Doctoral theses at NTNU, 2022:234

Magnus Karsten Oplenskedal

Realizing Context-Aware Services through Intelligent Mobile Data Analysis

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of Science and Technology

Magnus Karsten Oplenskedal

Realizing Context-Aware Services through Intelligent Mobile Data Analysis

Thesis for the Degree of Philosophiae Doctor

Trondheim, January 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

ISBN 978-82-326-5987-6 (printed ver.)
ISBN 978-82-326-6899-1 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2022:234

Printed by NTNU Grafisk senter

Realizing Context-Aware Services through Intelligent Mobile Data Analysis

Thesis submitted for the degree of Philosophiae Doctor
Department of Information Security and Communication Technology

Norwegian University of Science and Technology, 2022

Applicant: Magnus Oplenskedal
Supervisors: Peter Herrmann and Amir Taherkordi

Abstract

In recent years there has been a rapid development of mobile technologies, Internet of Things (IoT) and cellular network infrastructures. In combination with the fast evolution of data analysis, and particularly within machine learning, this has led to new unprecedented opportunities for building smart environments. Central to smart environments lie *context-aware* systems. These systems *sense* the users environment, analyse the sensed data to deduce new insights and, from this, provide content, experiences, help and suggestions to the users. Accessing and providing such services have never been easier, since more than 6.5 billion people in the world own smartphones, a platform prime for supporting *context-aware services*.

A topical form of context-aware services, when using smartphones as sensing equipment, are so-called location-aware services. Their particular characteristic is the utilization of the user's location to provide useful suggestions related to their surroundings. Location-aware services, in particular, services used in retail (shopping) and public transport, are the main focus of this thesis. That are two domains, people generally interact with on a daily basis.

In retail and shopping, so-called mobile recommender systems (MRS) are used to perform recommendations such as points of interest or products in the user's proximity. Crucial to these services are the accurate real-time locations of both, the user and the products they suggest. The location of the user can be provided using a real-time location system (RTLS) tracking the user's smartphone. However, an accurate, efficient solution to the location of the product is not as readily available.

In public transport on the other hand, the detection of the *mobile context* of vehicles and their passengers is key to realize intelligent transportation systems (ITS). A topical example of this is the in-vehicle presence of a passenger, essential for context-aware services such as automated-ticketing. In order for these systems to work, the accuracy of the solution needs to be incredibly high, something, the state-of-the-art solutions available today still lack.

To support these next-generation location-aware services, advanced *context reasoning* techniques, *i.e.*, algorithms extracting useful information from the data sensed from the user's environment, are of utmost importance. Machine learning algorithms form a very promising category of such context-reasoning techniques. In consequence, most of the work done in this Ph.D. project centers on them. Particularly, machine learning technology is used in this thesis to address challenges regarding context-aware services within the above mentioned fields retail and public transport. Altogether, we provided the following four contributions:

The *first* contribution introduced in this thesis, is an automatic product localisation algorithm. The product locator proposed in this thesis infers the location of the products in a store by accumulating the locations at which customers stop when picking up products as well as the list of purchased products. A simulation-based environment shows that 99.9% out of $8,000$ products in a typical large Norwegian grocery store can be correctly located by aggregating data from customers over a 12-day period.

The *second* contribution presented in this thesis is *DeepMatch*, a highly accurate in-vehicle presence detection algorithm.

The approach is utilizing the smartphone of a passenger to analyze and match the sensor event streams of the device against the streams of sensors embedded in an on-board reference unit installed in public transportation vehicles. The matching is facilitated by a new deep learning model employed in a distributed fashion, where the feature extraction and dimensionality reduction is offloaded to the smartphones and the reference unit, while the matching is performed on a remote server. The approach achieved an in-vehicle prediction accuracy of 0.9781 on a dataset consisting of real data gathered by volunteers.

The *third* contribution builds on the second one. It consists of the method *DeepMatch2*, the successor of *Deepmatch*, that increases its accuracy from 0.9781 to 0.9851. In addition, the algorithm improves its efficiency, effectively reducing the amount of data needed by the model by a factor of four. Furthermore, we propose a *travelling user inference system* based on *DeepMatch2* with the ability to infer if and for which period of time a passenger makes a trip in a public transport vehicle with a very low error rate.

The *fourth* contribution of this thesis is *Ataraxis*, a solution to hardware-less in-vehicle presence prediction. Through the collaboration with Public Transportation Authorities, we learned that some PTAs do not have the control to decide autonomously about the hardware that is installed in the vehicles they use. Therefore, the hardware-based solution proposed in *DeepMatch* and *DeepMatch2*, *i.e.*, the installation of additional hardware in the vehicles, is not always suitable. *Ataraxis* addresses this challenge. A deep

convolutional neural network to detect the transport mode of users from the sensor events generated by ordinary smartphones was developed. The user mode is used in combination with a GPS trace of the user and nearby public transport vehicles in order to infer the in-vehicle presence of the user. The deep learning model created for Ataraxis achieved an F1 Score of 98.69% when classifying the four user modes *driving a car, riding a bike walking and using public transport*.

To summarize, in this thesis, we contribute applied research through several learning algorithms, system designs, and software solutions in order to enhance and improve the intelligence and quality of location-based context-aware services within retail and public transport. Furthermore, we show through extensive empirical experiments that the proposed approaches can be used in practice without negatively impacting the users smartphones. It answered the main research objective as well as provided several business critical patents to the industrial partner.

Preface

This dissertation is submitted in fulfillment of the requirements for the degree of Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The presented work was carried out at the Department of Information Security and Communication Technology (IIK), Trondheim, conducted under the supervision of Professor Peter Herrmann and Associate Professor Amirhosein Taherkordi. The thesis was supported by the Norwegian Research Council within the Industrial Ph.D. scheme under Grant 276259 (MobiTrack project) and was done in a close collaboration between the industrial partner of the project, Forkbeard Techonologies[1] and the academic partner NTNU.

[1] https://www.forkbeardtech.com/

Acknowledgements

First of all, with a deep sense of gratitude, I thank my supervisor Professor Peter Herrmann, for the opportunity to pursue a PhD career under his supervision. Throughout the last six years, during both my masters degree and this PhD, Peter has been a great source of inspiration and a pillar to rely on. His ability to quickly attain new knowledge, to tackle any challenge head on and his dedicated support and guidance has helped me through any difficulties during my work on this thesis.

I am also extremely grateful to my co-supervisor, associate professor Amir Taherkordi. Amir was responsible for arranging this PhD and with his sharp and clever mind, his expertise in the topics covered by this PhD and rock solid understanding of the academic world made this journey a pleasure. With his unlimited ability to support when needed, he went above and beyond any expectations one could have for their co-supervisor.

I would also like to thank all my colleagues at IIK, especially Mona for being the omniscient being of administrative tasks at the faculty and to Mister Ergys for making my days at the university a pleasure and for being a sparring partner with me late at night when practicing for exams.

Additionally I would like to express my thanks towards colleagues at Forkbeard Technologies, especially Wilfred for his expertise and technical assistance and for making this thesis become reality, and to Endre for making my days at the office a pleasure through insightful discussions and interesting coffee talks.

I cannot even begin to express my thanks to my partner Marte, who have been nothing but patient and supportive to me throughout all these years, this journey would not have been possible without her.

I would also like to give my most profound thanks to my family for their never ending support and belief in me.

Finally, I would like to dedicate this work to my late mother. She was the main driving force behind my academic interests and career, and always believed that nothing was impossible as long as I gave it a shot.

Contents

List of Figures

List of Tables

Acronyms

AE	Autoencoder
ANN	Artificial Neural Networks
BDS	BeiDou Navigation Satellite Systems
BiBo	Be-In/Be-Out
BLE	Bluetooth Low Energy
CAE	Convolutional Autoencoder
CARS	Context-Aware Recommender System
CNN	Convolutional Neural Network
DNN	Dense Neural Network
DTW	Dynamic Time Warping
GPS	Global Positioning System
HAR	Human Activity Recognition
HMM	Hidden Markov Models
IoT	Internet of Things
ITS	Intelligent Transportation Systems
MEMS	Micro-Electronics-Mechanical Systems
ML	Machine Learning
MRS	Mobile Recommender Systems
NLP	Natural Language Processing

Acronyms

PTA	Public Transportation Authorities
RFID	Radio Frequency Identification
RNN	Recurrent Neural Network
RSS	Received Signal Strength
RTLS	Real-Time Location System
SNN	Siamese Neural Network
SVM	Support Vector Machines
TMD	Transportation Mode Detection

Part I

Thesis Summary

Chapter 1

Introduction

1.1 Motivation and Research Questions

Today, the convergence of research and development within mobile technologies, IoT, cellular network infrastrcutures and advanced data analyses tools sets the stage for new and improved *context-aware services*.

These services collect data from the users environment, *i.e.*, *context*, and use it to anticipate the users requirements in order to offer enriched, context-aware and usable content/functions and experiences. They can be found on a wide variety of platforms, such as on desktop, mobile, web and through IoT based applications. Moreover, development and research of such services is receiving intensive attentions in a plethora of domains such as healthcare, retail, transportation, vacation related service and so on. Context aware services are often categorized by their main source of information, and type of assistance/recommendation they provide. Prominent examples of categories are *social-based*, *shopping-based*, and *location-based* applications.

In this work, we focus on the last one, *i.e.*, location-based context-aware services. The application domains of our research is mainly within *Retail* and *Public Transportation*. This is a result of close ties between the industrial partner and relevant organisations and corporations within these domains. When working on context-aware services, access to relevant data and insights into the domains where they are used, is of the utmost importance. Additionally, as we will show throughout this work, existing context-aware services in these domains are lacking, and, the potential for useful services and interesting research within them is high.

One key source of information used by location-based services is the real-time location of the user (*i.e.*, the real-time location of the user's smart phone). Until recently, the adoption of this approach in indoor scenarios

has been hampered by the lack of an accurate indoor positioning system for smartphones. Now such systems are developed making it possible to track the location of a phone with centimeter level precision. This allows us to engineer new types of context-aware systems, which was previously unattainable. One of these new in door real-time location systems has been created by Forkbeard Technologies. It realizes this much coveted level of accuracy using a low cost infrastructure and full backwards compatibility with billions of smartphone devices running iOS or Android.

In parallel to the development of better sensing equipment, improving the acquisition of the contextual information of the user, there has been an almost revolutionary improvement in the field of *context reasoning*, *i.e.*, in the analysis of the contextual data. A prominent embodiment of this is the current, expansive research and development of learning algorithms, specifically within the field of Artificial Neural Networks.

1.1.1 Primary Research Objective

Considering the motivations elaborated above, the following primary scientific research objective was proposed:

To conduct applied research and develop design concepts, software frameworks, and algorithms to utilize the real-time position, and sensor data from smartphones in order to improve the intelligence and quality of context-aware services that are directly or indirectly location-dependent.

In order to tackle this research objective, within the two domains mentioned above *Retail* and *Public Transportation*, six research questions were posed.

In the following, these research questions will be presented and the background for each question will be discussed.

1.1.2 Retail Stores

Through a thorough related work study, and close collaborations with industry stakeholders, we discovered that the field of *Mobile Recommender Systems* was worth investigating. Mobile recommender systems are a type of context-aware systems which use the location of the user in order to provide it with suggestions such as product recommendations. Moreover, two important inputs for these systems are the real-time locations of the user and the location of the products in the stores. While the first input can be provided

by Real-Time Location Systems (RTLS), the second, *i.e.*, the location of the products, are not readily available.

With this in mind, the first two research questions of this thesis are formed:

RQ1: *Can the data captured by an RTLS of customers moving through stores, together with their shopping receipt, be used to infer the location of the products?*

There are typically two categories of solutions proposed to this problem: The first is to attach *tags* transmitting some signal to the products and receiver technology througout the store in order to track their exact position [73]. However, this approach is not feasible for *e.g.*, grocery stores with a vast amount of products, including several product categories, such as vegetables and fruit, that are often unlabeled. The solutions in the second category suggest registering and updating the position of the products manually [18, 104]. However, this requires a lot of human effort, and especially in cases that products are regularly relocated.

Thus, in order to support Mobile Recommender Systems, an automated, reliable, efficient way to localize the products in the store is needed. Given that a store is equipped with a RTLS for localizing customers and a cashier system, providing an overview over the products purchased by each individual customer, we wanted to find out if the data generated by these systems could be used as a source of contextual information in order to infer the location of the products. To answer this, it was necessary to research ways of finding correlations between where the customers moved, what products they purchased, and the location of these products.

RQ2: *How can a product localization algorithm quickly update its knowledge after a product it has already located is moved?*

When accumulating large amounts of data from customers in order to localize the products in the store, it might become biased towards established products located at an *established* location, *e.g.*, if the first 800 customers found a product p in location x, the algorithm should establish the location of the product to be in location x. However, if the product is moved, it should not afford data from more than 800 customers to update the location of the product. The product localization algorithm needs to have some inbuilt trade-off mechanism in order to correctly localize a product from accumulated data, but at the same time be able to overcome the bias, this accumulation might entail.

1.1.3 Public Transport

A key feature of modern public transportation systems is the accurate detection of the mobile context of transport vehicles and their passengers. A prominent example is automatic in-vehicle presence detection which allows, *e.g.*, passenger flow analysis, dynamic vehicle allocation and automated ticketing of passengers. Most existing in-vehicle presence solutions are based on either using active RFID or Bluetooth Low Energy (BLE) technology respectively, or mobile data analysis. Unfortunately, the spatial and/or temporal accuracy of these systems are too low to be usable in practice.

These existing systems do not utilize that today most passengers use smartphones equipped with various types of sensors and high processing power. This potential of the smartphones for in-vehicle presence detection has not yet been thoroughly investigated, and has been central to the work presented in this thesis.

To provide the next-generation of context-aware services within public transport, a solution to highly accurate in-vehicle presence detection is needed. To this end, the following four research questions are posed:

RQ3: *Can the sensors of modern smartphones be used to provide a highly accurate in-vehicle presence detection system?*

As mentioned above, early approaches use communication technologies such as RFID and BLE to perform in-vehicle presence detection [36, 37, 47]. However, these approaches yield a too low accuracy to be used in practice. Other approaches, such as the ones proposed in [100] and [39], utilize sensor events from smartphones to infer in-vehicle presence. Even though these approaches promise a better accuracy than the ones based on communication technologies, they are still too low to be used in practice. For more details on the related work, see Section 3.2.

Hence, we posed the question on how sensor data generated by smartphones can be utilized in order to provide a sufficiently high in-vehicle presence detection accuracy.

RQ4: *How can a highly accurate in-vehicle presence detection algorithm be built whilst at the same time minimize the resource demands of the algorithm on the user's smartphone (*e.g.*, computation, battery consumption, data transmission)?*

In the process of answering RQ3, it is important to keep the potential trade-off between power consumption, computational overhead and accuracy in mind. For instance, an approach providing a high in-vehicle presence detection accuracy, while at the same time draining the smartphones battery or requiring all the computational capabilities of the users smartphone, will most likely not be accepted by the smartphone user. Context-aware services

are often expected to be running over extended periods in the background on users smartphones and thus it is very important that the context-aware service is not interfering with other applications. Furthermore, as described by the authors of [44], smartphone embedded sensors are found to be *highly power hungry*, *i.e.*, they are a major source of battery power consumption.

It is therefore of the utmost importance to research and perform empirical studies on the minimal number of sensors needed, the optimal *awake* time and collection frequency of these sensors and to find a suitable trade-off between off-loading communicating to other devices, while at the same time making sure the communication is not hurting more than helping.

RQ5: *How can a highly accurate in-vehicle presence detection algorithm be used to infer the time period over which a passenger is traveling in a vehicle?*

When using in-vehicle presence detection to provide, *e.g.*, automated ticketing, it is important for the algorithm to infer the duration of the passenger's trip to be flexible. For instance, one passenger might take a trip lasting 50 seconds, while another traveler spends 2 hours in the vehicle. In general, an in-vehicle presence detection algorithm will typically require some minimum amount of data in order to be accurate, *e.g.*, 10 seconds of sensor data. Then, if the duration of a passenger's trip is 10 minutes, the algorithm can carry out 60 independent in-vehicle predictions. Even with an in-vehicle presence detection accuracy of 98%, the likelihood that all these predictions are correct is only around 30%. Thus, an algorithm is needed, that can infer passenger trips with varying lengths from sequences of in-vehicle presence predictions with a high degree of precision. This algorithm should be able to tolerate occasional matching errors.

RQ6: *How can the issue of in-vehicle presence detection be solved without requiring the installation of additional hardware in the vehicles of the Public Transportation Authorities?*

Most state-of-the art approaches to in-vehicle presence detection rely on installing hardware such as BLE-transmitters [79], RFID receivers [36,37] or comparing features extracted from smartphone sensor data against the corresponding features extracted from a reference unit installed in the vehicle [62].

However, in some cases, it might not be feasible to install such hardware in the vehicles. Conversations with various Public Transportation Authorities (PTA) revealed that some PTAs, whilst providing services such as ticketing passengers, are not in control of the hardware-installations in the operated vehicles. Additionally, a solution not requiring the installation of hardware can potentially improve the maintainability and scalability of the service.

Solutions to this problem are often based on so-called Transportation Mode Detection (TMD) techniques. In TMD the goal is to recognize the

type of transportation mode performed by the carrier of the smartphone, often including modes such as *Walking, Riding a bike, driving a car and riding in a bus, train or tram.* Early approaches to this problem were solved by using the GPS or wireless network [87, 105]. However, the accuracy of these approaches were generally to low, 70% to 85% [26, 105], and the power consumption too high. Later approaches using sensor-based transportation mode detection were shown to be more reliable and energy-efficient [21, 38, 55, 75]. However, these sensor-based approaches with an high accuracy tend to either impose unacceptably high power consumption and computational overhead, or require very long sequences of data.

These insights have motivated us to conduct research on ways of providing in-vehicle detection, using sensors and computational capabilities of the smartphones carried by the passengers, whilst still providing the necessary accuracy as stated in RQ3, and not making the approach impracticable to be used in practice as stated in RQ4.

1.2 Research Methodology

In this section, the research methodology used as an inspiration for the work in this Ph.D. thesis is described. The methodology follows the six steps of *Design Science Research* described by Peffers, Ken, et al. in [69]:

1. **Identification** of the research problem,
2. **Definition** of research objective,
3. **Design and development** of artefacts,
4. **Demonstration** by solving problems using the artefacts,
5. **Evaluation** of the demonstrations,
6. **Communication** of the problem, artefacts and their ability to solve the problem.

As described in Section 1.1, this thesis was conducted as a joint project between the industrial partner Forkbeard Technologies, and the research partner NTNU. Thus, the work in this project was started by the *identification* and *definition* of the main goals of both stakeholders. These goals were condensed into the primary scientific objective described in 1.1. Furthermore, through brainstorming and discussions with the project stakeholders, and through a thorough study of related works, the research questions in

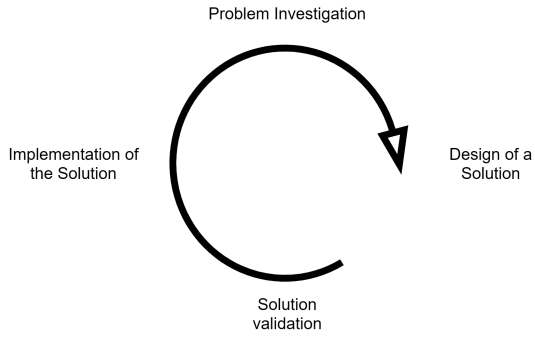

Figure 1.1: The regulative cycle

Section 1.1 were identified. Throughout the project, the *regulative cycle*, described in [99], has been roughly followed. The circle typically starts by investigating a practical problem, which can itself be the result of solving a previous practical problem. Thereafter, a solution is designed for the problem, followed by an evaluation of this design by, *e.g.*, developing and testing a proof-of-concept. Based on the evaluation results, the design is implemented and can thereafter trigger a new round through the regulative cycle.

In the first step, the *problem investigation*, the research object posed in Section 1.1 was investigated and decomposed into six research questions. These questions were analyzed and prioritized together with the projects stakeholders. From this prioritized list, the solution design was performed through the design and development of solutions and artifacts such as theoretical models, algorithms, and software design models. In the validation phase, various validation strategies were used depending on the problem, questions, the goals of the stakeholders and the availability of real data. However, the main concern of the evaluations was always to answer whether the suggested solutions would bring the stakeholders closer to their goals. For evaluation, frameworks such as the Python frameworks, Tensorflow, Keras, Pandas and Numpy as well as proprietary simulation and analysis tools were used. After a complete turn of the regulative cycle, the knowledge gained from the evaluations and the input from the stakeholders were used to start a new iteration of the cycle.

Furthermore, throughout this work, the design solutions, their implementations and evaluation results were communicated through paper and journal submissions and finally presented in this thesis.

1.3 Thesis Structure

This thesis was conducted as a collection of papers in line with NTNU's regulations for the doctoral degree. The thesis is organized in two main parts:

Part 1 presents a summary of the thesis, consisting of the following chapters:

- *Chapter 1 Introduction:* Establishing the motivation for the work in this thesis, the proposed research questions, and the applied research methodology.
- *Chapter 2 Background:* Giving the theoretical background for understanding the research done and contributions provided by this thesis.
- *Chapter 3 Related Work:* Presenting the state-of-the art literature and works related to the research objective, research questions and contributions of this thesis.
- *Chapter 4 Results and Implications:* Presenting the paper contributions of this thesis and concluding the thesis with a discussion on the results followed by suggestions for future work.

Part 2 consists of the four papers published throughout this work, representing the contributions of this thesis.

Chapter 2

Background

In this chapter, the theoretical background for the work in this thesis is presented. The chapter starts with a short introduction to context-aware computing, followed by a thorough description of *location-based* context-aware services. Thereafter, context reasoning techniques are presented, and in particular *learning-based* strategies are discussed in detail. Finally, the chapter is concluded with a presentation of relevant related works together with a discussion of identified, open challenges related to the research questions presented in Chapter 1.

2.1 Context-Aware Computing

Today, context-aware computing can be found nearly everywhere. It is used in innumerable situations by desktop, web, and mobile applications, by IoT-based applications, in autonomous driving systems etc. In general, the goal of context-aware computing is to discover useful *facts* about the user's current environment without distracting or interacting with them. Then, based on these facts, the service should improve the users situation by, *e.g.*, providing guidance, recommendations, or helpful suggestions.

Other terms commonly used to describe this type of computing are ubiquitous or pervasive computing, invisible computing, proactive computing, ambient intelligence, and sentient computing [59].

2.1.1 Context-Aware Service

An application *using* context-aware computing is generally referenced to as a *context-aware service*. In [59], Loke defines a context-aware service to consist of the three components *sensing*, *thinking* and *acting*, see Table 2.1. In

Table 2.1: Components of Context-Aware Systems

Component	**Example**
Sensing	Weather Sensors Motion Sensors Location Sensors
Thinking	Pattern Recognition Mathematical Modeling Machine Learning
Acting	Actuator Recommendation Assistance

Figure 2.1 these components are illustrated as three yellow boxes. Moreover, the figure presents how the raw contextual information *sensed* by the sensing component is passed on to the *thinking* component. This component infers new contextual knowledge from its input and passes this knowledge on to the *acting* component. Finally, the *acting* component is utilizing this knowledge in order to provide some recommendation, assistance or suggestion to the user.

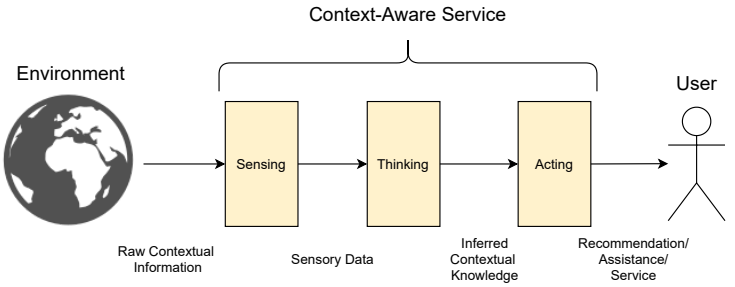

Figure 2.1: Context-Aware Computing

A prominent example of a context-aware service is the smartphone application and finger ring provided by Oura [67]. In this context-aware service, a ring is performing context-sensing through a wide variety of embedded sensors such as heart-beat, O_2, and body-temperature sensors and an accelerometer, magnetometer and a gyroscope. These sensors provide the *sensing* component of the service. Further, the output from these sensors are analysed by

advanced machine learning techniques, representing the *thinking* component of the system. For instance, the thinking component can infer that the user is sitting still for longer periods of time or that the temperature and 0_2 levels of the user indicate some sickness.

Based on these inferences, the *acting* component can provide suggestions such as *"Time to move around a bit"* or *"Your body temperature is slightly elevated, take it easy today"*, through their proprietary smartphone application.

In the following the three components proposed by Loke will be described in closer detail.

Sensing

As previously described, the sensing component is responsible for acquiring information from the context of the entity, *i.e.*, it is gathering information from the environment of the user. Contextual information can be sensed using many different sensors depending on the context, the entity and the goal of the context-aware service. For instance MEMS (micro-electronics-mechanical systems) sensors such as the magnetometer, accelerometer, gyroscope, barometer, light sensor and GPS are commonly found in smartphones. Moreover, so-called *virtual sensors* such as tilt-compensated compass, or the shake and orientation sensors can be provided by fusing the data generated by the MEMS sensors [28]. In autonomous vehicle systems, similar sensors have been used in addition to cameras and LIDAR. Even social interactions can be said to be *sensed* by analysing the data generated on social media platforms.

In this thesis the context sensing has mainly been performed by the MEMS sensors found in smartphones.

Thinking

Thinking or reasoning is arguably the most important component of a context-aware system. It is the part of the system that grants it the ability to infer new knowledge and deduce a better understanding of the context. The process of a thinking component is typically to aggregate, transform or combine the input from the sensing component in order to recognize patterns and *hidden* insights in the data either through mathematical modelling or various types of learning algorithms. Thinking components can also be configured in an iterative manner, where the output from one thinking component is used as the input to the next. For instance, in the work proposed by this thesis on automatic product localization, the product localization algorithm infers the location of products based on, among several sources, the output from a human activity recognition algorithm. This algorithm, in turn, is

recognizing the activity of a human based on the output from the sensors in a smartphone. The usefulness of a context-aware system is often dictated by the thinking components' ability to *understand* the contextual information. Consequently, the main contributions of this thesis is research and work om various thinking components. In Section 2.3, context reasoning, *i.e.*, the thinking component, will be further elaborated and several context reasoning techniques will be described in detail.

Acting
The acting component is the final component of the context-aware service. It is here that the new knowledge regarding the context, inferred by the thinking component, is provided to the user in order to improve the users current situation. Typically, this will be realized as some mobile- or web-application from which the user can get suggestions, help, assistance, or recommendations based on the current context.

2.2 Location-Aware Computing

As previously described, this thesis was a joint project between the Industrial partner Forkbeard Technologies, and the university NTNU. Forkbeards' main business is to provide indoor navigation solutions, and thus specialize in building *location-aware* services. Consequently, the focus of this thesis was research in this type of context-aware services where the location of the user is part of the input to the system. In [90], the authors consider location-based context-services, in other words location-aware services, to be the standard applications for the early stages of context-aware computing. They define it as *"applications that deliver functions or services to the user based on their physical location"*. Location-aware applications can be anything from simple services providing the location of a user on a map, to more advanced services such as a personal shopping assistance. For instance, the application sketched in Figure 2.2 providing path-finding assistance to a user looking for milk. This type of application requires the position of the user and the location of the product the user is looking for.

2.2.1 Location-Sensing Technologies

In location-aware services, the *sensing* component, described in Section 2.1.1, is entrusted with the ability to sense the location of the user, usually through some technologies providing the location of the user. For instance, it can be the user's geo-location, *i.e.*, location somewhere on earth, or the user's

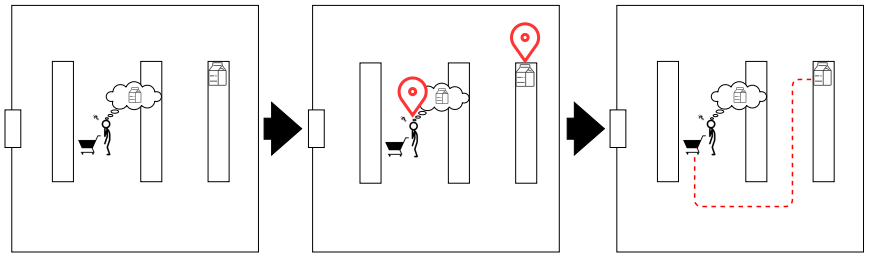

Figure 2.2: Example Context-Aware Application; Shopping Assistance

presence within a vehicle, building or more accurately the floor and room. It can even be a (x, y, z) coordinate within an indoor environment. There is a plethora of technologies providing the location of a user, with varying coverage, accuracy, accessibility, cost, etc. Research and development of these technologies are outside of the scope of this thesis. However, for clarity some topical technologies will be shortly sketched, and their strengths and weaknesses will be discussed.

Global Navigation Satellite Systems

Systems such as the Global Positioning System (GPS) [40], Galileo [29], BeiDou Navigation Satellite Systems (BDS) [6] or GLONASS [33] are well known solutions to track devices in outdoor environments. Devices that can be located by these systems are embedded with a receiver capable of reading signals transmitted from satellites. The device then estimates its position/location by measuring variances in different properties in the signals received from multiple satellites. The accuracy of these systems is generally accepted to be approximately around a couple of meters, however, this varies based on the surroundings of the device, the device itself and the availability of satellites in the area.

Short-Range Wireless Technologies

In indoor environments, either natural or human built, the signals from the satellites used by the GPS technology is disrupted and the accuracy of the location estimates falls drastically. In order to combat this, other signal sources can be used to transmit the signal used by devices to estimate their position. Transmitters transmitting signals such as RFID, IR, WLAN, Bluetooth and Ultrasound have all been used successfully in this aspect. Depending on the

number of transmitters and the type and amount of indoor obstacles, these technologies can offer a location detection accuracy on the centimeter level.

2.2.2 Location-Aware Applications

In this section, some prominent applications domains will be presented, illustrating how the location of a user, sensed by various sensing technologies embedded in smartphones, can be used to create location-aware smartphone applications.

Outdoor/Indoor Navigation

Navigation assistance is a service most people around the world have gotten used to, for instance, guiding a user to his/her desired destination. It was previously achieved using specialized GPS devices. However, nowadays it is provided by the GPS embedded in the user's smartphone. A well known example of an outdoor maps provider is Google Maps [35]. In addition to outdoor navigation assistance, indoor navigation assistance is a popular research and development field and has experienced great improvements in the last decade. Forkbeard Technologies is a provider of technologies enabling indoor navigation [27] using Ultrasound and BLE in order to infer the location of a users smartphone.

Mobile Gaming

The mobile gaming industry has seen an exponential growth the last couple of years. It accounted for 50% of the global video game revenue [88] in 2020. Lately games such as Pokemon Go [66] and Geocaching [30] use the real-time location of the player as a core component of the gaming experience. In Pokemon Go, the player needs to travel around in the real world finding and "catching" digital creatures called pokemons. In Geocaching, the players use their mobile phones as a "treasure map" leading them to real physical boxes placed in various places all around the world. The goal is to find these boxes, sign the physical log-books inside them, and register the achieved *treasure* in the application.

Mobile Recommender Systems (MRS)

In contrast to regular recommender systems that only rely on some recommendation algorithms, mobile recommender systems also take the user's mobile context into account. As the authors explain in [72], "*mobile recommender systems are based on a recommendation algorithm and contextual*

information to provide recommendations of items or services to users of mobile devices."

For instance, in retail and shopping, personal shopping assistance applications such as *The Personal Shopping assistance* and *IRL SmartCart* proposed in [45] and [42], respectively, enrich the target recommendation algorithm by taking the current location of the user into consideration.

In some other works, media recommendations are provided to the smartphone carrier based on their mobile context. For example, in [103] the authors propose a framework for media recommendations, whilst in [4], a music recommendation system is provided for passengers in a car.

Another interesting category of works aims to recommend information to the user based on the user's location. For instance, a mobile recommender system recommending photos to users from their mobile context was proposed in [53]. In [76], the authors present *SMARTMUSEUM* providing tourists with context-aware on-site access to cultural heritage.

Furthermore, within the health and fitness domain, works such as *Motivate*, proposed in [57], aim to change the behavior of the user by suggesting simple daily activities based on the user's contextual information such as weather, user location, geo-information and user agenda.

Public Transportation

Public Transport Authorities (PTAs) already utilize several location-aware systems such as pathfinding to the nearest bus stop/metro station, etc. In addition, some PTAs provide services using the current location of all nearby vehicles combined with the location of the user in order to suggest the optimal travel route, *e.g.*, in cases where there are both a bus stop and a metro stop nearby, however, the next bus is arriving 10 minutes later than the next metro, the application will suggest using the metro station rather than the bus. Another area of location-aware applications within public transportation is applications providing *automated ticketing*, *i.e.*, a so-called Be-In/Be-Out (BIBO) applications [64]. These applications aim to reduce the complexity for users when they purchase public transport tickets. This can be achieved if the system can recognize the in-vehicle presence of passengers, *i.e.*, knowing for sure when a person enters a certain vehicle, and when the person leaves. Early in-vehicle presence detection systems were based on active RFID tags carried by the passengers communicating with a single communication unit in the center of the vehicles. Solutions such as EasyRide [37], developed by the Swiss Railways Association and Allfa [36] proposed by T. Gründel, H. Lorenz, and K. Ringat to be used in Dresden, Germany, used this technology. Later on, solutions such as [63], [49] and [47]

used BLE communication between the smartphones and devices installed in the vehicles in order to detect in-vehicle presence of the smartphones carrier.

2.3 Context Reasoning

All the location-aware applications mentioned above have one important thing in common, and that is they all depend on the contextual information of the user, in particular their location. Further, the system's ability to retrieve new knowledge from the sensed data determines the complexity and type of service the context-aware application can provide to its users. In other words, the most central feature a context-aware service is the *thinking* component and its ability to perform context reasoning. Due to the importance of this component, and that most of the work in this thesis is on context reasoning, we have dedicated the following section to carefully describing various context reasoning techniques.

2.3.1 Context Reasoning Techniques

Depending on the data sensed by the sensing component, and the goals of the acting component, *i.e.*, the context-aware application, various context reasoning techniques can be employed.

Perera et al. [70] categorize these techniques into the following six classes;

- Rules
- Probabilistic logic
- Fuzzy Logic
- Ontology Based
- Unsupervised Learning
- Supervised Learning

In the following, we describe each of these techniques. However, since the learning-based techniques, unsupervised and supervised learning, are at the center of this thesis, we elaborate more on these in their own dedicated section, Section 2.4.

Table 2.2: Rule-Based Context Reasoning

Inferred Context	**Reasoning Rules**
Flying	Barometer (low) ∧ Speed (very high) ∧ Cell signal (zero)
Sleeping	Moving (no) ∧ Light (dark) ∧ Pulse (low)
Workout	Moving (yes) ∧ Pulse (high)

Rules

Rule-based techniques are used to build the most straightforward context-aware services. They are usually built based on IF-THEN-ELSE structures and they are generally easy to develop. However, their strength is also their weakness, *i.e.*, their reasoning algorithm is hard coded and thus they struggle to adapt to dynamic and complex environments.

In table 2.2, three examples are presented on how higher level contexts can be inferred from lower level contextual information using rule-based reasoning. As can be seen from the table, a rule is defined by combining a set of lower-level contextual inputs, *e.g.*, the context *flying* is inferred when the barometer sensor is in "low-state", the speed is "high" and the received cell signal is "zero". However, a potential weakness of rule-based reasoning is when several contexts can be inferred from the same reasoning rules, *e.g.*, the reasoning rules for *Sleeping* could also be used to infer *Watching TV* if the light-sensor was embedded on the users smartphone, and the smartphone was located in the users pocket.

Creating suitable, unambiguous rules for any given context, can often be challenging. Additionally, determining a set of rules to use in an application can be difficult. However, even with these limitations Perera et al. [70] believe that rules are expected to play a significant role in, *e.g.*, IoT.

Probabilistic Logic

The publications [11] and [70] describe techniques belonging to this category as methods for reasoning when the *facts* composing the context have probabilities attached to them. For instance, when two different sources are providing contextual data in conflict with each other. *e.g.*, the accelerom-

eter sensor shows movement while the GPS sensor senses a user standing still. Moreover, *Dempster-Shafer theory* [81], also known as belief functions, provides a general framework for reasoning with uncertainty, allowing one to arrive at a conclusion with a degree of certainty based on combined evidence from different sources. For instance [80], [56] and [3] proposed solutions to Human Activity Recognition (HAR) using *Dempster-Shafer* in order to fuse the output from multiple sensors in smart-environments. Another popular probabilistic technique used to infer the contextual information from observable evidence is Hidden Markov Models (HMM), *e.g.*, the authors of [8] use this technique in order to learn situation models in smart homes.

Fuzzy Logic

In fuzzy logic, partial truths are acceptable in contrast to boolean logic where acceptable truth values are either *true(1)* or *false(0)*. Furthermore, it provides the ability to represent a context more naturally by supporting *vague* descriptions of the context. Propositions can be seen as elastic constraints, and reasoning can be thought of as elastic constraint propagation, in contrast to regular propositional logic where reasoning is precise and a certain proposition leads to an exact conclusion. Fuzzy logic is often used in combination with other reasoning techniques such as ontological, probabilistic, or rule based reasoning. For instance in [43], the authors propose a context-aware access control solution using fuzzy logic and ontology-based reasoning. As an example of their approach, their system gives the employees in a hospital access to medical records of the patients based on *fuzzy facts* as the context elements, *e.g.*, "The patient's health status is 95% critical".

Ontology-Based Reasoning

Reasoning techniques belonging to the ontology category are based on description logic, a family of logic-based knowledge representations. Within computer science, ontology modelling encompass a formal way of representing a set of concepts within a domain and the relationships between these concepts. Ontological reasoning uses a combination of ontology modelling and logic-based reasoning, *e.g.*, the authors of [97] propose *OWL* for modeling context in pervasive computing environments, and for supporting logic-based context reasoning. One shortcoming of ontology-based reasoning is that it is not capable of handling missing values or ambiguous information. Ontological reasoning have been used in a wide range of applications such as intelligent handling of customer complaints in [52], activity recognition in [12] and event detection in [91].

2.4 Learning-Based Reasoning

In this thesis, the context-aware services have been built using learning-based context-reasoning techniques. Thus, these techniques will be described in greater detail. However, the field of learning algorithms is broad and covers more than just context-reasoning algorithms. In this section, we will first give an overview over learning algorithms, unsupervised and supervised, in general. Thereafter, we take a closer look at a specific branch within learning algorithms; *Artificial Neural Networks (ANNs)*, and topics within the field of ANN that are relevant to this thesis.

In contrast to the categories described in the previous section, which mainly reason using human-made and hand-crafted algorithms, techniques belonging to this category are instead *learning* to extract higher level contextual information from lower-level context data. Learning algorithms learn from being exposed to large quantities of data from their environment. Moreover, it is important that the data, they are exposed to, consists of a sub-set of the environment that is large and varied enough to capture its intrinsic properties. Without a suitable dataset, a learning algorithm is not able to learn useful mappings between what it *senses* and the properties it is supposed to *infer*.

Important concepts

In this paragraph, some important concepts for learning algorithms, that are not explained in the text, will be shortly described.
Weights/Trainable Parameters: Most learning algorithms consists of a function, where some of the variables in the function are so-called *Trainable Parameters* or *Weights*. The goal during training of a algorithm is to *tune/change* these parameters in order to achieve a desired outcome, given a certain input.
Training and Validation Sets: Learning algorithms, especially supervised, learn by being exposed to a *training set*. The training and validation sets usually consist of samples of some input and a corresponding, expected output, *i.e.*, a label. For instance, the training and validation sets for an image classifier will contain a certain number of images, and a label describing the content of the images.
Classification: A typical task of learning algorithms is to recognize the class of a sample, *e.g.*, recognizing whether an image is of a dog or a cat. In these tasks, it is common to describe the prediction of the algorithm to: *True Positive(TP)*: a correctly classified positive sample; *True Negative(TN)*: a correctly classified negative sample; *False Negative(FN)*: a positive sample

wrongly classified as negative; *False Positive(FP)*: a negative sample falsely classified as positive. Further, in order to measure the performance of the learning model when performing a classification task, the follow metrics are typically used:

- *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 are actually positive:

$$PR \triangleq \frac{TP}{TP + FP} \tag{2.1}$$

- *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:

$$RE \triangleq \frac{TP}{TP + FN} \tag{2.2}$$

- *Accuracy (ACC)*: In a dataset with a 50/50 class distribution, the accuracy describes how good the model is at classifying samples from all classes, *i.e.*, it describes how many of all predictions are correct:

$$ACC \triangleq \frac{TP + TN}{TP + FP + TN + FN} \tag{2.3}$$

- *F1-score (F1)*: The harmonic mean between precision and recall. The F1-score is useful in cases where the distribution of the classes in the dataset is not 50/50.

$$F1 \triangleq 2 \cdot \frac{PR \cdot RE}{PR + RE} \tag{2.4}$$

Learning-based algorithms are typically divided into two categories; *unsupervised* and *supervised learning*. In some cases, two additional categories are included; *semi-supervised* and *reinforcement*, however, they are not covered in this section since they are of less relevance for the work in this thesis.

In the following two sections, the most archetypical techniques, concepts and algorithms related to supervised and unsupervised learning will be shortly described, and those most relevant for this thesis will be elaborated in detail.

2.4.1 Unsupervised Learning

An unsupervised learning algorithm is a type of algorithm that learns patterns and intrinsic structures in data. The data used to train the learning algorithm is typically not labelled, in contrast to data used to train supervised learning algorithms, hence it is called *unsupervised*. The typical problem solved by unsupervised learning algorithms is *clustering* of data. Clustering is the process of grouping similar objects into so-called clusters, and the *training* aspect of the process is to learn what *similarity* entails in any given environment. A well-trained clustering algorithm, such as *K-means*, *Hierarchical Cluster Analysis(HCA)* and *Expectation Maximization* have the ability to group un-labeled samples together with other *similar* samples in the given context. For instance, training a clustering algorithm on a dataset consisting of unlabeled images of cats and dogs should result in two distinct clusters. Then, when exposing the algorithm to three new images, one of a cat and two of dogs, the algorithm should be able to recognize that the two dog images are dissimilar from the cat image, even though the algorithm is not able to classify the image as an image of a cat.

Unsupervised learning has been used for a plethora of use cases such as data exploration, customer segmentation [20], recommender systems [92], data visualization [65] and input noise reduction [94].

2.4.2 Supervised Learning

In contrast to unsupervised learning, supervised learning uses so-called labeled data during training. They can be described as a type of function approximation technique, where they try to mimic a function f mapping from an input X to some output y. In order to do this, the learning algorithm is presented with a training set of inputs labeled with their expected output. From the training set the learning algorithm is expected to detect and learn some underlying structures or patterns in the dataset such that it finds a *generalized* way of mapping any input, within the bounds of the problem, to any output. If we have a function $f(X) = y$, we want the learning algorithm to build a solution $f'(X) = y'$, where the distance between y' and y is as small as possible for any given X.

A typical task in supervised learning is to solve a *classification*. For instance the ability to recognize the content of an image, the meaning of a text, the words in a sound wave, or the activity performed by a person carrying a smartphone, etc. In addition to these typical multi-class classification problems, where there are more than two potential outputs, binary classification problems are also common. For a binary classification algorithm, the goal is

typically to train some functions in order to map some input, to one of two potential binary outputs. It can be used to answer typical yes or no questions such as "Is this user allowed to receive a loan?", "Is this a fraudulent transaction?" or " Does this patient have cancer?"

In addition to the classification problems mentioned above, supervised learning can also be used in *regression* problems, where the goal of the learning algorithm is to map from some input to a numerical value, such as the expected price of a house given a set of features as input (*e.g.*, number of rooms, location, age, etc.), the future cost of a stock given the stock value in the previous six months, and the probability that it will snow tomorrow given a set of meteorological, geographical features.

Some of the most important supervised learning algorithms worth noting are:

- Linear and Polynomial Regression,
- Logistic Regression,
- Support Vector Machines,
- Decision Trees and Random Forests,
- Artificial Neural Networks.

Linear and Polynomial Regression

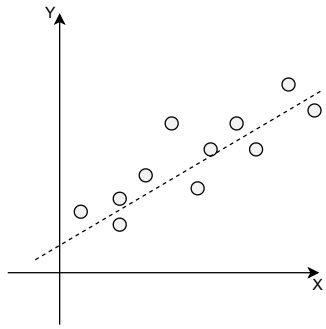

Figure 2.3: Linear Regression

In linear and polynomial regression the goal is to find a function f capable of mapping an input X, typically called the *independent variables* to a value

y', typically called the *dependent variable*. Specific to regression models is that their output, *i.e.*, the dependent variable is expected to be some scalar value. In linear regression you have either *simple linear regression*, where the input is only one independent variable, or you have *multiple linear regression*, where the input consists of two or more variables. Following is a formal description of the two linear regression models; simple linear regression model in Equation 2.5 and multiple linear regression in Equation 2.6:

$$y' = \beta_0 + \beta_1 X \tag{2.5}$$

$$y' = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_n X_n \tag{2.6}$$

Here, y' denotes the dependent variable and the predicted output from the model. β_0 and β_n are the *trainable* parameters of the model which can be tuned in order to *fit* the model to its training data. β_0 describes the y-intercept and β_1 and β_n the slope of the line.

A shortcoming of linear regression algorithms is that they only work when the relationship between their dependent and independent variables are linear. In Figure 2.4 this problem is illustrated by the plot on the left, where the line is not able to fit the samples, however, this is resolved in the plot on the right by using *polynomial regression*.

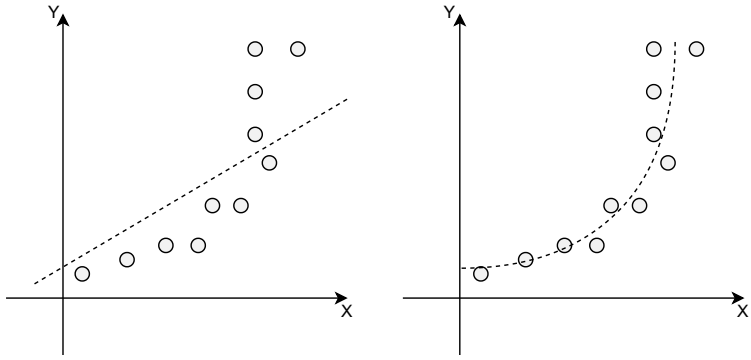

Figure 2.4: Linear vs. Polynomial Regression

Polynomial regression is formally described with the following equation:

$$y' = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_n X^n \tag{2.7}$$

When the relationship between the dependent and the independent variables are not linear, one can use try to describe their relationship using n-th degree polynomials instead.

Common to both linear and polynomial regression is the way they *learn*. As previously mentioned, they learn by adjusting their trainable parameters, *i.e.*, all relevant β parameters. In order to improve the model, we first need to figure out *how* wrong it is, in other words we need to quantify the disagreement between the predicted and the actual y values for any known X in our training set. Formally, this is typically described as the *loss function* or sometimes referred to as the *cost function*.

A typical cost function used for linear and polynomial regression is the Mean Squared Error cost function:

$$L = \frac{1}{m}\sum_{i=1}^{m}(f'(x_i) - y_i)^2 \tag{2.8}$$

Where $f'(x_i)$, equals the predicted output of the model for the ith sample x_i in the training set, y_i denotes the true value for that sample and m denotes the total number of samples in that training set. L is the mean squared error over all m samples of the training set, quantifying the mean error of the model given the current set of trainable parameters.

When adjusting the parameters of the model, it is important to adjust them in proportion to the influence, they had on the outcome, *i.e.*, the amount of loss. To do this one typically uses *Gradient Descent*. The process of gradient descent is to calculate the partial derivates of the loss function with regards to the parameters of the model. The partial derivates of the function quantifies the effect each parameter had on the loss, and, thus, by adjusting the parameters correspondingly, one can effectively reduce the amount of error they induce in the model.

Support Vector Machines

Support Vector Machines (SVMs) are another very popular and useful type of supervised learning algorithm, capable of solving both linear and non-linear, *binary* classification and regression problems. The core idea of SVMs is to separate instances of the two classes by one or several *hyperplanes* in a higher dimensional space. The hyperplanes can be thought of as the widest possible *road*, separating the two instances of the two classes closest to each other in the training set. In Figure 2.5, an illustration of a SVM is depicted, where the solid line represents the center and the dotted-lines represent the edges of the hyperplane. There are two instances circled at the edges of the hyperplane, which are called the *support vectors*, and as long as new training instances are added "off" the hyperplane, the boundaries of the hyperplane will not be affected. In general, the wider the *hyperplane* is, the better the

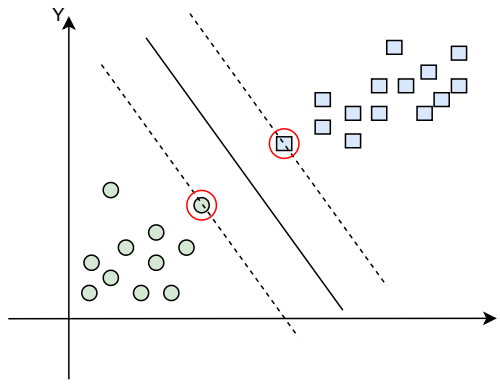

Figure 2.5: Support Vector Machine

model is at *generalizing* to the problem, *i.e.*, the model is expected to have a high accuracy when classifying new, previously unseen samples.

SVMs have been used successfully to solve various context reasoning problems such as human activity recognition in [68], context based recommendations in [48], and patient fall detection in hospitals in [19].

Decision Trees

Decision Trees are a type of supervised learning technique mainly used to solve classification problems. The core concept is to distribute the features of the input as nodes in the tree, always one root node, and one or several intermediate nodes. During training, the distribution of features to nodes is performed. This is done using certain metrics such as the *Gini Index* for categorical decision trees, or the *Mean Square Error* for regression trees. The metrics are used to evaluate which feature is best at splitting the training set into pure sub-sets. The features best capable of doing this are placed higher up in the decision tree. In Figure 2.6, a toy-example of a decision tree for deciding the family of an animal is depicted. Inputs to the decision tree consists of five *attributes*, *i.e.*, if the animal has a backbone, is cold blooded, has wings, lives in water and if it has scales. There are in total four intermediate nodes and one root node representing the attributes of the samples in the dataset, and six leaf nodes, representing the classes.

Prominent works where various forms of decision tree algorithms have been used to perform context reasoning are for instance the CARS (context-aware recommender system) framework proposed in [58], the location-based

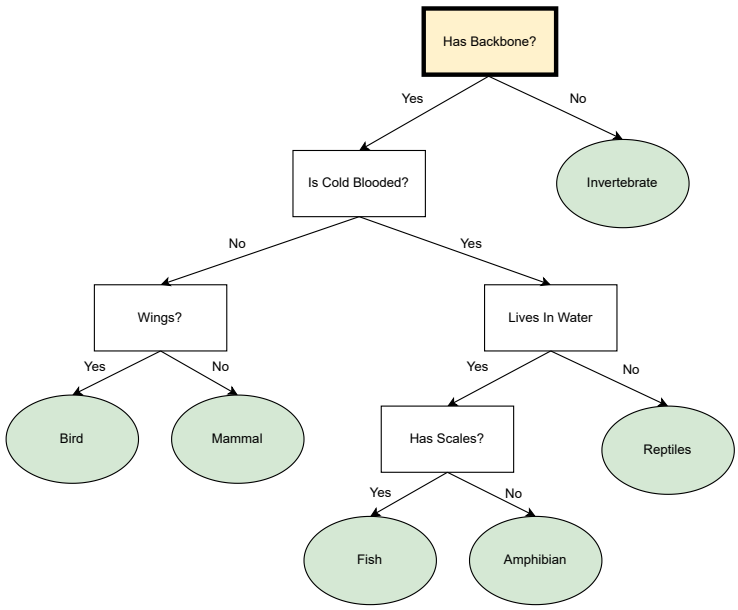

Figure 2.6: Simplified Decision Tree for classifying animals

recommendation system *"I'm feeling LoCo"* proposed in [77], and the human activity recognition algorithm proposed in [61].

2.5 Artificial Neural Networks

Artificial Neural Networks (ANNs) encompass a huge number of learning algorithms, and include a vast sub-set of so-called machine learning (ML) models. ANNs have been used to solve different problems in different domains. In this section, we will give a short introduction to some important concepts within artificial neural networks, followed by a description of the most popular network types, and finally some advanced neural network architectures that have been important in order to answer some of the research questions presented in this thesis.

2.5.1 The Neuron

All artificial neural networks consist of a structured organization or network, of small computational units called *neurons*. These neurons typically accept

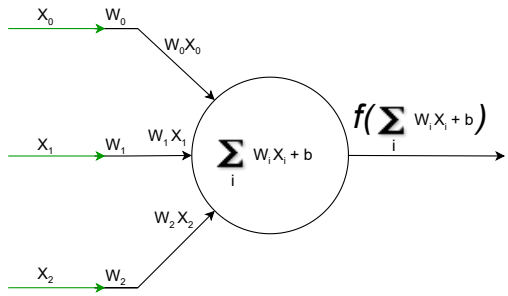

Figure 2.7: Simple artificial neuron

one or more input scalars, each of which are multiplied by a trainable parameter called a *weight*. Thereafter, the products from these multiplications are added together and used as input to an *activation function*, which outputs a single scalar representing the *activation* of the neurons.

Figure 2.7 depicts a simple neuron accepting three inputs, X_0, X_1 and X_2, all being multiplied by their corresponding weights W_0, W_1 and W_2 before the resulting products are summed and passed to the activation function f.

2.5.2 A Network of Neurons

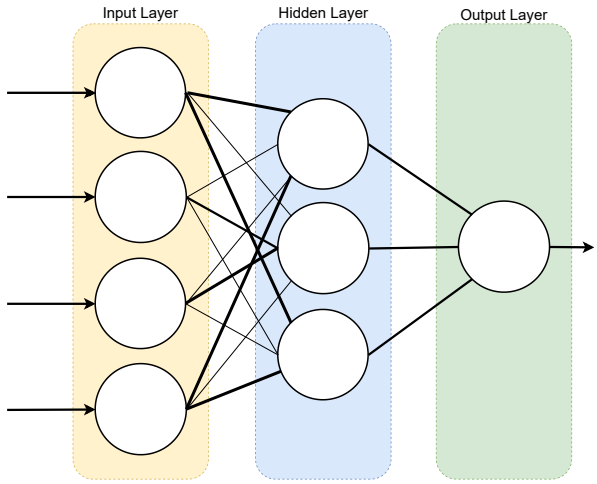

Figure 2.8: Artificial Neural Network (ANN) with fully connected layers

A neural network is modeled as a collection of neurons connected in various configurations as an acyclic graph. The neurons are usually organized into distinct *layers*, where the most basic networks, also called *single layer neural networks*, consist of two layers, an *input* layer and a *output* layer. In these networks, each neuron in the input layer is *connected* to the neurons in the output layer. In other words, the output from a neuron in the input layer is passed on to *all* neurons in the output layer. In order for a model to learn more complicated functions, there are usually more layers than these two. Layers in-between the input and output layers are usually called the *hidden layers*, and networks consisting of multiple hidden layers are often called *Deep Neural Networks*. The way neurons are connected between one layer and the next is defined by their layer type. In the next section, the most well-known layer types are discussed.

2.5.3 Network Layer Types

The way a layer is connected to the previous layer in the network defines the type of layer. The three most popular and well known types of layers are *Dense layers*, *Convolutional Layers*, and *Recurrent Layers*. Typically, if a network consists of mainly one of the layer types, the overall network is named after that layer type *e.g.*, a network consisting of mainly convolution layers is called a *Convolutional Neural Network*.

In the following subsections, the three most popular layer types will be described.

Dense Layer

Dense layer, or commonly called *Fully Connected Layers*, are layers in which each neuron in one layer is connected to all neurons in the next. Neural networks consisting of only dense layers are typically called Dense Neural Networks(DNN) or previously called Multi-Layered Perceptrons, and are the quintessential deep learning models. These types of models have been successfully used for a wide variety of context reasoning use-cases such as activity recognition in hospitals in [25] to in-building localization in [1]. A shortcoming of fully-connected networks, however, is their incapability to handle spatio-temporal properties in their input. Moreover, all samples must be structured equally. Therefore, these networks are well suited for problems where the input to the network consists of attributes that are spatio-temporally independent from each other. For instance, if the input to the network is a set of attributes describing a human, *i.e.*, age, gender, height, income, etc., the order of the attributes in the input vector is irrelevant for the

performance of the model, as long as the order of the attributes in the vector is the same for all samples. However, in image classification or time-series related problems where the input vector represents pixels or points in time, the topology of the input is important. Understanding the spatio-temporal properties of the input is crucial for the model to be able to work for these types of inputs. In these cases, a DNN is not necessarily the best option, and it can be better to use models consisting of the following two layer types instead.

Convolutional Layer

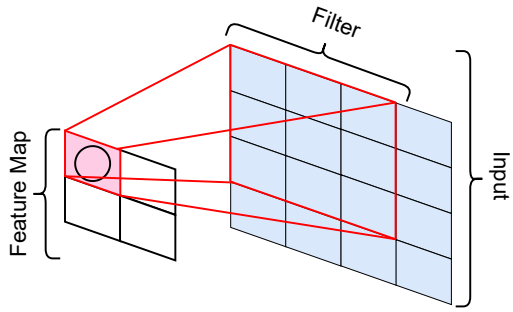

Figure 2.9: Convolutional Layer

Convolutional Networks, also called Convolutional Neural networks (CNNs) are networks consisting mainly of convolutional layers (Conv layers). Their speciality is to process input data configured in a certain multidimensional grid-like topology data such as images (2D) or time series (1D). These types of networks are based on three fundamental ideas to ensure shift and distortion invariance, *i.e.*, to be able to recognize features in their input data, no matter, where in the input *grid* the feature reside. Examples are local receptive fields, shared weights and spatio-temporal subsampling [51]. As input, the neurons in a convolutional layer receive the result from applying some n-dimensional *filter* on-top of the input. This filter represents *the receptive field* of the neuron, *i.e.*, the field of the input a neuron is capable of observing. In Figure 2.9 as an illustration of a 3x3 filter, the filter is illustrated as the red square on top of the *blue* input image. The filter consists of a multidimensional array of *weights* that is applied (convolved), step-wise, across the layers input. The convolution of the filter is done by computing the dot product between the weights of the filter and the input at any position.

Thereafter, the resulting value is passed through an activation function in the same manner as with the dense layers described in the previous section. Convolving the filter across the input results in a multidimensional array called a *feature map*, illustrated as the white two-by-two square in Figure 2.9. A convolutional layer usually contains several n-dimensional filters resulting in an $n + 1$-dimensional array of feature maps. The size of the additional dimension equals the number of filters in the layer, and correspondingly the number of generated feature maps.

In convolutional neural networks, it is common to stack several convolutional layers on-top of each other. Consequently, the input of one layer is the feature maps from the previous, *i.e.*, the filters of a *hidden* layer in a CNN is convolved over the feature maps from the previous layer. A consequence of this is that the *earlier* layers, *i.e.*, the layers closer to the input usually learn to detect lower level geometric features such as curves, lines etc. Layers deeper in the network learn to recognize more high level features, such as eyes, a mouth, fur and so on. Whilst layers close to the output layer learns to recognize concepts such as heads, bodies etc.

Another important aspect of CNNs is the so-called *Pooling Layers*. It is common to to periodically insert a *Pooling layer* in between consecutive conv. layers. Doing this progressively reduces the size of the data transmitted through the network and consequently reduces the number of trainable parameters and computations in the model. A pooling layer works similar to a convolutional layer, in the way that it is applied to its input. It is convolved across its input, and, for every step, it calculates some summary statistic of the values within the range of the filter. Typical calculations performed by a pooling layer are either the *max operation*, finding the largest value within the range of the filter and returning it, or calculating the *average*. For instance, a 2x2 max pooling layer would for every four values in its input return one value, effectively discarding 75% of its input.

Convolutional networks have been tremendously successful in practical applications. Prominent examples of context reasoning using CNNs are the various solutions to human activity recognition in [2, 13, 74, 101], sleep stage scoring in [89], abnormal status detection by an intelligent surveillance robot in [83] and context-aware dialoguing services for a Cloud-Based Robotic System in [41].

Recurrent Layer

The two previous types of layers are generally referred to as *feed-forward* layers, *i.e.*, the data from one layer is fed directly to the next one. Recurrent layers on the contrary, are not only feeding data from one layer to the next,

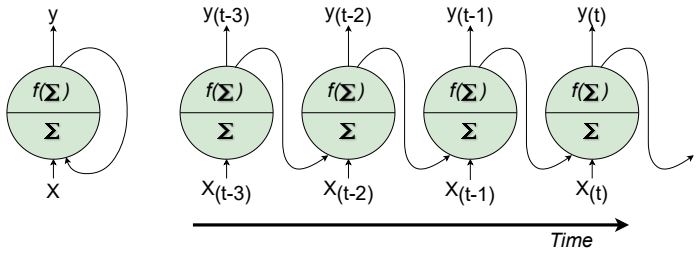

Figure 2.10: Recurrent Neuron(left), unrolled through time (right)

it also has connections pointing *backwards* such that the data processed from one sample is fed back into the same layer together with the next sample, see Figure 2.10. This provides the neural network with a basic memory-like functionality, *i.e.*, the ability to maintain state across input samples. Therefore, with these memory like capabilities, RNNs excel at processing long sequences of input, where the temporal relationships between the elements of the sequence are important and the relationship between the elements can span multiple elements.

For instance, a topical field for RNNs is Natural Language Processing (NLP). In NLP, the goal is to provide a computational unit with the ability to infer *meaning* from text of spoken words. For instance, in the sentence "The cat is chasing the mouse around the house", the relationship between the words "cat" and "mouse" are important to understand the meaning of the sentence, even though these two elements are not right next to each other in the input sequence.

RNNs can be structured differently to solve different kinds of problems. The first and most basic type is the *one-to-one* structure, which simultaneously take a sequence as input. From that, it continuously produces an equally sized sequence as output, see Figure 2.11.a.

Alternatively, the network *many-to-one* can be used, here the network is fed a sequence of inputs and from the whole sequence produce one single output, for instance a movie review used as as input can output a sentiment score (*e.g.*, 1-5 stars), see Figure 2.11.b.

In contrast, a so-called *one-to-many* structure receives a single sample as input, and produces an output sequence from it. For example, the input can be an image from which the network produces a caption (sequence of words) as output, see Figure 2.11.c.

Lastly, a *many-to-many* network configuration can be used in instances where a full sequence will be used as an input, resulting in a sequence as

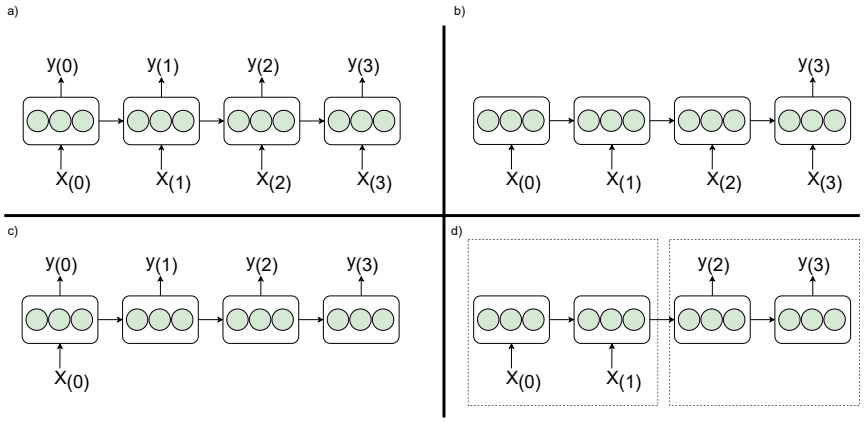

Figure 2.11: one-to-one(a), many-to-one(b), one-to-many(c) and many-to-many(d)

output. For instance in machine translation, where the meaning of the first word in a sentence can be dictated by the last, it is important for the model to receive the whole sentence before it makes its translation, see Figure 2.11.d.

RNNs have been successfully applied to solve problems related to in context in several works, such as predicting future movements of humans/objects in [16], human activity recognition in [71,85] and emotion detection and sentiment analysis in conversations in [82].

2.5.4 Network Architectures

In the previous sections, we described the most important building blocks of any neural network and the various forms, these building blocks usually take. In many cases a reasoning problem can be resolved by starting with a small simple neural network, consisting of as few neurons and layers as possible, and then expanding the number of neurons and layers in order to improve the performance of the model. However, there are some problems which are usually solved by a certain type of architecture, *i.e.*, the configuration of the layers of the model improves the networks ability to solve the problem.

There are countless types of neural network architectures, and creating and optimizing these architectures is on its own a huge research field. In this section we describe two concepts which are important to solve some of the research questions posed in this thesis.

Autoencoders

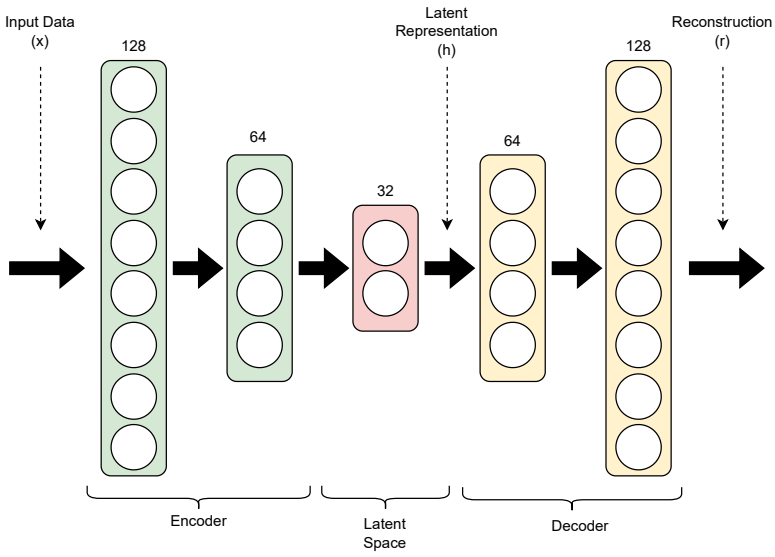

Figure 2.12: An simple Stacked Autoencoder

Autoencoders (AEs) are artificial neural networks used to learn *efficient* representations of their input data, often called *latent representation* or *codings*. They are usually used for dimensionality reduction and feature extraction [34]. Moreover, AEs learn by copying their inputs to their outputs. However, this is not the end goal of an autoencoder. Typically when training an autoencoder, it is constrained in various ways, such that the copy process becomes difficult. Therefore, when copying its input, the model is forced to learn underlying properties in the data. For example, a constraint can be to reduce the size of the latent representation in comparison to its original form. By doing this, the autoencoder is forced to learn the most important features and properties of its input data in order to be able to reconstruct it later.

To better understand this process, it can be useful to view an autoencoder as consisting of two parts: an *encoder* function $h = f(x)$ and a *decoder* function $r = g(h)$. The encoder function converts the models' input data to a latent representation h and the decoder uses h to create a reconstruction r. The cost function used to train an autoencoder is some measure of the dissimilarity between the models' original input x and r, its reconstruction.

In Figure 2.12, a typical *Stacked Autoencoder* is depicted. The name of the model is prefaced with the term *Stacked* since both the encoder and the decoder consist of several layers. As with other artificial neural network architectures, adding more depth to the network, *i.e.*, adding more hidden layers, is useful for the models ability to solve problems of higher complexity.

Depending on its input data, the type of layers used to create the autoencoder can vary, similar to that of other ANN architectures. For instance, in cases where the topology of the input data has no relevance for the data, a fully connected autoencoder can be used successfully, similar to the one illustrated in Figure 2.12. However, when the topology of the input data is important for the networks' ability to infer underlying properties, convolutional layers can be used. An autoencoder, also called *Stacked Convolutional Autoencoder* [60], suitable for processing input data with spatio-temporal properties can be created by *stacking* convolutional layers and pooling/upsampling layers. In these types of architectures the convolutional layers are used to capture and learn the underlying topological structures in the input data, the pooling layers are used by the *encoder* for dimensionality/size reduction and the upsampling layers are used by the decoder in order to reverse the size reduction done by the encoder.

Autoencoders have been used in works such as anomaly detection [78], speaker-aware denoising [15] and context-aware recommendations [106].

Siamese Architecture

In classification tasks, the goal of the neural network is to *recognize* some properties in the input data in order to predict the class the input data belongs to. To train classifiers, one needs a large dataset consisting of many samples from each expected class in order for the model to generalize well. However, in problems where the goal is to recognize if two samples belong to the same class, *e.g.*, if two images are of the same person, or whether a signature belongs to a certain individual, this general approach can be difficult. In order to solve this with regular classification, one has to create one class for each person of interest, *e.g.*, for signature verification used by a bank we need one class for each customer. Additionally, we need to have a large set of signatures from each person, and further, to retrain the machine learning model for each new customer. A solution to problems of this type can be to create a *Siamese Neural Network* (SNN), see Figure 2.13 Siamese Neural Networks contain two or more *identical* sub-networks, in other words the sub-networks consist of identical layers sharing all trainable parameters/weights, represented as the green layers in Figure 2.13 . The output from these sub-networks, the latent representation of their input, are thereafter passed on to

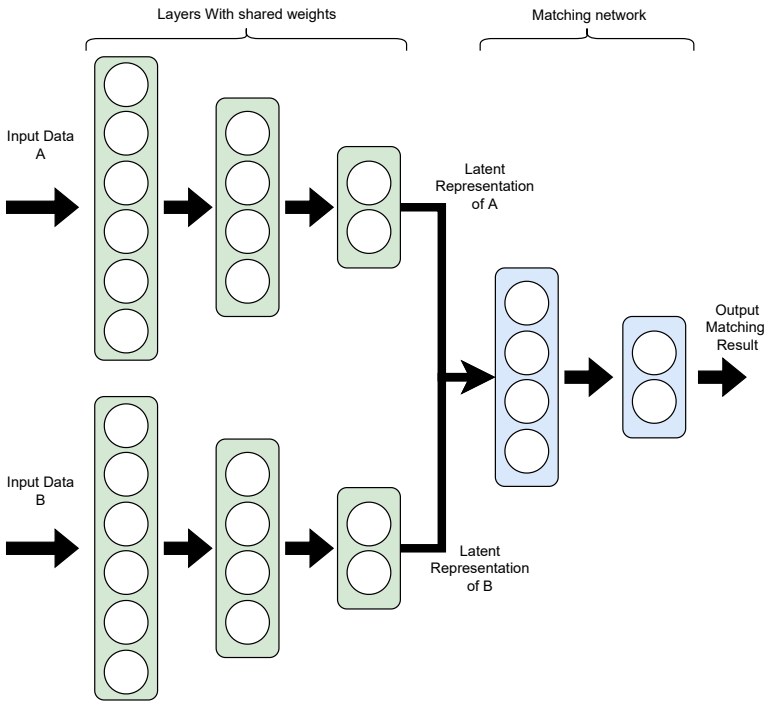

Figure 2.13: A simple Siamese Neural Network

an *objective* function performing matching of the models inputs, represented as blue layers in Figure 2.13. During training, the neural network will learn to ignore features in its input not relevant for matching, and to recognize and extract the most important features for matching detection. Siamese architectures have been used to solve various problems such as signature verification in [9], face verification in [14], person re-identification in [102], and Human Activity Recognition in [7].

2.5.5 Conclusion and Discussion

As mentioned above, selecting the appropriate reasoning technique for any given contextual input is not straightforward and usually one has no universally right answers. This is in particular true when it comes to choosing layers, structures, architectures and other hyperparameters for artificial neural networks. However, there are some basic rules, *e.g.*, as mentioned above,

one should try using convolutional layers when it is expected that some relevant patterns can be found in the spatial arrangement of the input to the model. Likewise, when elements far apart in a sequence are expected to be related with each other, *e.g.*, in time-series data, using RNN layers may be advised. Moreover, in some cases, it might be wise to design the model as an SNN giving the network the ability to accept two inputs at once, and thus the network will map the two inputs in an identical manner, enabling the rest of the network to perform matching detection on the inputs. Furthermore, one can use Stacked Convolutional Autoencoders as input layers providing several other advantages to the model. For instance, using autoencoders provides the capability to drastically reduce the dimensionality of its input, and thus the model learns to prioritize the most important features of its input filtering out unimportant information.

Chapter 3

Related Work

In this chapter, existing approaches relevant to the work carried out in this thesis are presented. First, we introduce the works related to location-aware mobile recommender systems in retail. Thereafter, the state-of-the art on in-vehicle presence prediction is presented. The chapter is concluded with a discussion of the works.

3.1 Mobile Recommender Systems in Retail

As described in Chapter 2, Mobile Recommender System (MRS) aim to provide services to users based on the context, they are in, specifically considering the location of the users. For instance in [24], the authors propose a mobile recommender system for indoor shopping using their own novel indoor mobile positioning approach. Moreover, they use the received signal patterns of mobile phones in order to localize them. Their proposed mobile recommender system implicitly captures user's preference by analysing their positions without requiring their explicit input. They show through comprehensive experimental evaluations that the proposed MRS achieves a high user satisfaction score.

Another mobile recommender system is *RecStore* [84] in which the authors propose a solution that assists customers in malls by proposing relevant shops based on two key elements; the location and historical purchase behavior of the user. Through empirical experiments, they also show that their approach meets the customers needs better than baseline models.

The authors of [10] verify that the products recommended by their location history-aware recommender system far exceeded the product recommendations of the two baseline approaches they studied. Additionally, several volunteers trying their systems gave positive feedback on the variability and

serendipity of their systems' recommendations.

The important finding in the above works is that location-based mobile recommender systems can improve shoppers experiences even though such approaches provide relatively *coarse grained* recommendations, *i.e.*, they suggest relevant stores or, in the best case, products in some nearby stores.

However, to enable more *fine grained* recommender systems, *e.g.*, recommending products at specific locations in a store, while the customer is inside, a more accurate position of the products is required.

Current solutions typically rely on one of two strategies, both using employees or system experts. The first is to attach tags to the products, *e.g.*, RFID-tags, and use a signal transmitted either by the tag or by an infrastructure to locate the product. The other strategy entails registering the location of the product in the system manually. For instance, *SugarTrail* [73] is a system providing both, positioning products and finding paths to their location in in-door environments. To this end, SugarTrail aggregates sequences of magnetometer readings and radio round-trip time-of-flight measurements from stationary nodes in order to automatically record the user's movement path while they move around in the store. These paths are used to construct so-called *Virtual Road Maps* (*VRMs*), which they in turn use to provide navigation assistance to the users in the store. The authors suggest combining this approach with RFID tags on products, and when an employee is placing products in the shelves, they simultaneously register the product in the location using their mobile unit.

Another approach to this problem is Travi-Navi [104], which combines high quality images and sensor readings from a *Guiders* smartphone and packs them into a navigation trace. Their approach is meant to be used by, *e.g.*, shop owners in order to record a *path* to their stores. Then *followers*, moving through an indoor environment, can use the path registered by the *Guider* to be guided to their destination.

In *Canoe* [18], the authors propose a solution where the Received Signal Strength (RSS) is measured in various parts of an indoor environment. The registered RSS values are compared with those of a user's device in order to direct them to points of interests.

Another approach relying on RFID readers and RFID tagged products is proposed in [11]. Their approach suggests products to customers in a store according to the customers' preferences, shopping records and current location.

Summary

In this section works on various mobile recommender systems have been presented, showing that these services tend to improve experience of users

while moving through retail environments. Yet, in order to develop solutions with a higher *fidelity* and more *fine-grained* recommendations, one of their key inputs, the location of the recommendation, needs to be highly accurate. However, current approaches to localizing products are either very costly, *i.e.*, when requiring tags on all products, or very labor intensive, *i.e.*, when using employees to manually register the location of the products. Additionally, these solutions do not suggest any efficient approach to updating the location of products when they are relocated, other than repeating the process used to localize them in the first place. These challenges will be handled by the answers to research questions RQ1 and RQ2, presented in Section 4.1.

3.2 In-Vehicle Presence Detection

As elaborated in Chapter 2, an important aspect of enabling future context-aware services within public transportation is their ability to detecting the in-vehicle presence of passengers with a very high degree of precision.

Early approaches to in-vehicle presence detection used Radio Frequency Identification (RFID) in the form of active tags carried by passengers and communications units installed in the vehicles. The active tags transmit a signal detected by the on-board communication unit and thus the tags' proximity to the unit could be used to infer in-vehicle presence. Approaches using this solution were EasyRide [37], made by the Swiss Railways Association and Allfa [36], made by Siemens VDO, Fraunhofer, GWT and VVO in Dresden, Germany. Unfortunately, tests unveiled that these systems were too unreliable to be used in practice. The authors of [37] attribute this to the weak transmitter strengths of the RFID-tags. In order to combat this weakness, they tried installing a vast number of receiver units in their vehicles, however, even this did not improve the accuracy of in-vehicle presence prediction sufficiently, *e.g.*, the accuracy of Allfa was reported to being 68%.

An alternative to RFID for in-vehicle presence detection arose with the advent of the technology *Bluetooth Low Engery* (*BLE*) in 2010. The solutions using BLE build on the same core principle: The passengers are either carrying a BLE transmitter propagating a signal which is received by on-board equipment or, the other way around, an on-board BLE-transmitter transmits signals that are received by devices carried by the passengers. In contrast to the approaches using RFID, the BLE-based solutions can utilize smartphones and their innate ability to both transmit and receive BLE signals. An early adopter of this approach was the solution proposed in [47], in which the authors used BLE to detect the end-to-end trip of passengers on public transport buses. Another interesting work was presented in [63]. The

authors' goal was to research whether BLE was applicable for Be-In/Be-Out systems in public transportation in general. They conclude their research with a *cautious* "yes". However, they expressed concerns regarding the accuracy of the solution in cases where people outside the vehicle could receive the signal, *e.g.*, people riding a bike or driving in a car next to the vehicle.

In contrast to the signal being too strong and thus being received by people outside the vehicle, the authors of [50] found that objects and other people will partly block or interfere with the BLE signal making the technology less suitable for indoor localization. It is therefore reasonable to assume similar problems can arise inside crowded vehicles.

The authors of [98] pose the question "*Is Bluetooth Low Energy feasible for mobile ticketing in urban passenger transport?*" and conclude, from the technology point of view, that it can be feasible. However, they also report problems related to weak beacon signal reception at long distances and in crowded stations and vehicles. In addition, some phone models and brands were limiting the frequency at which the devices were searching for beacons, negatively impacting the accuracy of the system.

A BLE-based solution called *SEAT* was proposed by the authors of [79], using the BLE capabilities of smartphones together with devices installed in vehicles to track the journeys of passengers in order to provide automatic ticketing. However, the main contribution of their work was on various non-functional demands such as security, performance and power consumption, rather than focusing on the accuracy of the in-vehicle presence solution.

Even though several works are cautiously optimistic in regards to using BLE-based approaches to in-vehicle presence detection, the potential for inaccurate signal detection, both false positives and false negatives, seem to make them inadequate as stand-alone solutions.

Some works, on the other hand, rely on the sensor events generated by embedded smartphones, analysing them using some form of context reasoning technique, and from this analysis directly or in-directly infer the in-vehicle presence of passengers.

Several works use mathematical modelling as their context reasoning technique. An example is Hybrid-Baro [100], that proposes a sensor-based system where the journey of the user inside a vehicle is inferred by using the Dymanic Time Warping (DTW) correlation technique on the sensor event stream of the barometer embedded in the users smartphone. In addition, in very flat areas where the barometer might struggle, they fuse the barometer events with GPS data in order to improve the accuracy of their solution. Another approach using DTW in order to infer the travelled path of users from their barometer trace is published in [39]. The authors demonstrate how the high correlation between barometer sensor events and elevation data can be used

to accurately track driving patterns. Their proposed solution can estimate the possible routes that users have travelled with an accuracy of more than 80%. In [62], the authors present their work on a ticketless transportation solution using features extracted from the passenger's sensor events matched against the corresponding features extracted from units installed in the vehicle. They report in-vehicle presence detection accuracy of 84 to 98% from experiments conducted in five bus-lines over more than 20 hours.

Other works rely on using machine learning techniques in order to infer in-vehicle presence from sensor data. Using machine learning techniques to infer contextual information from smartphone sensor data was surveyed extensively by the authors of [95]. The survey investigates the use of deep learning for sensor-based activity recognition in particular. The authors report on the accuracy of human activity recognition using deep learning models such as CNN, RNN, Restricted Boltzman Machines, autoencoders, and various combinations of CNNs and RNN. They conclude that deep learning models outperform classical *hand-crafted*, pattern recognition algorithms in all experiments. Furthermore, from a technical viewpoint, they state that no one model is outperforming all others. On the contrary, they recommend to try various models, and then select the one that is best suited to solve the task. In *RoadSphygmo* [17], the authors propose an approach to detect traffic congestion using Support Vector Machines (SVMs). *VLD*, proposed by the authors in [96], provides vertical location detection for vehicles in metropolises using SVMs.

Other works in this category focus on detecting the users mode of transportation, *i.e.*, *Transport Mode Detection (TMD)* using the sensor output of the user's smartphones. Works such as [93] demonstrate how the barometer can be used to classify user activities such as riding or leaving a cable-car using Bayesian networks, decision trees and RNNs. More advanced machine learning models such as the *Multimodal spectro-temporal resnet* proposed by the authors of [32] were able to recognize the six user activities *Still, Walk, Run, Bike, Car, Bus, Train, Subway*, from sensor data generated by smartphones, with a F1-score of 94.9% on the *Sussex-Huawei Locomotion(SHL)* dataset [31]. In [54], the authors propose a transportation mode detection algorithm in the form of an 1-dimensional CNN, using as input 10.24 seconds of a smartphone's accelerometer data. This approach provides an accuracy of 94.48%.

Summary

The important finding from the works presented in this section, is that the accuracy of existing in-vehicle detection solutions is not good enough to be used in practice. Even though the solutions using sensor event analytics

through various machine learning based inference techniques promise a better accuracy, it is still not at the level required to support next generation context-aware services. For instance in automated ticketing, false negatives will result in lost revenue, and false positives will result in user complaints, thus it is imperative to provide a high in-vehicle presence detection accuracy. Our solutions to research questions RQ3, RQ4, RQ5 and RQ6 aim to address these challenges, see Section 4.1.

Chapter 4

Results

This chapter presents a summary of the included papers, their contributions and how they relate to the research questions (RQs) posed in Section 1.1. Figure 4.1 illustrate these relationships, *i.e.*, the papers, their thesis contributions and the research questions they answer.

4.1 Summary of the Papers

The papers are introduced in the order they were published. The first paper describes a product localization algorithm for automated detection of the location of products in a retail store. In the second paper, *DeepMatch* is introduced, a deep learning model and design of an associated generic distributed framework offering highly accurate in-vehicle presence detection using smartphone sensors. This was followed by our third paper, published in a high-impact journal: It introduces *DeepMatch2*, a comprehensive deep learning-based approach for in-vehicle presence detection, in which multiple enhancements to the second work was made. Moreover, this paper contains a thorough discussion on travelling user inference systems. Our solution is capable of inferring if and for which period of time a passenger makes a trip in a public transport vehicle with a very low error rate. Finally, the fourth paper included in this thesis presents *Ataraxis*, a deep learning approach for hardwareless in-vehicle presence detection.

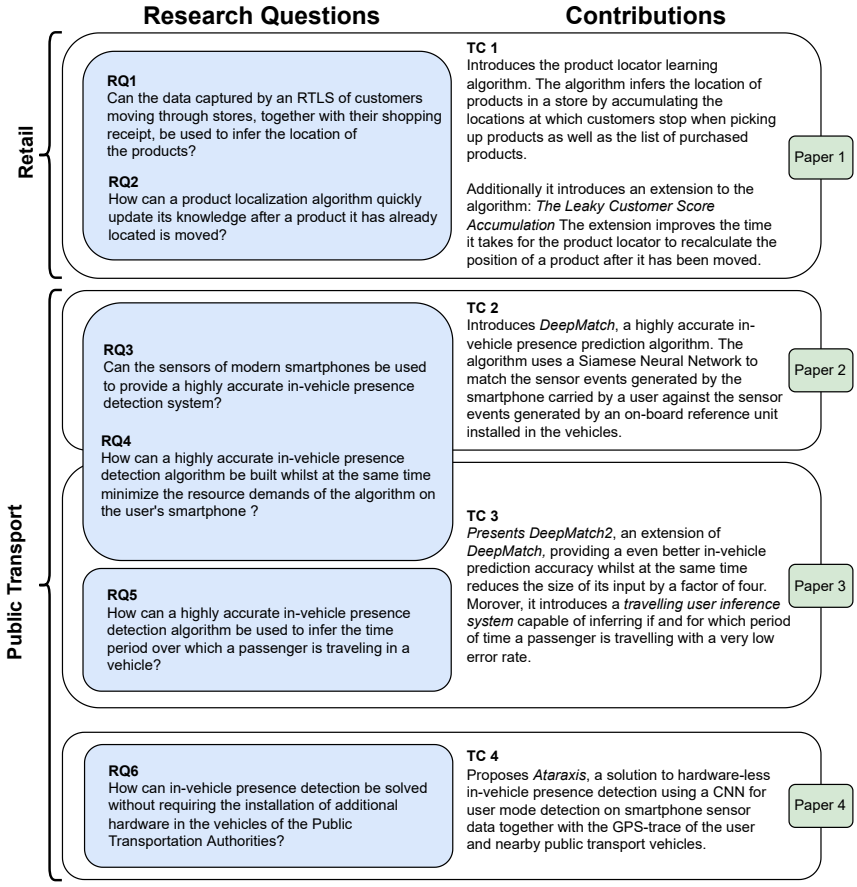

Figure 4.1: The relationship between the domains, research questions(RQs), thesis contributions(TCs) and the papers.

Paper 1

"Automated Product Localization Through Mobile Data Analysis"
Magnus Oplenskedal, Amir Taherkordi and Peter Herrmann
In: *Proceedings of IEEE International Conference on Mobile Data Management (MDM). (2019)*

Summary

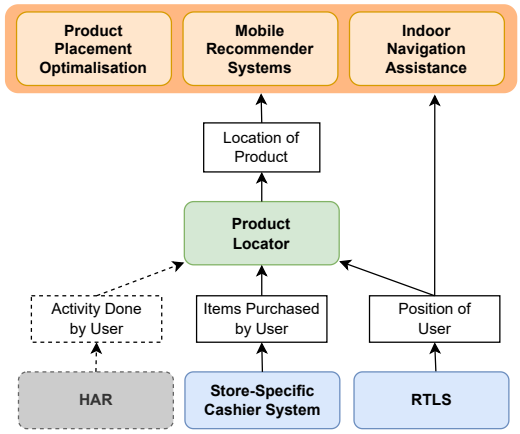

Figure 4.2: The scope of the Product Locator: from processing customer input data to potential applications (Paper 1)

In this paper, context reasoning in retail stores was explored. In particular, our solution uses the data gathered from a user's context to autonomously detect the location of products in the store. The work presented in this paper answers the research questions RQ1 and RQ2:

- ***RQ1:*** *Can the data captured by an RTLS of customers moving through stores, together with their shopping receipt, be used to infer the location of the products?*
- ***RQ2:*** *How can a product localization algorithm quickly update its knowledge after a product it has already located is moved?*

Our approach uses two sets of data from each customer, *i.e.*, the set of positions at which the customer stops while in the store, and the list of items purchased. These sets are accumulated from a large number of customers. Since customers have to walk to the locations of the products they purchase, the accumulated sets can be used to reveal distinct correlations between items and the places at which their buyers stop. These correlations can be utilized to infer the positions of the goods. To realize this, a basic *Customer Score Accumulation* learning algorithm was proposed. The algorithm first accumulates customer data and thereafter infers the locations of the products, thus answering *RQ1*. Moreover, we introduced two extensions of the algorithm (*i.e.*, Leaky Customer Score Accumulation and Softmax-based Inference). *Leaky Customer Score Accumulation* is used to improve the time it

takes for the algorithm to recalculate the position of the products after they have been relocated, effectively reducing the amount of data the learning algorithm needs in order to re-calibrate to new product positions, providing a solution to *RQ2*. *Softmax-based Inference* improves the algorithms ability to infer product locations from the accumulated data.

To evaluate the proposed approach suitable simulation tools simulating indoor stores and customers were developed. The tools were equipped with the possibility to configure the size and layout of the stores, the number of products and the number and frequency of customers using it. Additionally, the tools provide the possibility to induce errors in the simulation.

Considering a significant number of errors in the customer data sets, *e.g.*, customers randomly stopping without picking up products, the simulation-based evaluation results show that 99.9% of 8,000 products in a typical large Norwegian grocery store can be correctly localized after aggregating the data of around 12,000 customer trips.

Contributions. The contributions of Paper 1 are summarized as follows:

- A learning algorithm for automated product localization using the location at which customers stop when picking up products together with the shopping receipt of the purchased products as inputs.
- Two extensions to the learning algorithm (*i.e.*, Leaky Customer Score Accumulation and Softmax-based Inference), improving position calculation on relocated products, and improving the rate at which the algorithm learns the location of products. This effectively reduces the amount of data the algorithm needs to learn.

Paper 2

"DeepMatch: Deep Matching for In-Vehicle Presence Detection in Transportation"
Magnus Oplenskedal, Amir Taherkordi and Peter Herrmann
Published: July 2020
In: *Proceedings of ACM International Conference on Distributed and Event-Based Systems (DEBS).(2020)*

Summary
In this paper, *DeepMatch* is proposed as a solution to in-vehicle presence detection. The work answers the research questions RQ3 and RQ4:

- ***RQ3:*** *Can the sensors of modern smartphones be used to provide a highly accurate in-vehicle presence detection system?*

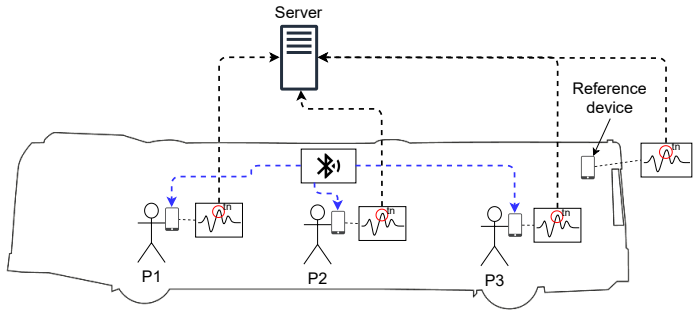

Figure 4.3: A sample scenario presenting DeepMatch(2) (Paper 2 and 3)

- ***RQ4:*** *How can a highly accurate in-vehicle presence detection algorithm be built whilst at the same time minimize the resource demands of the algorithm on the user's smartphone (*e.g.*, computation, battery consumption, data transmission, etc.)?*

The work proposes a highly autonomous solution that can detect in-vehicle presence with a very high degree of precision. The approach relies on equipping vehicles with a reference device (*e.g.*, an Android phone) supporting the ability to deduce the presence of passengers in the vehicle based on the similarity between the data gathered by sensors of the device and those registered by the sensors of the passengers' smartphones. The approach utilizes a deep learning model to verify the in-vehicle presence of a passenger. The model is created from two Stacked Convolutional Autoencoders (CAE), configured in a Siamese Architecture. The CAEs are used for feature extraction and dimensionality reduction, whilst similarity detection is performed by a fully connected deep neural network. The model is trained on real data traces gathered by a group of volunteers traveling 160 unique public transport trips. Evaluation results show that the statistical accuracy of *DeepMatch* is 0.978 for in-vehicle presence detection, providing an answer to *RQ3*.

Furthermore, the paper shows extensive testing on different input combinations, *i.e.*, varying which sensors are used as input to the inference algorithm. Moreover, it presents experiments on segment sizes, *i.e.*, the length of the input data, and illustrates its feasibility for practical use through extensive testing on the computational overhead and power consumption of the approach when run on used smartphones from various brands and with a wide age range, answering *RQ4*.

Contributions. The contributions of this paper are as follows:

- A novel, distributed framework, for in-vehicle presence detection with a very high degree of precision.
- Thorough evaluation using real data gathered by volunteers traveling in public transport.
- Extensive testing on the optimal choice of sensors, and sensor data lengths.
- Extensive testing on computational overhead and power consumption of the approach, when used on user's smartphones.

Paper 3

"DeepMatch2: A Comprehensive Deep Learning-based Approach for In-Vehicle Presence Detection"
Magnus Oplenskedal, Peter Herrmann and Amir Taherkordi
In: *Information Systems Journal. (2021)*

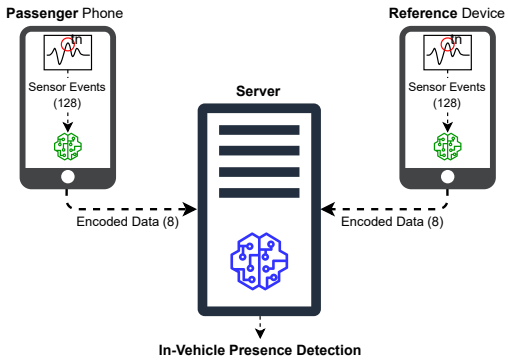

Figure 4.4: Overview of the DeepMatch(2) distributed framework

Summary
The third paper of this thesis presents *DeemMatch2*, an extension of the work done in Paper 2. The work *improved on the answers that Paper 2 proposed for RQ3 and RQ4, and provided an answer to RQ5.* In this paper several improvements were done to the deep learning model proposed in Paper 2. Changes were made to the structure of the Convolutional Autoencoders, the way the latent data from the two autoencoders were *concatenated* before the data was fed to the matching module, and lastly, the layers of the matching module was changed from dense to convolutional layers. These changes made

it possible for the model to increase its statistical accuracy from 0.9781 to 0.9851, and at the same time reduce the amount of data it needs to perform in-vehicle presence prediction from 512 to 128 barometer events, improving the answers posed to RQ3 and RQ4.

The paper also goes deep into the topic of using the output from the deep learning model to infer the duration of a passengers trip in public transport with a very low error rate, in order to answer RQ5.

Contributions. The summarized contributions of this paper are as follows:

- Improved efficiency by reducing the amount of data necessary for in-vehicle presence detection by a factor of four.
- In spite of the considerable reduction of the size of the input, the accuracy of DeepMatch2 was improved to 98.51% in comparison to the accuracy value of 97.81% in DeepMatch.
- Trip inference systems were thoroughly investigated and it showed how DeepMatch2 would be used to infer if and for which period of time a passenger is making a trip in a public transport vehicle.
- Finally, an in-depth description on how the accuracy of such an inference algorithm can be found theoretically was presented, as well as how the result of these calculations can be used to fine tune the parameters of such an algorithm in order to achieve a extremely low error rate.

Paper 4

"Ataraxis: A Deep Learning Approach for Hardwareless In-Vehicle Presence Detection"
Magnus Oplenskedal, Amir Taherkordi and Peter Herrmann
In: *Proceedings of International Conference on Cognitive Machine Intelligence (CogMI). (2021)*
Summary
In the fourth and final paper of this thesis research question RQ6 was investigated:

RQ6: *How can the issue of in-vehicle presence detection be solved without requiring the installation of additional hardware in the vehicles of the Public Transportation Authorities?*

In contrast to the solutions presented in Papers 2 and 3, the solution proposed in this paper does not rely on the installation of any additional

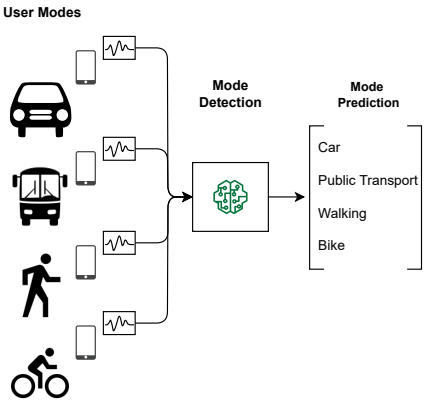

Figure 4.5: The different user modes Ataraxis is capable of recognizing

hardware in the vehicles. Instead, it presumes the presence of some external service providing the real-time and historical geolocation of the vehicles. In Norway, the government-funded organization Entur [22] has set strict regulations and requirements to what kind of hardware all vehicles operated by the public transport authorities have to be equipped with. One of these requirements is that all vehicles should submit real-time data to the publicly available Entur API [23] based on the SIRI 2.0 standard [86]. The SIRI standard includes the real-time location of the vehicle, and as such, all vehicles operated by public transportation providers in Norway are equipped with GPS. In Ataraxis, a solution to hardware-less in-vehicle presence detection is proposed, utilizing a novel deep convolutional neural network. The model was trained on a dataset collected by volunteers performing the four user modes "*walking*", "*riding a bike*", "*driving a car*", and "*traveling in public transport*". From this, Ataraxis can recognize whether the carrier of a smartphone is inside a public transport vehicle or not. If the user is assumed to be inside a public transport vehicle, the trace of their position can be compared with those of the vehicles in their vicinity using the Entur API [23] or similar services. Finally, this information can be used to infer which public transport vehicle the user is traveling in, answering research question RQ6. However, only the transport mode detection model is included in this paper, while the alignment of the user and vehicle GPS traces are work to be done in the future.

The proposed deep learning model achieved an accuracy of 98.69% when predicting the user mode of users from 12.8 seconds of data generated by the magnetometer, gyroscope, and accelerometer sensors typically found in

modern smartphones. Additionally, the paper shows the results of extensive empirical experiments on important non-functional aspects of the solution such as the size of the input data of the model, which sensors data yield the highest accuracy, and the energy consumption and CPU overhead when run on smartphones.

Contributions. The contributions of this paper can be summarized as follows:

- A deep learning model for highly accurate transport mode detection using the embedded sensors in user's smartphones.
- The results from testing on the optimal choice of sensors, and sensor data lengths.
- The results from testing showing that Ataraxis incurs an negligible computational overhead and power consumption on user's smartphones.

4.2 Complementary Aspects

Besides answering the research questions posed in this thesis, some complementary aspects were also discovered. We did not answer them because they are out of the scope of this work. Nonetheless, they are important aspects worth mentioning as descried below.

When it comes to the work carried out in Paper 1 on the product locator, the simulation tools used to train and evaluate the proposed approach simulates only the expected output from customers moving through a store. These stores are, in general, represented by a two-dimensional array of *locations* in which a pre-defined number of products could reside. An interesting further development of this approach could be to create more realistic digital environments in which the virtual customers actually walk through the store. For instance, some retail stores are rectangular with tall, long shelves, whereas others are more quadratic, with lower, smaller shelves. These configurations affect the way customers move through the environment and can influence the precision and learning rate of the algorithm. Furthermore, improvements to the simulation of virtual customers can provide various customer behaviours such as hasty purchases after work or strolling through the store at a leisurely pace while trying to discover something new.

Regarding the work performed in Paper 4, on a hardware-less in-vehicle presence detection approach, further work can be done on the actual trip detection algorithm. Ataraxis provides a transport mode detection deep

learning model, and outlines an algorithm for trip detection, however, finding an optimal way of providing trip detection using GPS trace matching is an interesting research topic that deserves attention.

Another aspect related to the work in this thesis, is the preservation of privacy concerns for the users data. As it is the case for most context-aware services, all approaches proposed in this thesis rely on contextual data regarding the user for providing services. Gathering this data can potentially breach the privacy and data security of the users, and thus finding solutions to this can be an interesting research topic. A solution to this problem is using *Federated Learning* [46] in order to train the machine learning models used for context reasoning. Federated learning is a technique to train the machine learning model across many decentralized devices at ones. Within each device, a local model is trained on a small dataset only persisted in the device. Then, at a certain frequency, the parameters, *i.e.*, the weights of the neural network, are transmitted from the devices, to a central server, in order to create a global model. This way, it is possible to train one model on all of the distributed data while still keeping the data decentralized and never shared with the central server. Thus, this addresses critical issues such as access rights, privacy and data security.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The rapid evolution of mobile technologies, IoT and cellular network infrastructures in combination with the mass adoption of smartphones (*i.e.*, 6.64 billion people in the world own one [5], equal to 83.89% of all people on earth), has set the stage for the development of new types of context-aware services. These services sense the users environment, lean new knowledge from it and use these new insights to provide the user with usable content, functions and experiences. A sub-set of these services use the location of the user as a key source of input, so-called *location-aware* services. Central to these services is that they depend on a high degree of precision and sophistication in their input. Additionally, since these services rely on the computational capabilities of the user's smartphone, it is important to always consider the impact the services have on non-functional requirements such as power consumption, CPU-overhead and data-transmission.

Through a thorough related work investigation and collaborations with the industrial and research partners of the project, it was uncovered that the finesse and accuracy of many of these services are inadequate. In consequence, they cannot be used in practice. This guided us to to define the main research focus of this Ph.D. thesis as follows:
To conduct applied research and develop design concepts, software frameworks, and algorithms to utilize the real-time position, and sensor data from smartphones in order to improve the intelligence and quality of context-aware services that are directly or indirectly location-dependent.

We derived the six research questions introduced in Section 1.1 from this objective. The results and contributions achieved in the work answering the research questions can be summarized as follows:

- The Product Locator: A learning algorithm for the automatic detection of location of products in stores. The key information for mobile recommender services is the location of the user and the location of the suggested point of interest or product. The location of users can be provided using Real-Time Location Systems (RTLS). However, a reliable, efficient and automatic way of locating the products was missing before. The learning model proposed in this work solves this issue. It provides the location of the products using as input the aggregated trajectory of users moving through the store together with the shopping receipts generated by a typical cashier system.

- DeepMatch: A deep learning-based distributed framework for in-vehicle presence detection of users in public transport. A key feature of context-aware services, *e.g.*, for automated ticketing (Be-In/Be-Out) in public transport is the in-vehicle presence detection of passengers. *DeepMatch* provides a solution to this coveted mobile context with a very high accuracy such that is can be used in practice. Moreover, the system does not impact the computational capabilities and battery of the users smartphone negatively, such that there is high probability that it will be accepted by the users.

- DeepMatch2: An extension to DeepMatch, further improving the accuracy of the in-vehicle presence detection while at the same time reducing the amount of data it needs to do so by a factor of four. Furthermore, a travelling inference system building on the DeepMatch learning model for accurately detecting the start, end, and duration of whole trips for passenger travelling in public transport with an extremely low error rate is proposed.

- Ataraxis: A deep learning based solution to hardware-less in-vehicle presence prediction of passengers. This approach offers the ability to perform in-vehicle presence prediction also in cases, where the installation of hardware in the vehicles is not possible, such that solutions such as DeepMatch and DeepMatch2 cannot be used. Ataraxis proposes an system based on a convolutional neural network (CNN) trained to detect the transportation mode of a user from the sensor output of the user's smartphone. The work also shows how this can be achieved without imposing too much strain on the user's smartphone and draining its battery.

5.2 Future Work

To conclude Part 1 of this thesis, some future research directions related to the contributions of this work will be presented.

Currently the product locator proposed in this thesis is able to provide the (x, y) coordinates for the product based on where the customer stops, *i.e.*, 2D product localization. In most retail stores, however, many products are also distributed along the z axis, they are placed vertically on different shelves. Hence, an interesting research topic would be to find a solution to the detection of the z coordinate of the product, *i.e.*, the shelf the product is located on. In other word, the extended system shall provide *3D product localization*. To achieve that, one has to answer the question whether it is possible to detect the actions *standing*, *bending* and *reaching* from the output of the smartphones' sensors such that it should be able to infer which shelf the product was on.

Another interesting research direction is the possibility of creating an efficient, *automatic infection-tracing* algorithm using the matching capabilities of the DeepMatch deep learning model together with some clustering algorithms. The learning model of DeepMatch is tailor made to predict the in-vehicle presence of a passenger with a very high precision by matching the barometer events generated by his/her phone against the corresponding data generated by a reference unit installed in the vehicle. Instead of matching the data from a user against a fixed reference device, it can also be matched against the data created by other users at the same time. This would allow us to build *temporal clusters*, to which two or more users belong if they stay in the same vehicle at the same time. Then, if one of the users were infected with some pathogen, other users that participated in the same temporal-clusters as this user, can be informed. Such a system would have proven very helpful in the COVID19 pandemic.

Finally, the event matching capabilities of DeepMatch opens a new horizon for future event-based systems in which the similarity between events is not clearly and easily visible, implying that the sate-of-the-art matching techniques would not applicable. For instance, a system comparing the energy consumption between inhabitants in a neighborhood. The output of these comparisons (*i.e.*, matching) can be used to provide customized intelligent offers to a category of consumers (*i.e.*, consumer classification), or categorised recommendations on how to use energy in different hours to reduce the overall cost. The DeepMatch learning model proposed in this thesis can provide a good basis for meeting event matching requirements in systems where streams of data should be compared intelligently and efficiently.

Bibliography

[1] Uzair Ahmad, Andrey Gavrilov, Uzma Nasir, Mahrin Iqbal, Seong Jin Cho, and Sungyoung Lee. In-building localization using neural networks. In *2006 IEEE International Conference on Engineering of Intelligent Systems*, pages 1–6. IEEE, 2006.

[2] Zeeshan Ahmad and Naimul Khan. Cnn-based multistage gated average fusion (mgaf) for human action recognition using depth and inertial sensors. *IEEE Sensors Journal*, 21(3):3623–3634, 2020.

[3] Fadi Al Machot, Heinrich C Mayr, and Suneth Ranasinghe. A hybrid reasoning approach for activity recognition based on answer set programming and dempster–shafer theory. In *Recent Advances in Nonlinear Dynamics and Synchronization*, pages 303–318. Springer, 2018.

[4] Linas Baltrunas, Marius Kaminskas, Bernd Ludwig, Omar Moling, Francesco Ricci, Aykan Aydin, Karl-Heinz Lüke, and Roland Schwaiger. Incarmusic: Context-aware music recommendations in a car. In *International conference on electronic commerce and web technologies*, pages 89–100. Springer, 2011.

[5] BankMyCell. How Many Smartphones are in the World? `https://www.bankmycell.com/blog/how-many-phones-are-in-the-world`, 2022. Accessed: 2022-02-22.

[6] BeiDou. BeiDou (Global Navigation Satellite Systems). `http://en.beidou.gov.cn/SYSTEMS/System/`. Accessed: 2022-02-22.

[7] Samuel Berlemont, Grégoire Lefebvre, Stefan Duffner, and Christophe Garcia. Class-balanced siamese neural networks. *Neurocomputing*, 273:47–56, 2018.

[8] Oliver Brdiczka, James L. Crowley, and Patrick Reignier. Learning situation models in a smart home. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(1):56–63, 2009.

[9] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

[10] Thomas Chatzidimitris, Damianos Gavalas, Vlasios Kasapakis, Charalampos Konstantopoulos, Damianos Kypriadis, Grammati Pantziou, and Christos Zaroliagis. A location history-aware recommender system for smart retail environments. *Personal and Ubiquitous Computing*, 24(5):683–694, 2020.

[11] Chia-Chen Chen, Tien-Chi Huang, James J. Park, and Neil Y. Yen. Real-time Smartphone Sensing and Recommendations towards Context-awareness Shopping. *Multimedia Systems*, 21:61–72, 2015.

[12] Liming Chen and Chris Nugent. Ontology-based activity recognition in intelligent pervasive environments. *International Journal of Web Information Systems*, 2009.

[13] Heeryon Cho and Sang Min Yoon. Divide and conquer-based 1d cnn human activity recognition using test data sharpening. *Sensors*, 18(4):1055, 2018.

[14] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[15] Fu-Kai Chuang, Syu-Siang Wang, Jeih-weih Hung, Yu Tsao, and Shih-Hau Fang. Speaker-aware deep denoising autoencoder with embedded speaker identity for speech enhancement. In *Interspeech*, pages 3173–3177, 2019.

[16] Enric Corona, Albert Pumarola, Guillem Alenya, and Francesc Moreno-Noguer. Context-aware human motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6992–7001, 2020.

[17] A. Dimri, H. Singh, N. Aggarwal, B. Raman, D. Bansal, and K. K. Ramakrishnan. RoadSphygmo: Using Barometer for Traffic Congestion Detection. In *8th Inter. Conf. on Communication Systems and Networks (COMSNETS)*, 2016.

[18] K. Dong, H. Ye, W. Wu, M. Yang, Z. Ling, and W. Yu. Canoe: An Autonomous Infrastructure-Free Indoor Navigation System. *Sensors (Basel)*, 17(5):996, 2017.

[19] Charalampos Doukas, Ilias Maglogiannis, Philippos Tragas, Dimitris Liapis, and Gregory Yovanof. Patient fall detection using support vector machines. In *IFIP international conference on artificial intelligence applications and innovations*, pages 147–156. Springer, 2007.

[20] J Du Toit, R Davimes, A Mohamed, K Patel, and JM Nye. Customer segmentation using unsupervised learning on daily energy load profiles. *Journal of Advances in Information Technology Vol*, 7(2):69–75, 2016.

[21] Alexandros Efthymiou, Emmanouil N. Barmpounakis, Dimitrios Efthymiou, and Eleni I. Vlahogianni. Transportation mode detection from low-power smartphone sensors using tree-based ensembles. *Journal of Big Data Analytics in Transportation*, 1(1), 2019.

[22] Entur. About entur. `https://om.entur.no/bedrift/om-entur`, 2021. Accessed: 2021-10-07.

[23] Entur. Entur API. `https://developer.entur.org`, 2021. Accessed: 2021-10-07.

[24] Bing Fang, Shaoyi Liao, Kaiquan Xu, Hao Cheng, Chen Zhu, and Huaping Chen. A novel mobile recommender system for indoor shopping. *Expert Systems with Applications*, 39(15):11992–12000, 2012.

[25] Jesus Favela, Monica Tentori, Luis A Castro, Victor M Gonzalez, Elisa B Moran, and Ana I Martínez-García. Activity recognition for context-aware hospital applications: issues and opportunities for the deployment of pervasive networks. *Mobile Networks and Applications*, 12(2):155–171, 2007.

[26] Tao Feng and Harry J P Timmermans. Comparison of Advanced Imputation Algorithms for Detection of Transporation Mode and Activity Episode using GPS Data. *Transportation Planning and Technology*, 39(2), 2016.

[27] Forkbeard Tech. `https://forkbeardtech.com/`. Accessed: 2022-01-15.

[28] Manish Gajjar. *Mobile sensors and context-aware computing*. Morgan Kaufmann, 2017.

[29] Galileo. Galileo (Global Navigation Satellite Systems). `https://www.gsc-europa.eu/galileo/what-is-galileo`. Accessed: 2022-02-22.

[30] Geocaching. `https://www.geocaching.com`. Accessed: 2022-01-15.

[31] Hristijan Gjoreski, Mathias Ciliberto, Lin Wang, Francisco Javier Ordonez Morales, Sami Mekki, Stefan Valentin, and Daniel Roggen. The university of sussex-huawei locomotion and transportation dataset for multimodal analytics with mobile devices. *IEEE Access*, 6:42592–42604, 2018.

[32] Martin Gjoreski, Vito Janko, Gašper Slapničar, Miha Mlakar, Nina Reščič, Jani Bizjak, Vid Drobnič, Matej Marinko, Nejc Mlakar, Mitja Luštrek, et al. Classical and deep learning methods for recognizing human activities and modes of transportation with smartphone sensors. *Information Fusion*, 62:47–62, 2020.

[33] GLONASS. GLONASS (Global Navigation Satellite Systems). `https://www.glonass-iac.ru/en/about_glonass/`. Accessed: 2022-02-22.

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[35] Google Maps. `https://www.google.no/maps/`. Accessed: 2022-01-15.

[36] T. Gründel, H. Lorenz, and K. Ringat. The ALLFA Ticket in Dresden. Practical Experience of Fare Management Based on Be-In/Be-Out & Automatic Fare Calculation. IPTS Conference, Seoul, South Korea, 2006.

[37] T. Gyger and O. Desjeux. EasyRide: Active Transponders for a Fare Collection System. *IEEE Micro*, 21(6), 2001.

[38] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based Transportation Mode Detection on Smartphones. In *11th ACM Conference on Embedded Networked Sensor Systems*, 2013.

[39] Bo-Jhang Ho, Paul Martin, Prashanth Swaminathan, and Mani Srivastava. From Pressure to Path: Barometer-based Vehicle Tracking. In *2nd ACM Inter. Conf. on Embedded Systems for Energy-Efficient Built Environments (BuildSys)*, Seoul, South Korea, 2015.

[40] Bernhard Hofmann-Wellenhof, Herbert Lichtenegger, and James Collins. *Global positioning system: theory and practice*. Springer Science & Business Media, 2012.

[41] Jhih-Yuan Huang, Wei-Po Lee, and Tsu-An Lin. Developing context-aware dialoguing services for a cloud-based robotic system. *IEEE access*, 7:44293–44306, 2019.

[42] Gerrit Kahl, Lübomira Spassova, Johannes Schöning, Sven Gehring, and Antonio Krüger. Irl smartcart-a user-adaptive context-aware interface for shopping assistance. In *Proceedings of the 16th international conference on Intelligent user interfaces*, pages 359–362, 2011.

[43] ASM Kayes, Wenny Rahayu, Tharam Dillon, Elizabeth Chang, and Jun Han. Context-aware access control with imprecise context characterization through a combined fuzzy logic and ontology-based approach. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 132–153. Springer, 2017.

[44] Inayat Khan, Shah Khusro, Shaukat Ali, and Jamil Ahmad. Sensors are power hungry: An investigation of smartphone sensors impact on battery power from lifelogging perspective. *Bahria University Journal of Information & Communication Technologies (BUJICT)*, 9(2), 2016.

[45] Ting Loong Khor. *Personal Shopping Assistant*. PhD thesis, UTAR, 2016.

[46] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[47] V. Kostakos, T. Camacho, and C. Mantero. Wireless detection of end-to-end passenger trips on public transport buses. In *13th IEEE International Conference on Intelligent Transportation Systems*, 2010.

[48] Aansi A Kothari and Warish D Patel. A novel approach towards context based recommendations using support vector machine methodology. *Procedia Computer Science*, 57:1171–1178, 2015.

[49] Sriharsha Kuchimanchi. Bluetooth low energy based ticketing systems. Master's thesis, Aalto University, Espoo, Finland, 2015.

[50] Andrzej Kwiecień, Michał Maćkowski, Marek Kojder, and Maciej Manczyk. Reliability of Bluetooth Smart Technology for Indoor Localization System. In *Inter. Conf. on Computer Networks (CN)*. Springer, 2015.

[51] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[52] Ching-Hung Lee, Yu-Hui Wang, and Amy JC Trappey. Ontology-based reasoning for the intelligent handling of customer complaints. *Computers & Industrial Engineering*, 84:144–155, 2015.

[53] Fabrício DA Lemos, R Carmo, Windson Viana, and R Andrade. Towards a context-aware photo recommender system. In *Context-aware recommender system workshops*, 2012.

[54] Xiaoyuan Liang and Guiling Wang. A convolutional neural network for transportation mode detection based on smartphone platform. In *2017 IEEE 14th international conference on mobile Ad Hoc and sensor systems (MASS)*, pages 338–342. IEEE, 2017.

[55] Xiaoyuan Liang, Yuchuan Zhang, Guiling Wang, and Songhua Xu. A deep learning model for transportation mode detection based on smartphone sensing data. *IEEE Trans. on Intelligent Transportation Systems*, 21(12), 2019.

[56] Jing Liao, Yaxin Bi, and Chris Nugent. Using the dempster–shafer theory of evidence with a revised lattice structure for activity recognition. *IEEE Transactions on Information Technology in Biomedicine*, 15(1):74–82, 2010.

[57] Yuzhong Lin, Joran Jessurun, Bauke de Vries, and Harry Timmermans. Motivate: Context aware mobile application for activity recommendation. In *International Joint Conference on Ambient Intelligence*, pages 210–214. Springer, 2011.

[58] Sonal Linda and Kamal Kant Bharadwaj. A decision tree based context-aware recommender system. In *International Conference on Intelligent Human Computer Interaction*, pages 293–305. Springer, 2018.

[59] Seng Loke. *Context-aware pervasive systems: architectures for a new breed of applications*. CRC Press, 2006.

[60] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pages 52–59. Springer, 2011.

[61] Merryn J Mathie, Branko G Celler, Nigel H Lovell, and Adelle CF Coster. Classification of basic daily movements using a triaxial accelerometer. *Medical and Biological Engineering and Computing*, 42(5):679–687, 2004.

[62] R. Meng, D. W. Grömling, R. R. Choudhury, and S. Nelakuditi. RideSense: Towards Ticketless Transportation. In *2016 IEEE Vehicular Networking Conference (VNC)*, 2016.

[63] 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*, 2015.

[64] Wolfgang Narzt, Stefan Mayerhofer, Otto Weichselbaum, Stefan Haselböck, and Niklas Höfler. Be-in/be-out with bluetooth low energy: Implicit ticketing for public transportation systems. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 1551–1556. IEEE, 2015.

[65] Tim W Nattkemper and Axel Wismüller. Tumor feature visualization with unsupervised learning. *Medical Image Analysis*, 9(4):344–351, 2005.

[66] Nintento. Pokemon Go. `https://pokemongolive.com/`. Accessed: 2022-01-15.

[67] Oura. Oura. `https://ouraring.com/`, 2019. Accessed: 2022-02-22.

[68] Shwetak N Patel, Thomas Robertson, Julie A Kientz, Matthew S Reynolds, and Gregory D Abowd. At the flick of a switch: Detecting and classifying unique electrical events on the residential power line (nominated for the best paper award). In *International Conference on Ubiquitous Computing*, pages 271–288. Springer, 2007.

[69] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[70] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys Tutorials*, 16(1):414–454, 2014.

[71] Schalk Wilhelm Pienaar and Reza Malekian. Human activity recognition using lstm-rnn deep neural network architecture. In *2019 IEEE 2nd wireless africa conference (WAC)*, pages 1–5. IEEE, 2019.

[72] Elias Pimenidis, Nikolaos Polatidis, and Haralambos Mouratidis. Mobile recommender systems: Identifying the major concepts. *Journal of Information Science*, 45(3):387–397, 2019.

[73] Purohit, A., Sun, Z., Pan, S., Zhang, P. Sugartrail: Indoor Navigation in Retail Environments without Surveys and Maps. In *10th annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 300–308, 2013.

[74] Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. Ahar: Adaptive cnn for energy-efficient human activity recognition in low-power edge devices. *IEEE Internet of Things Journal*, 2022.

[75] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using Mobile Phones to Determine Transportation Modes. *ACM Transactions on Sensor Networks (TOSN)*, 6(2), 2010.

[76] Tuukka Ruotsalo, Krister Haav, Antony Stoyanov, Sylvain Roche, Elena Fani, Romina Deliai, Eetu Mäkelä, Tomi Kauppinen, and Eero Hyvönen. Smartmuseum: A mobile recommender system for the web of data. *Journal of Web Semantics*, 20:50–67, 2013.

[77] Norma Saiph Savage, Maciej Baranski, Norma Elva Chavez, and Tobias Höllerer. I'm feeling loco: A location based context aware recommendation system. In *Advances in Location-Based Services*, pages 37–54. Springer, 2012.

[78] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pages 4–11, 2014.

[79] C. Sarkar, J. J. Treurniet, S. Narayana, R. V. Prasad, and W. de Boer. SEAT: Secure Energy-Efficient Automated Public Transport Ticketing System. *IEEE Transactions on Green Communications and Networking*, 2(1), 2018.

[80] Faouzi Sebbak, Farid Benhammadi, Abdelghani Chibani, Yacine Amirat, and Aicha Mokhtari. Dempster–shafer theory-based human activity recognition in smart home environments. *annals of telecommunications-annales des télécommunications*, 69(3):171–184, 2014.

[81] Glenn Shafer. *A mathematical theory of evidence*, volume 42. Princeton university press, 1976.

[82] Aman Shenoy and Ashish Sardana. Multilogue-net: A context aware rnn for multi-modal emotion detection and sentiment analysis in conversation. *arXiv preprint arXiv:2002.08267*, 2020.

[83] Moonsun Shin, Woojin Paik, Byungcheol Kim, and Seonmin Hwang. An iot platform with monitoring robot applying cnn-based context-aware learning. *Sensors*, 19(11):2525, 2019.

[84] Diogo Vinícius de Sousa Silva, Renato de Santana Silva, and Frederico Araújo Durão. Recstore: Recommending stores for shopping mall customers. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, pages 117–124, 2017.

[85] Nikhil Kumar Singh and Koduru Sriranga Suprabhath. Har using bi-directional lstm with rnn. In *2021 International Conference on Emerging Techniques in Computational Intelligence (ICETCI)*, pages 153–158. IEEE, 2021.

[86] SIRI. SIRI Standard. `http://www.transmodel-cen.eu/standards/siri/`, 2020. Accessed: 2020-10-07.

[87] Timothy Sohn, Alex Varshavsky, Anthony LaMarca, Mike Y. Chen, Tanzeem Choudhury, Ian Smith, Sunny Consolvo, Jeffrey Hightower, William G. Griswold, and Eyal de Lara. Mobility Detection using Everyday GSM Traces. In *International Conference on Ubiquitous Computing*. Springer, 2006.

[88] Statista. Mobile Gaming Statistics. `https://www.statista.com/topics/1906/mobile-gaming/#dossierKeyfigures`. Accessed: 2022-01-15.

[89] Akara Supratak, Hao Dong, Chao Wu, and Yike Guo. DeepSleepNet: A Model for Automatic Sleep Stage Scoring based on Raw Single-channel EEG. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(11), 2017.

[90] Punnarumol Temdee and Ramjee Prasad. *Context-aware communication and computing: Applications for smart environment*. Springer, 2018.

[91] Kia Teymourian, Olga Streibel, Adrian Paschke, Rehab Alnemr, and Christoph Meinel. Towards semantic event-driven systems. In *2009 3rd International Conference on New Technologies, Mobility and Security*, pages 1–6. IEEE, 2009.

[92] Moshe Unger. Latent context-aware recommender systems. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 383–386, 2015.

[93] Salvatore Vanini, Francesca Faraci, Alan Ferrari, and Silvia Giordano. Using Barometric Pressure Data to Recognize Vertical Displacement Activities on Smartphones. *Computer Communications*, 87, 2016.

[94] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[95] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep Learning for Sensor-based Activity Recognition: A Survey. *CoRR*, arXiv:1707.03502:10 pages, 2017.

[96] X. Wang, L. Kong, T. Wei, L. He, G. Chen, J. Wang, and C. Xu. Vld: Smartphone-assisted vertical location detection for vehicles in urban environments. In *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2020.

[97] Xiao Hang Wang, D Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. In *IEEE annual conference on pervasive computing and communications workshops, 2004. Proceedings of the second*, pages 18–22. Ieee, 2004.

[98] Bartosz Wieczorek and Aneta Poniszewska-Marańda. Towards the creation of be in/be out model for smart city with the use of internet of things concepts. In *International Conference on Service-Oriented Computing*, pages 156–167. Springer, 2019.

[99] Roel Wieringa. Design science as nested problem solving. In *Proceedings of the 4th international conference on design science research in information systems and technology*, pages 1–12, 2009.

[100] M. Won, A. Mishra, and S. H. Son. HybridBaro: Mining Driving Routes Using Barometer Sensor of Smartphone. *IEEE Sensors Journal*, 17(19), 2017.

[101] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. Deepsense: A Unified Deep Learning Framework for Time-series Mobile Sensing Data Processing. In *26th Inter. Conf. on World Wide Web*, 2017.

[102] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Deep metric learning for person re-identification. In *2014 22nd international conference on pattern recognition*, pages 34–39. IEEE, 2014.

[103] Zhiwen Yu, Xingshe Zhou, Daqing Zhang, Chung-Yau Chin, Xiaohang Wang, and Ji Men. Supporting context-aware media recommendations for smart phones. *IEEE Pervasive Computing*, 5(3):68–75, 2006.

[104] Y. Zheng, G. Shen, L. Li, C. Zhao, M. Li, and F. Zhao. Travi-Navi: Self-Deployable Indoor Navigation System. *IEEE/ACM Transactions on Networking*, 25:2655–2669, 2014.

[105] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. Understanding Mobility based on GPS Data. In *10th International Conference on Ubiquitous Computing*, 2008.

[106] Jin Peng Zhou, Zhaoyue Cheng, Felipe Pérez, and Maksims Volkovs. Tafa: Two-headed attention fused autoencoder for context-aware recommendations. In *Fourteenth ACM Conference on Recommender Systems*, pages 338–347, 2020.

Part II

Papers

Paper 1

Automated Product Localization Through Mobile Data Analysis

Automated Product Localization Through Mobile Data Analysis

Magnus Oplenskedal[1,3], Amir Taherkordi[1,2,3], Peter Herrmann[1]

[1]Norwegian University of Science and Technology (NTNU), Trondheim, Norway

[2]University of Oslo, Norway

[3]Forkbeard Technologies, Oslo, Norway

{magnukop, amirhost, herrmann}@ntnu.no

Abstract

Recent developments in the field of indoor Real-Time Locating Systems (RTLS) using mobile devices stimulate decision support for users. For instance, smartphone-based navigation in shops can enable location-aware recommendations of certain products to customers. An impeding factor to realize such systems is that they need the exact position of products. Existing product localization solutions, however, are based on tagging or manual location registering which tend to be quite costly and laborious. In this paper, we propose an *automated product localization* approach solving this problem. Our system infers the location of products based on the results of accumulating two sets of customer data, i.e., the locations at which the customers stop for picking up items as well as the list of the items, they purchase. These two data sets are accumulated for a large number of users, making it possible to build correct mappings between the products and their positions. We introduce a basic version of our localization algorithm and two extensions. One helps to improve calculating the position of relocated products while the other one fosters a faster localization using a smaller number of user data sets. We discuss the results of various simulation runs which give evidence that our system has a good potential to work in practice.

Keywords— Automated Localization, Real-Time Location Sensing, Intelligent Data Analysis, Mobile Services, Dynamic Leaky Accumulation, Softmax-based Inference.

1 Introduction

In recent years, the rapid development of mobile technology has created unprecedented opportunities to realise intelligent environments. In particular, the wide presence of smartphones in our daily life with their powerful sensors and processing capabilities creates new opportunities to support the users with context-aware systems [1]. These types of applications use information gathered from the users' environments to support their decision making process in various fields, such as mobile recommendations. A *Mobile Recommender System* (MRS) analyzes information retrieved from the environment in which a user is moving through, and uses the analysis results to provide meaningful suggestions [2]. The recommendations can be made based on the information collected from the surrounding context and the user behavior.

A key element of context information for an MRS is the real-time position of the user, which is basically the current location of her/his smartphone. For context-aware outdoor systems, global navigation satellite systems like GPS provide this information, e.g., the pathfinding algorithm of Google Maps [3]. Until recently, the development of indoor MRS was hampered due to the lack of accurate smartphone-based indoor positioning technology. However, new indoor *Real-Time Locating Systems* (RTLS) are emerging that are able to detect the exact location of a user's smartphone with an accuracy of a few centimeters. For instance, the ultrasound-based Forkbeard technology [4] is a novel RTLS solution promising to locate a smartphone with a precision of less than 10 *cm*.

A typical MRS application using an indoor RTLS is navigation support in shops helping customers to find products as well as giving them useful recommendations. Similar to the navigation systems in vehicles, customers are provided with a path that guides them to the places where desired goods are stored. The realization of such a system, however, requires the accurate position of products, while in many shops this type of information is not available yet. The reason for that is that only two solutions seem feasible [5] that are both not optimal: One possibility is to attach special computer-readable tags to the products in a store and use them to let the RTLS track their exact positions. The provision of the products with such tags, however, is a major cost factor. The other solution is to register and update the position of the products manually. Yet, this needs a lot of human effort, in particular, if products are regularly relocated which is the case in many grocery stores [6]. Considering indoor localization, existing work is mostly focused on coarse-grained indoor navigation in stores, e.g., to guide the customer by means of different sensor types on the user's smartphone and basic localization technologies (e.g., signal strength measured at various places) [6–9].

In this paper, we aim at proposing a cost-efficient approach to automatically locate products in retail stores. Instead of tagging products or registering their positions manually, product localization is performed using two sets of customer data: the set of positions at which the customer stops while shopping, and the list

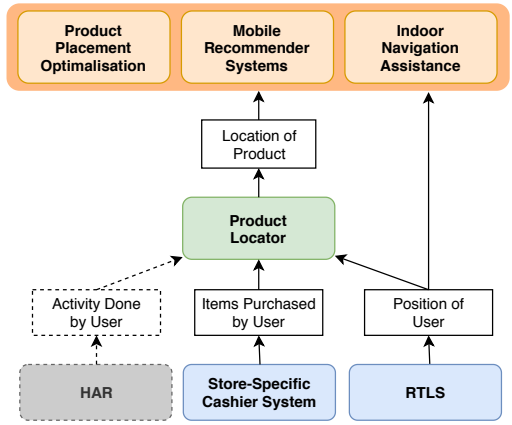

Figure 1: The scope of Product Locator: from processing customer input data to potential applications

of items purchased by her/him. Then, we accumulate these data pairs over a large number of customers. Since the customers can purchase products only by passing through the location of products, the accumulation reveals distinct correlations between items and places at which their buyers stop. These correlations are utilized to infer the positions of the goods. To realize this, we propose a basic *Customer Score Accumulation* algorithm to accumulate customer data and to infer product locations based on the accumulation results. Further, we introduce two extensions of the algorithm (i.e., *Leaky Customer Score Accumulation* and *Softmax-based Inference*) to improve position calculation for relocated products, respectively to reduce the number of customers needed to infer the location of products.

Considering a significant number of errors in the customer data sets, e.g., stopping at places without picking up items, our simulation-based evaluation results show that 99.9% of the 8,000 products in a typical large Norwegian grocery store [10] can be correctly located after aggregating the data of around 12,000 customers. Assuming 1,000 customers a day, our algorithm needs to run about 12 days to calculate the location of items, which seems practically feasible.

The rest of this paper is organized as follows: In Sect. 2, we give an overview of our aggregation and inference approach and discuss its scope. Thereafter, we present the approach in Sect. 3 followed by introducing both, the simulation technology used and the results revealed by the various test runs in Sect. 4. The article is completed with a discussion of related work in Sect. 5 followed by some concluding remarks in Sect. 6.

2 Overview and Scope

The scope of our approach is shown in Fig. 1. Intelligent location aware systems, such as mobile recommender systems, product placement optimization, and customer guidance, rely on the current position of users that can be obtained using an RTLS. As mentioned in the introduction, MRS also needs the location of the offered products which can be retrieved with our *Product Locator*. For a large number of users, the Product Locator collects data sets consisting of the places at which the customers stop while passing through the store as well as the list of items, they purchase. The customer stops are deduced from the RTLS data, while the list of purchased items can be obtained from the cashier systems or special apps provided by the stores. In addition, the activities done by the user during shopping can be a useful source of information for the Product Locator. In the rest of this section, we discuss these three sources of customer data and their relevance to our approach.

2.1 User Activity Recognition

Interpreting sensor inputs of mobile phones to find out certain activities is of growing importance, e.g., in health care [11]. This is usually done by *Human Activity Recognition* (HAR) systems that use pattern recognition algorithms to classify human activities from sensor data. Recently, the adoption of machine learning algorithms in HAR research have shown great results [12] such that HAR classifiers with an accuracy of greater than 90% can now be provided (see [13–16]).

The development of HAR systems, however, tends to be complex. In our context, it would be nice if one could detect whether a product is picked up or not from the movements sensed by the customer's smartphone. Yet, to allow an HAR system to learn the relation between sensor data and this activity, one needs example data sets for the activity "item pickup". Unfortunately, we could not find any openly available HAR data sets for this activity, and creating our own set would be a highly complex and laborious challenge. Therefore, we decided to use a much simpler way to recognize the picking up of items in a shop.

We simply record stops performed by customers while they are in the store. There are plenty of openly available datasets containing the activity "stop" and/or "standing" such that an HAR model trained on these datasets would provide the required stop locations. Using an RTLS, however, even a simpler approach might suffice: The stop activity is classified as "staying within a small area for a certain amount of time", e.g., staying in an area of one meter diameter for at least three seconds. Since customers perform those stops when they pick up products to buy, this classifier seems to be a good replacement for more complex activity recognitions.

Nevertheless, the price for the simplicity to just register stops instead of using more complex HAR mechanisms is a potentially larger number of errors in the

customer data sets. For example, customers can also stop at places at which they only look at certain products but decide not to buy them, or simply conduct other activities like checking their phones. Moreover, they may pick up products without stopping such that the location of a product pickup is not registered by the RTLS. The simulations carried out, however, revealed that our algorithm is sufficiently robust against these errors. The only effect is that the number of customers needed to locate all products correctly is slightly growing but, the errors do not lead to false localizations (see Sect. 4.2).

2.2 User Position Detection

Most existing indoor RTLS are not sufficiently precise for applications that need an accuracy at the level of less than a meter. They are based on different radio communication technologies such as UWB, RFID, Bluetooth, Ultrasound, Visible Light, SigFox, and LoRA. Among these, Ultrasound-based systems seem to be particularly promising since they are less affected by interference from metallic objects than electromagnetic methods. They are less influenced by opaque objects in the environment than optical technology while being cheaper to realize [17]. An RTLS usually comprises of fixed units located at particular points in a closed room which receive wireless signals from tags or badges attached to persons or objects of interest. The units measure the arrival times of a signal which vary due to the different distances between the units and a tag transmitting the signal. From the travel time spread and the knowledge about the locations of the fixed units, the position of the signal transmitters can then be computed using triangulation.

A practical and efficient solution to locate a user in a confined area such as a retail store is to utilize the microphone of his/her smartphone as a localization tag. This is, for instance, done by the Forkbeard technology [4]. It uses fixed units which are usually installed at the ceiling of a room and emit ultrasound signals in precisely coordinated intervals. The various signals are received by smartphones in the environment which measure the time lags between the signals. Further, the fixed units send their exact positions in the room such that the smartphone can triangulate its own position. This technology promises to reach a precision level of at least 10 *cm*. For retail stores, this precision is more than enough since a location accuracy of around 30 *cm* is sufficient to differentiate between different product-containing compartments in the store.

The user stop detection mechanism is realized as follows: An app in a customer's smartphone measures constantly its position using the RTLS. If the phone rests for a certain time in an area as discussed in Sect. 2.1, this is classified as a stop and the position is stored in a data base. In this way, all places at which the customer stops while shopping are retained. When the shopping activity is finished, the data set will be sent in anonymous form (to preserve the user's privacy) to the server running the Product Locator.

2.3 List of Purchased Items

The second data set to be used by the Product Locator is the list of all items bought by the customer in the shopping environment. A way to achieve this is to let the customer's smartphone retrieve the list from the cash register used for paying. An alternative is to use special customer apps. Lately, a trend among grocery stores has been to provide such apps giving additional digital experiences to their customers, e.g., systems providing bonuses, sales or discounts based on products purchased [18, 19]. These apps register which items each customer buys every time she/he visits the store. This information can be easily used to create the list of products purchased by a customer during a single store visit. Further, it is simple to combine this list with the set of stops also registered in the smartphone. From a judicial point of view, this solution is also helpful since customers who do not want their purchases being stored for inferring product locations, can simply switch of the tracking functionality or avoid to use this app at all.

3 Product Locator

Before describing the product localization approach in detail, we give a short introduction to the various designators used in this section. We define a particular indoor *Environment* of interest as E. The space covered by E is restricted by artificial boundaries in which a finite set of positions and items can be found. The set of all unique *Positions* in E is described by the set P while I refers to all unique *Items* available in E. The positions of the various items in E are described by the *Localization* function $L : [I \to P]$ that maps each item to its actual position. L is not an one-to-one mapping such that a position p can be assigned to several items.

Our approach accumulates the data of a large number of customers $C = \{c_1, c_2, \ldots\}$ moving in E over time. For a single customer $c \in C$, we define $P_c \subseteq P$ as the trajectory of positions, c passes while moving through E for shopping. The set $S_c \subseteq P_c$ is the set of positions at which the RTLS registers *Stops* for c. Further, we define $I_c \subseteq I$ as the set of items purchased by customer c during a traversal through E. Based on that, we can now define for a customer c the pair of *data sets* $ds_c \triangleq \langle S_c, I_c \rangle$ that will be utilized by the Product Locator.

In the following, we describe how the user data sets ds_c are analyzed and processed in order to obtain the localization mapping L for all items $i \in I$ in E. Our inference algorithm consists of three steps:

1. Calculate the score for each customer $c \in C$ from her/his data set $ds_c \triangleq \langle S_c, I_c \rangle$.
2. Accumulate the scores of all users in C and store the result in a data store.
3. Infer mapping $L : [I \to P]$ from the computation in step 2 by comparing the scores for each item that it is placed at a certain location, and return the position that has the highest score.

The three steps are described below. Thereafter, we introduce two extensions of the score accumulation step. The one is the *Leaky Score Accumulation* algorithm that weights newer customer data higher than older. This makes the algorithm more flexible against moving certain items to other positions in a store. The second improvement of the score accumulation is *Softmax-based Inference*. It uses the Softmax function [20] which makes the inference of correct item-to-position mappings faster and more reliable.

To clarify the various aspects of our Product Locator algorithm, we use a simple scenario. In a grocery store, bread is at position p_1, milk at p_2, eggs at p_3, and cheese at p_4. Further, we assume a user buying milk, bread, and eggs while a second one purchases only milk and a third one milk, bread, and cheese. For simplicity, we assume that all three customers stop at all the positions at which their purchased products are, and at no other positions. We can formalize that by using the sets $C = \{c_1, c_2, c_3\}$ referring to the three customers, $I = \{i_1 \text{ (milk)}, i_2 \text{ (bread)}, i_3 \text{ (eggs)}, i_4 \text{ (cheese)}\}$ describing the four products, and $P = \{p_1, p_2, p_3, p_4\}$ representing the four positions of the products. The data sets for our three customers can then be defined as follows:

$$ds_{c_1} \triangleq \langle\{p_1, p_2, p_3\}, \{i_1, i_2, i_3\}\rangle, \quad ds_{c_2} \triangleq \langle\{p_2\}, \{i_1\}\rangle,$$
$$ds_{c_3} \triangleq \langle\{p_1, p_2, p_4\}, \{i_1, i_2, i_4\}\rangle$$

3.1 The Basic Algorithm

Let us assume that our indoor environment E comprises n different positions $P \triangleq \{p_1, \ldots, p_n\}$ at which products can be stored and offers m different items $I \triangleq \{i_1, \ldots, i_m\}$ for sale.

3.1.1 User Score Calculation

In the first step, we take the data gathered for a customer c and stored in form of the data set $ds_c \triangleq \langle S_c, I_c\rangle$. The set ds_c is transformed into a matrix M_c that contains a row for each item and a column for each position. Using m items and n positions, this matrix has then the size $m \times n$. In M_c, we mark all matrix elements considering an item purchased by c and a position at which c stopped as 1 and else as 0:

$$M_{c_{xy}} \triangleq \begin{cases} 1 & \text{if } i_x \in I_c \wedge p_y \in S_c \\ 0 & \text{else} \end{cases}$$

For the three customers in our example, this leads to the three matrices M_{c_1}, M_{c_2}, and M_{c_3} depicted in Fig. 2.

c_1	p_1	p_2	p_3	p_4
i_1	1	1	1	0
i_2	1	1	1	0
i_3	1	1	1	0
i_4	0	0	0	0

c_2	p_1	p_2	p_3	p_4
i_1	0	1	0	0
i_2	0	0	0	0
i_3	0	0	0	0
i_4	0	0	0	0

c_3	p_1	p_2	p_3	p_4
i_1	1	1	0	1
i_2	1	1	0	1
i_3	0	0	0	1
i_4	1	1	0	1

Figure 2: Matrices of the three customers c_1, c_2 and c_3 in the example.

3.1.2 Accumulation of Customer Scores

In this step, the customer matrices M_{c_i} are accumulated to a *Data Store* matrix DS which can be achieved by simple matrix addition:

$$DS = \sum_{c \in C} M_c$$

For our example, the following matrix is computed:

$$DS = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \tag{1}$$

An advantage of the algorithm is that not all customer scores have to be received when running it. Instead, DS can be calculated based on just the currently available customer data. When data from a new customer c_{new} arrives, DS can be augmented by adding the corresponding customer score matrix $M_{c_{new}}$ to its previous version.

3.1.3 Inference of the Localization Mapping

The final step of the algorithm is inferring the locations of the items in the indoor environment from the data in matrix DS. To achieve that, we attach to each item the position that according to DS has the highest value.

Thus, we can define the localization mapping $L : [I \to P]$ as follows:

$$L[i_x \in I] \triangleq \text{choose } p \in P | \exists y \in \{1, \ldots, n\} : p = p_y \wedge \\ \forall l \in \{1, \ldots, n\} : DS_{xy} \geq DS_{xl}$$

If a product is placed at several positions in the store, we will locate only one of them which, however, is sufficient for indoor navigation and recommendation assistance.

Looking at our example data store DS (see eq. 1), we can only unambiguously infer that item i_1 is at position p_2 while the exact positions of the other items

are not clear-cut after considering just three customers. The inferences would be getting unambiguous if some more customer results were considered. Nevertheless, we will see in Sect. 3.3 that there are improvements to the accumulation process possible that allow us to infer the correct positions even if only our three customers are considered.

3.2 Leaky Customer Score Accumulation

The goal of this optimization of the customer score accumulation is to reduce the *bias*, the system has for older data compared to newly collected customer scores. The data registered for a product in DS, i.e., the value of element e_{xy} for an item x at the position y, will become very large over time as more customers purchase the product, and to be able to do so, stop at its position. This can be seen as positive as long as the position of the product is never moved. If products, however, are regularly relocated which happens often in grocery stores [6], the algorithm needs quite long until inferring the correct position again. The reason is that many customer scores considering the new location of the item have to be accumulated until those considering the old place are outpaced. The *Leaky Customer Score Accumulation* algorithm allows us to mitigate this problem.

We define the so-called *leaky factor* α with $0 < \alpha < 1$. When we now add the score matrix M_c of a new customer c to the data store matrix DS, we reduce all the elements of DS referring to items bought by c and places that she/he did not visit. Using DS^{old} to describe the value of DS before adding M_c and DS^{new} for the one afterwards, we can express this operation as follows:

$$DS_{xy}^{new} \triangleq \begin{cases} DS_{xy}^{old} + M_{c_{xy}} & \text{if } M_{c_{xy}} \neq 0 \\ DS_{xy}^{old} \cdot \alpha & \text{if } M_{c_{xy}} = 0 \wedge \\ & \quad \exists l \in \{1, \ldots, n\} : M_{c_{xl}} \neq 0 \\ DS_{xy}^{old} & \text{else} \end{cases}$$

The intuition for this improvement is that for all items $i \in I_c$ purchased by c, the correct position for the items will most likely be among the positions $p \in S_c$ at which the customer stopped. Based on this assumption, all matrix elements in DS that describe products in I_c, and positions not in S_c, are less likely to refer to correct product locations. Thus, it is useful to reduce the values of these matrix elements. If an item is now moved to another place, there will be several customers buying it but not stopping at its old location anymore. Thus, the value for the old position will decline relatively quickly such that it will be faster passed by the value for the new location of the product.

Finding the right balance between prioritizing new over old data was done by empirical experimentation. The value $\alpha = 0.985$ achieved the best results in our tests.

The simulation of different values of α inspired us to a further improvement that we call the *Dynamic Leaky Score Accumulation*. Here we use a *confidence level*

to indicate our trust in the data of a single customer c that is allocated her/his own leaky factor α_c. For instance, a customer buying one item and stopping at one location probably provides more accurate data than a customer stopping at 20 locations to buy a single product. Thus, the confidence level is based on two parameters, the number of items purchased, i.e., $|I_c|$, and the relation between the number of items purchased and number of stops performed by the customer, i.e., $\frac{|I_c|}{|S_c|}$. Our simulation runs showed the following dynamic leaky factor α_c to provide good results:

$$\alpha_c = \begin{cases} 0.750 & \text{if } 5 < |S_c| \leq 10 \wedge \frac{|I_c|}{|S_c|} \geq 0.5 \\ 0.650 & \text{if } |S_c| \leq 5 \wedge \frac{|I_c|}{|S_c|} \geq 0.5 \\ 0.985 & \text{else} \end{cases}$$

Thus, customers buying few products and stopping at most twice as often as the number of items purchased, are granted greater influence than other ones.

3.3 Softmax-based Inference

The intuition behind the second improvement strategy for the score accumulation in Sect. 3.1.2 is to amplify the differences between the values of the elements in our data store DS. Moreover, we want to consider the general numbers of stops at a certain position to evaluate the likelihood that a certain product resides there. Look for instance on the third row of the data store DS in our example (see eq. 1). There we have the value 1 in all of the first three columns such that we cannot claim a direct winner. On the other hand, the sums of the values in the first two columns of DS are much larger than the sum of the values in the third column. This indicates that fewer people not procuring item i_3 stopped at position p_3 than at p_1 or p_2 which makes it more likely that p_3 is, indeed, the place where i_3 is placed. The extension to the score accumulation algorithm presented here follows this consideration. It is based on the *Softmax* function (see, e.g., [20]) and will be carried out in three steps:

1. To amplify the more significant relations between items and positions, we transform matrix DS to DS^{exp} that uses exponential values. A problem to be solved is that the elements of DS may contain large numbers when the inputs of many users are stored such that computing the exponential value leads to an arithmetic overflow. To avoid that, we simply subtract the value ds_{max} of the largest matrix element in DS from all elements. Thus, no element will have a value larger than 0 and arithmetic overflow is prevented. The matrix DS^{exp} is computed as follows:

   $$DS^{exp}_{xy} \triangleq e^{DS_{xy} - ds_{max}}$$

   The use of the exponential value makes it possible to get rid of the zeros in data store DS since a product might be located in a position not yet visited.

DS^{exp}

	p_1	p_2	p_3	p_4
i_1	0.37	1.00	0.14	0.14
i_2	0.37	0.37	0.14	0.14
i_3	0.14	0.14	0.14	0.05
i_4	0.14	0.14	0.05	0.14

DS^{row}

	p_1	p_2	p_3	p_4
i_1	0.22	0.61	0.08	0.08
i_2	0.37	0.37	0.13	0.13
i_3	0.30	0.30	0.30	0.11
i_4	0.30	0.30	0.11	0.30

DS^{sm}

	p_1	p_2	p_3	p_4
i_1	0.18	0.39	0.13	0.13
i_2	0.31	0.23	0.21	0.21
i_3	0.25	0.19	0.48	0.18
i_4	0.25	0.19	0.18	0.48

Figure 3: The three matrices DS^{exp}, DS^{row}, and DS^{sm} of our example.

2. Thereafter, we perform the last step of the Softmax function by normalizing the rows of the exponential matrix DS^{exp}. This is done in order to find, for each product, the probability distribution based on the stops performed by customers buying this product. The matrix reached in this step is named DS^{row} and computed as follows:

$$DS^{row}_{xy} \triangleq \frac{DS^{exp}_{xy}}{\sum_{l=1}^{n} DS^{exp}_{xl}}$$

3. Finally, we normalize the columns of matrix DS^{row} such that the sum of the values in each column is 1. The resulting *Data Store* DS^{sm} is computed as follows:

$$DS^{sm}_{xy} \triangleq \frac{DS^{row}_{xy}}{\sum_{k=1}^{m} DS^{row}_{ky}}$$

Thus, we realize the consideration discussed above, that a product is more likely at a position if relatively few people not buying it stopped there.

Then we take matrix DS^{sm} instead of DS in the inference step described in Sect. 3.1.3.

Starting with matrix DS of our example (see eq. 1), the extended algorithm provides the matrices shown in Fig. 3. In contrast to the result of the basic algorithm in eq. 1, the adapted algorithm makes it possible to infer the mapping L unambiguously since every row in matrix DS^{sm} has a unique maximum value (marked in green). Thus, the mapping L from items to positions can be correctly determined:

$$L[i_1] = p_2, \quad L[i_2] = p_1, \quad L[i_3] = p_3, \quad L[i_4] = p_4$$

For i_1, i_3, and i_4, this localization mapping can even be inferred if we demand a difference of 0.1 between the highest and second highest value to prevent wrong localizations.

4 Evaluation

We evaluate the proposed approach through simulating the data sets. The issue is that real data traces for the discussed shopping use case are not currently available.

For that, the necessary RTLS hardware for high precision indoor location sensing must be provided, and the existing store apps on the smartphones of the customers have to be extended to provide the list of purchased items per shopping. To evaluate the core functionalities of our proposed solution, and to learn more about it in order to fine-tune it (e.g., selecting the correct leaky factor α, see Sect. 3.2), we carried out hundreds of simulations of various shop environments. To achieve that, we developed a suitable tool simulating indoor environments and the required customer data. For simulation, we considered data sets based on several distributions in order to cover many different realistic shopping scenarios. Therefore, we believe that the reported simulation results provide practical and relevant insights to our product localization solution. The first subsection describes the three modules of the simulation tool, in detail. Thereafter, we discuss the results of the evaluation runs.

4.1 Simulator

Our simulation tool consists of a creator for the shop environment, a creator for customer behavioral inputs, and the simulator core.

The *Environment Creator* module creates models of an indoor environment E to be simulated. For typical grocery stores in Norway, 8,000 items offered at 800 positions are realistic values to be used. The difference between the two numbers results from the fact that many products are pooled in shelves. From these inputs, the Environment Creator builds the sets I and P (see Sect. 3) as well as the data store matrix variable DS (see Sect. 3.1) with $|I|$ rows and $|P|$ columns.

The second module of the simulation tool is the *Customer Creator*. It is used to simulate customers moving through the simulated environment and purchasing the offered items. For each simulated customer c, the Customer Creator generates a random data set $ds_c = \langle S_c, I_c \rangle$ (see Sect. 3). To simulate realistic customer behavior, the demand for certain goods differs vastly. This reflects that some products like low-fat milk are bought much more often than others, for example, mustard with truffles. We also like the number of items purchased by the customers to be in line with normal shopping behavior. To solve these two requirements without having access to real data of a retail store as a foundation for the simulations, the Customer Creator uses a truncated gaussian distribution when selecting the number of items purchased by a customer. This distribution is also used to select with which likelihood the customers buy a certain item.

As discussed above, we need to consider errors in the data sets since customers may stop at positions without buying the goods available there, and they might pick certain products on the fly without being detected as a stop by the RTLS. To consider these errors in our simulations, the Customer Creator works as follows: When creating a new customer c, it first determines the number of items the customer purchases using the corresponding truncated gaussian distribution. Thereafter, it selects which items the customer buys using the other gaussian dis-

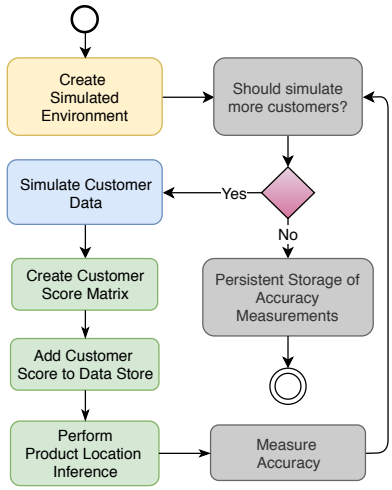

Figure 4: Flow chart illustrating the simulation tool

tribution. This leads to the set I_c, followed by querying the Environment Creator for the correct locations of the selected items. Using these data directly as S_c would produce a "perfect" data set for the Product Locator, i.e., the customer would stop at exactly the positions at which the purchased products are located.

This perfect user set is then altered by an error function. At first, a random number of additional locations from the simulated environment is selected and added to set S_c. This function follows a certain distribution that can be parameterized prior to a simulation run. At second, to simulate the event that the RTLS misses stops at places at which items are picked, elements of S_c from the perfect data set are removed following also a parameterizable function.

The third module is the *Simulator Core* which is responsible for initiating both the Environment Creator and the Customer Creator. Further, this module realizes the Product Locator and feeds it with the data sets created by the Customer Creator. To give meaningful information about the accuracy of the Product Locator during the learning phase, i.e., its ability to predict the correct location for the products, the core module compares the produced function L with the real positions in predefined intervals and stores this information.

The different steps performed during a simulation run are depicted in Fig. 4. The simulation is started by the Environment Creator building an indoor environment. If more customers shall be simulated, the Customer Creator creates a new data set that may contain errors followed by the three steps of the Product Locator (see Sect. 3.1). If the predefined interval is finished, moreover, the accuracy of the location mapping is checked and stored.

When all desired customers are simulated, the simulator terminates by storing

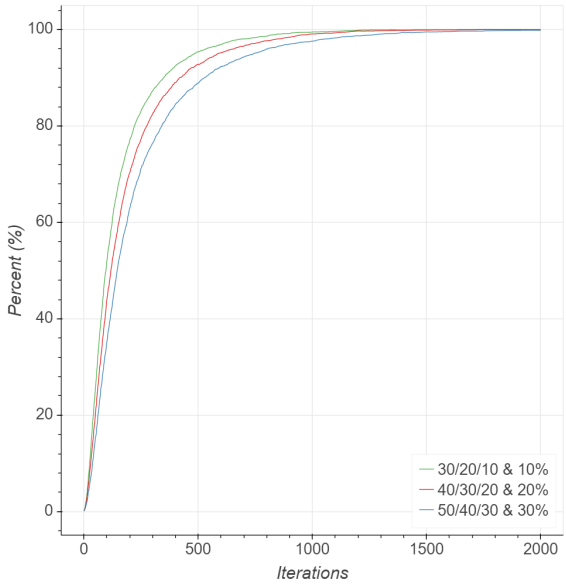

Figure 5: The increase in accuracy as customer data is added to the system depending on different error rates.

the accuracy measurements in a JSON file. This data can be used to evaluate our proposed solution. We report about some simulation results in the following.

4.2 Results

Our Product Locator was thoroughly tested through hundreds of simulation runs, wherein various store configurations were used to further analyze the system behavior in varying environments. To evaluate the quality of our solution, we use the *accuracy* metric *acc*. Be $I_{loc} \subseteq I$ the set of all items that can be unambiguously assigned a location, since in the rows of the data store DS_{sm} (see Sect. 3.3) the largest value is at least 0.1 larger than the second largest one. We then define the *accuracy* as $acc \triangleq \frac{|I_{loc}|}{|I|}$. The accuracy is measured and stored after adding the data of ten customers to the data store. Some of the simulation results are discussed in the following.

4.2.1 Testing the Basic Product Locator

First, we show the gradual increase in accuracy using what we claim to be "realistic" parameters. We simulated a shopping environment with 8,000 items, 800 positions, and 35,000 customers.

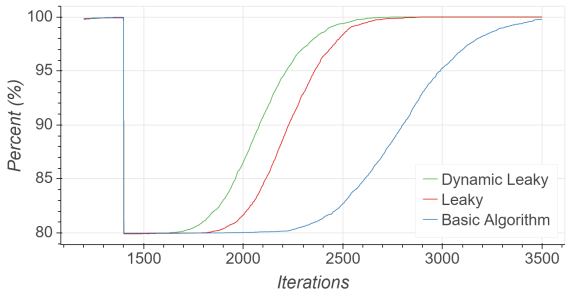

Figure 6: The time to recover after moving 20% of the items in the environment.

Besides stopping at the correct positions of the bought items, each customer can stop at various places without picking up any products. We use the so-called 30/20/10 likelihood pattern to select the number of such "erroneous" stops at which no products are picked up. It means that we choose a value between 0 and 10 erroneous stops with a probability of $\frac{4}{7}$. A number of stops in the interval between 11 and 20 of such stops is selected with a likelihood of $\frac{2}{7}$, i.e., half of the probability for the interval 0-10. With a likelihood of $\frac{1}{7}$, i.e., half of that for 11-20 stops, we select a value between 21 and 30 erroneous stops. When one of the three intervals has been chosen, the exact value within this interval is selected following a uniform distribution.

Further, we assume with a probability of 10% that a stop at a correct position is not noted by the RTLS. Then, we get the hyperbola depicted in Fig. 5 with the *green* curve. With the simulation parameters described above, an accuracy of 99% is achieved after $8,420$ customers, 99.9% after $12,170$, and finally, all products were correctly located after $18,630$ customers.

4.2.2 Increased Error Rates

The second group of simulations intends to fathom the robustness of our algorithm by vastly increasing the likelihoods of additional stops and not detected pickings of products. We added the curves of these tests also to Fig. 5. The *red* curve describes a likelihood of 40/30/20 for additional stops and 20% for not registering pick ups, and the *blue* one is even more extreme assuming probabilities of 50/40/30 resp. 30%. It is interesting that the increased error rates do not change the hyperbola forms of the curves. The Product Locator still locates all items in the simulated environment, albeit with a slightly greater delay.

4.2.3 Handling Relocated Products

The third group of simulations considers the behavior of our algorithm when products are relocated. Figure 6 shows the results from simulations in which 20% of the items in the store were moved to a new location. All three simulations shown in the figure use the simulation configuration with the "realistic" parameters introduced in Sect. 4.2.1. The *green* curve shows the results from the *Dynamic Leaky Accumulation* algorithm, described in Sect. 3.2. The result using plain *Leaky Accumulation* is plotted in as the *red* curve, while the *blue* one depicts the results when using only the base algorithm without any leaky accumulation. These curves clearly show that the *Dynamic Leaky Accumulation* has a clear advantage against the simple *Leaky Accumulation* and even more against the simple algorithm when we have to expect significant relocation of items during the lifetime of the Product Locator.

4.2.4 Comparing the Score Accumulation Variants

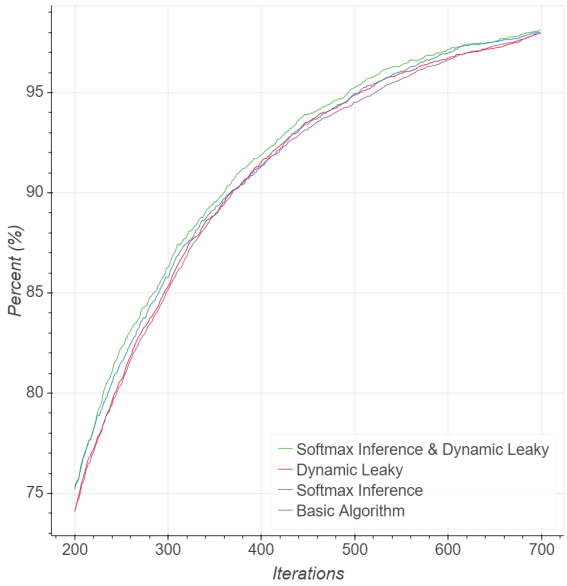

Figure 7: Results of using different score accumulation variants.

Finally, we compare the customer score accumulation of the basic variant (see Sect. 3.1.2) with the *Dynamic Leaky Customer Score Accumulation* (see Sect. 3.2), and the *Softmax-based Inference* (see Sect. 3.3) for our "realistic" shop scenario without product relocations.

The *green* curve in Fig. 7 describes the behavior when using both Dynamic Leaky Customer Score Accumulation and Softmax-based Inference, while the *red* refers to customer score accumulation through Dynamic Leaky Customer Score Accumulation alone. Softmax-based Inference alone, is shown as a *blue* curve, and the basic algorithm is provided in *purple*. The curves reveal that the differences between the four scenarios are minimal. Just with relatively few customer data sets like in the example used in Sect. 3, the Softmax-based extension gives a better accuracy score which becomes nearly irrecognizable when the data sets of more than 15,000 customers are considered. On the other hand, Softmax-based Inference has an advantage for localizing items that are rarely purchased. Thus, if a store has a fair number of such items, Softmax-based inference is more efficient than the basic algorithm.

5 Related Work

Being an important enabler for Mobile Recommender Systems (MRS), a fair amount of research has been conducted on indoor navigation systems. With respect to locating products in stores, most existing technologies propose manual tagging or registration of product locations performed by employees or system experts. To our best knowledge, previous work has neither been done on the automated localization of products in large stores nor on preserving the accurate location information when products are subject to relocation. Below, we discuss existing work of special interest.

Purohit et al. developed SugarTrail [6], a system providing the location of items and guidance to them in indoor environments without depending on existing maps. SugarTrail aggregates movement paths registered from users carrying mobile nodes utilizing magnetometers in addition to collecting data from stationary radios while traversing the indoor environment. The aggregated paths are used to construct Virtual Road Maps, which, in turn, can provide navigation assistance to items for customers in the store. The main contribution of SugarTrail is indoor navigation for stores, which is different from the goal of this paper, i.e., product localization.

Some other approaches leverage technologies such as computer vision, sensors and RSS to locate points of interests in indoor environments. Travi-Navi [7] combines high quality images and sensor readings from a *Guiders* smartphone and packs them into a navigation trace. This can be done, e.g., by a shop owner to provide navigation assistance to their stores. While moving through the indoor environment, the *followers* (customers) can follow the navigation trace defined by the Guider to the point of interest. In Canoe [8], the Received Signal Strength (RSS) is measured in various parts of a shop. Then, the observed RSS values are compared for directing users to points of interest. In Shopper Observer [9], the Redpin indoor localization framework which allows its users to create virtual fingerprints of locations, is used to find paths of customers in a shop which can be utilized, e.g., for product placement. In these approaches, the precision of localiza-

tions is lower. Therefore, they are better suited for scenarios where only relatively coarse-grain localization is needed.

In [21], Chia-Chen Chen et al. use RFID readers on smart devices together with RFID tags on products to recommend products to users according to their preferences, previous purchasing records, and current location. The recommendation mechanism works by a K-means algorithm, clustering customers based on their shopping behaviors. Augmented Reality (AR) is deployed in [22] to recommend healthy foods in grocery stores. This approach utilizes the camera on customer smartphones to both localize the customer within the store, and to display AR overlay *tagging* recommended products. The authors mention that product information is available in electronic product databases, however, not necessarily associated with their locations. This category of work is mainly focused on algorithms and techniques on intelligent product recommendation. In our view, the Product Locator proposed in this paper can be a good add-on to these approaches since it creates precise information about the position of products that can be leveraged to provide more intelligent user recommendations.

6 Conclusion and Future Work

The paper presents an algorithm for product location detection in indoor environments. Our approach only requires the trajectory of stops and items purchased by customers while they move through an environment. The data collection can be done during the normal shopping routines of customers without any other participation than installing an app on their smartphones.

We developed a basic algorithm consisting of score calculation for individual customers, accumulation of the scores of various customers, and inference of product locations from the accumulated scores. Furthermore, we propose improvements to the score accumulation algorithm by *Static* and *Dynamic Leaky Accumulation* as well as *Softmax-based Inference*. Evaluating the various aspects of the algorithm, we found out that the *Dynamic Leaky Accumulation* can be very helpful in scenarios with relocated goods while *Softmax-based Inference* is more efficient when rarely purchased items have to be located and if only a relatively small number of customer data sets are available.

The next step is to create a simulator that considers the spatial aspects of a shopping environment. Using retail data sets, we plan to create a simulated shop layout and let customers "roam" through it simulating the trajectories performed while shopping. In this way, we can simulate varying customer behaviors, e.g., hasty purchases on the way home or cozy strolling being inspired by the available products. Also spatial aspects like the contorted ways performed when searching a hidden product can be simulated in this way. Thus, the extended simulator should allow us to test our approach with more realistic customer data.

Finally, when the Forkbeard technology is fully implemented, a prototype installation with a Norwegian grocery store operator is planned. This will make it

possible to find out even more about user activity recognition, in particular, how long the waiting time of a user in an area should be in order to classify that as a stop. Moreover, we will investigate the accuracy of the assumptions we made in the simulations about the likelihood of errors in stop detection.

References

[1] Ö. Yürür, C. H. Liu, Z. Sheng, V. C. M. Leung, W. Moreno, and K. K. Leung, "Context-awareness for mobile sensing: A survey and future directions," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 68–93, 2016.

[2] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems: Introduction and Challenges*. Boston, MA, USA: Springer US, 2015, pp. 1–34.

[3] R. Gibson and S. Erle, *Google Maps Hacks*. O'Reilly Media, 2006.

[4] "Forkbeard Technology," https://www.sonitor.com/forkbeard/, accessed: 2018-12-04.

[5] P. Bolliger, "Redpin - Adaptive, Zero-configuration Indoor Localization Through User Collaboration," in *First ACM International Workshop on Mobile Entity Localization and Tracking in GPS-less Environments (MELT)*. San Francisco, CA, USA: ACM, 2008, pp. 55–60.

[6] Purohit, A., Sun, Z., Pan, S., Zhang, P., "Sugartrail: Indoor Navigation in Retail Environments without Surveys and Maps," in *10th annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2013, pp. 300–308.

[7] Y. Zheng, G. Shen, L. Li, C. Zhao, M. Li, and F. Zhao, "Travi-Navi: Self-Deployable Indoor Navigation System," *IEEE/ACM Transactions on Networking*, vol. 25, pp. 2655–2669, 2014.

[8] K. Dong, H. Ye, W. Wu, M. Yang, Z. Ling, and W. Yu, "Canoe: An Autonomous Infrastructure-Free Indoor Navigation System," *Sensors (Basel)*, vol. 17, no. 5, p. 996, 2017.

[9] M. Bourimi, G. Mau, S. Steinmann, D. Klein, S. Templin, D. Kesdogan, and H. Schramm-Klein, "A Privacy-Respecting Indoor Localization Approach for Identifying Shopper Paths by Using End-Users Mobile Devices," in *8th International Conference on Information Technology: New Generations*. Las Vegas, NV, USA: IEEE Computer, 2011, pp. 139–144.

[10] Norgesgruppen, "Innenfor flere varegrupper er matvareutvalget større i Norge enn i Sverige," https://www.norgesgruppen.no/presse/nyhetsarkiv, in Norwegian, accessed: 2019-01-27.

[11] M. N. K. Boulos, A. C. Brewer, C. Karimkhani, D. B. Buller, and R. P. Dellavalle, "Mobile Medical and Health Apps: State of the Art, Concerns, Regulatory Control and Certification," *Online Journal of Public Health Informatics*, vol. 5, p. 229, 2014.

[12] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep Learning for Sensor-based Activity Recognition: A Survey," *CoRR*, vol. arXiv:1707.03502, p. 10 pages, 2017. [Online]. Available: https://arxiv.org/abs/1707.03502

[13] Z. He and L. Jin, "Activity Recognition from Acceleration Data based on Discrete Consine Transform and SVM," in *IEEE International Conference on Systems, Man and Cybernetics*, 2009, pp. 5041–5044.

[14] Z.-Y. He and L.-W. Jin, "Activity recognition from acceleration data using ar model representation and svm," in *International Conference on Machine Learning and Cybernetics*, vol. 4, 2008, pp. 2245–2250.

[15] A. Khan, Y. Lee, and S. Lee, "Accelerometer's Position Free Human Activity Recognition using a Hierarchical Recognition Model," in *IEEE International Conference on e-Health Networking Applications and Services (Healthcom)*, 2010, pp. 296–301.

[16] O. Lara and M. Labrador, "A Survey on Human Activity Recognition using Wearable Sensors," *IEEE Communications Surveys & Tutorials*, vol. 15, pp. 1192–1209, 2013.

[17] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, "The Anatomy of a Context-Aware Application," *Wireless Networks*, vol. 8, pp. 187–197, 2002.

[18] Rema 1000, "Æ App," https://www.rema.no/ae/, in Norwegian, accessed: 2018-12-04.

[19] Trumf, "Trumf App," https://www.trumf.no/, in Norwegian, accessed: 2018-12-04.

[20] D. Heckerman and C. Meek, "Models and Selection Criteria for Regression and Classification," in *Thirteenth conference on Uncertainty in Artificial Intelligence (UAI)*. Providence, RI, USA: Morgan Kaufmann Publishers, 1997, pp. 223–228.

[21] C.-C. Chen, T.-C. Huang, J. J. Park, and N. Y. Yen, "Real-time Smartphone Sensing and Recommendations towards Context-awareness Shopping," *Multimedia Systems*, vol. 21, pp. 61–72, 2015.

[22] J. Ahn, J. Williamson, M. Gartrell, R. Han, Q. Lv, and S. Mishra, "Supporting Healthy Grocery Shopping via Mobile Augmented Reality," *ACM Trans Multimedia Comput Commun Appl*, vol. 12, p. 16:1–16:24, 2015.

Paper 2

DeepMatch: Deep Matching for In-Vehicle Presence Detection in Transportation

DeepMatch: Deep Matching for In-Vehicle Presence Detection in Transportation

Magnus Oplenskedal[1,3], Amir Taherkordi[1,2,3], Peter Herrmann[1]

[1]Norwegian University of Science and Technology (NTNU), Trondheim, Norway

[2]University of Oslo, Norway

[3]Forkbeard Technologies, Oslo, Norway

{magnukop, amirhost, herrmann}@ntnu.no

Abstract

A key feature of modern public transportation systems is the accurate detection of the *mobile context* of transport vehicles and their passengers. A prominent example is automatic in-vehicle presence detection which allows, *e.g.*, intelligent auto-ticketing of passengers. Most existing solutions, in this field, are based on either using active RFID or Bluetooth Low Energy (BLE) technology, or mobile sensor data analysis. Such techniques suffer from low spatiotemporal accuracy in in-vehicle presence detection. In this paper, we address this issue by proposing a deep learning model and the design of an associated generic distributed framework. Our approach, called DeepMatch, utilizes the smartphone of a passenger to analyze and match the event streams of its own sensors and the event streams of the counterpart sensors in an in-vehicle reference unit. This is achieved through a new learning model architecture using Stacked Convolutional Autoencoders for feature extraction and dimensionality reduction, as well as a dense neural network for stream matching. In this distributed framework, feature extraction and dimensionality reduction is offloaded to the smartphone, while matching is performed in a server, *e.g.*, in the Cloud. In this way, the number of sensor events to be transmitted for matching on the server side will be minimized. We evaluated DeepMatch based on a large dataset taken in real vehicles. The evaluation results show that the statistical accuracy of our approach is 0.978 for in-vehicle presence detection which, as we will argue, is sufficient to be used in, *e.g.*, auto-ticketing systems.

Keywords— Mobile Context, Sensor Event Streams Analysis, Deep Learning, Event Matching, Intelligent Transportation

1 Introduction

Within logistics and public transportation, there is a strong need for accurate and intelligent detection of mobile context situations of users, smart devices and vehicles. By *mobile context*, we refer to any kind of information that can be used to characterize spatiotemporal properties of a mobile entity [1]. An example is the position data of a moving vehicle making it possible to find out its current location. Public transportation providers in many countries (*e.g.*, in Northern Europe) are providing smartphone applications for their passengers, in which the user can, *e.g.*, purchase tickets and get route guidance. A key feature to enhance the next generation of these mobile context-aware applications is the integration of information about the presence of a passenger in a vehicle. An advantage of this extension is that it becomes easier to determine the exact flow of passengers between particular places. This makes it possible to plan optimal public transport networks in which passengers are offered rides when they need them and are brought to their destinations without the need to change vehicles often if at all. Further, peak hours can be detected and the supply of vehicles is optimized accordingly.

If the in-vehicle presence detection is highly accurate, we can also make the ticketing of passengers considerably simpler. In existing smartphone applications, the passengers have to remember buying tickets before starting a ride. Moreover, in order to buy the right ticket for the intended trip, they need to have fair knowledge about the ticketing system of the transport provider. In contrast, using a highly accurate in-vehicle presence detection solution, a so-called Be-In/Be-Out (BIBO) system [2], tickets can be automatically issued to the passengers based on the exact duration of their ridings on vehicles. Thus, they can conveniently enter and leave public transport vehicles without having to deal with the transportation provider in advance. If this seamless way of travel is accepted by many passengers, there is no need for cost intensive ticket checkpoints, ticket machines, and passenger controls anymore.

Existing solutions for in-vehicle presence detection are based on two different approaches. The first group applies communication systems such as Radio Frequency Identification (RFID) or Bluetooth Low Energy (BLE). While travelling, temporary connections are built up between the user's mobile device and certain fixed vehicle equipment through which evidence can be established that the passenger is within the vehicle. Prominent examples for such systems are EasyRide [3] and SEAT [4]. Approaches in the other group use event streams from smartphone sensors to analyze for certain properties. Modern smartphones are provided with a variety of sensors such as magnetometers, accelerometers, gyroscopes, GPS, and barometers which offer unprecedented opportunities to analyze mobile context information from the user's environment. Two prominent solutions for sensor-based

analysis are HybridBaro [5] and RideSense [6]. We will argue later that the accuracy of both categories above is still not high enough.

In this paper, we address the accuracy problem and propose a highly autonomous approach that can detect in-vehicle presence with a sufficient degree of precision. A central aspect of our approach is to equip each vehicle with a *reference device* (*e.g.*, an Android phone). This allows us to deduce the presence of passengers in the vehicle based on the match between the stream of events generated by sensors of the reference device and those measured by the sensors of the passengers' smartphones. We propose the system DeepMatch which provides a deep learning model to verify in-vehicle presence. The new learning architecture is based on Stacked Convolutional Autoencoders, used for feature extraction and dimensionality reduction. Matching is provided using a fully connected neural network. DeepMatch is a distributed framework, where feature extraction and dimensionality reduction is offloaded to the users' smartphones, while the matching process is performed in a server, *e.g.*, a cloud server. Using Stacked Convolutional Autoencoders, our model not only learns the most essential features of the sensed data stream (*i.e.*, encoded data), but also finds out which part of the stream can be omitted without deteriorating matching. The part of the model running on a server compares the input encoded data of a smartphone with that of the corresponding reference device and outputs a value indicating the probability of the two data sources being present in the same vehicle. The server can be realized as a centralized unit out of the vehicle. Alternatively, it can be locally installed in the vehicle, *e.g.*, in the reference device, or on a local router in the vehicle following the principle of fog computing [7].

The model is trained on real data traces gathered by a group of people using public transportation in two large Norwegian cities. The evaluation results show that the statistical accuracy, the so-called F1-score, of DeepMatch is 0.978 for in-vehicle presence detection outperforming the two well-known technologies Normalized Correlation by 4%, and Dynamic Time Warping by 16%. This can provide a significant advantage for practical in-vehicle presence detection as we will discuss later.

The rest of this paper is organized as follows. In Section 2, we discuss existing solutions followed the presentation of our in-vehicle presence detection approach in Section 3. In Section 4, the experimental evaluation results are reported. We conclude the paper with a discussion of our future plans in Section 5.

2 Related Work

As mentioned above, existing solutions for in-vehicle presence detection are based on either utilizing communication technologies or analyzing mobile sensor events. These approaches are presented in this section followed by the discussion of some recent works leveraging deep learning for mobile context detection.

2.1 Communication Technology-based Solutions

Early in-vehicle presence detection systems were based on active RFID tags, carried by the passengers, and a single communication unit in the center of vehicles. A contactless, mid-range radio-based identification and communication protocol was used for tracking. One of the first solutions was EasyRide [3], developed by the Swiss Railways Association. Allfa [8] is another RFID-based system, tested for half a year in busses, trams and trains in Dresden, Germany. Due to the weak transmitter strengths of the active tags, it is difficult to guarantee that all of them are detected in the vehicle. As discussed in [3], this affords a vast number of readers in the vehicle, at least one at each door. However, such approaches still suffer from lack of enough detection precision. For instance, Allfa has an accuracy rate of just 68% making it unsuitable for practical use.

Another category is based on BLE. Compared with active RFID approaches using battery-powered tags, BLE-based BIBO systems can utilize smartphones with additional monitoring options, the possibility to measure signal strengths for proximity determination, larger distribution channels, etc. The first BLE-based solution is proposed in [2], while [9] suggests a ticketing system adding a custom profile on top of the BLE to fulfill the payment procedure. In SEAT [4], a BLE-enabled smartphone communicates with devices installed in vehicles to track the journey for automatic pricing. Its main focus is on security, performance, and battery friendliness, but not on the accuracy of the in-vehicle presence detection. The authors of [2] are cautiously optimistic that BLE might work for BIBO systems. However, the chassis of a vehicle does not limit the accessibility of a BLE transmitter which makes it possible that somebody close to it, *e.g.*, a person in another vehicle, is wrongly detected. On the other hand, things in a vehicle may inhibit a BLE connection such that devices in the vehicle are not detected. This is confirmed by the authors of [10] when using BLE for indoor localization. While precise indoor location seems to be more complex than "just" finding out if somebody is in a vehicle, we expect similar accuracy problems.

2.2 Mobile Sensor Event Analytics-based Solutions

Existing work in this field analyzes the data stream of the sensors in user smartphones to detect mobile contexts in transportation. Our extensive experiments and the obtained results, reported in Section 4.4.1, show that the smartphone barometer is the only useful sensor for in-vehicle presence detection since the position and orientation of the phone as well as the movements of its carrier influence the measurements of other sensors. This is also confirmed by related approaches that mostly focus on analyzing the barometer data. In [11], Sankaran et al. demonstrate that the barometer can be applied to detect user activities of IDLE, WALKING, and VEHICLE at low-power through their context-detection algorithm, using four stages; pre-processing, jump detection, peak detection and walk detection. Likewise, in [12], user activities are classified using the barome-

ter sensor on smartphones. This approach leverages Bayesian networks, decision trees, and RNN as inference models to predict user action, *e.g.*, riding or leaving a cable-car. In [13], the authors demonstrate how the pressure data collected from a smartphone barometer can be utilized to accurately track driving patterns. An expansion is to correlate pressure time-series data sensed by the barometer against topographic elevation data and road maps for a given region. This allows a centralized server to estimate the possible routes through which users have driven. An example is HybridBaro [5], featuring a hybrid algorithm to adaptively utilize GPS data to increase the detection accuracy in flat areas. RoadSphygmo [14] uses the barometer in smartphones to detect traffic congestion. RideSense [6] is aimed to match a passenger's sensor trace against the traces of buses to determine the riding and leaving times.

The important finding of the above approaches is that the barometer can be used as a reliable sensor on smartphones for mobile context detection scenarios. However, they all require continuous sensor event measurements and transmission. Further, while better than RFID and BLE, the accuracy promised by these approaches is still not good enough to fulfill the demands of transportation systems. For example, the accuracy of RideSense for more than 20 hours of traces from five bus lines is between 84 to 98%. As pointed out in Section 4.5, only the uppermost value of 98% would be sufficient for using this technology in practice. Lower levels of accuracies are not acceptable considering the large number of daily trips made through different public transport modes in a city, *e.g.*, 950,000 daily trips in Oslo as an average sized city.

2.3 Mobile Sensor Events and Deep Learning

In some recent works, deep learning has been leveraged to analyze sensor events for detecting mobile contexts. In [15], the authors report on the accuracy of models such as RNN, CNN, various Hybrid models, Restricted Boltzman Machines, and Autoencoders with respect to their ability to classify human activities from body-worn sensors. They conclude that, compared to traditional pattern recognition methods, deep learning reduces the dependency on human-crafted feature extraction and achieves better performance by automatically learning high-level representations of the sensor events. The authors also state that, from a technical viewpoint, there is no model outperforming all the others in general. Thus, they recommend to choose the models based on the requirements of specific scenarios. *DeepSense* [16] uses CNN and RNN to provide an estimation and classification framework for car tracking with motion sensors and human activity recognition. In [17], *DeepSleepNet*, a deep learning framework for automatic sleep stage scoring based on electroencephalogram data, is proposed. The authors show that the model automatically learns features for different datasets without utilizing any hand-engineered features. The model achieves an accuracy that is similar to the state-of-the-art methods using hand-engineering. From the ML-based stream

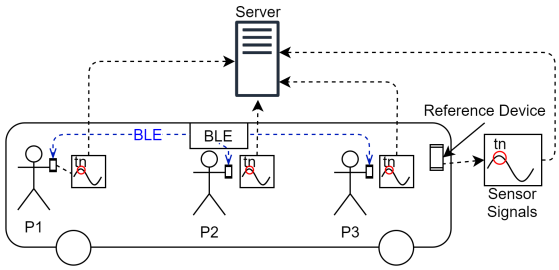

Figure 1: A sample scenario presenting DeepMatch.

matching perspective, StreamLearner [18] is a distributed Complex Event Processing (CEP) system proposed for scalable and low-latency event detection on streaming data that uses neural networks. StreamLearner is mainly designed for systems with multiple event sources causing diverse patterns in the event streams. Its case study is detecting anomalies (*i.e.*, abnormal sequences of sensor events) in smart factories.

The important finding of most works in this category is that deep learning can outperform hand-crafted feature extraction methods when applied to mobile sensor event streams. This can be used to deduce valuable information about the mobile context. We aim to exploit this power of deep learning in DeepMatch to build a model capable of highly accurate in-vehicle presence prediction solely based on sensor event streams. In addition, the limited work carried out on ML-based sensor stream matching, is more focused on the quality of stream matching, *e.g.*, to provide higher throughput.

3 DeepMatch

In this section, we first provide an overview of our approach. Then, we explain the hardware and software settings on which DeepMatch is built, followed by the presentation of our mobile stream data analysis and matching approach and a detailed description of the associated design and architecture model.

3.1 Overview

Figure 1 shows a simple scenario that we use to outline our approach. Three passengers are traveling with a bus. Everybody carries a smartphone with an app featuring the DeepMatch learning model. As fixed equipment, the bus is provided with a BLE-transmitter and a reference device (RefDev) that uses the same type of sensors as found on the smartphones. When a passenger enters the bus, the mobile app is awoken by the OS based on detecting the BLE signal. The

application then immediately starts to retrieve sensor data. Moreover, it performs feature extraction converting the sensed data to a lower dimensional representation. The compressed version of the data is timestamped and tagged with the ID of the BLE signal awakening the application, before it is transmitted to a remote server. Simultaneously, RefDev is measuring, transforming and transmitting event streams of its own sensors to the same server. Thus, the server receives two sets of data that are compared to infer whether the two sensors are in the same vehicle. For that, a special module carrying out the matching analysis is employed. In Section 3.4, we explain the learning model used for the matching analysis in detail. If the result of this analysis is that two data sets with the same BLE-transmission ID are collected in the same vehicle, and one of them is produced by RefDev, the person carrying the smartphone producing the other one is assumed to be in the same vehicle.

3.2 Hardware Requirements and System Settings

As indicated above, a RefDev equipped with equal sensors as a typical smartphone, is required to collect the same sensor events with an identical frequency. The RefDev is used to provide a ground truth for the in-vehicle presence detection. Through our empirical experiments, we found out that using only the barometric sensor provides both the best matching accuracy as well as a very low power consumption. This is discussed in Sections 4.4.1 and 4.8. The barometric sensors was initially introduced in smartphones to reduce GPS delay by providing the z coordinate. As described in Section 2, this sensor can also be applied to provide highly accurate contextual information. It guarantees position-independence, resistance to vibrations, and high sensitivity to changes in elevation that are properties of high value to implement the matching process (see [11]). *Position-independence*, *i.e.*, the sensor's ability to provide useful data independently of the sensor's location, is particularly important for underground transportation in tunnels, subways and trains, where for instance GPS is very inaccurate. *Vibration resistance* is important for the ability to measure the movements of the vehicle rather than the movements of the user. With respect to this property, the barometer clearly outmatches the accelerometer and gyroscope sensors, which are often more sensitive to the movements of a user's hands than to those of the vehicle. This results in the fact that DeepMatch with the barometer renders a precision of 97.8% while, with the two other sensors, only around every other matching is correctly detected (see Table 4). Finally, a high *elevation sensitivity* is critical for extracting useful context data in flat terrain, as demonstrated in Section 4.6. In [13], Bo-Jhang H. et al. report that relative pressure sensitivity for the Bosch BMP280 sensor used in the iPhone 6 and Nexus 5 is sensitive to elevation changes of 10 to 20 cm, even better than the specified vertical resolution of about one meter reported by Bosch in [19].

Besides the RefDev, the vehicle is provided with a BLE transmitter that, in

Table 1: Example datapoints

Sensor	Value	Timestamp	Trip	Device
Accelerometer	0.117311	3366...	15	75i3...
Magnetometer	21.835773	3366...	15	75i3...
Gyroscope	0.059957	3366...	15	75i3...
Barometer	993.281097	3366...	15	75i3...

contrast to the communication technology-based approaches discussed in Section 2, is not directly used for in-vehicle detection. Instead, its task is to wake up the app in the passenger's smartphone when entering a vehicle as well as to align the data produced by this smartphone with those sensed by the RefDev. Both Android and iOS provide the ability to start "sleeping" applications when a BLE-signal with a pre-defined ID is detected. Thus, our application only turns on and collects the sensor events when the phone is close to a BLE-transmitter registered in the application. Due to the imprecise nature of BLE, a transmitter may not only be readable in its own vehicle but also in its environment. In this case, *e.g.*, in a bus terminal, a smartphone may read several BLE transmitter inputs simultaneously. The IDs of these BLE transmitters are sent together with the collected data to the server. In this way, the service running on the server does not need to compare the user data with those of all RefDevs in the transport network, but only with those related with detected BLE transmitters. This significantly reduces the workload of the server. Further, if we use local servers in the vehicles, *e.g.*, letting RefDev conducting matching, the BLE transmitter can be used to forward the data from the user's smartphones to the server.

If a vehicle enters a *dead spot*, *i.e.*, an area with no cellular network coverage, and we use a central server, the encoded data will be temporarily stored on the device and tagged with timestamps and BLE IDs. When the vehicle leaves the dead spot, the locally stored data will then be transmitted to the server for a delayed in-vehicle presence detection.

3.3 Mobile Data Analysis

The deep learning model of DeepMatch performing the in-vehicle prediction has to be trained based on real sensor events collected from RefDev and passenger devices. In this subsection, we describe how the real sensor events are collected and converted to the training and evaluation datasets used to train the model.

3.3.1 Data Collection and Preprocessing

The sensor events used to train our deep learning model are collected by means of the *DataCollector*, an Android application that we developed for this purpose. The application can be configured to listen to events from any available sensor

Table 2: An Example of interpolated data

Timestamp	Accel.	Magneto.	Barom.	Gyrosc.
0 ms	5.62421	21.83577	989.28109	0.05995
20 ms	5.58418	22.83491	989.28610	0.13596
40 ms	5.53032	24.54790	989.27981	0.07716
60 ms	5.67377	25.12537	989.26586	0.08019

in the smart device, and to store and timestamp them locally as datapoints, see Table 1. The data from various runs can then be uploaded to a computer running our *Data Analysis* tools. Moreover, the application contains a simple server-client communication protocol using websockets. This allows us to connect several devices and to synchronize their clocks. In this way, the collection of sensor events can be carried out synchronously. The data collection is performed between two stops along the route of a public transportation provider, where all datapoints collected between the two stops are stored as a *Trip*. All trips are registered with a unique *trip* ID, propagated from the server device to all clients.

The sensor framework provided by Android allows developers to determine the sampling rate of each available sensor. The sensors generate events, using this sampling rate as a guideline, usually with a standard deviation of one to two milliseconds. To perform sensor event matching, however, we need a fixed sampling rate across all sensors and devices for a trip. This is achieved through our *Data Analysis* tool by *interpolating* the data collected by each device individually. The interpolation of a trip's data is done by first defining a global start time extracted from the data. Thereafter, this start time is substracted from the timestamps of all datapoints to get a relative timestamp. In the next step, we interpolate the values for each sensor event set with a fixed frequency, and finally remove the original data. With these fixed timestamp and interpolated values, we can now create a new table where the rows represent timestamps and each column contains the value for the given timestamp.

3.3.2 Dataset Creation

An important goal of DeepMatch is to minimize the amount of data needed to perform in-vehicle detection which reduces the amount of data to be transmitted between the devices and the server as well as the number of calculations performed by the server. To this end, we trained our model to perform predictions based on smaller *segments* of the trip data. Our *Data Analysis* tool converts the interpolated data from a trip, shown in Table 2, into *trip segments* by splitting the trip data into smaller segments of a fixed length. Furthermore, all segments are tagged with the ID of the trip they belong to, in addition to a segment number following the naming convention *¡trip id¿_¡segment nr¿*, *e.g.*, the first segment of a trip with id 15 becomes 15_0 and the second 15_1. This will be the same for all devices used to

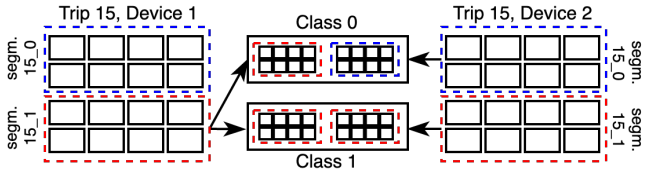

Figure 2: Matching samples created from trip segments.

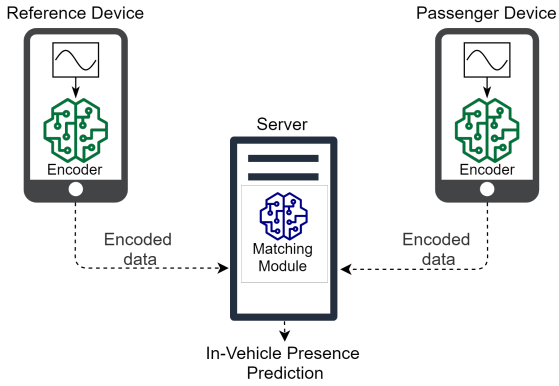

Figure 3: Overview of the DeepMatch model design.

gather data for Trip 15. The tool allows us to configure the length of the segments freely to find out which one renders the best matching results. However, when applying the tool to train and use the deep learning model, all segments must have the same length.

The created segments are used to build samples for a *matching dataset*. The samples in this dataset belong to either *Class 1* or *Class 0*. *Class 1* consists of samples from segments with the same trip id and segment number, *i.e.*, sensor events captured by two devices at the same time in the same vehicle. Samples from *Class 0* are created from segments with different trip ids or segment numbers. They represent sensor events not captured at the same time or in the same vehicle, as shown in Figure 2.

3.4 Design and Architecture of the Learning Model

The main goal of the DeepMatch learning model is to perform feature extraction, dimensionality reduction, and matching. As already mentioned, the overall in-vehicle presence detection process will be performed in a distributed fashion that is depicted in Figure 3. The feature extraction and dimensionality reduction take place both in the smartphones of the passengers and the reference devices fixed in

the vehicles. They are performed by *Encoder Modules*, which are shown in form of green networks in Figure 3. These encoders reduce the size of the original sensor events stream by a factor of four. In consequence, the bandwidth necessary to transmit the sensor data from the devices to the server will be reduced to a fourth in comparison to sending all the originally sensed data. The main objective of the encoder is to guarantee the preservation of characteristics and features of the data necessary for accurate matching.

The encoder is part of a neural network topology, called *Autoencoder* [20]. It is composed of two parts, an *encoder* and a *decoder*. Autoencoders are used to learn efficient, often lower-dimensional representations of their input through unsupervised training. The encoder maps the autoencoders input to a *latent representation* in *latent space*, *i.e.*, an internal representation of its input. The decoder maps this latent representation to a reconstructed representation of the Autoencoder's original input. The amount of information passed from the encoder to the decoder is typically restricted, forcing the Autoencoder to prioritize the most relevant information in its input. In DeepMatch, we use dimensionality reduction to restrict the encoder in order to achieve the size reduction by the factor four.

The matching predictions are performed on the server by a fully connected deep neural network, called the *Matching* module, depicted as a blue network in Figure 3. To achieve a high in-vehicle presence detection accuracy, this module has to learn and fine-tune the spatiotemporal thresholds to distinguish the samples in *Class 1*, *i.e.*, segments taken in the same vehicle at the same time, from those in *Class 0*, *i.e.*, segments sensed during different trips or at different locations.

The Matching module and the Autoencoder are developed and trained jointly using the architecture shown in Figure 4. Different types of Autoencoders exist. In DeepMatch, we use a *Stacked Convolutional Autoencoder* (CAE) [21] in which the *encoder* is created from *stacks* of alternating convolutional (conv) and maxpool layers. The conv layers are responsible for feature extraction and the maxpool layers for dimensionality reduction.

As previously mentioned, the *decoder* is the part of the Autoencoder responsible to recreate a copy of its input from the latent representation output by the encoder. It is created from stacks of alternating conv and upsample layers. Conv layers are specially suited to detect and extract time-invariant features in sequences, see [15, 21–23]. The maxpool layers perform dimensionality reduction using the *max* operator. The upsampling layers reverse this process by duplicating each value in its input sequence, *e.g.*, the sequence $1, 2, 3$ would become $1, 1, 2, 2, 3, 3$.

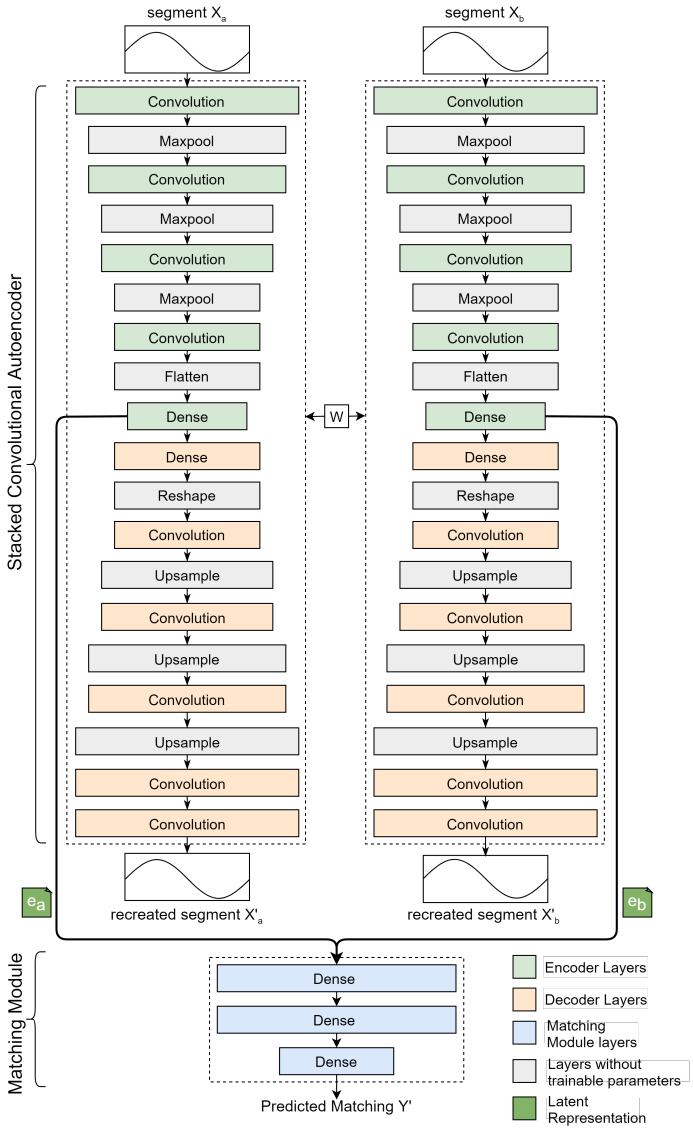

Figure 4: DeepMatch model architecture.

In Fig. 4, the specifics of our deep model are shown. The green boxes represent the layers of the encoder and the orange ones the layers of the decoder. The grey boxes show layers without trainable parameters and the blue ones the layers

of the Matching module. Each convolution layer represents the following three operations sequentially: convolution, rectified linear unit activation (ReLU, *i.e.*, $relu(x) = max(0, x)$) and batch normalization [24]. Every other layer in the encoder is a maxpool layer, using a stride size of 2, and every other layer in the decoder is a upsample layer, with size of 2, doubling the size of their input. The task of the flatten layer is to reshape any N-dimensional input to an 1-dimensional output, whilst the reshape layer reverse this process. In our model, the encoder consists of four convolutional layers, three maxpooling layers, one flatten layer and one dense layer. The decoder consists of five convolutional layers, three upsample layers, one reshape layer and one dense layer. The last part of the learning model in DeepMatch, the Matching module, consists of three consecutive fully connected dense layers, all using ReLU activation and batch normalization.

The DeepMatch model is distributed amongst the server, the reference device, and the passenger devices. The encoder module is embedded in the smartphones and reference devices whilst the Matching module is implemented in the server. The decoder module is only used during training and not in the execution of in-vehicle presence detection.

To train the overall model depicted in Fig. 4, the CAE is duplicated, sharing all trainable parameters W in a network topology known as a *Siamese Architecture*. This architecture has been applied with great success in matching problems like face recognition [25], signature verification [26], and human identification using gait recognition [27]. The Siamese architecture allows the model to accept two sensor data segments at the same time, *e.g.*, segment X_a and X_b. Since the two CAEs share the same weights, the encoder performs an identical mapping of the segments. Therefore, if the segments are matched (*i.e.*, they belong to a sample of *Class 1*), the latent representations e_a and e_b should also be matched. Likewise, e_a and e_b should be different for samples belonging to *Class 0*. Through joint training of both the CAE and the Matching module, the encoder learns to prioritize both, features of the segments that are necessary for the decoder to recreate them, and features needed by the Matching module for matching.

3.5 Model Training

This subsection describes the training routine for the model shown in Fig. 4. We describe two sensor data segments belonging to a matching sample as X_a and X_b (see Section 3.3.2) while the binary label Y refers to the ground truth class of a sample, *i.e.*, $Y = 1$ for *Class 1*, and $Y = 0$ for *Class 0*. Through the encoder layers of the Siamese CAEs, X_a and X_b are mapped to lower-dimensional latent representations e_a and e_b, shown as dark green squares in Fig. 4. Thereafter, we map e_a and e_b through the decoder layers which results in the segment recreations X'_a and X'_b. Finally, we feed X'_a and X'_b to the Matching module which returns the class prediction Y', *i.e.*, $Y' = 1$ if the segments are matched and $Y' = 0$ otherwise.

The goal of the model training, of course, is to reduce the disagreement between

the ground truth label Y and the class prediction Y' for as many samples as possible. To achieve that, we also need to reduce the disagreement between the original segments X_a and X_b and the recreated ones X'_a and X'_b. To quantify the disagreements between original and recreated segments, we use *Mean Squared Error*:

$$L = \frac{1}{n}\sum_{t=1}^{n}(X'_a[t] - X_a[t])^2$$

Here, n is the overall time span of segment X_a while $X'_a[t]$ is the recreation of the datapoint $X_a[t] \in X_a$ at the point of time t. As a loss function for the Matching module to quantify disagreements between Y and Y', we apply *Binary Cross Entropy*:

$$L = -Y \cdot log(Y') + (1 - Y) \cdot log(1 - Y')$$

Y' is the predicted label of the sample containing segments X_a and X_b, and Y its ground truth.

The disagreements found by the loss functions described above are used to update the trainable parameters of the model through Stochastic Gradient Descent. We emphasize that the gradients from *both* loss functions are backpropagated to the encoders. This enables the encoders to extract not only the most defining features of its input, but also the features relevant for matching prediction.

3.6 Design Rationale behind the DeepMatch Model

To find out the best model, we conducted hundreds of experiments on various model configurations. Every configuration was evaluated using the performance metrics described in Section 4.1 on the dataset described in Section 4.2. To obtain a useful model architecture, we tried increasing as well as decreasing the number of convolutional layers in the CAEs and swapping the convolutional layers for dense layers. Moreover, we tried multiple variants of the Matching module, using convolutional layers instead of dense layers, varying the size and number of dense layers, and also exchanging the Matching module with a function calculating the Euclidean Distance between the latent representations and using this for matching predictions. We tried stacking convolutional layers as feature extractors instead of using Autoencoders, removing the need for loss calculations between the input and recreated segments. In addition to different model architectures, we tested various hyperparameter settings such as adjusting the number and sizes of filters in each conv layer, and trying various output sizes on the dense layers of the Matching module. From all our experiments, the architecture in Fig. 4, using the hyperparameter settings described in 3.4, rendered the best performance.

All experiments (*i.e.*, training and evaluation) were performed on a desktop PC with an Intel i7 4.00GHz CPU, 16 GB memory, and a Nvidia GTX 1080 GPU. The models were created, trained and evaluated using Google Tensorflow 2.0, version 2.0.0-rc0 [28].

4 Evaluation

In this section, we first describe the performance metrics chosen to evaluate our learned models. Thereafter, we explain how the data used during training and evaluation was collected and pre-processed. Moreover, we show the performance results from seven sensor modality variations of our model. For that, we investigated not only the barometer but also accelerometer, magnetometer, and gyroscope sensors as well as various combinations. Further, the performance results for DeepMatch 5, DeepMatch 10 and DeepMatch 15 are compared. These variants refer to three different segment sizes of the best sensor modality with lengths of 5, 10 and 15 seconds, respectively. Afterwards, we compare DeepMatch with two well-known baseline methods. The results of evaluating these methods against our datasets are reported and the performance comparison between DeepMatch and those methods is discussed. To further illustrate the accuracy of barometer-based DeepMatch, we look also at the special case of very flat terrain, a worst case scenario when only barometer data is used. In addition, we investigate the execution time overhead of the Matching module carried out on the server. Finally, the battery consumption as well as the CPU and run-time overhead for the passenger smartphones are evaluated.

4.1 Definitions and Metrics for Evaluation

A *positive sample* represents segments belonging to *Class 1*, and a *negative sample* those from *Class 0*. Furthermore, according to the common denominations in binary classification, we define the following terms: *True Positive (TP)*: a correctly classified positive sample; *True Negative (TN)*: a correctly classified negative sample; *False Negative (FN)*: a positive sample wrongly classified as negative; *False Positive (FP)*: a negative sample falsely classified as positive.

The following four metrics are used for evaluation:

- *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*:

$$PR \triangleq \frac{TP}{TP + FP} \tag{1}$$

- *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:

$$RE \triangleq \frac{TP}{TP + FN} \tag{2}$$

- *Accuracy (ACC)*: In a dataset with a 50/50 class distribution, the accuracy describes how good the model is at classifying samples from all classes, *i.e.*,

it describes how many of all predictions made are correct:

$$ACC \triangleq \frac{TP + TN}{TP + FP + TN + FN} \tag{3}$$

- *F1-score (F1)*: The harmonic mean between precision and recall. The F1-score is useful in cases where the distribution of classes is not 50/50:

$$F1 \triangleq 2 \cdot \frac{PR \cdot RE}{PR + RE} \tag{4}$$

We plot the results of our evaluation in so-called Receiver Operating Characteristics (ROC)-graphs which describe how good a function and/or a model are at distinguishing between the classes in the dataset. The measurements for the three DeepMatch variants using barometer data and two baseline methods according to these metrics will be discussed in Section 4.4.

4.2 Data Collection and Dataset Creation

The data was collected by volunteers, each carrying one to three Android phones. All phones were connected through the Android application discussed in Section 3.3.1. The following seven Android devices were used: Huawei Nexus 5X, two Huawei Nexus P6, Samsung S8, Sony Z3 Compact, Google Pixel XL and Google Pixel 3a. The data was collected during trips made by public transportation (*i.e.*, trains, subways, busses and trams) in Oslo and Trondheim, two Norwegian cities. In total, we collected 21,252 unique 10 second sensor data segments that consist of events from the magnetometer, accelerometer, gyroscope, and barometer sensors[1]. Following the common practice in machine learning, 70% of the segments were used for training and 30% for evaluation. Thereafter, matching sets were created separately for both training and evaluation, resulting in a training dataset of 180,408 and an evaluation set of 67,304 unique samples.

The creation of the matching sets was performed separately for the training and evaluation sets to avoid using the same sensor event segments in both phases. In this way, any segment used in the evaluation set has never previously been seen by the model. In both sets, we selected each 50% of the segment pairs from *Class 0* and *Class 1*. Following this approach, we created seven datasets containing data from various sensor modality combinations:

- **A**: Accelerometer
- **M**: Magnetometer
- **B**: Barometer

[1]The datasets will be available via GitHub. In this version of the paper, we do not share the GitHub link due to the double-blind review policy of the DEBS conference.

- **BA**: Barometer and Accelerometer
- **BM**: Barometer and Magnetoer
- **AMG**: Accelerometer, Magnetometer and Gyroscope
- **AMGB**: Accelerometer, Magnetometer, Gyroscope and Barometer

After training the models on these datasets, and evaluating their performance on the evaluation sets, the best performing model and sensor modality were selected for further testing. The next goal was to test how models trained on segments of varying lengths would perform. To do this, we created two additional datasets of 5 and 15 second segments from the best performing sensor modality data.

4.3 Baseline Methods

Two baseline methods were chosen for comparison with DeepMatch: *Normalized Correlation (NORM_CORR)* which calculates the correlation between two sequences by comparing datapoints in the same temporal position, and *Dynamic Time Warping (DTW)* which compares all datapoints in two sequences by warping the temporal dimension to find the best correlation for any datapoint.

Since DTW describes the distance between two sequences, where a large distance equals a small correlation, we inverse the results from this function. The goal is to find a way to classify instances belonging to the two classes in the dataset, using these methods. The assumption is that applying either method on samples belonging to *Class 1*, should provide a large value, while samples belonging to *Class 0* should return a small value. To this end, we used the following equation:

$$c = f(X_a, X_b), \quad Y' = \begin{cases} 1 & \text{if } c > \alpha \\ 0 & \text{else} \end{cases}$$

The function f represents either of the two baseline methods, and c the result of applying f to the segments X_a and X_b in a sample from the dataset. The delimiting value α is used to classify instances of the two classes from their c values. To find α, we first apply f to all samples in the training set and add the resulting c-values to a sorted array. Thereafter, we search for the optimal delimiting value α, best able to separate instances in the sorted array. If the value c for a sample is larger than the delimiting value α, the sample is assumed to belong to *Class 1*. Otherwise, it should belong to *Class 0*. Optimal α values were searched for both NORM_CORR and DTW using the training set. Then, we evaluated the functions and their corresponding α values on the evaluation set. The results of our experiments are discussed in the following.

Table 3: Confusion matrix for the barometer-based DeepMatch 10

	Predicted Positive	Predicted Negative
Actual positive	33018	634
Actual negative	842	32810

Table 4: Performance comparison various sensor combinations

Model	PR	RE	ACC	F1
DeepMatch 10 A	0.5065	0.9531	0.5122	0.6615
DeepMatch 10 M	0.5064	0.9280	0.5118	0.6553
DeepMatch 10 B	**0.9751**	0.9812	**0.9781**	**0.9781**
DeepMatch 10 BA	0.7332	0.9697	0.8082	0.8350
DeepMatch 10 BM	0.7081	0.9708	0.7853	0.8189
DeepMatch 10 AMG	0.5011	0.9646	0.5020	0.6595
DeepMatch 10 AMGB	0.7079	**0.9892**	0.7905	0.8253

4.4 Experimental Results

During the development of our model, we continuously evaluated our results using the metrics described above. The confusion matrix, *i.e.*, the overall number of TP-, TN-, FN-, and FP-rated samples, for barometer-based DeepMatch 10 is listed in Table 3. The values of the confusion matrices for the learned models and the two baseline models allowed us to compute the outcomes according to the four metrics introduced in Section 4.1. The results are presented in Tables 4 and 5, and discussed below.

4.4.1 Sensor Modality Experiments

Table 4 depicts the results from training DeepMatch on various sensor modality combinations as described in Section 4.2. The numbers show that all models trained on datasets containing barometer data outperform all other models. Moreover, the DeepMatch 10 B, trained on barometer data alone, outperforms all other models. As described in Section 3, the barometer sensor is precise independently of the position of the vehicle. In particular, it is resistant to vibrations and sudden user movements as well as highly sensitive to elevation changes. This makes it perfectly suited to capture the movements of the vehicle rather than the movements of the individual user.

The accelerometer and gyroscope, on the other hand, are more sensitive to the movements of the users. The magnetometer is more sensitive to magnetic objects in the proximity to the user as well as to the power unit of the vehicle than to the movements of the vehicle which makes it also a poor source of data for the model. All these factors impact the performance of the models, and the results in Table 4

Table 5: Performance comparison of the barometer-based DeepMatch with baseline methods

Model	PR	RE	ACC	F1
DeepMatch 5	0.9408	0.9765	0.9574	0.9583
DeepMatch 10	0.9751	0.9812	**0.9781**	**0.9781**
DeepMatch 15	0.9348	**0.9816**	0.9566	0.9576
NORM_CORR	0.9174	0.9595	0.9393	0.9380
DTW	**0.9810**	0.7350	0.8136	0.8404

show that, with one exception, DeepMatch 10 B renders the best results. That holds particularly for the important *ACC* metric that shows the share of correct versus all matchings. The high *RE* value of the AMGB model indicates that the model correctly classifies most of the positive samples as positive. It seems, however, that it has a bias towards false positives, *i.e.*, classifying also negative samples as positive, which results in a low *PR* value. Altogether, limiting ourself to using only the barometer data seems to be the most promising way to conduct in-vehicle presence detection.

4.4.2 Segment size experiments with the barometer-based DeepMatch

From the numbers in Table 5, we can conclude that for all performance metrics, DeepMatch *10* is outperforming DeepMatch *5*. This is caused by the difference in segment sizes for the two models, 512 and 256 data points respectively. Thus, the former model has more data to learn from than the latter, which explains the higher quality of its performance. According to this explanation, however, DeepMatch 15 with its 768 data points should outperform the two other models. This is true for RE but not for the other three metrics where it underperforms at least DeepMatch 10. Due to the bad PR value in comparison with the good RE result, the model seems to be biased towards classifying samples as positive which leads to an extended number of false positives. Probably, the composition of 15 seconds long segments of our learning set is non-representative which leads to learning a sub-optimal classifier. Using a larger dataset, we believe DeepMatch 15 would outperform DeepMatch 10. We will follow this up in future experiments.

4.4.3 Baseline Methods

From Table 5, we can see that RE, ACC, and F1 of both baseline methods are lower than the corresponding metrics for the learned DeepMatch models. The sole exception is the metric PR for which DTW gave a better result than both the DeepMatch variants and NORM_CORR. The reason for this is a correlation of DTW to negative samples that we discuss below. That causes the consequence

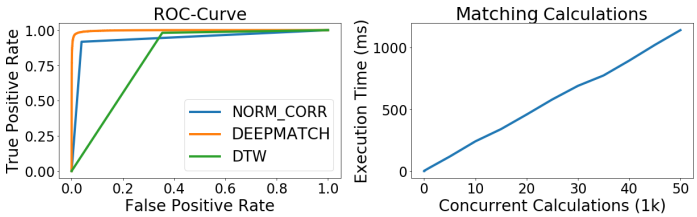

Figure 5: ROC-curve for baseline methods (left) and matching calculations execution time (right).

that DTW produces only relatively few false positives which renders the good result for PR. Instead, it generates a significant number of false negatives spoiling the values for the other metrics.

Altogether, the two baseline methods seem to be less suited for in-vehicle presence detection than DeepMatch. For NORM_CORR, we believe this is due to the sensitivity of the function to time-lag between its input sequences, *e.g.*, a passenger sitting a couple of meters behind the RefDev in the vehicle, will experience a lag between the signals which will result in a lower correlation value for positive samples. Therefore, the correlation value for some of the positive samples will be mixed with the correlation value for negative samples resulting in a less optimal delimiter.

The low performance of DTW is most likely caused due to lacking sensitivity to the temporal dimension. DTW warps the temporal dimension between the two sequences to find the shortest distance. This will result in a very high correlation value for some negative samples, making it difficult for the delimiter to separate samples from the two classes. As a result of this, there are relatively few false positives at the expense of many false negatives which explains the discrepancy of DTW's results for the different metrics in Table 5. Similar results can be observed in the ROC-graphs for the models. The left graph in Fig. 5 depicts the ROC-curve for DeepMatch 10, NORM_CORR and DTW. A property of these curves is that, as larger the areas under the curve are, as better the performance of the corresponding model will be. According to that, DeepMatch 10 is better than NORM_CORR and much better than DTW what our RE, ACC and F1 results also reflect.

4.5 Discussion of the Experimental Results

At a first glance, the differences between the accuracies of barometer-based DeepMatch 10 ($ACC = 0.9781$) and the baseline model NORM_CORR ($ACC = 0.9393$) may not seem to be considerable. In practice, however, they may have a great effect. Let us take an auto-ticketing system for city busses. Reflecting short distances of just one or two minutes journey time between two bus stops in

an inner city environment, we assume that six in-vehicle prediction runs (*i.e.*, six segments of 10 seconds each) are conducted during this period. To reduce the risk of wrongly billing people who are not riding in a bus but being, *e.g.*, in a car next to it, the bus operator may apply a policy to ticket somebody only if at least five of these six runs predict the user's smartphone being in the bus. The likelihood P_{cr} that our policy correctly detects a passenger can be computed as follows:

$$P_{cr} = ACC^6 + 6 \times \ ACC^5 \, (1 - ACC)$$

Thus, taking the ACC value of NORM_CORR, $P_{cr} = 95.31\%$ of all passengers are ticketed on average while the rest travels for free. This system leads to a revenue reduction of nearly 5% which few bus operators would accept. With DeepMatch 10, however, $P_{cr} = 99.32\%$ of the passengers are correctly billed. The loss of revenue of less than one percent seems to be acceptable since it will be easily outweighed by reducing the number of ticket machines and other infrastructure.

Additionally, for the case of wrongly billing non-passengers, DeepMatch 10 has a significant advantage over NORM_CORR. Using the policy mentioned above, the likelihood P_{er} of erroneous ticketing can be calculated by the following formula:

$$P_{er} = (1 - ACC)^6 + 6 \times \ ACC \, (1 - ACC)^5$$

That leads to the values $P_{er} = 0.000003\%$ with DeepMatch 10 and $P_{er} = 0.000469\%$ with normal correlation. In the latter case, around 171 people are wrongly billed in a year if we assume a 100,000 non-passengers being checked for in-vehicle presence every day which seems reasonable for a larger city. Thus, more than three such cases arise every week leading to a lot of compensation claims and bad press. In contrast, using DeepMatch 10, only a single person is wrongly billed in a year which seems acceptable.

4.6 Performance in Flat Terrain

In Section 4.4.1, we showed evidence that DeepMatch works best when only barometer data is used. This, however, may cause a problem in level areas. Therefore, to measure the accuracy of DeepMatch in such worst case scenarios, we made different trips in a very flat region in the central district of Trondheim. Some results of these experiments are shown in Fig. 6. Plot 1 shows the pressure measured by two different phones during the same trip, while Plot 2 depicts pressure measurements from two trips using the same phone. The low amplitudes in all curves show the flatness of the area. Plot 3 shows the match prediction of DeepMatch 10 based on the sensor output from the two devices in Plot 1. Plot 4 depicts the match prediction of our model based on the sensor output from the two trips in Plot 2. As indicated by Plot 4, DeepMatch detects dissimilarity at a high accuracy, despite the very similar pressure values measured by Google Pixel 2 in Plot 2.

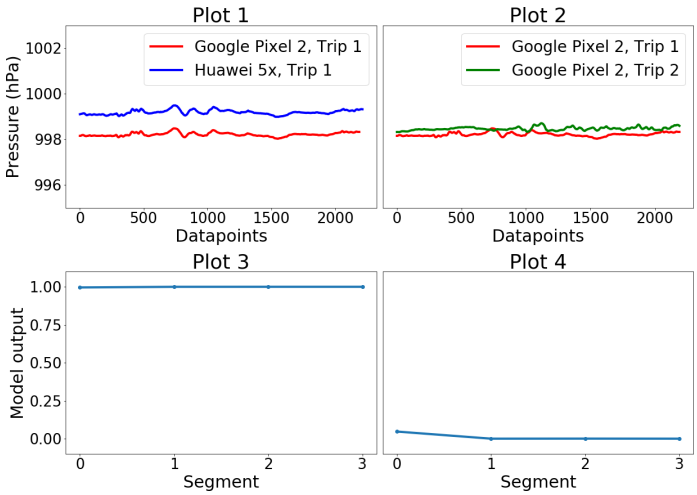

Figure 6: Match prediction tests for trips on flat roads.

4.7 Matching Execution Time

To use DeepMatch-based in-vehicle prediction also in real environments, the server needs to be able to do matching calculations from a large number of concurrently travelling passengers. That holds particularly for centralized server structures. The right graph in Fig. 5 shows the execution time of a central server as a function of increasing concurrent calculations. To increase the operational speed of our system, we exploited the feature of Tensorflow models to make several simultaneous predictions on multiple inputs. This resulted in an execution time of 1,140 milliseconds for 50,000 concurrent matching calculations, all running on one desktop equipped with a single GTX 1080 GPU. Since all trips between two stops are far longer than the 1,140 milliseconds, a data center consisting of just 19 of such computers could serve a city like Oslo with its 950,000 daily passengers even if all of them travel at the same time.

4.8 Battery Consumption on Smartphones

In this subsection, we evaluate the battery consumption of DeepMatch which is crucial for the adoption of our approach in practice. In general, there are three main sources of battery drain in our framework, *i.e.*, collecting barometer data, the encoder module for data processing, and transmitting the processed data to the server.

For our tests, we selected five Android phones from five different manufacturers that are listed in Table 6. To consider age diversity, we used phones that

Table 6: Android phones used in the tests

Type	Battery capacity	Age
Samsung S8	3000 mAh	2 years
LG Nexus 5X	2700 mAh	3 years
Huawei Nxus 6P	3450 mAh	4 years
Google Pixel 3a	3000 mAh	1 year
Sony Z3 compact	2600 mAh	5 years

Table 7: Battery consumption per hour

Brand	Data collection	Learning	Complete
Samsung	25 mA	26 mA	31 mA
LG	23 mA	24 mA	26 mA
Huawei	22 mA	23 mA	25 mA
Google	16 mA	17 mA	18 mA
Sony	15 mA	18 mA	21 mA

are between one and five years old. Besides the battery capacity, the environment temperature is an important factor that can influence the performance of batteries. Therefore all tests were run in an experimental environment with a temperature of 19° Celsius representing the indoor temperature of typical transportation vehicles. Since barometer-based DeepMatch 10 promises the best overall performance (as discussed above), we consider this version of our model for the battery measurements.

The battery status is collected from the app using the *Batterystats* and *Battery Historian* tools included in the Android framework [29]. These tools provide functionality to extract details on battery consumption for all applications running on the device. In order to ensure that the app can listen to barometer events and process them in intervals of ten seconds, we run the tests in the background with the *wake lock* parameter enabled to keep CPU processing on. Reflecting the above mentioned battery consumption factors, we use the following three scenarios for our experiments:

- *Complete scenario*: All three factors of battery consumption, *i.e.*, the barometer data collection, data processing by the encoder, and data transmission.
- *Learning scenario*: Data collection and data processing.
- *Data collection scenario*: Only barometer data collection.

The results of our tests are depicted in Table 7. The numbers show clearly that for all five devices, DeepMatch influences the battery consumption only

Table 8: Run Time and CPU overhead

Brand	CPU	Mean Run Time	Overhead
Samsung	2.3 GHz + 1.7 GHz, Cortex-A53	49 ms	1-2 %
LG	1.4 GHz + 1.8 GHz, 64-Bit Hexa-Core	46 ms	1-2 %
Huawei	2.0 GHz + 1.55 GHz, 64-Bit Octa-Core	52 ms	1-2 %
Google	2.0 GHz + 1.7 GHz, 64-Bit Octa-Core	19 ms	0-1 %
Sony	2.5 GHz Quad-Core, 400 Krait	73 ms	3-4 %

marginally. For all phones, the battery usage will be less than 62 mA considering a total travel time of two hours a day. With a battery capacity of 3000 mAh, this equals 2.1%. This value is considerably lower than most smartphone apps, as reported in [30]. Therefore, we believe that the battery consumption of DeepMatch is satisfactory.

4.9 Computational Overhead on Smartphones

In this subsection, we evaluate the computational overhead of the feature extraction and dimensionality reduction performed by the barometer-based DeepMatch 10 on smartphones. For these experiments, we used the same smartphones as in the battery consumption analysis. We registered both the run-time and CPU usage of the *encoder* module, when it processed sensor events with intervals of 10 seconds. The results of our tests are depicted in Table 8. The numbers show clearly that, for all phones, the mean run-time and CPU overhead of the encoder is barely noticeable. Even for the oldest model in the tests, the five year old Sony Z3 Compact, the mean run time of the encoder is 73 ms which affords a CPU usage of only three to four percent.

5 Conclusions and Future Work

In this paper, we proposed a machine learning-based approach, called DeepMatch, to address the challenge of in-vehicle presence detection as an important aspect of mobile context. It utilizes the sensor event streams of a smartphone to estimate its presence in a public transport vehicle at the very high accuracy of nearly 98%. DeepMatch is based on utilizing Stacked Convolutional Autoencoders for feature extraction and dimensionality reduction, and a dense neural network for event stream matching. The feature extraction and dimensionality reduction run on the smartphone and the reference device, while the event matching is performed on a server. Through dimensionality reduction, the datapoints are reduced by the factor four such that the bandwidth of the data transfer to the server is considerably reduced without losing the information of the data necessary to perform matching.

Our future plan is to improve DeepMatch 10 with further data gathering and model optimization. During 2020, we will implement a pilot of DeepMatch in a Norwegian city together with a public transportation provider. Moreover, we intend to research on the optimum length of the data segments and the frequency of data gathering (from the reference devices and the smartphones) in order to minimize the amount of data needed for in-vehicle presence detection.

References

[1] S. Kang, Y. Lee, C. Min, Y. Ju, T. Park, J. Lee, Y. Rhee, and J. Song, "Orchestrator: An Active Resource Orchestration Framework for Mobile Context Monitoring in Sensor-rich Mobile Environments," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*. Mannheim, Germany: IEEE Computer, 2010, pp. 135–144.

[2] W. Narzt, S. Mayerhofer, O. Weichselbaum, S. Haselböck, and N. Höfler, "Be-In/Be-Out with Bluetooth Low Energy: Implicit Ticketing for Public Transportation Systems," in *IEEE 18th International Conference on Intelligent Transportation Systems*. Las Palmas, Spain: IEEE, 2015, pp. 1551–1556.

[3] T. Gyger and O. Desjeux, "EasyRide: Active Transponders for a Fare Collection System," *IEEE Micro*, vol. 21, no. 6, pp. 36–42, 2001.

[4] C. Sarkar, J. J. Treurniet, S. Narayana, R. V. Prasad, and W. de Boer, "SEAT: Secure Energy-Efficient Automated Public Transport Ticketing System," *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 1, pp. 222–233, 2018.

[5] M. Won, A. Mishra, and S. H. Son, "HybridBaro: Mining Driving Routes Using Barometer Sensor of Smartphone," *IEEE Sensors Journal*, vol. 17, no. 19, pp. 6397–6408, 2017.

[6] R. Meng, D. W. Grömling, R. R. Choudhury, and S. Nelakuditi, "RideSense: Towards Ticketless Transportation," in *2016 IEEE Vehicular Networking Conference (VNC)*. Columbus, OH, USA: IEEE, 2016, pp. 1–8.

[7] 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, pp. 13–16.

[8] T. Gründel, H. Lorenz, and K. Ringat, "The ALLFA Ticket in Dresden. Practical Experience of Fare Management Based on Be-In/Be-Out & Automatic Fare Calculation," 2006, iPTS Conference, Seoul, South Korea.

[9] S. Kuchimanchi, "Bluetooth low energy based ticketing systems," Master's thesis, Aalto University, Espoo, Finland, 2015.

[10] A. Kwiecień, M. Maćkowski, M. Kojder, and M. Manczyk, "Reliability of Bluetooth Smart Technology for Indoor Localization System," in *International Conference on Computer Networks (CN)*, ser. CCIS 522. Br'unow, Poland: Springer-Verlag, 2015, pp. 444–454.

[11] K. Sankaran, M. Zhu, X. F. Guo, A. L. Ananda, M. C. Chan, and L.-S. Peh, "Using Mobile Phone Barometer for Low-power Transportation Context Detection," in *12th ACM Conference on Embedded Network Sensor Systems*. Memphis, TN, USA: ACM, 2014, pp. 191–205.

[12] S. Vanini, F. Faraci, A. Ferrari, and S. Giordano, "Using Barometric Pressure Data to Recognize Vertical Displacement Activities on Smartphones," *Computer Communications*, vol. 87, pp. 37–48, 2016.

[13] B.-J. Ho, P. Martin, P. Swaminathan, and M. Srivastava, "From Pressure to Path: Barometer-based Vehicle Tracking," in *2nd ACM Inter. Conf. on Embedded Systems for Energy-Efficient Built Environments (BuildSys)*. Seoul, South Korea: ACM, 2015, pp. 65–74.

[14] A. Dimri, H. Singh, N. Aggarwal, B. Raman, D. Bansal, and K. K. Ramakrishnan, "RoadSphygmo: Using Barometer for Traffic Congestion Detection," in *8th International Conference on Communication Systems and Networks (COMSNETS)*. Bangalore, India: IEEE Computer, 2016, pp. 1–8.

[15] 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.

[16] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A Unified Deep Learning Framework for Time-series Mobile Sensing Data Processing," in *26th International Conference on World Wide Web*. Perth, Australia: ACM, 2017, pp. 351–360.

[17] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Exploiting Multi-Channels Deep Convolutional Neural Networks for Multivariate Time Series Classification," *Frontiers of Computer Science*, vol. 10, no. 1, pp. 96–112, Feb. 2016.

[18] C. Mayer, R. Mayer, and M. Abdo, "StreamLearner: Distributed Incremental Machine Learning on Event Streams: Grand Challenge," in *11th ACM International Conference on Distributed and Event-Based Systems*. Barcelona, Spain: ACM, 2017, pp. 298–303.

[19] Bosch, "Bosch BMP280," https://www.bosch-sensortec.com/bst/products/all_products/bmp280, 2020, accessed: 2020-04-01.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, ch. Autoencoders, pp. 505–528, http://www.deeplearningbook.org.

[21] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked Convolutional Auto-encoders for Hierarchical Feature Extraction," in *International Conference on Artificial Neural Networks (ICANN)*, ser. LNCS 6791. Espoo, Finland: Springer-Verlag, 2011, pp. 52–59.

[22] 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.

[23] A. Supratak, H. Dong, C. Wu, and Y. Guo, "DeepSleepNet: A Model for Automatic Sleep Stage Scoring based on Raw Single-channel EEG," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 11, pp. 1998–2008, 2017.

[24] 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.

[25] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition," 2015, https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf.

[26] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature Verification using a "Siamese" Time Delay Neural Network," in *Advances in Neural Information Processing Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1994, pp. 737–744.

[27] C. Zhang, W. Liu, H. Ma, and H. Fu, "Siamese Neural Network based Gait Recognition for Human Identification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Shanghai, China: IEEE, 2016, pp. 2832–2836.

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

[29] B. Historian, "Batterystats and Battery Historian," https://developer.android.com/studio/profile/battery-historian, 2019, accessed: 2019-10-23.

[30] X. Chen *et al.*, "Smartphone Energy Drain in the Wild: Analysis and Implications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1, pp. 151–164, 2015.

Paper 3

DeepMatch2: A Comprehensive Deep Learning-based Approach for In-Vehicle Presence Detection

DeepMatch2: A Comprehensive Deep Learning-based Approach for In-Vehicle Presence Detection

Magnus Oplenskedal[1,3], Peter Herrmann[1], Amir Taherkordi[1,2,3]

[1]Norwegian University of Science and Technology (NTNU), Trondheim, Norway

[2]University of Oslo, Norway

[3]Forkbeard Technologies, Oslo, Norway

{magnukop, amirhost, herrmann}@ntnu.no

Abstract

The accurate detection of the *mobile context* information of public transportation vehicles and their passengers is a key feature to realize intelligent transportation systems. A topical example is *in-vehicle presence detection* that can, *e.g.*, be used to ticket passengers automatically. Unfortunately, most existing solutions in this field suffer from low spatiotemporal accuracy which impedes their use in practice. In previous work, we addressed this challenge through a deep learning-based framework, called DeepMatch, that allows us to detect in-vehicle presence with a high degree of accuracy. DeepMatch utilizes the smartphone of a passenger to analyze and match the event streams of its own sensors with the event streams of counterpart sensors provided by a reference unit that is installed inside the vehicle. This is achieved through a new learning model architecture using Stacked Convolutional Autoencoders to compress sensor input streams by feature extraction and dimensionality reduction as well as a deep convolutional neural network to match the streams of the user phone and the reference device. The sensor stream compression is offloaded to the smartphone, while the matching is performed in a server. In this paper, we introduce DeepMatch2. It is an amended version of DeepMatch that reduces the amount of data to be transferred from the user and reference devices to the server by the factor of four. Further, DeepMatch2 improves the already good accuracy of DeepMatch from 97.81% to 98.51%. Moreover, we propose a travel inference algorithm, based on DeepMatch2, to detect the duration of whole passenger trips in public transport vehicles with a high

degree of precision. This is needed to create intelligent and highly reliable auto-ticketing systems. Thanks to the high accuracy of 98.51% by DeepMatch2, the inferences can be carried out with a negligible error rate.

Keywords— Mobile Context, In-Vehicle Presence Detection, Sensor Event Streams Analysis, Deep Learning, Event Matching, Intelligent Transportation

1 Introduction

In recent years, the rapid development of mobile technologies, IoT and cellular network infrastructures has led to new unprecedented opportunities for making public transportation a very environment-friendly mode of travelling more attractive. The fact that more than 3.8 billion people in the world own smartphones [1], provides a very worthwhile research and development area already utilized by public transportation providers in many areas of the world, *e.g.*, in Northern Europe. For instance, smartphone applications, that provide passengers the option to buy tickets and offer them other context-aware services such as path-finding and travel planning, are quite common nowadays.

However, the next generation of context-aware service within public transportation will require data sources providing an extremely high degree of precision and sophistication. In particular, one may consider the *mobile context*, *i.e.*, all kinds of spatiotemporal properties of the participating passengers and vehicles [2]. For example, if we know whether a person is inside a vehicle or not at a certain time and place, services such as dynamic vehicle-route planners based on passenger load and route optimization can be realized. We can detect this kind of mobile context by precise *in-vehicle detection* systems. These systems can also make the ticketing of passengers considerably simpler. In today's smartphone applications, the passengers have to remember buying tickets before starting a ride. Further, they often need in-depth knowledge about the ticketing system to buy the correct ticket for the planned trip. In contrast, using a highly accurate in-vehicle presence detection solution, a so-called Be-In/Be-Out (BIBO) system [3], tickets can be issued automatically to the passengers based on the exact duration of their journey. This way, the passengers can conveniently enter and leave public transport vehicles without having to deal with planning and purchasing tickets in advance.

In-vehicle presence detection has attracted the attention of the research community and industry. Early approaches like [4, 5] utilize communication systems such as Radio Frequency Identification (RFID) or Bluetooth Low Energy (BLE). While travelling, temporary connections are built up between the user's mobile device and certain fixed vehicle equipment to detect the passengers presence inside the vehicle. Other approaches, for instance [6, 7], analyze event streams from smartphone sensors for certain properties. Modern smartphones are equipped with a variety of sensors such as magnetometers, accelerometers, gyroscopes, GPS, and

barometers which offer unprecedented opportunities to analyze mobile context information from the user's environment. Finally, machine learning techniques have recently been leveraged to analyze sensor events for detecting mobile contexts, *e.g.*, [8]. We will argue later that the accuracy of works within the above categories is still not good enough to make them suitable for auto-ticketing in practice.

To realize in-vehicle presence detection with a high degree of accuracy, in *our previous work*, we proposed a deep learning-based framework, called DeepMatch [9]. Each vehicle is equipped with a stationary *Reference Device (RefDev)* (*e.g.*, an Android phone). We record the streams of sensor events gauged in both, the RefDev and the smartphones of potential passengers. In a so-called *in-vehicle presence detection process*, the stream generated in a smartphone is then compared with the one from the RefDev to find out if both devices are in the same vehicle. If that is the case, the owner of the smartphone is necessarily a passenger in the vehicle containing the stationary RefDev and can, *e.g.*, be billed for the journey. The in-vehicle presence detection process is realized by *data compression* using Stacked Convolutional Autoencoders as well as a deep neural network *matching* component that matches compressed sensor samples to find out if they were taken from within the same vehicle. The data compression is offloaded to the users' smartphones and reference devices, while the matching process is performed in a server that can be external, *e.g.*, in a cloud, or within the vehicle realizing an Edge computing solution [10]. By training both parts of our model together, *i.e.*, the compression and matching parts, we achieve that the matching process does not need the full smartphone and reference data for its comparison but can rely on the compressed versions.

Since the design and development of DeepMatch, we continuously iterated and improved our deep learning model to improve its *efficiency* and *accuracy*. Moreover, we enhanced the original framework with inference algorithms that allow us to deduce the *period of time that passengers travel* in public transportation with a very high accuracy. DeepMatch lacks this feature which is highly needed, *e.g.*, in automatic ticketing. The result of the improvements is a new version of our deep learning-based framework that we call DeepMatch2. It is introduced in this paper. In contrast to the original framework, DeepMatch2 incorporates the following amendments:

- The efficiency was enhanced by reducing the amount of data necessary for in-presence detection by a factor of four, *i.e.*, from previously 512 float values in DeepMatch to just 128 float values in DeepMatch2.
- Considering accuracy, we gradually amended the original layer structure, and for each change, trained and evaluated the results using the designated performance metrics. In spite of the concomitant reduction of the size of the input parameters, we further managed to increase the accuracy of DeepMatch2 to 98.51% in comparison to the accuracy value of 97.81% in DeepMatch.

- In [9], we provided only a short sketch about how one can use the results of DeepMatch to detect whole trips of passengers in public transport vehicles with a high degree of precision. In this paper, we go much deeper into this topic and discuss travelling user inference systems that are based on DeepMatch2 and can infer if and for which period of time a passenger makes a trip in a public transportation vehicle with a very low error rate.

The rest of this paper is organized as follows. In Sect. 2, we discuss existing solutions followed by the presentation of the original method DeepMatch in Sect. 3. In Sect. 4, we elaborate on the improvements made in DeepMatch2. Thereafter, we report the experimental evaluation results for the variants of our deep learning model and some baseline methods in Sect. 5. The travelling user inference algorithms that allow us to detect whole passenger trips, are introduced in Sect. 6. Finally, we conclude our paper in Sect. 7 with a discussion on the results gained so far and a look at our future plan.

2 Related Work

In-vehicle presence detection solutions can be classified into three different categories. The first category is focused purely on utilizing communication technologies, while the second one is based on analyzing mobile sensor events to detect in-vehicle presence. The third category consists of some recent works that leverage deep learning to analyze mobile contexts. In the following, we discuss each category in detail.

2.1 Communication Technology-based Solutions

Early in-vehicle presence detection systems were implemented using Radio Frequency Identification (RFID) with active tags carried by the passengers, and a single communication unit in the center of a vehicle. To track the RFID devices, contactless, mid-range radio-based identification and communication protocols were used. One of the first solutions was EasyRide [4], developed by the Swiss Railways Association. Allfa [11] is another RFID-based system, tested in busses, trams and trains in Dresden, Germany, for half a year. In total, the system covered about 120,000 trips carried out by 2,000 users. Unfortunately, testing these systems proved that they were too unreliable to be used for in-vehicle presence detection in practice. The main reason for that is the weak transmitter strengths of the active RFID-tags, which makes it difficult to detect them reliably in all areas of the vehicle. As discussed in [4], this affords not only one but a vast number of readers in the vehicle, at least one at each door. But even that does not seem to be sufficient to make the passenger assignment sufficiently predictable. For instance, Allfa has an accuracy rate of just 68% making it unsuitable for practical use.

Other approaches focus on Bluetooth Low Energy (BLE)-based automated in-vehicle detection. Compared with active RFID approaches with battery-powered tags, BLE-based BIBO systems can utilize smartphones with additional monitoring options, the possibility to measure signal strengths for proximity determination, larger distribution channels, etc. One of the early works on Bluetooth-based public transport ticketing system was carried out by the authors of [12]. Their system was in charge of collecting only the source and destination of each passenger journey. The first BLE-based solution is proposed in [3], where the authors are cautiously optimistic that BLE might work for BIBO systems. Nevertheless, the chassis of a vehicle does not limit the accessibility of a BLE transmitter which makes it possible that somebody close to it, *e.g.*, a person in another vehicle, is wrongly detected. On the other hand, objects in a vehicle may inhibit a BLE connection such that devices in the vehicle may not be detected. This is confirmed by the authors of [13] who found out that BLE is not well suited for indoor localization. As reasons preventing connections, they name the position of a device as well as human body obstacles like the hand carrying the device. The authors of [14] suggest a ticketing system adding a custom profile on top of the BLE specification to fulfill the payment procedure. In SEAT [5], a BLE-enabled smartphone communicates with devices installed in the vehicles to track the journey for automatic pricing. The main focus of the authors, however, is on security, performance, and battery friendliness but not on the accuracy of the in-vehicle presence detection. To conclude, BLE-based solutions are also suffering from low accuracy values making them less suited for in-vehicle presence detection scenarios.

2.2 Mobile Sensor Data Analytics-based Solutions

Works in this category analyze the data of the sensors in user smartphones to detect mobile contexts in transportation. The authors of [15] focus on context detection using only the smartphone barometer as it is independent of the phone's position and orientation. They demonstrate that the barometer can be applied to detect user activities of IDLE, WALKING, and VEHICLE at low-power. Likewise, in [16], user activities are classified using the barometer sensor on smartphones. This approach leverages Bayesian networks, decision trees, and RNN as inference models to predict user action, *e.g.*, riding or leaving a cable-car. The authors of [17] demonstrate how the pressure data collected from a smartphone barometer can be utilized to accurately track driving patterns based on the pressure data collected from the smart phone's barometer. By correlating pressure time-series data against topographic elevation data and road maps for a given region, a centralized server can estimate the possible routes through which users have travelled. Another barometer-based mobile system is HybridBaro [7] that features a hybrid algorithm to adaptively utilize GPS data to increase the detection accuracy in flat areas. RoadSphygmo [18] uses the barometer in smartphones to detect traffic congestion. RideSense [6] is aimed to match a passenger's sensor trace against the traces of

busses to determine the riding and leaving times. The authors of [19] present a vertical location system for vehicles in metropolises. In particular, they utilize the barometers and gravity sensors of smartphones to remedy the deficiency of vertical localization such as GPS. To achieve that, several novel algorithms are used (*e.g.*, height and angle detection, relative height measurement, and tracking) to build a highly accurate detection system.

While better than RFID and BLE, the accuracy promised by the approaches mentioned above is still not good enough to fulfill the demands of transportation systems. For example, the accuracy of RideSense [6] collected from five bus lines over more than 20 hours, is between 84 to 98%. As pointed out in [9], only the uppermost value of 98% would be sufficient for using this technology in practice.

2.3 Mobile Sensor Events and Deep Learning

Recently, deep learning has been leveraged to analyze sensor events for detecting mobile contexts. In [20], the authors report on the accuracy of models such as RNN, CNN, various Hybrid models, Restricted Boltzman Machines, and Autoencoders with respect to their ability to classify human activities from body-worn sensors. They conclude that, compared to traditional pattern recognition methods, deep learning reduces the dependency on human-crafted feature extraction and achieves better performance by automatically learning high-level representations of the sensor events. The authors also state that, from a technical viewpoint, there is no model outperforming all the others in general. Thus, they recommend to choose the models based on the requirements of specific scenarios. *DeepSense* [8] uses CNN and RNN to provide an estimation and classification framework for car tracking with motion sensors and human activity recognition. In [21], *DeepSleepNet*, a deep learning framework for automatic sleep stage scoring based on electroencephalogram data, is proposed. The authors show that the model automatically learns features for different datasets without utilizing any hand-engineered features. The model achieves an accuracy that is similar to the state-of-the-art methods using hand-engineering. Some works in this category focus on detecting the transportation mode using machine learning techniques and sensors data on smartphones such as [22,23]. From the ML-based stream matching perspective, *StreamLearner* [24] is a distributed Complex Event Processing (CEP) system proposed for scalable and low-latency event detection on streaming data that uses neural networks. It is mainly designed for systems with multiple event sources causing diverse patterns in the event streams. As a case study, the authors discuss anomaly detection (*i.e.*, finding abnormal sequences of sensor events) in smart factories.

The important finding of most works in this category is that deep learning can outperform hand-crafted feature extraction methods when applied to mobile sensor event streams. This can be used to deduce valuable information about the mobile context. We aim to exploit this power of deep learning in DeepMatch2

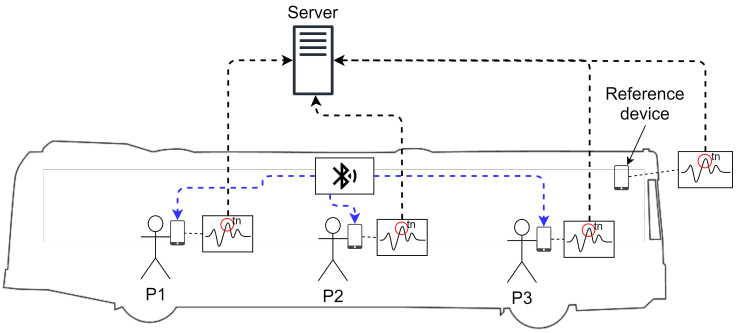

Figure 1: DeepMatch bus scenario

to build a model capable of highly accurate in-vehicle presence prediction solely based on sensor event streams. On the other side, the limited number of works, yet carried out on machine learning-based sensor stream matching, focus mainly on the quality aspects of stream matching, *e.g.*, to provide high throughput. In contrast to this paper, the efficiency and accuracy of these approaches are only superficially discussed in most cases.

3 DeepMatch

As we will discuss later, our deep learning method DeepMatch2 enhances its predecessor DeepMatch in both, in-vehicle presence detection accuracy and in the amount of data transfer needed. Nevertheless, the basic structures of DeepMatch and DeepMatch2 are very similar. Therefore, we decided to introduce the fundamentals for the architecture of both DeepMatch and DeepMatch2 in this section. Thereafter, we discuss the changes leading to DeepMatch2 in Sect. 4.

In the following, we start with a general overview of the model architecture of our deep learning model, followed by a discussion of the hardware and software settings, on which our approach has been built. Thereafter, we describe how DeepMatch conducts the analysis of the mobile data sensed, followed by the design considerations and architecture of our learning model, and a discussion on how the learning model is trained. Finally, we present the design rationale and experimental settings behind the DeepMatch deep learning model.

3.1 Overview

Figure 1 depicts the equipment needed in a bus[1] to realize our approach. The bus is equipped with a Bluetooth Low Energy (BLE) transmitter as well as a so-called *Reference Device (RefDev)*. The RefDev can be a smartphone mounted onto the bus or any other kind of hardware providing the same type of sensors, that can be found in a modern smartphone. These sensors include but are not limited to *accelerometers*, *magnetometers*, *gyroscopes*, *barometers*, and *GPS receivers*. The passengers travelling with the bus carry smartphones in which a special application is installed realizing parts of the DeepMatch deep learning model.

The BLE-transmitter continuously transmits a special ID that is unique to the bus it is installed in. Due to the low signal strength, this signal can be only detected by devices that are either inside the bus or nearby. When a passenger's phone picks up the BLE-transmitted signal for the first time, its operating system starts the DeepMatch application. From that moment, the sensors of the phone sample values that are forwarded to the deep learning model of DeepMatch running in the smartphone application. DeepMatch extracts relevant features from the sensed events and compresses them through dimensionality reduction. Finally, the compressed data are timestamped and tagged with the IDs of the BLE-transmitters, the phone is currently receiving.

In a similar way, the RevDef is used to continuously stream events from its own sensors and compresses them through DeepMatch. The compressed data is also timestamped and tagged but, in contrast to the user phones, only the tag of the BLE transmitter installed in the same vehicle is used.

Both, the RevDef and the user phones send the compressed sensor data to a server. Following the wishes of the public transport operators, we may realize DeepMatch using different hardware configurations. For instance, the server functionality can be realized using a cloud provider. Alternatively, following the principle of fog computing [10], it can be a unit locally installed in the vehicle, *e.g.*, together with the RevDef.

The server matches the data of the RevDef and the user phones, that carry the same BLE-transmitter IDs and timestamps, against each other by a special module of DeepMatch. If the module reports a match, we assume that both data sets were sensed within the same vehicle. Since the RevDef can be unambiguously allocated to a particular bus, we can then assume that the smartphone and its carrier are in the same bus.

3.2 Hardware Requirements and System Settings

As mentioned above, all vehicles using DeepMatch to provide automated in-vehicle presence prediction, require both, a BLE-transmitter and a Reference De-

[1]The realization of DeepMatch in other types of public transport like subways, trams, or trains is similar.

vice (RevDef). In contrast to the communication technology-based approaches discussed in Sect. 2, the BLE-transmitter is not directly used for in-vehicle detection. Instead, we apply it to perform a coarse-grained guess in which vehicle a passenger might be inside. In this way, the server only needs to match the user data with those from RefDevs, that are related to the sensed BLE ID received by the user phone, and not with the data of all RefDevs in the transport network. An additional advantage of this approach is that we can reduce the time DeepMatch is required to run on a user phone. Both Android and iOS provide the ability to awaken applications in smartphones when detecting a BLE-signal with a pre-defined ID. This provides us with the ability to run the application only if the user is either very close to a vehicle, or inside it. Thus, both computation overhead and battery consumption is at a minimum.

The authors of [25] show that BLE offers a good reliability also in noisy in-house environments like those we might come across in public transport vehicles. In their tests, at least 99.45% of all packets were transmitted within the expected delay bounds. Based on these numbers we expect that the phone of a passenger receives a fair number of the packets broadcasted by the BLE-transmitter within in the first seconds after entering the vehicle. Therefore, DeepMatch will almost certainly be started timely.

As we discuss to greater detail in Sect. 5.3, we found out through experiments, that using only the barometric sensor provides by far the best matching accuracy. Using DeepMatch alone with the barometric sensor provides an accuracy of 97.81% while no other combination of sensor data exceeds 80.82%. Moreover, performance tests show that registering barometer events with a frequency of 10 Hz incurs a very low battery consumption. The battery drain on the phones we tried in our tests is between 15 and 25 mAh while continuously registering events from the barometer. This equals a drain of between 0.6% and 0.8% of the total battery capacity per hour. That is described more closely in Sect. 5.8.

In the case that a vehicle enters a *dead spot*, *i.e.*, an area with no cellular network coverage, we temporarily store the compressed data locally until connectivity is regained and the data can be transmitted to the server for a delayed matching.

3.3 Mobile Data Analysis

The deep learning model performing the in-vehicle presence prediction, *i.e.*, the matching of sensor events, is trained on real sensor events that were collected from Android-based smartphones in the public transport systems of the Norwegian cities Oslo and Trondheim. In the following, we sketch the process of collecting the data and converting it to training and evaluation sets.

The data sets used to train the deep learning model were built from sensor events gathered by the means of an Android application, called *Datacollector*, that we developed for this purpose. *Datacollector* registers events from all sensors available in the phone, timestamps them, and stores them locally as *data points*.

Further, we can use the application to upload the data points to our *Data Analysis* center. There, the data points can be processed further into training and testing samples that are used to train and evaluate our deep learning network.

To allow the parallel collection of sensor data by several phones, multiple devices running the *Datacollector* can be connected using a simple client-server communication protocol. This allows us to synchronize the clocks of the various phones. Further, we can tag all events registered by the connected devices with a unique *trip ID*. When a data collection session is initiated, the *trip ID* is generated by the initiating device and propagated to all devices taking part in the collection.

Android provides developers with a sensor framework where the sampling rate of each available sensor can be separately defined. The effective sampling rate, however, comes usually with a standard deviation of one to two milliseconds. In addition, even though each sensor is collecting events at the provided sampling rate, there is often a shift of the exact sensing times (*e.g.*, while both, the barometer and accelerometer sensors collect data every 20 ms, the exact points of time, the samplings take place, deviate from each other by a few milliseconds). On the other hand, two data streams can be matched best when the sensors in both devices carry out their sampling steps at exactly the same points of time t. To dissolve this contradiction between precise sampling times and the aforementioned shortcomings of the Android framework, we implemented an interpolation technique in our *Data Analysis* tool which is described in detail in [9].

The deep learning model in DeepMatch was created to support travel times of varying lengths, and to reduce the amount of data to be transmitted from the devices in the public transport vehicles to the server, as well as the number of operations required by the server. To fulfill these requirements, we train our model to perform predictions on smaller *segments* of the collected events. In Sect. 5.4, we report the results from training the model on segment sizes of five, ten and 15 seconds. As elaborated in Sect. 5.4, our tests showed that the model being trained on segments consisting of ten seconds of barometer sensor events provides the best results.

3.4 Design and Architecture of the Learning Model

The goal of the deep learning model of DeepMatch is to predict the in-vehicle presence of a device by matching its sensor events against the sensor events generated by the on-board Reference Device (RefDev). The deep learning model consists of three modules, an *encoder*, a *decoder*, and a *matching module*. All three modules are trained jointly as one large neural network. In practice, however, the in-vehicle presence predictions can be achieved by utilizing only the encoder and the matching module. Therefore, we use the full model that also includes the decoder, *only* during the training phase. When our deep learning model is sufficiently trained, we extract the encoder and matching modules from it. Copies of the encoder are then used in the RefDev and the passenger devices, while the matching module is

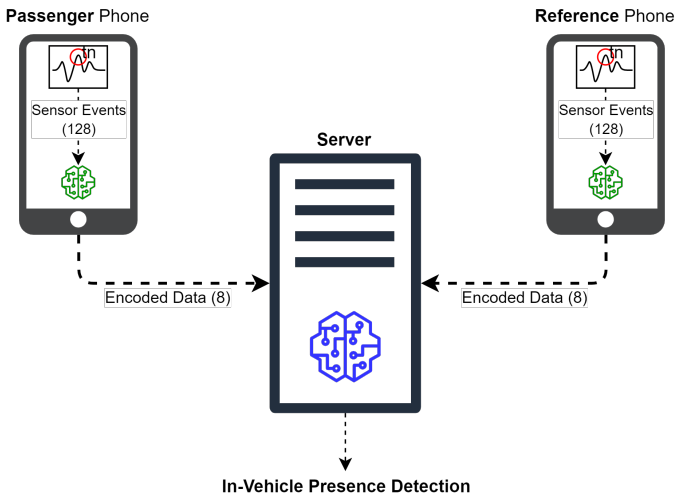

Figure 2: Overview of the DeepMatch distributed framework

executed in the server.

The distribution of the modules is depicted in Fig. 2. Here, the green networks in the passenger and reference devices illustrate that the encoders are residing on these devices. In contrast, the blue network illustrates the matching module that runs on the server.

Figure 3 provides a sketch of the neural network used in DeepMatch. The colored boxes represent layers of the neural network that can be trained while the grey boxes refer to model layers that do not contain trainable parameters. The green boxes describe trainable layers of the encoder, the orange ones trainable layers of the decoder, and the blue boxes trainable layers of the matching module. The hyperparameters of each layer in the neural network are represented as numbers next to the description of the layer. More specifically, in the boxes representing the conv layers, the size of each filter in the layer is represented as *width* x *height*, whilst the number on the right side of a box refers to the number of filters used. For instance, in the uppermost layers of the Stacked Convolutional Autoencoders in Figure 3, there are 128 filters of size 8x1. Furthermore, for the dense layers, the number of neurons in the layer is described by the number at the end of the layer description, *e.g.* in the first dense layer in the matching module the number of neurons is 256. The properties and utilization of the various layers are discussed below.

Since the matching module has to compare the sensor data from two devices, the RefDev and a passenger phone, we show two copies of the encoder and decoder in Fig. 3. This type of neural network topology is generally known as a *Siamese*

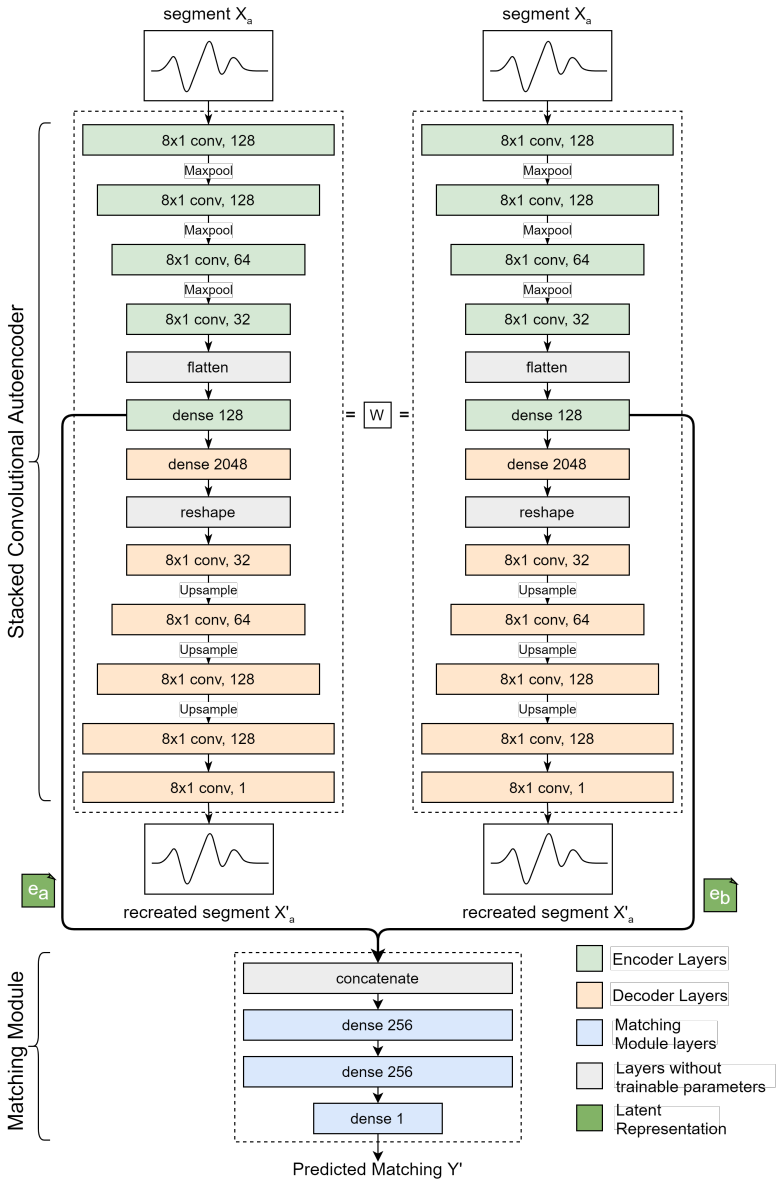

Figure 3: Original architecture of DeepMatch

Architecture and has been successfully used to solve other matching problems such as face recognition [26], gait recognition for person identification [27], and signature verification [28].

Configuring the deep learning model as a Siamese architecture provides the model with the ability to receive two simultaneous inputs, *e.g.* sensor data segments X_a and X_b. Since the two Convolutional Autoencoders share the same weights, the mapping performed by the encoders on the two inputs are identical. In consequence, two matching input segments, *i.e.*, samples of *Class 1*, result in latent representations e_a and e_b that are also matching. The same is true for not matching samples belonging to *Class 0*. Here, the two latent representations are dissimilar as well.

3.5 Encoder and Decoder

We use an architecture called *autoencoder* [29]. This kind of neural network consists of two parts, that directly reflect our encoder and decoder. The encoder transforms the input of the autoencoder into an internal representation often referred to as the *latent representation* in latent space, whilst the decoder aims to reconstruct the original input from its latent representation.

The loss, *i.e.*, the error of encoding and later decoding data, is calculated by comparing the input with the output of the autoencoder. The goal of training the autoencoder with a large set of samples is to keep this error minimal. As depicted in Fig. 3, autoencoders usually consist of several encoder and decoder layers through which the input data is sequentially forwarded. The quality of an autoencoder often depends on the arrangement of these layers. Usually, the length of the data forwarded between two layers is restricted such that the neural network needs to learn to prioritize the most important characteristics of its input, *i.e.*, the encoder must learn which features of its input are most relevant to perform a correct matching.

This *feature extraction*, can be seen as a compression algorithm. Then, the latent representation corresponds to the compressed data while the decoder is the corresponding decompression algorithm. In addition, the encoder provides *input noise reduction* since the compression forces it to learn the most important features of its input and to discard irrelevant features.

In DeepMatch, the ability to compress data is utilized to reduce the amount of transmitted information from the user phones and the RefDev to the external server. To let the autoencoder learn to prioritize those parts of the input data, that are most relevant, we train it together with the matching module in a single neural network. In this way, it learns to discard only those parts of the sensor events that are less important for the in-vehicle presence detection, but to keep all relevant data in the latent representation. This allows us to run the matching module based on the latent representation of the sensor such that the server does not need to decode them first.

As shown in Fig. 3, we created our autoencoder using alternating *convolutional* (*conv*) and *maxpooling* layers in the encoder part. Here, the conv layers are responsible for the feature extraction while the maxpool layers reduce the size resp. dimensions of the input. In the decoder, the conv layers alternate with *upsample* layers that are responsible for reverting the maxpool operation in the decoder.

We use the convolutional layers since they are especially suitable to detect and extract time-invariant features in sequences, see [20, 30–32]. Of course, this time-invariance is very important for our in-vehicle presence detection problem since we want to find out whether two devices are at the same place, *i.e.*, the same vehicle, independent from temporal influences like those caused by the distance between the phones in the vehicle. The maxpool layers reduce the size of their input data by a factor of two using the *max* operator. The upsample layers make it possible to reverse this process by duplicating each value in its input sequence, *e.g.*, $upsample(x, y, z) = x, x, y, y, z, z$. An autoencoder consisting of conv, maxpool, and sample layers, is called a *Stacked Convolutional AutoEncoder* (CAE) [31].

Each layer of our CAE consists of three or four more elemental machine learning operations. At first, a convolution is carried out followed by a Rectified Linear Unit (ReLU) activation. If the layer has maxpooling or upsampling functionality, this is executed after the ReLU. Finally, in each layer a batch normalization is carried out.

3.6 Matching Module

As previously mentioned, the matching predictions to find out if a smartphone is in the same vehicle as a RefDev, are performed by the matching module residing in a server. To match the sensor data previously compressed by the encoders, without having first to decompress them, the matching module needs to be able to compare the latent representations. To achieve this, we trained it to learn an accurate spatiotemporal threshold for separating instances of *Class 1*, *i.e.*, sensor events gathered by two devices in the same vehicle at the same time, from instances of *Class 0*, *i.e.*, a pair of sensor event sequences collected during different trips or at different locations.

The functionality of the matching module is represented by the blue boxes in Fig. 3. It consists of two *Dense* layers using the ReLU activation function, and finally another dense layer using the *Sigmoid* activation function. The Sigmoid function converts any real number into a value between zero and one. It is used in DeepMatch to describe whether the deep learning model believes that its input pairs belong to Class 1 or Class 0.

3.7 Model Training

As discussed above, the three modules of the DeepMatch deep learning model are jointly trained using the configuration shown in Fig. 3. In our Siamese architecture,

we observe two separate copies of the CAE that compress the sensor data segments X_a and X_b. Both CAEs share the same set W of trainable parameters causing the model to perform identical mappings for its two inputs. The CAEs produce the latent representations e_a and e_b that we illustrate as dark green squares in the figure. In the training phase, the latent representations are propagated through the layers of the decoders mapping them to the recreated segments X'_a and X'_b. In parallel, e_a and e_b are also sent to the matching module that is depicted by the blue boxes in Fig. 3. Here, the latent representations are being matched and the class prediction Y' is being produced. It assigns the value $Y' = 1$ if the model predicts that e_a and e_b are matches, and $Y' = 0$ if they are not. The values of Y' correspond to the ground truth labels Y of our sample pairs since we assign $Y = 1$ to a pair X_a and X_b if the samples are from *Class 1*. Likewise, for a sample pair of *Class 0*, we use $Y = 0$.

The goal of the training regime is now to train the layers of the matching module such that their computed values Y' are mostly identical to the ground truth labels Y of the training sample pairs. This kind of training utilizing external information about the training samples, *i.e.*, the ground truths, is usually called *Supervised Learning*. Formally, we describe the deviation between the ground truth Y and its prediction using the *Binary Cross Entropy* L:

$$L = -Y \cdot log(Y') + (1 - Y) \cdot log(1 - Y') \tag{1}$$

The goal of the training is to find weights for the layers of the matching module such that the prediction renders values of L close to zero.

In the same step, the layers of the encoder and decoder are trained by reducing the disagreements between the original segment pairs X_a and X_b and the recreated ones X'_a and X'_b. This improves the quality of the sample compression and decompression steps. Since neither the ground truths of the samples nor other external information is used, this training regime is referred to as *Unsupervised Learning*. We quantify the disagreements between the original and recreated segments using the *Mean Squared Error* MSE:

$$MSE = \frac{1}{n} \sum_{t=1}^{n} (X'_a[t] - X_a[t])^2 \tag{2}$$

In formula (2), n is the duration of the sensor event sequence X_a while $X'_a[t]$ is the recreation of the value $X_a[t] \in X_a$ at time t.

The disagreements between original and recreated samples as well as those between the ground truths and the values predicted by the matching module are used to update the weights of the neural network through the machine learning technique *Stochastic Gradient Descent*. In this technique, both the gradients from the mean squared error calculations and the Binary Cross Entropy loss functions are backpropagated to the neurons of the encoders. This enables our encoder to extract both, the most important features of the input segments for a good

recreation and the features that are relevant for an accurate matching prediction. In consequence, it is sufficient to use the latent representations e_a and e_b instead of the original sample pairs X_a and X_b to conduct the matchings. This is the reason that, when executing DeepMatch to detect real in-vehicle presence of passengers, we only need to use the encoder of the CAE in passenger smartphones and in the RefDev as well as the matching module in the server, while the functionality of the decoder module is not needed.

3.8 Design Rationale and Experimental Settings behind the DeepMatch Model

In our quest to find the best model, we conducted hundreds of experiments on various model design and hyperparameter configurations. Our approach relied on starting with smaller, shallower neural networks, before expanding them by adding layers, filters within the CONV layers, and by increasing the size of these filters. To keep track of the various experiments, every configuration and design of the network was evaluated using the performance metrics described in Section 5.2. During this work, we also experimented with various activation functions for both CONV and dense layers. Moreover, we tried swapping the CONV layers in the Autoencoders with dense layers and exchanging the Matching Module with a function calculating the Euclidean Distance between the latent representations, respectively. Furthermore, we experimented on how we trained the modules of the network, *e.g.* we tried training the autoencoders separately from the matching module. This was done by first training the autoencoders, and thereafter using the autoencoders to create datasets consisting of encoded samples. These encoded samples where then used to train the matching module. In addition to design, architectural, and training experiments, we tested a large number of hyperparameter settings, *e.g.* the number of neurons in each dense layer, the size of batches used during training, the number of epochs for each training session and so on. From the experiments conducted for our previous work in [9], the model architecture shown in Figre 3, using the hyperparameter settings described in Section 3.4, gave us the pest performance. All experiments were conducted on a desktop PC with an Intel i7 4.00GHz CPU, 16 GB memory, and a Nvidia GTX 1080 GPU. The models were created, trained and evaluated using Google tensorflow 2.0, version 2.0.0-rc0 [33].

4 DeepMatch2

Since the original publication of DeepMatch in [9], we continuously iterated and improved our deep learning models. That has led to several improvements that we could incorporate into the new version DeepMatch2 of our in-vehicle presence detection system. In particular, we amended the layer structure of the architecture.

In the following, we will discuss the improvements in greater detail.

4.1 Design Rationale and Experimental Settings of the DeepMatch2 Model

We started the work on our new model basing it on the original architecture of DeepMatch, depicted in Figure 3. We followed the approach described in Sect. 3.8 by gradually making incremental changes to the model architecture, and for each change, training and evaluating the results using the performance metrics described in Sect. 5.2. Moreover, we investigated adapting the numbers of convolutional layers and filters within the CONV layers as well as changing the sizes of the individual filters. Thereafter, we experimented with the ratio between the numbers of CONV and maxpool layers used by the CAEs. In particular, the structure of the matching module was modified. DeepMatch2 uses another method to concatenate the two latent representations, the module receives from the autoencoders. This is described in detail in Sect. 4.3. Our experiments revealed that the model architecture and the model parameters depicted in Fig. 4 yielded the best results. Also for these experiments, we used the desktop PC and Google tensorflow version described in Sect. 3.8.

4.2 Dimensionality Reduction

During the initial development of our original deep learning model, we were not aware that the best in-vehicle detection accuracy could be achieved using events from the barometer sensor alone. We learned this fact through the empirical studies described in Sect. 5.3, for which a first version of DeepMatch was needed.

After gaining the insight that just barometer events would be sufficient for inference, we started to refine and restructure parts of the model in order to better accommodate samples containing only this type of data. Our aim hereby was to utilize the limitation to barometer inputs for a reduction of the size of data transmitted between the distributed computational units. Of course, this should happen without worsening the accuracy of the in-vehicle prediction, but rather with an improvement.

Our first major change was to reduce the size of the model inputs, from previously 512 float values in DeepMatch to just 128 float values in DeepMatch2. The model of the previous version was laid out to accept events from multiple smartphone sensors that publish events at fixed rates but possibly with different maximum frequencies and varying reading points in time. In each layer of our model, the maxpool operator reduces the size of its input by the factor of two such that the input length needs to be a multiple of two. However, the output of a layer must also be a multiple of two since it is directly used as the input for the maxpool operator of the next layer. To guarantee this property for a number of subsequent layers, we therefore need the size of the initial encoder input to be

an exponent of two. The highest frequency of the sensors tested in DeepMatch was 50 Hz, and the first sample size, we aimed to create, was 10 seconds worth of sensor data resulting in 500 events. Considering the aforementioned requirement that the input must be an exponent of two, the closest sample size to 500 was 512 events, resulting in an actual sample size length of 10.24 seconds.

On the other hand, the barometer sensor conducts its sensing with a maximum frequency of 10 Hz such that a sample of 10 seconds only needs 100 events. To fulfill the demand of using a number of events that is an exponent of two, we decided to increase the sample period to 12.8 seconds such that a sample processed in DeepMatch2 contains 128 barometer events. That is just a quarter of the original sample size.

4.3 Accuracy Improvement

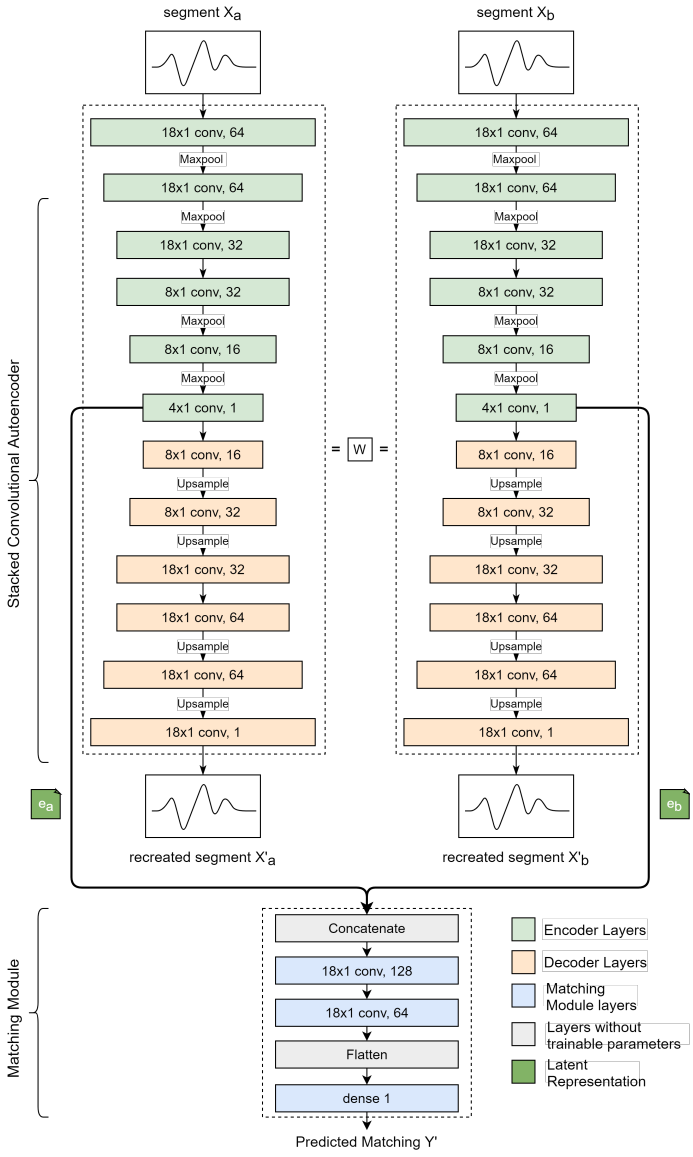

Figure 4: Architecture of DeepMatch2

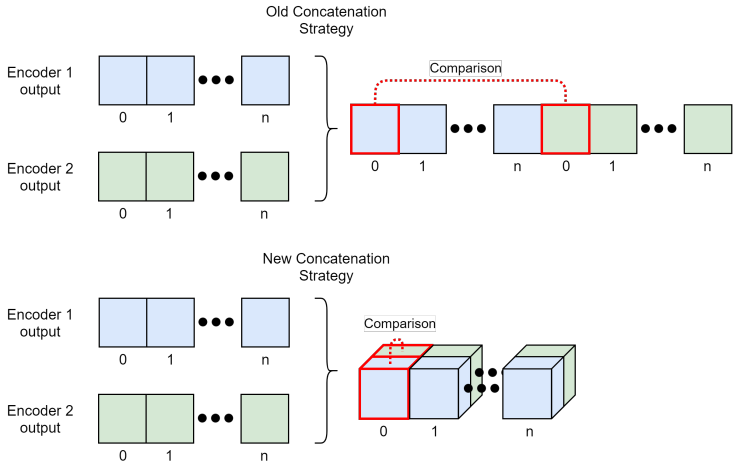

Figure 5: Old and new Matching module input concatenation strategy

While the accuracy of 97.81% of the original deep learning model is quite good, we aimed to make it even better in DeepMatch2. The most impactful change, we made to improve the accuracy of our deep learning model architecture, was altering the way the matching module concatenates its two inputs. These changes are illustrated in Fig. 5. As shown at the top of the figure, in DeepMatch, the two input samples were simply concatenated along the first axis. That means that two float vectors that both had the length x, were transformed into one vector of the length $2x$. This concatenation, however, usually creates a large spatial distance between the pairs of values that need to be compared by the matching module.

To reduce this distance, we altered the concatenation by transforming two input vectors of length x into a matrix with the size $(x, 2)$ which is shown at the bottom of Fig. 5. This adaptation to the input of the matching module allowed us to replace some of the dense layers by convolutional layers. As discussed in Sect. 3.5, convolutional layers are well suited to handle time-invariant features in different sequences such that they promise to be a better fit than the dense layers.

As a result, we achieved the layer structure that is illustrated by the blue boxes in Fig. 4. Its uppermost layer is a concatenate layer that concatenates the inputs from the two encoders by transferring them into a matrix of the size $(x, 2)$. This matrix then forms the input of the first of two convolutional layers. Thereafter, a *Flatten* layer flattens its n-dimensional input into a one-dimensional vector. Finally, a dense layer uses the Sigmoid function explained in Sect. 3.6. This amended layout led to an improved accuracy of 98.51% which we discuss in greater detail in Sect. 5.6.

Table 1: Smartphones used by volunteers to collect data

Brand	Model
LG	Nexus 5X
Huawei	Nexus P6
Samsung	Galaxy S8
Sony	Z3 Compact
Google	Pixel XL
Google	Pixel 3a

5 Evaluating the Deep Learning Models

Our evaluation effort includes three main steps. *First*, we explain how we created our training data, and present the performance metrics used to evaluate different models. Thereafter, we report on the experiments carried out to find which sensor data are best for in-vehicle presence detection and the best segment size (cf. Subsect. 5.1—5.4). *Second*, we evaluate the prediction performance of both DeepMatch and DeepMatch2. To this end, we compare several versions of the original deep learning model DeepMatch with varying sample lengths. This is followed by comparison of DeepMatch with two well-known baseline methods and, of course, with our updated version DeepMatch2 (cf. Subsect. 5.5 and 5.6). *Third*, we investigate and discuss the execution time overhead of the matching module on the server, and the battery consumption as well as the CPU run-time overhead for the smartphones of the passengers (cf. Subsect. 5.7—5.9). Note that the evaluation results, in this step, *apply to both versions of our learning models*, namely, DeepMatch and DeepMatch2.

5.1 Data Collection and Dataset Creation

The data used to develop and evaluate our deep learning model was collected in various public transportation vehicles (*i.e.*, trains, subways, busses and trams) in the Norwegian cities of Oslo and Trondheim by volunteers. They used different Android phones that are listed in Tab. 1. All these phones were provided with the *Datacollector* application introduced in Sect. 3.3 such that their clocks could be synchronized and common *trip IDs* assigned.

In total, our volunteers collected 212,520 seconds of unique sensor data events from the magnetometer, accelerometer, gyroscope and barometer sensors. Segments of sensor events from two different sources were paired in each data set sample to model that DeepMatch and DeepMatch2 match pairs of sensor data segments from a user's phone and a RefDev. Moreover, the pairs of sensor event segments were classified as illustrated in Fig. 6. Thus, sensor event segments with identical trip IDs and beacon identifiers, *i.e.*, segments computed by different de-

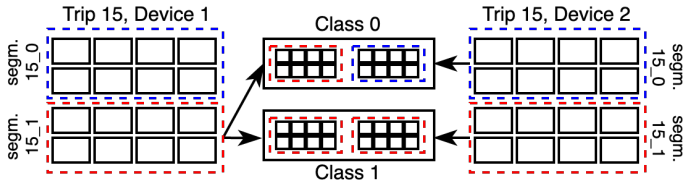

Figure 6: Matching samples created from trip segments

vices in the same vehicle at the same time, were labeled as *positive samples*, *i.e.*, *Class 1*. In contrast, event segment pairs with differing trip IDs or beacon identifiers were fetched at different times or in different vehicles. Therefore, they are labeled as *negative samples*, *i.e.*, *Class 0*.

Following common practice in machine learning, we shuffled the samples of our dataset. Thereafter, we normalized the data using minmax and, finally, we split our samples into training and testing ones. The training set consisted of around 70% of the overall data. The other 30% were used to create testing sets for the purpose of model evaluation. By completely separating the training from the testing samples, we avoided to use the same sensor events in both phases. Further, we made sure that the distribution of *Class 1* and *Class 0*, were about 50/50 in both data sets.

5.2 Metrics to Evaluate Learning Models

To evaluate the different versions of our deep learning model with each other and with other methods introduced later, we use the four metrics *precision*, *recall*, *accuracy*, and *F1-score* that all are popular means to evaluate machine learning models. In order to define these metrics, we use four binary classifiers that describe if a sample is positive or negative in reality, and if it is correctly classified:

- *True Positive(TP)*: A correctly classified positive sample,
- *True Negative(TN)*: A correctly classified negative sample,
- *False Positive(FP)*: A negative sample that is falsely classified as a positive one,
- *False Negative(FN)*: A positive sample that is wrongly classified as a negative one.

With the help of these classifiers, we can now introduce the four metrics used to evaluate the models:

- *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*:

$$PR = \frac{TP}{TP + FP} \tag{3}$$

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

$$RE = \frac{TP}{TP + FN} \tag{4}$$

- *Accuracy (ACC)*: This metric states how good the model classifies samples from all classes, *i.e.*, it describes how many of all predictions are correct:

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

  The accuracy results are usually only reliable if the number of members from *Class 0* and *Class 1* are about equal. That is the reason for using equal representation of *Class 1* and *Class 0* in our training and testing sets (see Sect. 5.1).

- *F1-score (F1)*: The harmonic mean between precision and recall.

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

In the various experiments introduced in the next subsections, we compare the different methods using these four metrics.

5.3 Sensor Modality Experiments

In a first set of experiments, we wanted to find out which combinations of smartphone sensors are most suited to be used in our deep learning model. To be able to compare various sensor mixes, we created the following seven sensor modality combinations: Accelerometer (A), Magnetometer (M), Barometer (B), Barometer and Accelerometer (BA), Barometer and Magnetometer (BM), Accelerometer, Magnetometer, and Gyroscope (AMG), and Accelerometer, Magnetometer, Gyroscope, and Barometer (AMGB).

We trained and tested each of these modality combinations with our original deep learning method DeepMatch using sample lengths of 10 seconds. The test results of these experiments are depicted in Tab. 2. From the table, we can see that the models trained on datasets of type DeepMatch 10 B containing sensor events only from the barometer, have a significantly higher precision, accuracy, and F1-scores than the other models. The recall values are much closer for all

Table 2: Performance comparison between various sensor combinations

Model	PR	RE	ACC	F1
DeepMatch 10 A	0.5065	0.9531	0.5122	0.6615
DeepMatch 10 M	0.5064	0.9280	0.5118	0.6553
DeepMatch 10 B	**0.9751**	**0.9812**	**0.9781**	**0.9781**
DeepMatch 10 BA	0.7332	0.9697	0.8082	0.8350
DeepMatch 10 BM	0.7081	0.9708	0.7853	0.8189
DeepMatch 10 AMG	0.5011	0.9646	0.5020	0.6595
DeepMatch 10 AMGB	0.7079	**0.9892**	0.7905	0.8253

models but also here DeepMatch 10 B finishes in the top two, albeit a little behind DeepMatch 10 AMGB.

We believe, the reason for the good result of just using the barometer is that this sensor particularly suited to capture the movements of the vehicle rather than those of the smartphone user. For instance, it is position independent, *i.e.*, the precision of the barometer events is not affected by the location of the sensor. This is especially important for the use in underground transportation, *e.g.*, in subways, where the GPS performs poorly. Further, the barometer is highly resistant to vibrations as well as movements of the smartphone user. This in stark contrast to the accelerometer and gyroscope which are more strongly affected by the movements of the user than by those of the vehicle. Unlike the barometer, the magnetometer is highly sensitive to magnetic objects in the environment like the power unit of the vehicle. All these traits make the barometer perfectly suited to capture just the movements of the vehicle and to ignore the movements and immediate surroundings of the carrier of the smart device.

The absence of the mentioned weaknesses of the other sensors makes it easier for our deep learning model to learn how to separate instances of *Class 1* and *Class 0* only using the barometer events. This is confirmed by the overall good values for DeepMatch 10 B in comparison to the other sensor combinations. The fact, that DeepMatch 10 AMGB has a little better recall value, results probably from a tendency to classify samples as positive even if they are negative in reality. This is not punished by the recall value but by the precision and accuracy values that are meager for DeepMatch 10 AMGB.

Altogether, our sensor modality experiments led to the decision to consider only the barometer sensor data for our deep learning method. This is in accordance with most of the other works introduced in Sect. 2.2 which also claim that the barometer data are best for classifying in-vehicle detection, *e.g.*, [15, 16].

Table 3: Performance comparison of the barometer-based DeepMatch with various sample lengths as well as with baseline methods and DeepMatch2

Model	PR	RE	ACC	F1
DeepMatch 5	0.9408	0.9765	0.9574	0.9583
DeepMatch 10	0.9751	0.9812	0.9781	0.9781
DeepMatch 15	0.9348	0.9816	0.9566	0.9576
NORM_CORR	0.9174	0.9595	0.9393	0.9380
DTW	**0.9810**	0.7350	0.8136	0.8404
DeepMatch2	0.9769	**0.9935**	**0.9851**	**0.9851**

5.4 Segment Size Experiments

After deciding to base the deep learning model just on the barometer input, we wanted to find the sample length for which DeepMatch renders the best results. Therefore, we trained and tested the model with different sample lengths of five, ten, and 15 seconds. The results are shown in the first three lines of Tab. 3.

We see that variant DeepMatch 10 with its ten seconds long samples outperforms DeepMatch 5, the model trained on matching samples of five seconds. The cause is most likely the greater number of sensor events contained in a DeepMatch 10 sample. This provides a better foundation for training the neural network in DeepMatch 10 than in DeepMatch 5.

Following this logic, however, we should expect that DeepMatch 15, the model trained on 15 seconds long matching data, outperforms DeepMatch 10 since it has an even higher number of sensor events available in a sample. Yet, this is not the case for the precision, accuracy, and F1-score performance metrics, while the recall values are basically even. Like with DeepMatch AMGB in the tests discussed in Sect. 5.3, it seems that DeepMatch 15 is biased towards classifying samples as positive since it produced good recall but bad precision values. The most likely reason for this surprising effect is that we have fewer 15 seconds long samples available than shorter ones in our training set. In consequence, there might be simply too few sample pairs available to train the neural network well.

Due to the ongoing Covid-19 pandemic it is currently too dangerous to send our volunteers into public transport vehicles. When the pandemic is over, however, we will collect a larger number of longer samples expecting that DeepMatch 15 will outperform DeepMatch 10 when also these can be used for training and testing.

5.5 Comparing DeepMatch with two Baseline Methods

In order to get a better comparison of our deep learning method with other possible approaches, we also employed two well-known baseline methods that seem to be suited to perform sensor event matching for in-vehicle presence prediction. One of the selected baseline methods is *Normalized Correlation (NORM_CORR)*. It calculates the correlation between two vectors by comparing the values in the same position. The other baseline method is *Dynamic Time Warping (DTW)*. In DTW, all values in the two vectors are compared by warping the temporal dimension until the best correlation for any data point is found. In consequence, DTW does not inherently describe the correlation between two vectors but the distance between them, and a large distance equals a small correlation. Thus, to use DTW as a measure of correlation on par with NORM_CORR, we had to inverse the results produced by it.

In both baseline methods, we need to find a threshold value α such that all sample pairs with a correlation c larger than α are from *Class 1* and all others from *Class 0*. This corresponds to the following equation:

$$c = f(X_a, X_b), \quad Y' = \begin{cases} 1 & \text{if } c > \alpha \\ 0 & \text{else} \end{cases} \tag{7}$$

Here f represents the baseline method used. To find a good threshold α, we first applied f to all samples of our training set and added the resulting c-values to a sorted array. Thereafter, we searched the sorted array for an optimal delimiting value that minimizes the number of falsely grouped sample pairs. In the final step, α was set to this delimiter.

The results of our baseline methods tests are shown in the fourth and fifth lines in Tab. 3. We see that, except for the precision of DTW, the two baselines are clearly outperformed by DeepMatch with the different sample sizes.

The reason for the poor performance of NORM_CORR is most likely its sensitivity to potential time-lags between its input sequences. This is due to the fact that two passengers, who are at different locations in a public transportation vehicle, register changes in altitude with a time-lag that corresponds with the quotient of the spatial distance between them and the speed of the vehicle. This time-lag produces a lower correlation value for NORM_CORR even though the passengers are in the same vehicle.

On the other hand, the poor results of DTW are most likely caused by its insensitivity to the temporal dimension. Warping the temporal dimension can cause the function to achieve a very high correlation value for some negative samples which increases the number of false positives. While this error is rewarded by the precision metric, it makes it very hard to find a good delimiter value α. This is the likely reason that the other three metrics are particularly bad for DTW.

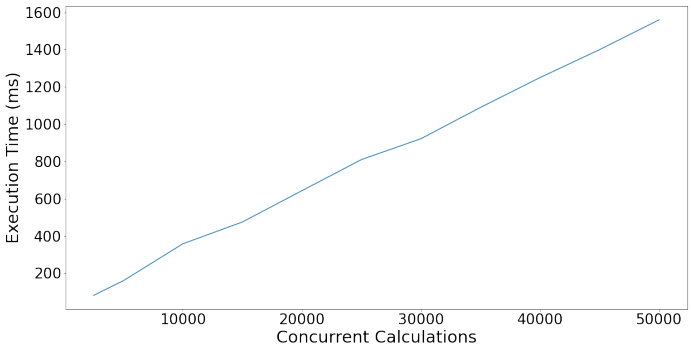

Figure 7: Execution time to execute matchings in parallel

Altogether, it seems that the baseline methods are less suited to perform in-vehicle presence detection on barometer data than DeepMatch.

5.6 Prediction Performance of DeepMatch2

We also trained our updated version DeepMatch2 with the available data. The performance results of our tests are listed in the last row of Tab. 3. We can see that, except for the precision value of DTW that was already discussed above, DeepMatch2 outperforms all other tested models for all four performance metrics used. If we take a closer look at the results, we see that the largest change between DeepMatch 10 and DeepMatch2 is the increase of the recall value by more than a percentage point. The likely reason is that the improvements we made to the model (see Sect. 4), increased the ability of DeepMatch2 to correctly classify positive samples of the dataset. This could be achieved without classifying too many samples as positive, as can be seen by its precision value that is also slightly better than the one of DeepMatch 10.

Achieving good results for both the precision and recall means that DeepMatch2 is good at separating the samples in our evaluation dataset. That is also proved by the very high accuracy and F1-scores of DeepMatch2 which are both around 0.7% better than their counterparts in DeepMatch 10. Thus, in spite of increasing the compression of the sensor data by the factor of four, we managed to improve the performance metrics of DeepMatch2.

5.7 Execution Time in the Server

Since usually a lot of passengers use public transportation throughout the day, particularly during the rush hour, the server of a public transport authority performing the matching calculations of DeepMatch or DeepMatch2, needs to

be able to handle large amounts of concurrent data simultaneously. To test the expected load for such a central server, we exploited a feature of Tensorflow that allows the matching module to accept multiple inputs. Further, we used the powerful parallel computational capabilities of Tensorflow that make it possible to calculate the matchings for all received inputs simultaneously.

Fig. 7 depicts the execution time of the matching module performing its matches based on latent data, after it has been extracted from the overall deep learning model of DeepMatch2. It reveals that 50,000 matching calculations can be executed in parallel in 1,560 milliseconds all running on a single five years old GTX 1080 GPU. Due to the the fact that the matching calculation is only performed once for every 12.8 seconds of collected data per passenger, a data center consisting of only three such GPUs could serve a city like Oslo with its 960,000 daily passenger trips even if all passengers travel at the same time. Running these calculation on a newer GPU with improved concurrency and computational capabilities, would improve these results even further.

5.8 Battery Consumption on Smartphones

Power consumption is an important issue when using deep learning models on mobile devices that run on rechargeable batteries since the models often require a large number of calculations at high speed, which usually demands a lot of power. Fortunately, we do not require the *encoders* executed on the smartphones to run continuously in our approach. Even if the phone is in close proximity to a a BLE-transmitter (see Sect. 3.1), we only carry out the encoder in certain intervals corresponding to the lengths of the samples to be matched, *e.g.*, every 12.8 seconds when using DeepMatch2.

In addition to running the deep learning model, the continuous sensor event generation and the transmission of the compressed data to the server are power consuming tasks the smartphones will have to perform. To ensure that our approach does not cause an excessive drain on the batteries of the smartphones, we constructed three test scenarios that reflect the battery consumption factors mentioned above:

- *Complete scenario*: All three factors of battery consumption, *i.e.*, the barometer data collection, data processing by the encoder, and data transmission,
- *Learning scenario*: Barometer data collection and data processing,
- *Data collection scenario*: Only barometer data collection.

To ensure diversity in our testing devices, we used five Android smartphones from five different manufacturers. To ensure age diversity, these phones are between zero and six years old, as can be seen in Tab. 4. To make sure that the test results were not influenced by the environments in which the tests were run, we

Table 4: Android phones used in battery tests

Brand	Age	Battery Capacity
LG Nexus 5X	5 years	2700 mAh
Huawei Nexus P6	4 years	3450 mAh
Samsung Galaxy S8	3 years	3000 mAh
Sony Z3 Compact	6 years	2600 mAh
Google Pixel 3a	0 years	3000 mAh

Table 5: Battery consumption per hour

Brand	Data collection	Learning	Complete
Samsung	25 mA	26 mA	31 mA
LG	23 mA	24 mA	26 mA
Huawei	22 mA	23 mA	25 mA
Google	16 mA	17 mA	18 mA
Sony	15 mA	18 mA	21 mA

performed all tests indoors with a constant temperature of $19°C$, representing the indoor temperature of a typical public transportation vehicle in Norway.

To measure the battery power usage, we used the tools *Batterystats* and *Battery Historian* provided by Google to log the battery consumption of all processes running on an Android device [34]. The tests were run in the background with the *wake lock* parameter enabled. This allowed us to simulate an environment in which our application retrieving sensor inputs, encoding them, and forwarding the encoded data to the server, runs in the background of a user phone.

The results of our tests are depicted in Tab. 5. They clearly show that for all devices used in the tests, our learning models influence the battery consumption on a smart device only marginally. Considering a passenger travelling with a public transportation vehicle for over two hours, measuring, encoding and transmitting barometer sensor events, only around 62 mAh will be used for the Samsung Galaxy S8, the phone with the highest power usage. With a battery capacity of 3000 mAh, this equals to the use of 2.1% of the overall battery capacity. Compared to the numbers reported in [35], this is substantially lower than most smartphone applications.

As a result of our tests, we consider that our approach has no significant negative impact on the overall battery consumption of the user smartphones. For the reference devices, we expect that they have a power connection with the battery of the transport vehicle.

Table 6: Run Time and CPU overhead

Brand	CPU	Mean Run Time	Overhead
Samsung	2.3 GHz + 1.7 GHz, Cortex-A53	49 ms	1-2 %
LG	1.4 GHz + 1.8 GHz, 64-Bit Hexa-Core	46 ms	1-2 %
Huawei	2.0 GHz + 1.55 GHz, 64-Bit Octa-Core	52 ms	1-2 %
Google	2.0 GHz + 1.7 GHz, 64-Bit Octa-Core	19 ms	0-1 %
Sony	2.5 GHz Quad-Core, 400 Krait	73 ms	3-4 %

5.9 Computational Overhead on Smartphones

As mentioned above, deep learning models are often large complex computational units. Thus, in addition to affecting the battery consumption of the devices running the model, the computations might influence their CPU usage such that a smartphone is not able to support other applications, that the user likes to run in parallel to our learning models DeepMatch or DeepMatch2. Therefore, we evaluated also the computational overhead of the models executed on smartphones. For these experiments, we used the same five devices listed in Tab. 4. We analysed both, the CPU usage and the time, the encoder module needed to process one sample of input. Table 6 shows the results of these tests. We can clearly see from the mean run time and CPU overhead produced for the devices, that the overhead of our models are barely noticeable and should not impact other functions executed on the phone in parallel.

6 Travelling User Inference

Up to now, DeepMatch and DeepMatch2 can determine with a high accuracy whether a person's smartphone is in the same public transport vehicle as a RefDev over a fixed time interval of, *e.g.*, 12.8 *s*. To make our approach usable for, *e.g.*, automatic ticketing, however, a solution is required to find out, over which time period the owner of the phone effectively travels in the vehicle.

To infer the duration of being in the vehicle, we can utilize that a normal trip in a city bus may be up to an hour such that DeepMatch2 can conduct hundreds of samples. The samples of the user's phone taken on a trip and the corresponding ones produced by the reference device are paired and merged to a so-called *matching sequence*. This is illustrated in Fig. 8.

As an example, let us assume that a passenger travels with a bus for 20 minutes.

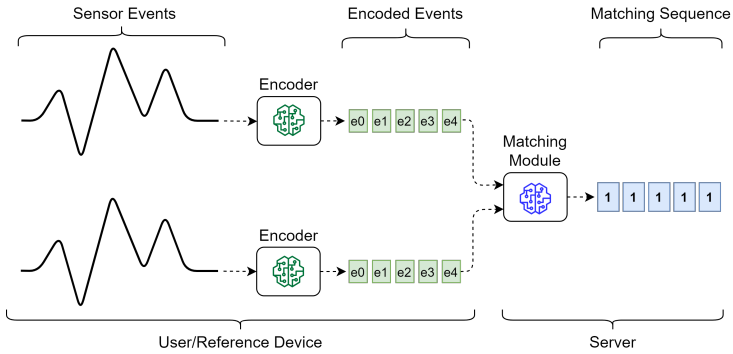

Figure 8: Matching Sequence output from the matching module

That gives DeepMatch2 the necessary time to generate a matching sequence containing 93 successive matches. Due to the accuracy of 98.51% for DeepMatch2, the likelihood that all these matches are correctly detected as being in-vehicle (*i.e.*, *Class 1*), however, is only 24.76%. Thus, in more than three quarters of trips with a 20 minute duration, at least one matching will be falsely declared as being out of the vehicle (*i.e.*, *Class 0*). Similarly, if a person rides in a car next to a bus for 20 minutes, e.g., due to slow moving traffic, the chance that all matches are *Class 0*, is also only around 25%. Thus, we need an algorithm that can infer passenger trips from matching sequences with a high degree of precision in spite of occasional matching errors. In this section, we describe how one can develop and evaluate such an inference algorithm.

In the following, we first describe how we gathered the data that explains the travel time between two adjacent stops for busses in Norway's capital Oslo. Thereafter, we introduce some concepts based on which the algorithm is designed and presented. A relevant parameter of this algorithm is the degree of fault tolerance it supports, which is investigated at the end of this section.

6.1 Travelling Times between Adjacent Bus Stops

To get the travelling times between two adjacent stops for busses in Oslo, we gathered all real-time data of the bus network of Oslo over twelve hours on a normal Monday. ENTUR, a government-funded organization, gathers and openly shares traffic data from all public transportation operators in Norway. In particular, it offers an API [36] which is based on the SIRI 2.0 standard [37].

The collected data allows us to understand the distribution of the travelling times between two adjacent stops. To this end, we aggregated the arrival and departure times for all buses at all stops throughout the recorded 12-hour slot. From this, we calculated for all bus trips the times needed to travel between two

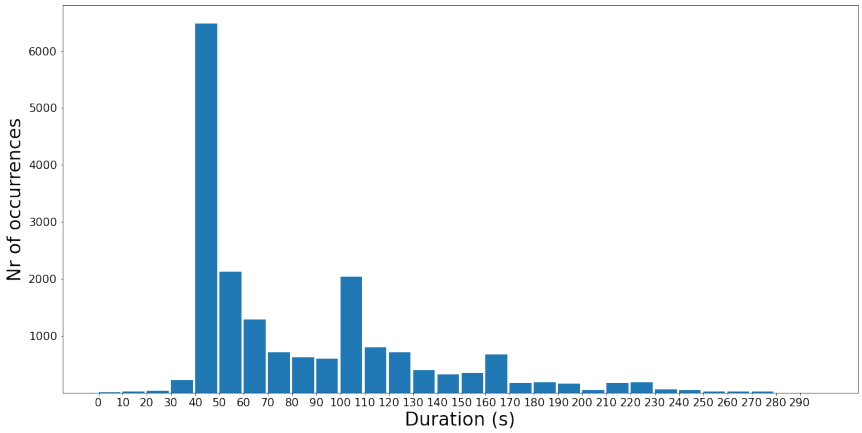

Figure 9: Number of times any vehicle traveled a route segment in x seconds

adjacent stops. The results were summed up, and we obtained the distribution depicted in Fig. 9. The x axis refers to travelling times in seconds while the y axis shows how often busses needed a particular time segment between two neighboring stops during the 12-hour slot. As can be seen, the most prevalent travelling time between two adjacent stops is between 40 s and 50 s. Another interesting fact is that in less than 2% of the gathered cases, the travelling time between two stops is less than 40 seconds. We will show how these facts can be utilized to provide good results.

6.2 Matching Sequences and Travel Inference Algorithms

As introduced above and illustrated in Fig. 8, a *matching sequence* is the sequence of outputs generated by the matching module of DeepMatch2. It is basically a sequence of *ones* and *zeros* that forms the input of the *inference algorithm*. Based on this input, the inference algorithm decides whether the user traveled in the vehicle for the duration of the matching sequence, or not.

To better understand the possible errors that the inference algorithm could make, we classify its results (in analogy to the true and false positives and negatives introduced in Sect. 5.2) as follows:

- *True Travelling user* (T_T) denotes an actual passenger who is correctly inferred as a traveller.
- *False Travelling user* (T_F) is a person not using the public transport vehicle, but who was falsely inferred to be travelling in it.

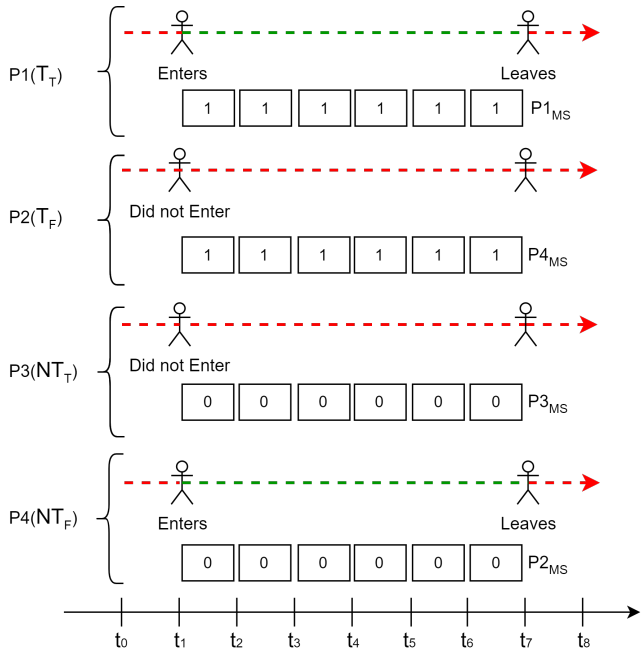

Figure 10: Example travellers and expected matching sequence

- *True Non-Travelling user* (NT_T) is a user not travelling with the vehicle in question who is correctly inferred as being not in the vehicle.
- *False Non-Travelling user* (NT_F) is an actual travelling user who was, however, falsely inferred as not travelling with the vehicle.

In Fig. 10, the four types of travellers are illustrated, where green dashed lines indicate the time period that the passengers are inside the vehicle, while red dashed lines refer to the phases they are outside. The orders of white boxes beneath the dotted lines represent the matching sequence for each passenger. The goal of the algorithm is to detect true travelling and true non-travelling users with a high accuracy. In particular, it should keep the number of false travelling users very low since billing people who do not use public transport, may easily lead to complaints, lawsuits, and bad press for the operator. Moreover, false non-travelling users shall be avoided since they end up with travelling for free.

6.3 User Travel Inference Algorithm

In this section, we describe our methodology to design and evaluate the inference algorithm. It shall take the accuracy of DeepMatch2 of 98.51% into account.

While this accuracy is relatively high, it is not perfect as matching sequences can occasionally contain a mix of *ones* and *zeros*, and the inference algorithm should be able to perform inference with a high accuracy even from such mixed sequences. Overall, our goal is to find an inference strategy that keeps the number of false non-travellers and false travellers low.

We first need to calculate the accuracy of an inference algorithm based on the accuracy of DeepMatch2. We use the *binomial experiment* to calculate the likelihood of occurring a certain ratio of *ones* and *zeros* in a matching sequence:

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{8}$$

Here, n denotes the number of trials, *i.e.*, the length of the matching sequence, while k refers to the number of successful trials, *i.e.*, the number of correct matchings in the sequence. Finally, p states the probability of a successful trial which, in our case, corresponds to the accuracy of DeepMatch2, *i.e.*, 0.9851.

With the binomial experiment, we can compute the two probabilities that a certain proportion of *ones* and *zeros* in a matching sequence is rightfully inferred to be in or out of the vehicle. From that, we can easily determine the likelihoods for the four user types introduced in Sect. 6.2.

As an example, let us assume that, in fear of inferring false travelling users, the public transport authority decides to bill a passenger when all the six entries in the matching sequence are *ones*. Using formula (8) leads to the following likelihoods for the four inference results:

$$\begin{aligned} P_{T_T}(6) &= 0.9851^6 = 0.9139 \\ P_{NT_F}(6) &= 1 - P_{T_T}(6) = 0.0861 \\ P_{T_F}(0) &= (1-0.9851)^6 = 1.09 \cdot 10^{-11} \\ P_{NT_T}(0) &= 1 - P_{NT_F}(0) = 0.999999999989 \end{aligned}$$

The likelihood, that DeepMatch2 produced all six *Class 1* predictions wrongly and with that infers a false travelling user, is $P_{T_F}(0) = 1.09 \cdot 10^{-11}$. In correspondence, the likelihood to detect a true non-traveller is very high since our strategy declares all passengers who produced at least one *zero* in their matching sequences, as not travelling. The corresponding probability is $1 - P_{T_{NF}}(0) = 0.999999999989$.

The price of this very rigid approach is the relatively high rate of false non-travelling users which is calculated as $1 - P_{T_T}(6) = 0.0861$. Thus, nearly every twelfth passenger gets a free ride which might be unacceptable for most operators. In consequence, the likelihood for a true travelling user is only $P_{T_T}(6) = 0.9139$. To avoid such a large rate of false non-travelling users, we need to bring some *tolerance* into our inference algorithm. To achieve that, we extend equation (8) for the binomial probability to the so-called cumulative binomial probability, that

is described by the following formulas:

$$P(k \geq M) = \sum_{k=M}^{n} \binom{n}{k} p^k (1-p)^{n-k} \tag{9}$$

$$P(k \leq M) = \sum_{k=0}^{M} \binom{n}{k} p^k (1-p)^{n-k} \tag{10}$$

In formula (9), M refers to the *minimum* and in (10) to the *maximum* number of trials that have to be successful in order to accept a trip as in-vehicle. The other symbols used in these formulas are identical to those introduced for formula (8).

The cumulative binomial probability provides the means to calculate the probabilities for more fault-tolerant inference algorithms, *e.g.*, accepting matching sequences of the length six as in-vehicle, when they contain at least five *ones*. This algorithm leads to the following results for the four inference results:

$$\begin{aligned} P_{T_T}(k \geq 5) &= 0.9851^6 + 6 \cdot 0.9851^5(1 - 0.9851) = 0.9968 \\ P_{NT_F} &= (1 - P_{T_T}(k \geq 5) = 0.0032 \\ P_{T_F}(k \leq 1) &= (1 - 0.9851)^6 + 6 \cdot 0.9851(1 - 0.9851)^5 = 4.35 \cdot 10^{-9} \\ P_{NT_T} &= (1 - P_{T_F}(k \leq 1)) = 0.9999999957 \end{aligned}$$

This more fault-tolerant algorithm reduces the likelihood of false non-travelling users to just around 0.3%. While the number of false travelling users is increased by two digits compared to the more rigid algorithm described above, it is still very low. Therefore, this more tolerant inference algorithm seems to be a better fit for a billing system than the stricter one discussed above.

The fact is that we still do not know if this algorithm is the best, or if more fault-tolerant policies render even better results. Moreover, for most passenger trips, DeepMatch2 produces much longer matching sequences than ones with just six matching pairs. Utilizing the data from the public transportation behavior analysis discussed in Sect. 6.1, we conducted several experiments to find out the best inference algorithm solutions, which are discussed below.

6.4 Considering Different Forms of Fault Tolerance

The aforementioned comparison of two algorithms for matching sequences of length six shows that there is a trade-off between false travellers and false non-travellers based on the degree of fault tolerance—the percentage of *zero* values that a matching sequence may contain whilst still being inferred as in-vehicle. In the rigid example in Sect. 6.3, the fault tolerance was 0, while for the second case it was $\frac{1}{6} = 16.\bar{6}\%$. The value M in the cumulative binomial probability formulas (9) and (10) can be calculated from the fault tolerance as follows:

$$M = \left\lfloor (1 - fault\_tolerance) \cdot n + \frac{1}{2} \right\rfloor \tag{11}$$

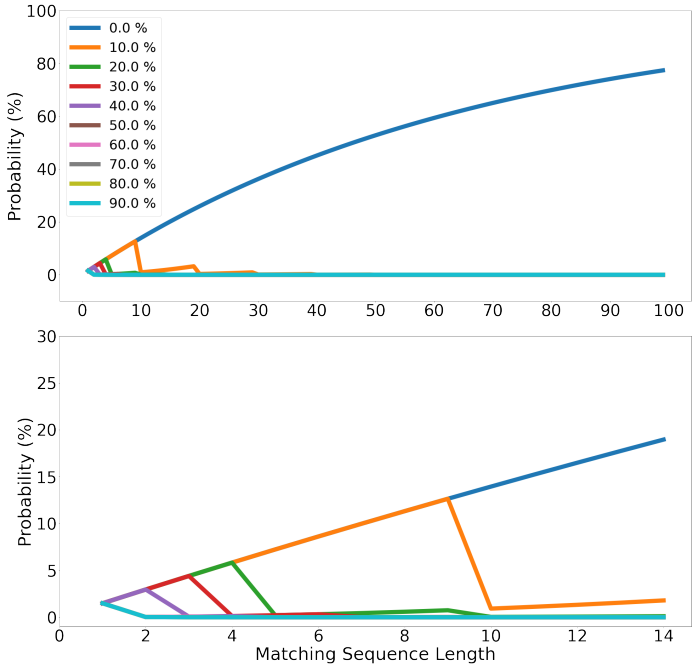

Figure 11: Probability of predicting a False Non-Travelling user (P_{NT_F}) over varying matching sequence lengths

Thus, the higher the fault tolerance is, the more occasional *zeros* are allowed in the matching sequence when the inference algorithm infers that the user is inside the vehicle. In consequence, by using a higher fault tolerance, we reduce the number of false non-travellers, albeit at the cost of more false travellers.

To analyze the influence, we calculated the likelihoods of false non-travellers and false travellers for different fault tolerances ranging from 0% to 90% in steps of 10% and for all matching sequence lengths from 1 to 100. Figure 11 depicts the probabilities for non-travelling users depending on the lengths of the matching sequences. The dark blue curve, showing no fault tolerance at all, rises towards 1 while all other trajectories converge towards 0. The interesting observation is how early the approximation towards 0 starts: Even with a fault tolerance of just 10%, the likelihood for producing non-travelling users is next to nothing for trips that produce more than 40 matchings. Thus, for trip lengths longer than nine minutes, this low fault tolerance would be sufficient.

The curves show that the selection of a good fault tolerance value is only relevant for short trips for which relatively short matching sequences are produced by DeepMatch2. To make the differences of the curves for short journeys more

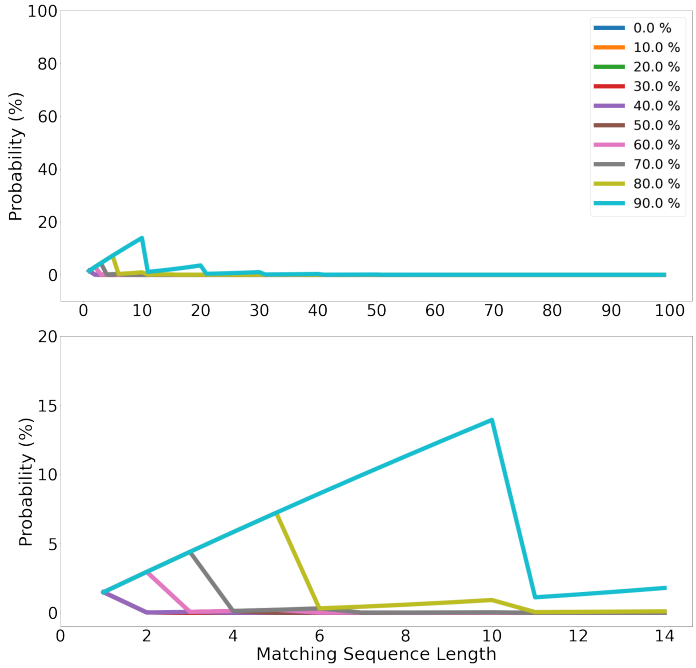

Figure 12: Probability of predicting a False Travelling user (P_{T_F}) over matching sequence lengths

legible, we show a zoomed-in version in the lower part of Fig. 11. In Sect. 6.1, we claimed that less than 2% of all time periods between adjacent bus stops is lower than 40 seconds. Since passengers can enter and leave the vehicle only at bus stops, we can therefore assume that for nearly every trip, at least three matchings are produced. From our curves, one can see that with a fault tolerance of 40%, it is enough to produce nearly no false non-travellers.

Figure 12 shows the same curves for false travelling users. Similarly to the false non-travelling users, the likelihood of creating false travelling users is negligible if a smartphone user is close to a vehicle for a longer time period. Thus, even with a fault tolerance of 90%, the inference is enough if the length of the matching sequence is at least 30, which corresponds to 6.4 minutes.

Yet, a non-travelling user will usually be close to a vehicle only for a relatively short time period, *e.g.*, less than a minute. Therefore, it is particularly important that the inference algorithm handles those cases correctly. The zoomed-in curves in the lower part of Fig. 12 show that the fault tolerance of 40%, that we already mentioned as sufficient for false non-travellers, is very close to zero for all matching sequences except for those consisting of just one matching.

Our findings about false non-travelling and false travelling users provide useful hints for the configuration of the inference algorithm. Since nearly no distances between two stops are less than 30 seconds (cf. Fig. 9), if our algorithm declares all matching sequences of lengths one and two as out-of-vehicle, it will hardly affect the overall accuracy of the algorithm. For all matching sequences of lengths three or larger, the policy may use a fault tolerance of 40%, which should lead to excellent results with very few false non-travellers and false travellers.

6.5 A More Formal Look at the Trade Off between NT_F and T_F

The discussion above to determine a good fault tolerance seems coherent, however, it is not very formal. A public transport authority that has resilient statistics about the travelling times of its passengers, can therefore go a step further and determine the trade offs between NT_F and T_F for different fault tolerance percentages more formally.

To be able to assess false travellers differently than the potentially less problematic false non-travellers, we use a weight factor $w \in [0, 1]$ that describes the weight that T_F shall have in comparison to NT_F. With $P^{ft}_{NT_F}(ft, msl)$ and $P^{ft}_{T_F}(ft, msl)$, we state the probabilities for NT_F resp. T_F for a certain fault tolerance ft and message sequence length msl that can be calculated using the formulas (9), (10), and (11). Moreover, p_l refers to the likelihood that the travelling time of a passenger has a certain length such that the corresponding matching sequence has l-many entries. Finally, Max_{msl} describes the maximum length of matching sequences that can occur in practice. These values can be computed from the operator's travelling time statistics. The *weighted average* Av between NT_F and T_F can be computed as follows:

$$Av(ft, w) = \sum_{l=1}^{Max_{msl}} p_l \cdot (w \cdot P^{ft}_{T_F}(ft, l) + (1 - w) \cdot P^{ft}_{NT_F}(ft, l)) \tag{12}$$

With formula (12), one can compare different fault tolerances and select the one ft that gives the lowest value $Av(ft, w)$ for the desired weight w.

Unfortunately, in spite of an in-depth search and requests to some public transportation authorities, we could not get access to meaningful statistics about usual travel durations of passengers. Thus, we based our calculations on the worst case scenario in which every passenger travels only between a stop and the adjacent one. We also assume that the number of people travelling are evenly distributed among the operator's buses. Then, we can use the distribution presented in Fig. 9 to calculate the likelihoods p_l for the lengths of the matching sequences.

The result of applying formula (12) to these values and the 10 different fault tolerant values 0% to 90% is listed in Tab. 7. The columns show the likelihoods for false non-travellers and false travellers as well as the weighted averages for the

Table 7: Weighted averages for all matching sequences with lengths between one and 21

Fault Tolerance	P_{NT_F}	P_{T_F}	$0.5 \cdot P_{NT_F} + 0.5 \cdot P_{T_F}$	$0.1 \cdot P_{NT_F} + 0.9 \cdot P_{T_F}$
0%	31.22957%	**0.15125%**	15.69041%	3.25908%
10%	8.49190%	0.15125%	4.32158%	0.98532%
20%	1.73878%	0.15126%	0.94502%	0.31001%
30%	0.95806%	0.15139%	0.55473%	0.23206%
40%	0.46660%	0.15828%	**0.31244%**	**0.18911%**
50%	0.15829%	0.46545%	**0.31187%**	0.43473%
60%	0.15796%	0.48792%	0.32294%	0.45492%
70%	0.15139%	0.96164%	0.55652%	0.88062%
80%	0.15125%	2.54341%	1.34733%	2.30419%
90%	**0.15125%**	10.11554%	5.13340%	9.11911%

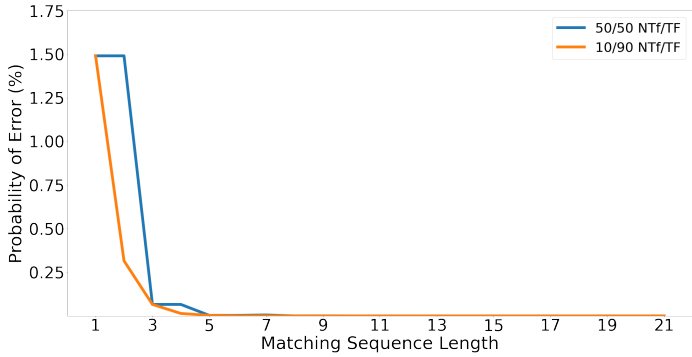

Figure 13: Total number of erroneous travel predictions over MS lengths using a Fault Tolerance of 40% and the average weight w set to 50% (blue) and 90% (orange)

weights $w = 0.5$ and $w = 0.9$. Non-surprisingly, the likelihood for NT_F is lower if a higher degree of fault tolerance is allowed, while it is the opposite for T_F. If we weight both error cases equally (*i.e.*, $w = 50\%$), fault tolerances of 40% and 50% render the best trade off results closely followed by 60%. When we consider false travellers as more important and set the weight to $w = 90\%$, the result is much clearer. Here, 40% is the winner followed by 30%. These results foster our assumption from Sect. 6.4 that 40% seems to be a efficient fault tolerance value.

In a final step, we checked the probability of errors for certain matching sequence lengths l, *i.e.*, $w \cdot P^{ft}_{T_F}(ft, l) + (1 - w) \cdot P^{ft}_{NT_F}(ft, l)$. The results for the

two weights and a fault tolerance of 40% are shown in Fig. 13. We see in both curves that the two shortest matching sequence lengths 1 and 2 are responsible for nearly all of the errors, while all the others give good results. This confirms the suggestion of our inference algorithm to bill only passengers for which at least three matchings were generated. The only losses are for passengers that travel only between two adjacent stops with less than 40 s distance. Assuming an average bus speed of 27 km/h between these stops, they are just 300 meters apart. In reality, people would walk this distance instead of waiting for a bus.

It would be nice to repeat our trade off calculations based on real travelling time statistics, and we will do so as soon as we get our hands on such data. However, we do not expect very different results since, in reality, the average travelling times will be much longer than in our thought experiment, and the graphs in Figs. 11 and 12 show that the inference is becoming better with longer matching sequences. Therefore, the likelihoods for the errors would probably be lower than in Tab. 7. However, based on the considerations from Sect. 6.4 and in this subsection, we still expect that 40% will be the winning fault tolerance value in most cases.

7 Conclusions and Future Work

To address the challenge of in-vehicle presence detection as an important aspect of mobile context analysis, we introduced our proposed deep learning-based approach, called DeepMatch and its improved version DeepMatch2. Our approach utilizes the sensor event streams of smartphones to predict their presence inside public transportation vehicles with an accuracy of 98.51% in the case of DeepMatch2. The deep learning model consists of custom made Stacked Convolutional Autoencoders for feature extraction and dimensionality reduction configured in a Siamese architecture, and a matching module consisting of several layers of stacked Convolutional Layers for event stream matching. The deep learning model is distributed among the smartphones carried by passengers, a reference device installed in public transport vehicles, and a central server. The Stacked Convolutional Autoencoders allow for compressing the sensor events through feature extraction and dimensionality reduction on the smartphone and the reference device, while the event matching is performed on the server. Through dimensionality reduction, the input data is reduced by the factor eight such that the bandwidth of the data transferred to the server is considerably reduced without losing the information of the data necessary to perform the matching.

Furthermore, we discussed how the deep learning-based approach can be used to create inference algorithms that deduce the travel time duration for passengers travelling in public transportation with a very high accuracy. In particular, we presented a theoretical framework that can be used to configure the inference algorithms to weight the various types of potential erroneous inferences, and thus accommodate the needs of the public transportation providers.

As our future plan, we intend to implement a pilot of DeepMatch2 together

with a public transportation provider in Norway. Moreover, we intend to research on the optimum length of the data segments and the frequency of data gathering (from the reference devices and the smartphones) in order to minimize the amount of data needed for in-vehicle presence detection.

References

[1] BankMyCell, "How Many Smartphones are in the World?" https://www.bankmycell.com/blog/how-many-phones-are-in-the-world, 2022, accessed: 2022-02-22.

[2] S. Kang, Y. Lee, C. Min, Y. Ju, T. Park, J. Lee, Y. Rhee, and J. Song, "Orchestrator: An Active Resource Orchestration Framework for Mobile Context Monitoring in Sensor-rich Mobile Environments," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*. Mannheim, Germany: IEEE Computer, 2010, pp. 135–144.

[3] W. Narzt, S. Mayerhofer, O. Weichselbaum, S. Haselböck, and N. Höfler, "Be-In/Be-Out with Bluetooth Low Energy: Implicit Ticketing for Public Transportation Systems," in *IEEE 18th International Conference on Intelligent Transportation Systems*. Las Palmas, Spain: IEEE, 2015, pp. 1551–1556.

[4] T. Gyger and O. Desjeux, "EasyRide: Active Transponders for a Fare Collection System," *IEEE Micro*, vol. 21, no. 6, pp. 36–42, 2001.

[5] C. Sarkar, J. J. Treurniet, S. Narayana, R. V. Prasad, and W. de Boer, "SEAT: Secure Energy-Efficient Automated Public Transport Ticketing System," *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 1, pp. 222–233, 2018.

[6] R. Meng, D. W. Grömling, R. R. Choudhury, and S. Nelakuditi, "RideSense: Towards Ticketless Transportation," in *2016 IEEE Vehicular Networking Conference (VNC)*. Columbus, OH, USA: IEEE, 2016, pp. 1–8.

[7] M. Won, A. Mishra, and S. H. Son, "HybridBaro: Mining Driving Routes Using Barometer Sensor of Smartphone," *IEEE Sensors Journal*, vol. 17, no. 19, pp. 6397–6408, 2017.

[8] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A Unified Deep Learning Framework for Time-series Mobile Sensing Data Processing," in *26th International Conference on World Wide Web*. Perth, Australia: ACM, 2017, pp. 351–360.

[9] M. Oplenskedal, A. Taherkordi, and P. Herrmann, "DeepMatch: Deep Matching for In-Vehicle Presence Detection in Transportation," in *Proceedings of the*

14th ACM International Conference on Distributed and Event-based Systems, 2020, pp. 97–108.

[10] 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, pp. 13–16.

[11] T. Gründel, H. Lorenz, and K. Ringat, "The ALLFA Ticket in Dresden. Practical Experience of Fare Management Based on Be-In/Be-Out & Automatic Fare Calculation," 2006, iPTS Conference, Seoul, South Korea.

[12] V. Kostakos, T. Camacho, and C. Mantero, "Wireless detection of end-to-end passenger trips on public transport buses," in *13th IEEE International Conference on Intelligent Transportation Systems*, 2010.

[13] A. Kwiecień, M. Maćkowski, M. Kojder, and M. Manczyk, "Reliability of Bluetooth Smart Technology for Indoor Localization System," in *International Conference on Computer Networks (CN)*, ser. CCIS 522. Br'unow, Poland: Springer-Verlag, 2015, pp. 444–454.

[14] S. Kuchimanchi, "Bluetooth low energy based ticketing systems," Master's thesis, Aalto University, Espoo, Finland, 2015.

[15] K. Sankaran, M. Zhu, X. F. Guo, A. L. Ananda, M. C. Chan, and L.-S. Peh, "Using Mobile Phone Barometer for Low-power Transportation Context Detection," in *12th ACM Conference on Embedded Network Sensor Systems*. Memphis, TN, USA: ACM, 2014, pp. 191–205.

[16] S. Vanini, F. Faraci, A. Ferrari, and S. Giordano, "Using Barometric Pressure Data to Recognize Vertical Displacement Activities on Smartphones," *Computer Communications*, vol. 87, pp. 37–48, 2016.

[17] B.-J. Ho, P. Martin, P. Swaminathan, and M. Srivastava, "From Pressure to Path: Barometer-based Vehicle Tracking," in *2nd ACM Inter. Conf. on Embedded Systems for Energy-Efficient Built Environments (BuildSys)*. Seoul, South Korea: ACM, 2015, pp. 65–74.

[18] A. Dimri, H. Singh, N. Aggarwal, B. Raman, D. Bansal, and K. K. Ramakrishnan, "RoadSphygmo: Using Barometer for Traffic Congestion Detection," in *8th International Conference on Communication Systems and Networks (COMSNETS)*. Bangalore, India: IEEE Computer, 2016, pp. 1–8.

[19] X. Wang, L. Kong, T. Wei, L. He, G. Chen, J. Wang, and C. Xu, "Vld: Smartphone-assisted vertical location detection for vehicles in urban environments," in *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2020.

[20] 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.

[21] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Exploiting Multi-Channels Deep Convolutional Neural Networks for Multivariate Time Series Classification," *Frontiers of Computer Science*, vol. 10, no. 1, pp. 96–112, Feb. 2016.

[22] P. Castrogiovanni, E. Fadda, G. Perboli, and A. Rizzo, "Smartphone data classification technique for detecting the usage of public or private transportation modes," *IEEE Access*, vol. 8, pp. 58 377–58 391, 2020.

[23] L. Wang, H. Gjoreski, M. Ciliberto, P. Lago, K. Murao, T. Okita, and D. Roggen, "Summary of the sussex-huawei locomotion-transportation recognition challenge 2020," in *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*, ser. UbiComp-ISWC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 351–358.

[24] C. Mayer, R. Mayer, and M. Abdo, "StreamLearner: Distributed Incremental Machine Learning on Event Streams: Grand Challenge," in *11th ACM International Conference on Distributed and Event-Based Systems*. Barcelona, Spain: ACM, 2017, pp. 298–303.

[25] M. Spörk, C. A. Boano, and K. Römer, "Performance and Trade-offs of the new PHY Modes of BLE 5," in *Proceedings of the ACM MobiHoc Workshop on Pervasive Systems in the IoT Era (PERSIST-IoT)*. ACM, 2019, pp. 7—12.

[26] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition," 2015, https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf.

[27] C. Zhang, W. Liu, H. Ma, and H. Fu, "Siamese Neural Network based Gait Recognition for Human Identification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Shanghai, China: IEEE, 2016, pp. 2832–2836.

[28] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature Verification using a "Siamese" Time Delay Neural Network," in *Advances in Neural Information Processing Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1994, pp. 737–744.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, ch. Autoencoders, pp. 505–528, http://www.deeplearningbook.org.

[30] 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.

[31] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked Convolutional Auto-encoders for Hierarchical Feature Extraction," in *International Conference on Artificial Neural Networks (ICANN)*, ser. LNCS 6791. Espoo, Finland: Springer-Verlag, 2011, pp. 52–59.

[32] A. Supratak, H. Dong, C. Wu, and Y. Guo, "DeepSleepNet: A Model for Automatic Sleep Stage Scoring based on Raw Single-channel EEG," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 11, pp. 1998–2008, 2017.

[33] Tensorflow, "Tensorflow 2.3," https://www.tensorflow.org/api_docs/python/tf, 2019, accessed: 2020-11-27.

[34] B. Historian, "Batterystats and Battery Historian," https://developer.android.com/studio/profile/battery-historian, 2019, accessed: 2019-10-23.

[35] X. Chen *et al.*, "Smartphone Energy Drain in the Wild: Analysis and Implications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1, pp. 151–164, 2015.

[36] Entur, "Entur public transport API," https://developer.entur.org, 2020, accessed: 2020-10-07.

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

Paper 4

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

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

Magnus Oplenskedal[1,3], Amir Taherkordi[1,2,3], Peter Herrmann[1]

[1]Norwegian University of Science and Technology (NTNU), Trondheim, Norway

[2]University of Oslo, Norway

[3]Forkbeard Technologies, Oslo, Norway

{magnukop, amirhost, herrmann}@ntnu.no

Abstract

Accurately detecting the mobile contexts of public transport vehicles and their passengers is a key requirement of intelligent context-aware services in such systems. A prominent example is *in-vehicle presence detection* which can be used to provide various services such as automated ticketing, dynamic vehicle distribution, and live route optimization. To use such services in practice, 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, we introduce Ataraxis in this paper—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%. This is higher than the accuracy of existing techniques for transport mode detection. We also made experiments on the battery consumption and CPU overhead. The results show that Ataraxis incurs a negligible computational overhead and power consumption on smartphones, even though we base our approach on sensor data collection and a deep learning model.

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

1 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, many providers are moving 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 main goal of this technological movement is to reduce infrastructure- and personnel-related costs. Another advantage is the improvement of the user experience and the removal of unnecessary steps for the passengers which make public transport more attractive.

The rapid development of mobile technologies such as 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 or outside a means of transportation, what type of vehicle it is, and more specifically, *in which* vehicle the person is travelling, we can provide various context-aware services such as route optimization, dynamic vehicle distribution, and live passenger flow analysis. With this type of information, user-centric services can be provided as well, *e.g.*, within the so-called Be-In/Be-Out (BIBO) systems [2]. They can automatically issue tickets based on the exact duration/distance traveled by users. Such systems alleviate the user from having in-depth knowledge regarding the fare rules 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], which provide a good accuracy of up to 98.51%. More details about these approaches and other BIBO technologies are provided in Sec. 2. An important disadvantage of these approaches, however, is that they require extra hardware, *e.g.*, reference devices and BLE transmitters as fixed equipment installed in busses, 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 spatiotemporal information, in particular, the real-time and historical locations of a certain vehicle. In Norway, the government-funded organization 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.

The core of 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 users based on the sensor event streams of their smartphones. From this, we can recognize whether the users are believed to be riding in a public transport vehicle. If they are assumed to be inside such a vehicle, we can then compare traces of their positions with those of the vehicles in their vicinity using the Entur API [5] or a similar service. This allows us to find out in which public transport vehicle users are effectively travelling. 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 detection 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. 2. In Sect. 3, we provide an in-depth presentation of our proposed approach followed by a report on experimental evaluation results in Sect. 4. In Sect. 5, we present related work before we conclude the paper with a discussion of future plans in Sect. 6.

2 DeepMatch and DeepMatch2

Existing solutions for in-vehicle presence detection can be separated into two different groups. The approaches of the first group 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 approaches of the other group rely on analyzing event streams produced by the sensors embedded in the passengers' smartphones. 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

our solutions, 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 those 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 our 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 this vehicle.

The deep learning models in DeepMatch and DeepMatch2 consist of several distributed modules. Apps running on the users' smartphones and the RefDev in the vehicle contain modules allowing for sensor data compression and feature extraction. These modules are the encoder part of a Stacked Convolutional Autoencoder. The most significant difference between both approaches is that the compression factor of DeepMatch2 is four times higher than the one used in DeepMatch.

The other part of each of the two models is a separate deep neural network that matches the data generated by the encoders in the RefDev and the passenger phone. It is supposed to run either on the RefDev or an external server. 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. Further, 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 swap their vehicles between areas handled by different PTAs. When moving a vehicle between PTAs, however, its operator has to exchange all devices working for just one of them which can be quite costly and laborious. That makes the operators reluctant to the use of additional hardware. Based on this insight, we investigated alternatives to the hardware-based solution proposed in DeepMatch and DeepMatch2.

3 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 the data used to train our deep learning models were collected

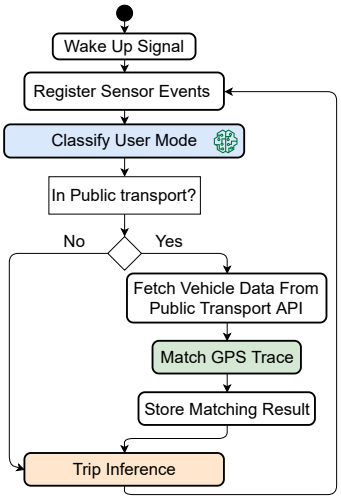

Figure 1: Outline of the ATARAXIS algorithm

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.

3.1 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 100 m. 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 m) 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.

3.2 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

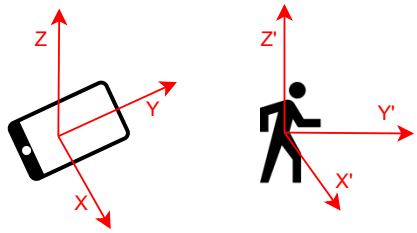

Figure 2: The orientation of a smartphone vs. that of the carrying user

provides the highest user mode detection 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 common, at least 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 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 being learned by the model itself, before the data is fed to the classifier [11]. The reorientation is calculated from the accelerometer data collected on the smartphone's coordinates and the output from the *gravity* sensor. That is a software-based sensor derived from the output of the accelerometer, 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 considering the orientation of the device in relation to its user. As pointed out in our empirical studies discussed in Sect. 4, 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.

Table 1: 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...

3.3 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.

3.3.1 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 1. 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.

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. To resolve this problem, we created a set of data analysis tools that are 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

Table 2: 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	...

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 1 are replaced by those in Table 2.

3.3.2 Dataset Creation

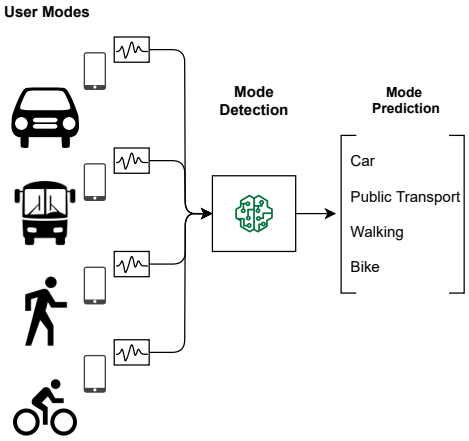

Figure 3: The different user modes ATARAXIS is capable of recognizing

The goal of this process 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.

Table 3: 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

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 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.

3.4 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 [12–15]. This is exactly what we need to solve our problem.

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 convolutional filter executes the three consecutive operations (1) convolution, (2) rectified linear unit activation (ReLu), and (3) batch normalization, see [16]. The inputs to the convolutional layers are two-dimensional. As pointed out by the example in Table 3, the rows of the matrix refer to sensor inputs, while the columns depict the points of time for which the sensor values were interpolated. The texts in the

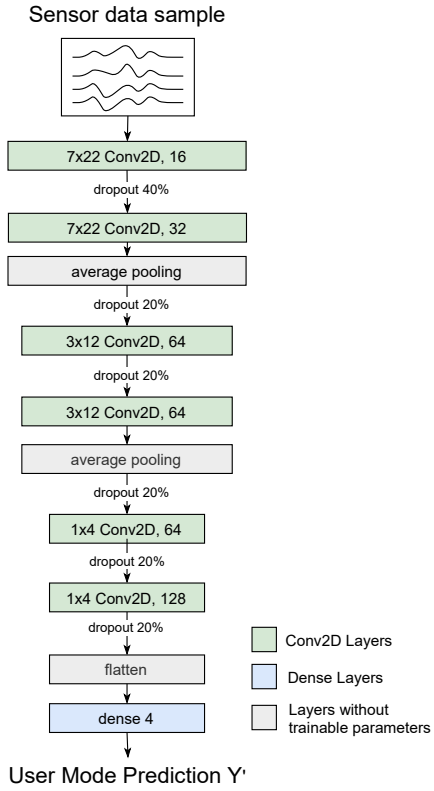

Figure 4: Ataraxis deep learning model architecture

green boxes describe the size of the convolutional filter matrices used in a layer. For instance, in the model's uppermost 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 uppermost 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. In this way, 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. Acting as the output of the model, this layer uses the softmax activation function such that a probability distribution over the four user modes is rendered.

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.

3.5 Model Training

In this subsection we describe the configurations and hyperparameters used when training our ATARAXIS deep learning models. As discussed 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 3). 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 into 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* [17] 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 the setting depicted in Table 4.

Table 4: ADAM optimizer settings

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

3.6 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. 3.3.2 and evaluated the results using the well-known performance metrics Precision, Recall, and F1-score which are described in greater detail in Sect. 4.1.

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 [18].

4 Evaluation

First, 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. Finally, the battery consumption as well as the CPU usage and run-time overhead of our approach for the passenger smartphones are discussed.

4.1 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 apply 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 and 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 and 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:

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

A fourth metric, that could have been used, is the accuracy which, however, is less sensitive to false positives and false negatives than the F1-score, see [19]. Since the avoidance of false positives and false negatives is a crucial feature of Ataraxis, we use it instead of the accuracy metric in our evaluations.

4.2 Data Collection and Dataset Creation

The datasets used to train the models in this work was collected by volunteers, each carrying a smartphone whilst performing one of the four activities using public transport, biking, riding in a car, or walking. The data was collected using our DataCollector application described in Sect. 3.3.1. 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 5, 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].

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

Table 5: 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 6: 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**

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. 4.2.1.

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. 4.2.2.

In Table 6, 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 [20]. In our experiments we set k to be 5, resulting in an distribution of 80% training samples and 20% testing samples of this algorithm.

4.2.1 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

[2]The datasets are available to interested readers on request.

Table 7: 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

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 7, 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 98.69%.

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. 3.2. 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.

4.2.2 Sample Size Experiments

The length of the input data used by the deep learning model in Ataraxis is important for a couple of reasons. Firstly, the size of the input of the model

Table 8: 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**

influences its performance, *i.e.*, the more data a model can base its predictions on, the better the chance it that it will succeed in making good predictions. 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 made for this purpose, 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. 3.1, 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 8, 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 10 Hz, 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 an F1-score of 98.69%. Nevertheless, the model trained on 6.4 second long samples performed nearly as well with an F1-score of 98.44%.

4.3 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

Table 9: 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

practice. In particular, we measure 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 9. 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, that our test environment simulates a typical public transport vehicle, we performed all tests indoors at an temperature of about 19°C. 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 [21]. 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 depicted in Table 9. 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 just 25 mAH. This equals to only 0.96 % of the total battery capacity. As a result from our tests, we consider that our approach has a negligible impact on the overall battery power consumption.

Table 10: 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 %

4.4 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 practice. To learn about this 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 10. We 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 32 ms for the P30 Pro. Furthermore, since the machine learning model is only executed every 12.8 s, the machine learning model should hardly impact any other applications or services that are executed on the phone in parallel.

5 Related Work

In Sec. 2, we briefly discussed our 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.

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 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 [22].

From another perspective, existing techniques can be arranged in two cate-

gories, *i.e.*, location-based [23] and sensor-based [24] approaches. Location-based solutions often rely on location data provided by the GPS or wireless network [25]. 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 [26]. 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 [27, 28].

Most works in this category rely on GPS data [29], while others combine GPS with the use of a Geographic Information System (GIS) platform or the map service APIs [30]. In another group of approaches, GPS, accelerometer and Bluetooth are combined with map-matching algorithms, see [31–33]. Finally, some approaches utilize the fusion of GPS and accelerometer data, see [32, 33].

Sensor-based transportation mode detection techniques can be used in a more energy-efficient and reliable manner than the location-based approaches [34]. 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 [35]. As described in [36], 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) [37].

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 [38], 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 [39]. 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. [40] 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 power 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 6 and 7. In addition, our approach uses relatively short sample sizes as depicted in Table 8. This allows for conducting a greater number of user mode classifications during a tour which makes the trip inference more precise, see [4]. Moreover, it induces a hardly noticeable power consumption as shown in Table 9 and a low computational overhead as presented in Table 10.

6 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 which 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 power 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 usage of these vehicles has recently exploded in many countries. 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. Kang, Y. Lee, C. Min, Y. Ju, T. Park, J. Lee, Y. Rhee, and J. Song, "Orchestrator: An Active Resource Orchestration Framework for Mobile Context Monitoring in Sensor-rich Mobile Environments," in *IEEE International Confonference on Pervasive Computing and Communications (PerCom)*. Mannheim, Germany: IEEE Computer, 2010.

[2] W. Narzt, S. Mayerhofer, O. Weichselbaum, S. Haselböck, and N. Höfler, "Be-In/Be-Out with Bluetooth Low Energy: Implicit Ticketing for Public Transportation Systems," in *IEEE 18th International Conference 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 *14th ACM International Conference 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, J. J. Treurniet, S. Narayana, R. V. Prasad, and W. de Boer, "SEAT: Secure Energy-Efficient Automated Public Transport Ticketing System," *IEEE Transactions 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] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

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

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

[20] M. Stone, "Cross-Validatory Choice and Assessment of Statistical Predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, 1974.

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

[22] A. Efthymiou, E. N. Barmpounakis, D. Efthymiou, and E. I. Vlahogianni, "Transportation mode detection from low-power smartphone sensors using tree-based ensembles," *Journal of Big Data Analytics in Transportation*, vol. 1, no. 1, 2019.

[23] T. Sohn, A. Varshavsky, A. LaMarca, M. Y. Chen, T. Choudhury, I. Smith, S. Consolvo, J. Hightower, W. G. Griswold, and E. de Lara, "Mobility Detection using Everyday GSM Traces," in *International Conference on Ubiquitous Computing*. Springer, 2006.

[24] 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.

[25] 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.

[26] Z. A. Lari and A. Golroo, "Automated Transportation Mode Detection using Smart Phone Applications via Machine Learning: Case Study Mega City of Tehran," in *Transportation Research Board 94th Annual Meeting*, Washington, DC, USA, 2015.

[27] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding Mobility based on GPS Data," in *10th International Conference on Ubiquitous Computing*, 2008.

[28] T. Feng and H. J. P. Timmermans, "Comparison of Advanced Imputation Algorithms for Detection of Transporation Mode and Activity Episode using GPS Data," *Transportation Planning and Technology*, vol. 39, no. 2, 2016.

[29] 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.

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

[31] J. Chen and M. Bierlaire, "Probabilistic Multimodal Map Matching with Rich Smartphone Data," *Journal of Intelligent Transportation Systems*, vol. 19, no. 2, 2015.

[32] B. D. Martin, V. Addona, J. Wolfson, G. Adomavicius, and Y. Fan, "Methods for Real-Time Prediction of the Mode of Travel using Smartphone-based GPS and Accelerometer Data," *Sensors*, vol. 17, no. 9, 2017.

[33] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using Mobile Phones to Determine Transportation Modes," *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, no. 2, 2010.

[34] S. Hemminki, P. Nurmi, and S. Tarkoma, "Accelerometer-based Transportation Mode Detection on Smartphones," in *11th ACM Conference on Embedded Networked Sensor Systems*, 2013.

[35] 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.

[36] M. Nikolic and M. Bierlaire, "Review of Transportation Mode Detection Approaches based on Smartphone Data," in *17th Swiss Transport Research Conference*, 2017.

[37] G. Yanyun, Z. Fang, C. Shaomeng, and L. Haiyong, "A Convolutional Neural Networks based Transportation Mode Identification Algorithm," in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2017.

[38] X. Liang and G. Wang, "A Convolutional Neural Network for Transportation Mode Detection based on Smartphone Platform," in *IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2017.

[39] T. H. Vu, L. Dung, and J.-C. Wang, "Transportation Mode Detection on Mobile Devices using Recurrent Nets," in *24th ACM International Conference on Multimedia*, 2016.

[40] G. Asci and M. A. Guvensan, "A Novel Input Set for LSTM-based Transport Mode Detection," in *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019.

ISBN 978-82-326-5987-6 (printed ver.)
ISBN 978-82-326-6899-1 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU

Norwegian University of
Science and Technology