *Recall (RE)*: The ratio of correct positive predictions to the total number of positive samples, *i.e.*, out of all available positive samples in the dataset, how many were correctly classified by the model:

$$RE \triangleq \frac{TP}{TP + FN} \tag{5}$$

- *F1-score (F1)*: The harmonic mean between precision and recall. The F1-score is useful in cases where the number of samples from each class is not distributed evenly:

$$F1 \triangleq 2 \cdot \frac{PR \cdot RE}{PR + RE} \tag{6}$$

TABLE V: Data sizes and smartphone positions collected for the different user modes

User Mode	Data Size	Smartphone Position
Public Transport	530 min	In hand, pocket, bag
Car	530 min	In hand, pocket, holder, bag
Bike	530 min	Pocket, holder, bag
Walking	530 min	In hand, Pocket, bag

TABLE VI: Performance metric values for the four user modes

Class	PR	Recall	F1-Score
Public Transport	0.9975	0.9897	0.9936
Car	0.9925	0.9826	0.9875
Bike	0.9759	0.9863	0.9810
Walking	0.9818	0.9890	0.9854
Macro Avg	**0.9870**	**0.9869**	**0.9869**

A fourth value is the accuracy that is more meaningful when the distribution between the classes of the dataset is relatively similar while the F1-score is better when they are imbalanced. Since there is significant imbalance in our data and, in addition, the F1-score is more sensitive to false positives and false negatives, which is particularly important to avoid in ATARAXIS, we use it instead of the accuracy metric in our evaluations, see also [20].

B. Datacollection and Dataset Creation

The data applied in this work was collected by volunteers, each carrying a smartphone whilst performing one of the four activities public transport, bike, car, or walking. The data was collected using our DataCollector application, described in Sect. III-C1. Copies of it were installed on the following seven Android devices: a Huawei Nexus 5X, two Huawei Nexus P6, a Samsung S8, a Sony Z3 Compact, a Google Pixel XL and a Google Pixel 3a. In Table V, the amount of data collected by our volunteers is listed. The Data Size column shows the total amount of data collected by all our volunteers for a given user mode, and the smartphone position describes the locations of the phones, while the data was collected[1].

The collection rate was set to 100 ms, *i.e.*, 10 Hz, for all sensors. This rate was selected since it is the fastest collection frequency offered by the slowest sensors in the smart devices, we used in our experiments. For each of the four user modes, we have a total of approximately 318,000 events per sensor.[2]

Altogether, we built six different datasets with the aim to find out which combination of input data provides the best result for our deep learning model. Based on the results from these experiments, we selected the best sensor modality combination, see Sect. IV-B1.

The next test step was to find an optimal sample size, *i.e.*, the duration of the sensor events segment used to build a sample of the datasets. Thus, we created three datasets to find out which sample lengths render the best performance results. This is discussed in Sect. IV-B2.

[1]In cars, the phone was only held by passengers but never by the driver.

[2]The datasets will be available via GitHub.

TABLE VII: Performance comparison sensor combinations

Model	PR	RE	F1
ATARAXIS 12.8 A	0.8919	0.8919	0.8919
ATARAXIS 12.8 B	0.7370	0.7368	0.7369
ATARAXIS 12.8 BA	0.9456	0.9451	0.9454
ATARAXIS 12.8 AM	0.9765	0.9765	0.9765
ATARAXIS 12.8 AMG	**0.9870**	**0.9869**	**0.9869**
ATARAXIS 12.8 AMGB	0.9836	0.9835	0.9835

In Table VI, the three performance metric values provided by our best performing model, *i.e.*, the one depicted in Fig. 4, are listed for the four user modes using the k-fold cross-validation algorithm [21]. In our experiments we set k to be 5, resulting in an distribution of 80% training samples and 20% testing samples of this algorithm.

1) Sensor Modalities: To make sure that we found a good trade off between the computational and battery consumption induced by our approach and the performance of the deep learning model, we created six datasets with sample lengths of 12.8 seconds consisting of the sensor combinations Accelerometer (A), Barometer (B), Barometer and Accelerometer (BA), Accelerometer and Magnetometer (AM), Accelerometer, Magnetometer, and Gyroscope (AMG) as well as Accelerometer, Magnetometer, Gyroscope, and Barometer (AMGB).

In Table VII, the results from training ATARAXIS on these datasets are presented. As can be seen from the performance metrics depicted in the table, the combination *AMG*, *i.e.*, the dataset built from samples consisting of sensor events from the accelerometer, magnetometer and gyroscope, rendered the best results. The F1-score of this modality is 0.9869.

We believe that the reason for achieving the best results by training the model on the *AMG* and the *AMGB* datasets is that data from the accelerometer, magnetometer, and gyroscope are required to *translate* the sensor events registered along the X, Y and Z axes of the device to movements along the X', Y' and $'$Z axes of the person carrying the device, see Sect. III-B. The slightly reduced performance of the model based on *AMGB* comes in our opinion from the additional noise that the barometer introduces to the input of the model.

Interestingly, this result is exactly opposite to our previous work on DEEPMATCH and DEEPMATCH2, where using only the sensor events generated by the Barometer rendered the best results. As we described in [3], the Barometer is great when the goal is to ignore the movements of the user, and rather capture the movements of the vehicle, the user is traveling in. This property makes it easy to compare the data of a smartphone with other devices present in the same vehicle, *i.e.*, the RefDev. When predicting the user mode, however, the Barometer does not seem to be a good source of input since here the movements done by both, the user and the vehicle are relevant characteristics in order to decide if the user travels in a public bus, a car, rides a bike, or walks.

2) Sample Size Experiments: The length of the input data used by the deep learning model in ATARAXIS is important for a couple of factors. Firstly, the size of the input of the

TABLE VIII: Performance comparison sample sizes

Sample Length	PR	RE	F1
3.2 sec	0.9506	0.9502	0.9504
6.4 sec	0.9845	0.9844	0.9844
12.8 sec	**0.9870**	**0.9869**	**0.9869**

model influences its performance, *i.e.*, the more data, a model can base its predictions on, the better will its accuracy be. This is true as long as the additional amount of data contains information still helpful to improve the pattern recognition. However, from a certain threshold on, the added data does not contain relevant new information, and it will not help to increase the sample size past that. From our experiments, the threshold is around 10 seconds.

Secondly, the size of the sample influences how often the model can predict the users mode anew. As discussed in Sect. III-A, the trip inference algorithm introduced in [4] works better if more single predictions can be considered. That also calls for using a sample size that is not too big. Lastly, running the machine learning model on the smartphone induces a computational overhead, that we want to keep minimally.

In Table VIII, the results from training the ATARAXIS model on three different sample lengths are presented. In our tests, the samples consist of datasets that are taken in time intervals of 3.2, 6.4, and 12.8 seconds, respectively. Since our data collection frequency was 10Hz, this results in sample size lengths of 32, 64 and 128 datasets, respectively. Applying datasets where the sample lengths are multiples of two, simplifies the creation of deep learning models using stacks of conv. and average pooling layers. The reason for this is that the average pooling operator divides the size of its input in half. Since we use more than one average pooling layer consecutively, it is therefore good to have an input size, the half of which is also a multiple of two.

We see that the variant trained on a dataset consisting of 12.8 second long samples yields the best performance with a F1-score of 0.9869. Nevertheless, the model trained on 6.4 second long samples performed nearly as well with an F1-score of 0.9844.

C. Power Consumption on Smartphones

As mentioned above, in ATARAXIS, the application responsible for inferring the in-vehicle presence of a user is expected to run on the user's smartphone. Therefore limiting the power consumption of ATARAXIS is crucial to guarantee a high degree of acceptance by the users.

In this subsection, we evaluate the power consumption of our approach in practice. In particular, we check the consumption of the three main sources of potential battery drain, namely, sensor data collection, prediction performed by the deep learning model, and communicating with the central server to fetch public transport vehicle data.

The tests were carried out using the three Android phones listed in Table IX. To consider age diversity, we used phones that are between two and seven years old. Moreover, an important factor on battery life is the temperature of the smartphone and its environment. To make sure our test environment simulates that of a typical public transport vehicle, we performed all tests indoors at an temperature of about 19° Celsius. Since ATARAXIS AMG with 12.8 seconds long samples yielded the best test results, we only considered this model for our power consumption runs. The tests were performed by collecting battery statistics from the phones using the *Batterystats* and *Battery Historian* tools provided by Android [22]. To quantify the aforementioned sources of power consumption, we constructed three scenarios for our experiments:

- *Data collection scenario*: The battery used by the three sensors continuously collecting data;
- *Prediction scenario*: The power consumed by the machine learning model processing data every 12.8 seconds;
- *Communication scenario*: The power consumption used to communicate with a server.

The results from our battery tests are also depicted in Table IX. They show clearly that, using just between 0.5 and 3 mAh, the deep learning model of ATARAXIS influences the overall battery consumption only marginally. Even for the oldest device used in the tests, the seven years old Sony Z3 compact, the total power consumption of all three scenarios was 25 mAH. This equals only to 0.96 % of the total battery capacity. As a result from our tests, we consider that our approach has only a negligible impact on the overall battery power consumption.

TABLE IX: Android phones used in the power consumption tests

Phone	Battery capacity [mAh]	Age [yrs]	Data coll. [mA]	Learning [mA]	Communication [mA]
Huawei P30 Pro	4200	2	12	1	1
Huawei Nexus 6P	3450	5	8	3	2
Sony Z3 compact	2600	7	24	0.5	0.5

TABLE X: Run Time and CPU overhead

Phone	CPU	Mean Run Time	Overhead
P30 Pro	2x 2.6 GHz, 2x 1.92 GHz, 4x 1.8 GHz Octa-Core	32 ms	2 %
P6	2.0 GHz + 1.55 GHz, 64-Bit Octa-Core	59 ms	5 %
Z3	2.5 GHz Quad-Core, Krait	48 ms	10.2 %

D. Computational Overhead on Smartphones

In addition to the battery consumption, the computational overhead induced by applications running passively on user smartphones is of utmost importance. In order to learn about overhead, we therefore ran tests registering the CPU usage and the mean run time of the machine learning model when the phone is used to process sensor data collected by its sensors.

In our tests, we used the same devices as in the battery tests. The results are depicted in Table X. We can see that

the CPU overhead and mean run-time is small at least for the newer models. This is particularly true since the computational overhead is only present during the execution of the deep learning model, *e.g.*, over the duration of 32ms for the P30 Pro. Furthermore, since the machine learning model is only executed every 12.8 seconds, the machine learning model should hardly impact any other applications or services that are executed on the phone in parallel.

V. Related Work

In Sec. II, we briefly discussed own as well other approaches that rely on additional equipment in order to realize in-vehicle presence detection. In this section, we mainly explore hardware-less solutions for in-vehicle presence detection. These types of solutions are often built upon Transportation Mode Detection (TMD) techniques. Although our work is more than just a TMD technique, we discuss existing TMD approaches in the following and compare them with ATARAXIS.

TMD has been addressed from different methodological angles and methods throughout the past two decades. Earlier techniques were limited to separating only motorized from non-motorized vehicles. In contrast, most more recent solutions aim to identify more than just these two basic transportation modes. Often, one distinguishes walking, bicycle, cars, and buses, where the main challenge is to separate cars from the rest of motorized transport [23].

From another perspective, existing techniques can be arranged in two categories, *i.e.*, location-based [24] and sensor-based [25] approaches. Location-based solutions often rely on location data provided by the GPS or wireless network [26]. The issue with these approaches is that they can induce high power consumption. Moreover, one may not always have sufficient cellular network accessibility, *e.g.*, when traveling in metros operating underground or on ferries [27]. In addition, the accuracy of detecting transport modes or vehicles such as walking, running, cycling, motorcycles, buses, and subways using the GPS-based techniques is reported to be just between 70% to 85%, see [28], [29].

Most works in this category rely on GPS data [30], while others combine GPS with the use of a Geographic Information System (GIS) platform or the map service APIs [31]. In another group of approaches, GPS, accelerometer and Bluetooth are combined with map-matching algorithms, see [32], [33], [34]. Finally, some approaches utilize the fusion of GPS and accelerometer data, see [33], [34].

Sensor-based transportation mode detection techniques can be used in a more energy-efficient and reliable manner than the location-based approaches [35]. The reason is that the data can be sampled from sensors at higher sampling frequencies with a considerably lower energy consumption.

Traditionally, rule-based or simple machine learning based approaches were applied for TMD. One of the works using shallow machine learning techniques and motion sensors on phones was proposed by Fang et al. in [36]. As described in [37], however, the accuracy rates decrease significantly with the increasing number of transportation classes. Therefore, more advanced machine learning mechanisms, in particular, deep learning approaches, have been introduced to enhance the classification success rate for models that shall distinguish between large numbers of different transportation modes. Fang et al. could increase the success rate from 83.57% to 95% using Deep Neural Networks (DNN) [38].

Convolutional Neural Networks (CNN) is another popular DNN-based approach, that was originally designed for image classification problems, but can be adapted to TMD. G. Yanyun et al. reported a success rate of 98 % for four classes using DNNs for TMD. In [39], the authors aim to classify seven classes using a CNN. They achieved 94.48 % success rate, but the high overlapping ratio of 87.5% is an issue in their model since it causes additional computational costs. T. Vu et al. introduced a Recurrent Neural Network (RNN) for TMD [40]. The authors used Vanilla RNN as well as some other variations of RNNs such as Control Gate-based Recurrent Neural Networks (CGRNN) and Long-Short Term Memory (LSTM). The most efficient one was CGRNN with an accuracy of 94.72% for a dataset consisting of 10 classes. In another LSTM-based approach, Asci et al. [41] use accelerometer, gyroscope, and magnetometer sensors as inputs into a recurrent neural network to classify ten different transport modes with an accuracy of 97.07%.

The important finding in the above approaches is that accurate transportation mode detection is complex, and usually a high accuracy entails either an unacceptable battery consumption and a high computational overhead, or requiring very long data sequences. In comparison to all these approaches, ATARAXIS provides excellent results as can be seen from Tables VI and VII. In addition, our approach uses relatively short sample sizes as depicted in Table VIII. Moreover, it induces a hardly noticeable power consumption as shown in Table IX and a low computational overhead as presented in Table X.

VI. Conclusion and Future Work

Providing accurate in-vehicle presence detection is an important step towards provisioning future context-aware services in public transport. In this paper, we addressed this challenge through ATARAXIS, a deep learning based approach to hardwareless in-vehicle presence detection. We presented the ATARAXIS deep learning model that can detect the four user modes walking, public transport, driving and riding a bike with an accuracy of 98.69 %. The model achieves this using only the raw sensor events generated by the accelerometer, magnetometer, and gyroscope sensors typically embedded in modern smartphones. We showed through empirical experiments that the combination of these three sensors and an input length of *12.8 seconds* yields the best results. Furthermore, we presented tests showing that the computational overhead and battery consumption of ATARAXIS is low, even for a 7-year old, used, Android phone.

In the future, we plan to implement the hardwareless in-vehicle presence detection, together with a public transportation provider in Norway. In particular, we will combine

the user mode predictor introduced in this paper with the open Entur API providing the real-time location of all public transportation vehicles in the country. Moreover, we plan to continue making improvements to the ATARAXIS deep learning model by further data collection and model optimizations. Here, we will also include the use of *electrical scooters* (e-scooters) in our datasets and model classification, since the use of these vehicles has exploded in Norway and in many other countries during the Covid-19 pandemic. Together with the localization systems of the e-scooters, our method will then allow their providers to automatically bill the users of these devices who can simply board and dismount them without having to think about ticketing.

REFERENCES

[1] S. K. *et al.*, "Orchestrator: An Active Resource Orchestration Framework for Mobile Context Monitoring in Sensor-rich Mobile Environments," in *IEEE Intr. Conf. on Pervasive Computing and Communications (PerCom)*. Mannheim, Germany: IEEE Computer, 2010.

[2] W. Narzt *et al.*, "Be-In/Be-Out with Bluetooth Low Energy: Implicit Ticketing for Public Transportation Systems," in *IEEE 18th Intr. Conf. on Intelligent Transportation Systems*. Las Palmas, Spain: IEEE, 2015.

[3] M. Oplenskedal, A. Taherkordi, and P. Herrmann, "DeepMatch: Deep Matching for In-Vehicle Presence Detection in Transportation," in *Proc. of 14th ACM Intr. Conf. on Distributed and Event-based Systems*, 2020.

[4] M. Oplenskedal, P. Herrmann, and A. Taherkordi, "DeepMatch2: A Comprehensive Deep Learning-based Approach for In-Vehicle Presence Detection," *Information Systems*, 2021, accepted.

[5] Entur, "Entur API," https://developer.entur.org, 2021, accessed: 2021-10-07.

[6] SIRI, "SIRI Standard," http://www.transmodel-cen.eu/standards/siri/, 2020, accessed: 2020-10-07.

[7] C. Sarkar *et al.*, "SEAT: Secure Energy-Efficient Automated Public Transport Ticketing System," *IEEE Trans. on Green Communications and Networking*, vol. 2, no. 1, 2018.

[8] T. Gyger and O. Desjeux, "EasyRide: Active Transponders for a Fare Collection System," *IEEE Micro*, vol. 21, no. 6, 2001.

[9] M. Won, A. Mishra, and S. H. Son, "HybridBaro: Mining Driving Routes Using Barometer Sensor of Smartphone," *IEEE Sensors Journal*, vol. 17, no. 19, 2017.

[10] R. Meng, D. W. Grömling, R. R. Choudhury, and S. Nelakuditi, "RideSense: Towards Ticketless Transportation," in *2016 IEEE Vehicular Networking Conf. (VNC)*. Columbus, OH, USA: IEEE, 2016.

[11] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its Role in the Internet of Things," in *1st Workshop on Mobile Cloud Computing (MCC)*. Helsinki, Finland: ACM, 2012.

[12] D.-N. Lu, D.-N. Nguyen, T.-H. Nguyen, and H.-N. Nguyen, "Vehicle mode and driving activity detection based on analyzing sensor data of smartphones," *Sensors*, vol. 18, no. 4, 2018.

[13] N. S. Madiraju, S. M. Sadat, D. Fisher, and H. Karimabadi, "Deep Temporal Clustering: Fully Unsupervised Learning of Time-domain Features," *arXiv*, vol. cs, no. arXiv:1802.01059, 2018.

[14] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep Learning for Sensor-based Activity Recognition: A Survey," *Pattern Recognition Letters*, vol. 19, pp. 3–11, 2017.

[15] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked Convolutional Auto-encoders for Hierarchical Feature Extraction," in *International Conf. on Artificial Neural Networks (ICANN)*, ser. LNCS 6791. Espoo, Finland: Springer-Verlag, 2011.

[16] A. Supratak, H. Dong, C. Wu, and Y. Guo, "DeepSleepNet: A Model for Automatic Sleep Stage Scoring based on Raw Single-channel EEG," *IEEE Trans. on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 11, 2017.

[17] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv*, vol. cs.LG, no. arXiv:1502.03167, 2015.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] Tensorflow, "Tensorflow 2.0 RC Tutorials," https://www.tensorflow.org/beta/, 2019, accessed: 2019-10-23.

[20] P. Huilgol, "Accuracy vs. F1-Score," https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2, 2019, accessed: 2021-10-14.

[21] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the royal statistical society: Series B (Methodological)*, vol. 36, no. 2, 1974.

[22] B. Historian, "Battery Stats and Historian," https://developer.android.com/studio/profile/battery-historian, 2021, accessed: 2021-10-10.

[23] A. Efthymiou *et al.*, "Transportation mode detection from low-power smartphone sensors using tree-based ensembles," *Journal of Big Data Analytics in Transportation*, vol. 1, no. 1, 2019.

[24] T. Sohn *et al.*, "Mobility detection using everyday gsm traces," in *Intr. Conf. on Ubiquitous Computing*. Springer, 2006.

[25] S. Wang, C. Chen, and J. Ma, "Accelerometer based transportation mode recognition on mobile phones," in *2010 Asia-Pacific Conf. on Wearable Computing Systems*. IEEE, 2010.

[26] A. Jahangiri and H. A. Rakha, "Applying machine learning techniques to transportation mode recognition using mobile phone sensor data," *IEEE trans. on intelligent transportation systems*, vol. 16, no. 5, 2015.

[27] Z. A. Lari and A. Golroo, "Automated transportation mode detection using smart phone applications via machine learning: Case study mega city of tehran," in *Proc. of the Transportation Research Board 94th Annual Meeting, Washington, DC, USA*, 2015.

[28] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding mobility based on gps data," in *Proc. of the 10th international conf. on Ubiquitous computing*, 2008.

[29] T. Feng and H. J. Timmermans, "Comparison of advanced imputation algorithms for detection of transportation mode and activity episode using gps data," *Transportation Planning and Technology*, vol. 39, no. 2, 2016.

[30] P. Sadeghian, J. Håkansson, and X. Zhao, "Review and evaluation of methods in transport mode detection based on gps tracking data," *Journal of Traffic and Transportation Engineering*, 2021.

[31] L. Zhu and J. D. Gonder, "A driving cycle detection approach using map service api," *Transportation Research Part C: Emerging Technologies*, vol. 92, 2018.

[32] J. Chen and M. Bierlaire, "Probabilistic multimodal map matching with rich smartphone data," *Journal of Intelligent Transportation Systems*, vol. 19, no. 2, 2015.

[33] B. Martin *et al.*, "Methods for real-time prediction of the mode of travel using smartphone-based gps and accelerometer data," *Sensors*, vol. 17, no. 9, 2017.

[34] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," *ACM Trans. on Sensor Networks (TOSN)*, vol. 6, no. 2, 2010.

[35] S. Hemminki, P. Nurmi, and S. Tarkoma, "Accelerometer-based transportation mode detection on smartphones," in *Proc. of the 11th ACM conf. on embedded networked sensor systems*, 2013.

[36] S.-H. Fang, H.-H. Liao, Y.-X. Fei, K.-H. Chen, J.-W. Huang, Y.-D. Lu, and Y. Tsao, "Transportation modes classification using sensors on smartphones," *Sensors*, vol. 16, no. 8, 2016.

[37] M. Nikolic and M. Bierlaire, "Review of transportation mode detection approaches based on smartphone data," in *17th Swiss Transport Research Conf.*, 2017.

[38] G. Yanyun, Z. Fang, C. Shaomeng, and L. Haiyong, "A convolutional neural networks based transportation mode identification algorithm," in *2017 Intr. Conf. on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2017.

[39] X. Liang and G. Wang, "A convolutional neural network for transportation mode detection based on smartphone platform," in *2017 IEEE 14th intr. conf. on mobile Ad Hoc and sensor systems (MASS)*. IEEE, 2017.

[40] T. H. Vu, L. Dung, and J.-C. Wang, "Transportation mode detection on mobile devices using recurrent nets," in *Proc. of the 24th ACM intr. conf. on Multimedia*, 2016.

[41] G. Asci and M. A. Guvensan, "A novel input set for lstm-based transport mode detection," in *2019 IEEE Intr. Conf. on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019.