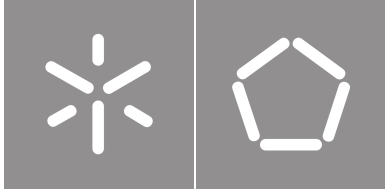
Universidade do Minho
Escola de Engenharia

Pedro Miguel Ferreira Guimarães

A Neurodynamics-Based System for Online Learning and Prediction of Traveling Routines in Intelligent Vehicles

Novembro de 2025

A Neurodynamics-Based System for Online Learning and Prediction of Traveling Routines in Intelligent Vehicles
Pedro M. F. Guimarães
UMinho | 2025



Universidade do Minho
Escola de Engenharia

Pedro Miguel Ferreira Guimarães

A Neurodynamics-Based System for Online Learning and Prediction of Traveling Routines in Intelligent Vehicles

Tese de Doutoramento
Programa Doutoral em Sistemas Avançados de Engenharia para a Indústria

Trabalho efetuado sob a orientação de:
Professor Doutor Sérgio Monteiro
Professora Doutora Flora Ferreira

Novembro de 2025

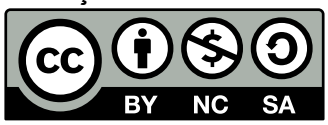
COPYRIGHT AND TERMS OF USE OF THIS WORK BY A THIRD PARTY

This is academic work that can be used by third parties as long as internationally accepted rules and good practices regarding copyright and related rights are respected.

Accordingly, this work may be used under the license provided below.

If the user needs permission to make use of the work under conditions not provided for in the indicated licensing, they should contact the author through the RepositoriUM of Universidade do Minho.

Licença concedida aos utilizadores deste trabalho



Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

CC BY-NC-SA 4.0

https://creativecommons.org/licenses/by-nc-sa/4.0/deed.en

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.19) .

Acknowledgments

First and foremost, I would like to thank my scientific advisors, Sérgio Monteiro and Flora Ferreira. I am truly grateful for the opportunity to work under your guidance, for your kindness, for believing in me, and for your invaluable support, expertise, clarity, and encouragement throughout this journey. I also wish to express my sincere gratitude to Estela Bicho for her warm, uplifting spirit and dedication. And an enormous thank you to Wolfram Erlhagen, for his endless patience and for always being there for me.

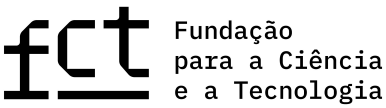
I also want to extend my gratitude to my colleagues from MARLab (University of Minho) for their collaboration, support, and camaraderie throughout this journey. A big thanks to Paulo Barbosa, Luís Louro, Carlos Fernandes, and Ankit Patel, and a special thanks to Weronika Wojtak and Paulo Vicente for your constant encouragement and insightful discussions. A warm acknowledgment to Dona Helena, who, since my master's, always took the time to ask how things were going and offered kind words of support while tidying up our lab at the University. I am also grateful to Emanuel Sousa and the HTIR group at CCG for their support during the final stretch of this journey.

I am grateful to Bosch Car Multimedia Portugal for their institutional support and for enabling the development of this PhD as part of the collaborative initiative between Bosch and the University of Minho. A dedicated thanks goes to Ana Carolina Silva, and a very special thanks to Carla Durães for her constant availability, and for always helping to navigate the organizational side of things with clarity and care.

I would also like to thank my longtime friend and old university colleague, David "Big Foot", whose occasional but timely perspective helped me move forward in key moments. To my fellow AESI PhD colleagues, a warm thank you for the shared struggles, encouragement, and study sessions. I'm especially grateful to my colleagues and friends Manuel Alves, Armindo Lobo, and Valdemar Leiras, who made this PhD journey lighter, more fun, and filled with shared motivation. And to my bro, Eduardo Mendes. We've shared (and still share) far more than just research advice: meaningful conversations about our lives, challenges, guitar sessions, and good times. Thank you for your friendship.

To my parents, Bete and Joca, and my closest person, Inês "Kate" Caetano. The heart of it all, my home. *Mãe, Pai, vocês têm sido incondicionais desde que me lembro. O meu alicerce. A nossa proximidade, construída com abertura e confiança, tem sido, desde sempre, uma fonte de força para mim. Estiveram (e estão) ao meu lado no silêncio e na conversa, na alegria e na frustração, sempre dispostos a carregar parte do meu peso como se fosse vosso.* And to my dear Kate, for your love, patience, sacrifice, encouragement, dedication, and inspiration, both as a person and as a professional. Your "scars" and your wise advice have protected and guided me more times than I can count. I'm also grateful for the silliness, the laughter, and the way you make life fun and richer. The future is bright...
To you three, this journey was not mine alone — it was ours. Thank you.

The work received financial support from FCT and Bosch Car Multimedia Portugal through the PhD scholarship PD/BDE/150498/2019.

















STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

Assinado por: **Pedro Miguel Ferreira Guimarães**
Num. de Identificação: 14578362
Data: 2025.11.27 00:42:42 +0000

(Pedro Miguel Ferreira Guimarães)

”

“Come up and feel the sun
A new morning has begun
Another day will make it clear
Why your stars should guide us here...”

— Mark Knopfler, *”Sailing to Philadelphia”*

Resumo

Sistema Baseado em Neurodinâmica para Aprendizagem Online e Previsão das Rotinas de Viagem em Veículos Inteligentes

A indústria automóvel tem prestado crescente atenção aos assistentes inteligentes integrados nos veículos, capazes de aprender e adaptar-se aos seus utilizadores. A previsão de destinos é uma das funções que tem sido amplamente explorada, aproveitando-se da regularidade espaço-temporal nos padrões de mobilidade humana. No entanto, prever informações sobre viagens além dos destinos permite formas de assistência mais abrangentes e impactantes, mas também implica lidar com a complexidade de prever aspetos espaciais e temporais. As abordagens existentes baseiam-se predominantemente em aprendizagem *offline*, exigindo conjuntos extensos de dados, e tipicamente inaptas para cenários realistas onde os dados de treino são incrementais ou os padrões de viagem evoluem. Esta tese propõe uma abordagem baseada em Campos Dinâmicos Neuronais para a aprendizagem e previsão dinâmica de informações sobre viagens. Esta abordagem suporta aprendizagem contínua e *online*, permitindo a adaptação a hábitos de viagem em evolução. Para além de determinar onde e quando ocorrem partidas e chegadas, identifica condutores, passageiros e objetos envolvidos. Assim, possibilita lembretes oportunos, configurações de conforto personalizadas, e sugestões proativas como avisos sobre objetos esquecidos. Isto contrasta com soluções que se concentram exclusivamente na aprendizagem dos destinos habituais do veículo ou fornecem sugestões apenas após a seleção de um perfil de condutor. A abordagem proposta foi implementada num sistema baseado em ROS (*Robot Operating System*), concebido para suportar operação em tempo real e simulação. Esta capacidade permite testes e validação controlados da abordagem, com rotinas pré-gravadas, garantindo confiança no desempenho para testes em contexto real. Avaliações realizadas em rotinas reais de 11 semanas, recolhidas através de uma aplicação móvel desenvolvida, demonstram a sua eficácia. A simulação e o desempenho em tempo real são otimizados com aceleração por *hardware* na NVIDIA Jetson Nano. A escalabilidade e modularidade do sistema oferecem uma base para diversas aplicações assistivas e progressão a níveis de maturidade tecnológica mais elevados.

Palavras-chave: aprendizagem adaptativa espaço-temporal, campos dinâmicos neuronais, previsão de informações sobre viagens, *robot operating system*, unidades de processamento gráfico

Abstract

A Neurodynamics-Based System for Online Learning and Prediction of Traveling Routines in Intelligent Vehicles

The automotive industry is witnessing growing attention to intelligent in-vehicle assistants capable of learning and adapting to user behaviors. Among these capabilities, destination prediction has been a widely explored feature, leveraging the spatio-temporal regularity observed in human mobility patterns. Predicting trip information beyond just destinations, however, enables more comprehensive and impactful forms of assistance, but it requires addressing the complexity of predicting both spatial and temporal aspects of mobility. Existing approaches predominantly rely on offline learning from stationary data distributions, requiring extensive datasets. Many also do not fully address realistic scenarios where training data is incrementally available, or travel patterns may change over time. This thesis proposes an approach based on Dynamic Neural Fields for dynamically learning and predicting trip information. It supports online and continual learning to adapt to evolving travel habits. Beyond determining where and when departures and arrivals occur, it identifies the specific drivers, passengers, and objects involved. It thus enables functionalities like timely reminders, personalized comfort settings prepared in advance, and proactive suggestions such as notifying about forgotten objects. This contrasts with some commercially available systems, which focus solely on learning the vehicle's habitual destinations or provide personalized suggestions only after a driver profile is selected. The proposed approach was implemented within a modular system based on ROS (Robot Operating System), designed for both real-time operation and simulation. This dual-mode capability enables controlled testing and validation of the approach with pre-recorded routines, ensuring confidence in its theoretical performance for real-time deployment. Evaluations on 11-week datasets of real traveling routines, collected via a tailored mobile application, demonstrate its efficacy. Simulation speed and real-time performance are optimized by leveraging hardware acceleration on the NVIDIA Jetson Nano. The system's scalability and modularity provide a foundation for diverse assistive applications and progression to higher Technology Readiness Levels.

Keywords: adaptive spatio-temporal learning, dynamic neural fields, graphics processing units, robot operating system, trip information prediction

Contents

List of Figures **xii**

List of Tables **xiv**

List of Listings **xv**

Acronyms **xvi**

1 Introduction **1**

1.1 Motivation 2

1.2 Objectives 3

1.3 State of the art 4

1.3.1 Mobility prediction in transportation 4

1.3.2 Processing units in automotive assistance systems 10

1.4 Main contributions 13

1.5 Thesis outline 15

2 Fundamentals **16**

2.1 Dynamic Neural Fields 17

2.1.1 Overview 17

2.1.2 Foundations of Dynamic Neural Fields 18

2.1.3 Example of a Dynamic Neural Field model architecture 27

2.2 GPU acceleration 31

2.2.1 Overview 31

2.2.2 NVIDIA Jetson Nano board 31

2.2.3 CUDA programming 33

2.3 Robot Operating System 2 . . . 37
2.3.1 Overview . . . 37
2.3.2 Architecture . . . 39
2.3.3 Core concepts . . . 40
2.3.4 Algorithms and tools . . . 42
2.4 Flutter . . . 44
2.4.1 Overview . . . 44
2.4.2 Architecture . . . 45
2.4.3 Building user interfaces . . . 46
2.4.4 Integration with Google Firebase . . . 47

3 *A data recording mobile application to create datasets of vehicle users' routines* 48
3.1 Introduction . . . 49
3.2 P29DR application . . . 51
3.2.1 Methods . . . 52
3.2.2 Design and user flow . . . 52
3.2.3 Recording data description . . . 56
3.3 Usage principle . . . 57
3.4 Conclusion . . . 58

4 *A Dynamic Neural Field approach for intelligent cockpits: online learning and prediction of traveling routines* 60
4.1 Introduction . . . 61
4.1.1 Motivation . . . 61
4.1.2 Contribution . . . 63
4.2 The Dynamic Neural Field framework . . . 64
4.3 DNF approach to learn and predict traveling routines . . . 66
4.3.1 Preliminaries . . . 66
4.3.2 Overview . . . 68
4.3.3 Details of the DNF cognitive model . . . 69
4.4 Experimental results . . . 75
4.4.1 Datasets . . . 75
4.4.2 Simulation process . . . 76
4.4.3 Specifications of the modules . . . 76
4.4.4 Performance . . . 77
4.5 Discussion . . . 80
4.6 Conclusion . . . 83

5 A neurodynamics-based system for traveling routine assistance in intelligent vehicle cockpits **84**
5.1 Introduction . . . 85
5.2 ROS-based system architecture . . . 85
5.2.1 Overview . . . 85
5.2.2 Real-Time mode . . . 87
5.2.3 Simulation mode . . . 90
5.3 System implementation details . . . 92
5.3.1 Engine Node implementation . . . 92
5.3.2 System interfaces . . . 99
5.3.3 System launching and parameters . . . 101
5.3.4 Filesystem organization . . . 104
5.4 Performance insights . . . 105
5.5 Conclusion . . . 110

6 Conclusion **112**
6.1 Thesis summary and highlights . . . 113
6.2 Limitations and future work . . . 117

Appendices

A Mathematical formulation of the approach **121**
A.1 Clustering model . . . 121
A.2 Cognitive models . . . 121

B Simulation parameters **127**
B.1 Clustering model . . . 127
B.2 Cognitive models . . . 127

Bibliography **130**

List of Figures

1 The fundamental building block of DNF theory: the dynamic neural field. 18
2 Activation dynamics in a DNF. 19
3 Example of a two-dimensional dynamic neural field tuned to color and size features. . . 20
4 Types of interaction kernels defining interaction strength based on the distance between two neurons. 22
5 Detection mechanism in a DNF with a Gaussian interaction kernel. 23
6 Memory mechanism in a DNF with a Gaussian interaction kernel. 24
7 Memory trace mechanism. 25
8 Multi-item memory mechanism in a DNF with an Oscillatory interaction kernel. 25
9 Encoding of the order and timing of perceived events in a DNF with an Oscillatory interaction kernel. 26
10 Decision-making mechanism in a DNF with a Gaussian interaction kernel. 26
11 Basic DNF model architecture illustrating inter-field interactions. 27
12 Autonomous agent endowed with cognitive and behavioral capabilities to learn and play a musical piece. 28
13 Architecture of the DNF model for learning a musical piece. 29
14 NVIDIA Jetson Nano Developer Kit. 32
15 Simplified architectural differences between a CPU (left) and a GPU (right). 33
16 The CUDA hierarchy, including the memory (left) and thread hierarchy (center), and the GPU mapping (right). 34
17 High-level ROS 2 software architecture. 40
18 Example of a simplified system architecture for a patrol robot. 41
19 UI examples developed with the Flutter toolkit. 44
20 Anatomy of a Flutter application. 46
21 A driver using the data recording mobile application. 51

22 Interaction flow of (A) the Main page (nonrecording mode), (B) the New Passenger page, and (C) the History page. 53
23 Interaction flow of (A) the Main page (recording mode), and (B) the Car Settings page. . 54
24 Interaction flow of the Users page. 55

25 A common weekday traveling routine of a private vehicle, including drivers, passengers and objects status, which can be learned and recalled by an intelligent driver assistant. . . . 62
26 Bi-stable field dynamics example. 65
27 Different neural field representations. 66
28 Schematic overview of the proposed DNF-based approach for learning daily traveling routines of a shared vehicle. 67
29 Clustering module. 69
30 Architecture of the cognitive model which is used independently for departure and for arrival events. 70
31 Example of a multi-peak pattern in the memory field at the end of a day, representing arrivals. 71
32 Time window implemented by the recall dynamics in the Anticipation and Tolerance Recall fields, u_{ar} and u_{tr}. 72
33 Snapshots of the activity patterns in the various fields of the vehicle cognitive model for the example of departure events on four consecutive Mondays. 74
34 Predicted and real arrival events. 79

35 The main ROS nodes of the system and their roles. 86
36 Real-Time mode system architecture. 88
37 Simulation mode system architecture. 91
38 Class diagram of the $\mathrm{Engine\ Node}$. 93
39 Sequence diagram outlining the workflow within the $\mathrm{Engine\ Node}$ when a day begins and ends. 95
40 Sequence diagram illustrating the workflow within the $\mathrm{Engine\ Node}$ for processing traveling events and the periodic model updates. 96
41 Sequence diagram illustrating the periodic readout process of the cognitive models. . . 97
42 Overview of the $\mathrm{routines_adas_system}$ ROS package directory structure. 104
43 Detailed directory structure of the system subfolder within the $\mathrm{routines_adas_system}$ ROS package. 105
44 Average execution time per time step versus the number of cognitive models for different field sizes, leveraging CUDA-based GPU acceleration. 106
45 GPU vs. CPU execution time comparison across varying configurations. 109

List of Tables

1 Trajectory collection. 56
2 Occupants collection. 56
3 Objects collection. 57
4 Car settings collection. 57

5 Sample of a Friday traveling routine from the datasets. 75
6 Results for (P1). 78
7 Results for (P2). 80

8 Summary of the ROS interfaces. 100
9 Average execution time (in milliseconds) per single update (time) step, along with the standard deviation, for different configurations of the number of cognitive models and stop locations. 107

10 Table of variables of the cognitive model. 122

11 Parameters and values for the fields of the clustering model. 127
12 Parameters and values for the Spatial Encoding field of the cognitive model. 128
13 Parameters and values for the Spatio-Temporal Encoding field of the cognitive model. . . 128
14 Parameters and values for the Consolidation field of the cognitive model. 128
15 Parameters and values for the Memory field of the cognitive model. 129
16 Parameters and values for the Recall fields of the cognitive model. 129
17 Parameters and values for the Working Memory field of the cognitive model. 129

List of Listings

2.1 CUDA C++ program, *simple_cuda_example.cu*. 36
5.1 Prototypes of CUDA kernels and auxiliary CUDA functions for neural field updates and interactions. 99
5.2 The TravelingEvent.msg message type. 101
5.3 The GetTravelingRoutine.srv service type. 101

Acronyms

ADAS Advanced Driver Assistance Systems
AI Artificial Intelligence
API Application Programming Interface
ASIC Application-Specific Integrated Circuit

BSP Board Support Packages

CAN Controller Area Network
CNN Convolutional Neural Network
CPU Central Processing Unit
CUDA Compute Unified Device Architecture

DDS Data Distribution Service
DFT Dynamic Field Theory
DL Deep Learning
DNF Dynamic Neural Field

ECU Electronic Control Unit
EMA Exponential Moving Average

FFT Fast Fourier Transform
FPGA Field-Programmable Gate Array

GDPR General Data Protection Regulation
GPS Global Positioning System

GPU Graphics Processing Unit
GUI Graphical User Interface

HMM Hidden Markov Model

IFFT Inverse Fast Fourier Transform
ITS Intelligent Transportation Systems
IVIS In-Vehicle Information Systems

LSTM Long Short-Term Memory

MCU Microcontroller Unit
ML Machine Learning
MM Markov Model

OS Operating System

POI Points of Interest

ROS Robot Operating System
RTOS Real-Time Operating System

SLAM Simultaneous Localization and Mapping
SM Streaming Multiprocessor
SoC System-on-Chip

TRL Technology Readiness Level

UI User Interface
URDF Unified Robot Description Format

Chapter 1

Introduction

1.1 Motivation

In recent years, the automotive industry has witnessed remarkable advancements in intelligent transportation and vehicle technologies [1]. These innovations have been largely driven by the escalating demand for solutions that ensure safety and enhance the driving experience [1–3]. At the forefront of this technological evolution is the development of Advanced Driver Assistance Systems (ADAS), which are crucial for reducing accidents, optimizing driving performance, and improving user comfort [4]. This includes, among many, lane keep assist, adaptive cruise control, collision avoidance, parking assistance, and in-vehicle assistants. The proliferation of ADAS is anticipated to continue its upward trajectory, driven by safety regulations and growing consumer interest in applications that enhance both comfort and efficiency [5, 6].

Traditionally, ADAS have been designed based on generic principles, intended to function uniformly for all users. However, there is an increasing awareness of the need for personalized solutions adaptable to different drivers [2, 7], e.g., personalized adaptive cruise control and personalized driver monitoring systems. This paradigm shift underlines the importance of considering user preferences and characteristics to increase user acceptance and trust. Since personalization in real-life applications cannot be viewed as a static process, it is imperative that assistance systems are capable of perceiving, reasoning, and evaluating data to continually adapt and improve through ongoing driver interactions [8–10].

In this context, intelligent in-vehicle information systems (or intelligent in-vehicle assistants) are an emerging and increasingly recognized category within personalized ADAS [2, 11]. Recommendation systems is a particular example. By learning users' preferences, those systems can provide precise suggestions and actions, enhancing the driving experience. The renowned automotive manufacturer Land Rover has proposed the concept of a "self-learning car", which leverages Artificial Intelligence (AI) to learn user habits and adjust settings like climate control, music preferences, and seat configuration accordingly [11]. Mercedes' MBUX Hyperscreen uses AI to offer context-aware suggestions based on user behavior, location, and time. For instance, if a user regularly calls a particular contact during their commute, the system will intelligently prompt them to make the call at the usual time [12]. Moreover, by identifying which driver is inside, it can also recommend the next destination. Another example is the Destination Prediction feature in TOMTOM navigation devices [13]. The system learns daily driving routes and predicts frequently visited locations based on the time of day, asking, for example, *Going to work?* in the morning. However, it is important to note that this feature typically learns from the vehicle's patterns rather than the behaviors of individual users.

A recent survey by [14] highlighted drivers' desire and acceptance of vehicles that can learn their habits. This capability could allow a vehicle to anticipate the departure and arrival of the next trip and, for instance, timely inform a specific driver about the predicted departure time and alert them if objects usually transported in the vehicle are being forgotten. In addition, based on the predicted departure time, the system could enhance comfort and convenience in advance by adjusting the interior temperature as

well as the position of mirrors and seats for the next driver. In fact, the application of such personalized systems may extend beyond private vehicles to ride-hailing services like Waymo [15] and community-shared vehicles. By learning the daily traveling routines of individual users, these systems could enable vehicles to anticipate pick-up times, improving overall efficiency, user experience, and fleet management.

To enable this level of assistive intelligence, vehicular systems must be capable of capturing both the spatial and temporal characteristics of travel patterns, including those of drivers, passengers, and objects. Indeed, numerous studies suggest that human mobility exhibits a high degree of spatio-temporal regularity [16–19], with individuals often visiting the same places at similar times on specific days. For instance, the weekday trajectories of a typical family car may involve dropping the kids off at school, going to work, picking the kids up from school, and returning home. Specific days of the week may include additional stops for scheduled gym classes or music lessons, as well as regular trips to the grocery store or the park. Nevertheless, it is important to recognize that travel patterns are dynamic and can evolve over time due to factors such as lifestyle changes, new commitments, or unforeseen events.

This dynamic nature of real-world routines highlights the limitations of many current mobility prediction models, which typically rely on large, stationary datasets and offline training. Most do not support continual learning or adaptation over time, nor do they handle fine-grained temporal prediction or multi-entity context (e.g., drivers, passengers, and carried objects). These gaps motivate the present work, which aims to explore a cognitively inspired approach capable of capturing and predicting traveling routines in a flexible, adaptive, and interpretable way.

1.2 Objectives

The concept of intelligent in-vehicle assistants that can learn about their users to provide personalized suggestions or take anticipatory actions is becoming increasingly significant in the automotive industry [3, 11]. While existing systems like Mercedes' MBUX Hyperscreen [12] and TOMTOM's Destination Prediction [13] have advanced in offering predictive capabilities, they remain bound by certain constraints. For instance, Mercedes offers suggestions only when a specific driver is already identified, and TOMTOM centers its predictions on the vehicle's traveling habits, rather than learning individualized behaviors.

From a scientific perspective, the literature on human mobility has extensively addressed next destination prediction, but relatively little attention has been given to the broader and more complex challenge of trip information prediction, that is, predicting when and where a person will depart or arrive [20]. Moreover, most current models rely on large, stationary datasets and offline learning processes that do not reflect real-world conditions, where data availability is incremental and mobility patterns are dynamic. Additionally, the common use of arbitrary temporal discretization poorly captures the continuous nature of time in human routines.

This thesis aims to address these limitations by investigating a neurodynamics-based approach for intelligent vehicle assistance. The following research objectives guide the work:

1. To investigate and design a computational model based on Dynamic Neural Fields (DNFs) capable of online, continual learning of traveling routines, capturing both spatial and temporal features.
2. To explore a strategy for modeling traveling routines involving multiple entities, including drivers, passengers, and objects, in a dynamic and evolving context.
3. To develop a modular, real-time capable system architecture, integrating the proposed approach within an embedded platform suited for deployment in intelligent vehicles.
4. To evaluate the neurodynamics-based approach using real-world traveling routine datasets, assessing its predictive performance, adaptability to changing patterns, and potential for assistive applications.

This thesis was developed in the context of a collaborative research initiative between the University of Minho and Bosch Car Multimedia S.A. Portugal, and as part of the joint project "Easy Ride: Experience is Everything". The work addressed an innovative research topic aligned with Bosch's strategic interest in developing intelligent systems capable of understanding and anticipating user routines to enhance the in-vehicle experience.

1.3 State of the art

This section explores the current state of the art in two domains that form the groundwork of the research presented in this thesis. First, the review focuses on mobility prediction in transportation, giving particular attention to the strategies, methodologies, and challenges on predicting trip information. Then, attention is turned to the processing units in automotive assistance systems, evaluating the hardware typically adopted to handle the computational demands of various automotive systems.

1.3.1 Mobility prediction in transportation

In the context of Intelligent Transportation Systems (ITS), the study of human mobility prediction has become increasingly pertinent [16, 18]. Mobility prediction can be generally defined as the study of predicting future locations and movements of people and other objects (e.g., vehicles), by the application of computational models that analyze historical and real-time mobility data, and identify spatial and temporal patterns and regularities. But, to what degree is the mobility of an individual predictable? Research indicates that human mobility exhibits significant spatio-temporal regularity and predictability. Studies by [16] and [17] have shown that individuals tend to follow reproducible travel patterns, often returning to a few frequented locations. Additionally, [18] found that urban vehicular mobility is highly predictable, with regular daily patterns.

As urban populations expand, the resulting strain on transportation infrastructures increases, demanding more sophisticated approaches to urban planning and resource management [18, 21, 22]. This need is further underscored by projections indicating that over 68% of the world's population will reside in urban areas by 2050 [23]. Understanding and predicting mobility patterns is thus critical for effective mobility management, which is essential for sustaining urban livability and functionality. From an environmental viewpoint, mobility prediction can lead to better management of urban energy consumption and help mitigate the environmental effects of overcrowded transport systems, such as air pollution and excessive fuel consumption [22, 24]. By optimizing transportation routes and reducing congestion, cities can achieve significant environmental benefits. Economically, efficient transportation systems can reduce travel time and costs, thereby increasing productivity and economic output. Reducing traffic congestion not only diminishes time spent in transit but also may decrease the gradual degradation of vehicles and roadways, leading to significant cost savings for both individuals and governments [24].

It is crucial to point out that technological advancements have been strongly shaping the ability to collect, analyze, and interpret large amounts of mobility information. In addition to the continuous progression of data storage/processing infrastructures, processing power and communication protocols [25, 26], the proliferation of devices enabled with Global Positioning System (GPS) and the ubiquity of mobile technology allow for the collection and handling of unprecedented volumes of data on human movement patterns. Indeed, despite the enormous potential and benefits of leveraging insights from human movement data, the continuous tracking of individuals through ubiquitous technologies raises significant privacy and security concerns, that must be continually recognized and mitigated [19, 27].

From the user perspective, mobility prediction offers significant advantages by enhancing travel efficiency and personalizing the driving experience [21, 28, 29]. This means optimized route planning, avoiding congestion, and reducing travel time. Predictive systems can suggest next trip information (e.g., next destination), best times to travel, alternative routes during peak hours, and even dynamic pricing for tolls to manage demand efficiently [21, 28]. Additionally, personalized travel and service recommendations have the potential to improve user satisfaction by considering individual preferences and habits, anticipating trip information, and making journeys more pleasant, informative, and less stressful. Furthermore, according to [30], predicting a user's next destination may enable advertisement companies to provide customized commercial advertisements. Indeed, within the scope of ride-hailing services, accurately predicting where and when a trip will start can help organize the vehicle fleet, reduce traffic congestion, and improve vehicle utilization [31]. For the end user, this translates into reduced wait times, increased availability, an enhanced user experience, and ultimately, better service quality.

In the context of ITS, the studies on mobility prediction can be classified by their level (or the unit of analysis): individual or collective. Additionally, they can be also characterized by the problem they address. In the following, these two dimensions are described. Then, greater depth is given to a mobility prediction

problem that is more relevant to the present thesis: trip information prediction. This includes a discussion on data specification and methodologies (see *Trip information prediction: data specification, methodologies, and challenges* below in this subsection).

Individual vs. collective mobility prediction

Mobility prediction models usually operate at the individual (or personal) level and collective (population, or general) level, each serving particular purposes [21, 32].

Collective-level models aim to forecast the movement patterns of groups. By analyzing aggregated data, collective prediction models can identify common patterns and trends, which are useful for applications such as traffic congestion management and on-demand transportation services [24, 31]. In [24], Deep Learning (DL) techniques are used to model and predict traffic congestion evolution, demonstrating significant accuracy in forecasting congestion patterns based on GPS data from taxis. Geng et al. [31] introduces a novel DL model for ride-hailing demand forecasting, which models complex spatial and temporal dependencies among different regions, thereby operating at a collective level to improve prediction accuracy.

Individual mobility prediction, on the other hand, focuses on forecasting the mobility patterns of an individual, typically based on his/her movement data. In [33], the authors specifically address individual-level prediction by utilizing a DL framework to analyze personal historical trip sequences and predict future trips. This work thus highlights the importance of individual-specific data and prediction times, thereby improving the accuracy of predictions for applications such as personalized travel recommendations and mobility management. Indeed, some works also employ collective information for individual mobility prediction, by clustering similar users or leveraging population-level patterns [27, 34–36]. W. Wang et al. [34] apply an attention-based DL framework to predict the most likely trip destination of a user's journey, from a specific sharing bike station, considering the check-in data of all the users and the geographical information of all sharing bike stations.

Mobility prediction problems

Mobility prediction in ITS involves addressing problems typically defined by the specific application they aim to support, such as reducing traffic congestion, enhancing safety, improving transportation efficiency, or providing personalized traveling recommendations. The level of mobility prediction — individual or collective — can influence the approach to solving these problems. Typically, collective-level models are effective for problems that support, e.g., congestion management, where understanding group movement patterns is crucial. Conversely, individual-level models are better suited for applications like personalized travel recommendations, where the focus is on predicting the behavior of a single user. Among the several existing problem domains, we described here a sample of them: trip information prediction, trajectory/route prediction, traffic flow prediction, and transportation mode prediction.

In the present review, we specifically refer to trip information prediction, which encompasses next destination (location or place) prediction, a popular mobility prediction problem often considered separately [21]. Some studies focus solely on predicting the next destination (e.g., [37]), while others address the broader scope of next trip prediction (e.g.,[20] and [38]). Mainly applied at the individual level, the former problem involves predicting the possible next visiting location of an individual. The latter includes that same purpose as well as predicting additional information, namely trip origin and time. The insights provided by including origin, destination, and time can be directly used in online applications such as personalized travel recommendations and management without extra processing procedures [21]. However, while next destination prediction offers relative simplicity, next trip prediction is usually more complex due to the difficulty in predicting when and where users will go next [20]. It is important to be mindful that these are distinct problems within the studies of mobility prediction, but since next trip prediction inherently includes next destination prediction, the next Subsubsection, *Trip information prediction: data specification, methodologies, and challenges*, will focus on the broader concept.

Trajectory prediction involves forecasting the future paths of vehicles or users, using historical data and real-time information. Addressing this problem is essential for enhancing autonomous driving [39, 40], traffic management, and navigation systems [41] by anticipating the trajectory (or moves) vehicles are likely to take, thus improving safety and efficiency [39, 40]. Liang and Zhao [41] introduce NetTraj, an approach designed to improve vehicle trajectory prediction in city-scale road networks by representing trajectories through intersections and movement directions, aiming to capture the complex spatial and temporal dependencies in road networks using advanced attention mechanisms. In the realm of autonomous navigation, [40] propose a model designed to predict the trajectories of other vehicles in complex traffic scenarios. This helps autonomous driving systems understand surrounding vehicle movements, improving decision-making and overall safety by capturing spatial and temporal dependencies in-vehicle interactions. In the same context, [39] propose a model for predicting the intentions of human drivers at intersections. This supports autonomous driving systems to anticipate whether a driver will turn left, right, or continue straight, improving decision-making and safety.

Traffic flow prediction involves forecasting the volume, speed, and density of traffic on road networks. It is crucial for reducing congestion, improving traffic management, and aiding in efficient transportation planning [42, 43]. On ITS, predicting traffic flow seems to be one of the more frequent use cases of DL [26]. In the work [43], the authors present a novel DL framework for traffic forecasting, which effectively captures spatio-temporal correlations in traffic data. It offers significant improvements in prediction accuracy and training efficiency over traditional methods, making it valuable for urban traffic control and guidance, and promising for large-scale industry deployment. In the same sense, [42] propose a novel DL approach using stacked autoencoders for traffic flow prediction, in which spatial and temporal correlations in traffic data are inherently considered. From experiments using real-world traffic data, they demonstrate the superior performance of the proposed method for traffic flow prediction.

Finally, transportation mode prediction is distinct from e.g., destination prediction, as it centers on determining how individuals are traveling instead of their future movements or locations. X. Liang et al. [44] demonstrate the potential to improve urban transportation planning by proposing a robust system that utilizes a smartphone's accelerometer, and employs a DL algorithm, specifically, a Convolutional Neural Network (CNN), to accurately detect users' transportation modes.

Trip information prediction: data specification, methodologies, and challenges

Trip information prediction is the research topic focused on predicting the next destination, trip origin, and trip times of individuals or vehicles. This area encompasses a significant corpus of work dedicated, particularly, to predicting future locations, employing a variety of state-of-the-art methodologies. The work of Ashbrook and Starner [45] comes as the cornerstone in the problem of next location prediction, introducing foundational methods that have influenced subsequent research. In their pioneering efforts, the authors utilized Markov Models (MMs), where raw GPS data is first clustered into frequent locations. Subsequently, a transition matrix is learned to predict users' most likely next location given the current position. This and other probabilistic approaches have become common strategies in the domain of mobility prediction, and are still employed in contemporary research [22].

Indeed, spatial information emerges as the most prominent feature of mobility data taken into account for trip information prediction. Spatial information can be concrete GPS coordinates, regions, or stay points, forming the backbone of the input data required for building predictive models. There is a notable trend towards utilizing models based on stay points, also referred to as Points of Interest (POI) or stop locations. The preprocessing of data often involves extracting these relevant points from raw GPS data, typically through clustering techniques [46], as they provide a simplified yet effective representation of the movement patterns of a person. Note that raw mobility data is usually obtained under two forms [21]: as trajectory data, where movement information is collected at a fixed sample rate, regardless if the individual moves or not; or as trip data, where data is recorded only when mobility activities/events happen, such as arrivals or departures, and check-ins or check-outs at specific locations.

A known drawback of the probabilistic approach in the pioneer work of Ashbrook and Starner [45] is its lack of flexibility and the difficulty in handling highly uneven sample sizes. For instance, if the learned most probable transition from *Home* is *School*, this prediction may be valid only during weekdays but not during weekends. According to [19], the spatial mobility of individuals is influenced by temporal context, meaning that the traveling behavior of a person varies depending on the day of the week. Thus, temporal information also represents a prominent source of context to combine with spatial information. Indeed, several human mobility studies have revealed that incorporating temporal information significantly increases predictability ([17], e.g., at 6 PM the most likely transition from *Work* is *Home*). Temporal information can be represented as timestamps, time intervals, day of the week, or the discretization of the day into hours or more abstract concepts such as morning, afternoon, evening, and night. Besides spatial

and temporal information, several works have utilized additional contextual features for trip information prediction, including, among many, smart card data (which usually offers additional contextual features such as the frequency and timing of trips, the mode of transportation, and user-specific travel habits), user activity patterns, semantic information from LBSN (location-based social networks), and land use functions [19, 20, 36, 38, 47, 48].

The studies from [20, 22, 49–51] address the issue of trip information prediction through probabilistic models in which temporal factors, e.g., the day of the week and the time of the day, are considered.

Probabilistic models, such as MMs and Hidden Markov Models (HMMs), capture the transition probabilities of mobility patterns, allowing for the prediction of future states based on the probability of different outcomes derived from historical data and observed patterns [21]. Such models offer advantages including efficiency, simplicity, and low computation cost [22]. As a note of reference, the studies in [28] and [51] highlight two main prediction approaches: early prediction (or pre-trip prediction), which involves predicting information of a trip before it begins, providing valuable information for travel planning and management; and during-trip prediction, which updates predictions in real-time as the trip progresses. Considering both strategies allows addressing the trade-off between accuracy and timing. The work by [52] extends the scope of destination prediction by developing an individual mobility prediction model, specifically an input-output HMM, to predict both the origin and destination of ride-hailing service trips for an individual. Furthermore, [20, 22, 53] not only predict the user's next locations but also trip times. However, in [53], the division of a day into arbitrary time intervals (e.g., one hour) defines the resolution of the temporal predictions and makes the system sensitive to categorical prediction changes at interval boundaries. This approach does not represent well the continuous nature of people's temporal choices. On the other hand, [20] captures the complex interaction of continuous time choices and discrete location choices, providing a more realistic representation of individual travel behavior.

More recently, Machine Learning (ML) techniques, such as DL models, have been widely used to predict next trip information [26, 29, 33, 54–56].

Deep learning represents a data-driven approach requiring human experts to define hyperparameter settings, learning rules, and cost functions. This technique has revolutionized trip information prediction, often surpassing traditional probabilistic methods [22]. DL models, predominantly those with specific recurrent architectures such as Long Short-Term Memory (LSTM) networks [57], utilize complex neural architectures with feedback connections to capture the non-linear, high-dimensional patterns in mobility data. As a result, there is a large consensus that deep learning offers a more robust and flexible framework for predicting movement patterns, as it enables handling large and complex datasets, and modeling intricate dependencies suited to the dynamic nature of human mobility [22]. Nevertheless, it is important to note that despite their superior predictive performance, DL models are generally black-box systems and difficult to interpret [20].

The studies [29] and [55] focus on enhancing the prediction accuracy of future locations by leveraging advanced DL models, capturing both short-term and long-term dependencies in individual mobility patterns. The DeepTrip model developed by [33] incorporates temporal information in the learning process, and is capable of predicting the origin, destination, and trip time of the next railway trip for an individual user at any time since the last mobility record. The results show that the best prediction accuracy is achieved when the prediction time is at about 80% of the real departure time. In [58], the authors apply ML algorithms to combine next location prediction of a vehicle with the associated status of the driver, passengers, and comfort settings.

The heterogeneity and sparsity of data recording human mobility are generally considered major challenges for deep learning models. According to [59], most data recording human mobility is low-sampling in nature, making it difficult to train a mobility model. Aggregating data from multiple users can negatively affect prediction accuracy due to the diverse and heterogeneous mobility patterns of individuals. The authors propose an attention recurrent network to predict human mobility from sparse trajectories. Similarly, [27] propose a federated learning framework that addresses heterogeneity by modeling individual mobility dynamics on local devices, clustering similar users on a parameter server, and using a third-party server to automatically learn a similarity graph. This graph enhances data-scarce users' representations by aggregating knowledge from neighboring users.

Indeed, state-of-the-art ML approaches typically learn representations offline from massive, stationary batches of training data [60]. They therefore cannot account for online learning scenarios in which information becomes incrementally available since there is no distinction between learning and test phases. Moreover, to account for the non-stationarity of mobility patterns which may show a systematic concept drift over time [61], additional adaptive mechanisms have to be introduced into ML-based models such as sliding time window methods [62], or tree-based stream learning algorithms [61].

1.3.2 Processing units in automotive assistance systems

Over the years, the automotive industry has evolved significantly to reach the advanced vehicles we see today. Historically, mid-20th-century vehicles were relatively simple, consisting primarily of an engine, a drive mechanism, wheels, a battery, an alternator system, and basic electrical components [63]. In the 1960s, an electronic fuel-injection system developed by Bosch was included in vehicles, marking the introduction of small computers in automotive technology [64]. By the end of the 20th century, it had become standard for vehicles to include a few dozens Electronic Control Units (ECUs). These embedded computing units were responsible for managing and controlling various functions, particularly those related to engine performance [64–66].

Fast-forward to modern times, where vehicles that were once hardware-defined are now rapidly transforming

into software-defined transportation platforms [67]. High-end vehicles now include more than 100 embedded ECUs [68, 69], connected via Controller Area Network (CAN), to manage core functions such as windows, lights, infotainment, comfort settings, engine parameters, and, with the shift towards electric vehicles, electric motor control and battery management systems [69, 70]. Furthermore, ADAS [2, 4], which, among many, include lane keep assist, adaptive cruise control, collision avoidance, parking assistance, and in-vehicle assistants, are also composed of ECUs running distributed real-time software algorithms to control specific functions within those systems, and interface with sensors and actuators [5]. Given the growing complexity and importance of vehicular electronic systems, some leading automotive companies are redefining the automotive computing system architecture from a traditional concept, i.e., various automotive functions distributed across multiple ECUs, to a more optimized software-based architecture [71]. Under this concept, ECUs are partitioned into multiple functionally related zones controlled by their zone controllers, which are interconnected and further partitioned into a set of domains controlled by domain controllers.

The Central Processing Unit (CPU), integrated within Microcontroller Units (MCUs) or System-on-Chips (SoCs), is the prominent processing unit for running automotive embedded software [4, 5, 64] . CPUs are favored for their versatility, affordability, low power consumption, and general-purpose processing capabilities. The first generation of automotive embedded systems relied on simple 8-bit MCUs [66]. While such resources may still be used for basic functionalities today, modern ECU architectures primarily consist of CISC and RISC-based 16/32-bit MCUs, including multicore processors [72, 73]. Automotive software typically runs on a Real-Time Operating System (RTOS), extensively used on automotive MCUs to meet the requirements of modern vehicles' safety-critical systems [71, 74]. Notwithstanding, like the in-vehicle monitoring system proposed by Sharara et al. [75], vehicular systems may employ MCUs running a bare-metal application. In contrast, modern ADAS, such as collision avoidance systems, require significant computational power and the ability to process data from various sensors in real time, making SoCs a preferred choice [76, 77]. Furthermore, automotive applications involving more complex functionalities, as well as soft real-time, non-time-critical applications (e.g., infotainment), predominantly run on embedded Linux systems [71, 74].

Cutting-edge SoCs integrate various components, from general-purpose CPU cores to high compute density accelerators, enabling heterogeneous computing that supports more intelligent and sophisticated ADAS [71, 76, 78, 79]. Also driven by advancements in artificial intelligence and data analytics, heterogeneous computing is becoming a crucial strategy for meeting the demands of innovative ADAS and the progression toward autonomous vehicles. This computing approach allows relieving the CPU, through the efficient allocation of complex and computationally intense tasks across specialized processing units, enhancing performance and efficiency [79]. These specialized processing units, commonly used as hardware accelerators, include Graphics Processing Units (GPUs), Field-Programmable Gate Arrays (FPGAs), and Application-Specific Integrated Circuits (ASICs) [71].

GPUs are prominent in state-of-the-art ADAS studies due to their ability to perform complex parallel calculations, making them ideal for tasks involving artificial intelligence, machine learning, and real-time data processing [80]. While CPUs excel at executing sequences of operations and managing a few tens of threads in parallel, GPUs are designed to handle thousands of threads simultaneously [81]. This is particularly useful because modern computational problems frequently have parallel structures, like those in video or signal processing. Embedded GPUs are already established in the automotive industry as suitable solutions for meeting the high-performance demands of automated driving [82]. NVIDIA is a key player in this sector, delivering platforms with GPU support for next-generation vehicles. Examples of automotive SoCs include NVIDIA Jetson TX2, NVIDIA AGX Xavier, and NVIDIA DRIVE, which leverage GPUs for e.g., deep learning tasks, including both training and inference, enabling sophisticated models to run efficiently and accurately [82, 83]. Moreover, developers can use C++ along with Compute Unified Device Architecture (CUDA), a parallel computing platform and Application Programming Interface (API) model created by NVIDIA, to harness GPU support on systems development [84].

FPGAs are also a high-performance computational alternative in the realm of embedded automotive technologies [4, 71]. These digital integrated circuits can be reconfigured, offering unparalleled flexibility and adaptability for implementing custom hardware logic tailored to specific applications. They stand out among hardware accelerators for their affordability, flexibility, ability to achieve extremely high operational frequencies, and low power consumption [4, 85, 86]. This makes them ideal for demanding applications, including deep learning [87] and real-time data processing [88]. FPGAs provide deterministic performance and can be optimized at the hardware level for specific tasks, leading to significant performance gains and efficiency improvements. An essential aspect of leveraging FPGAs effectively in automotive systems is the hardware-software co-design [86, 88]. This approach involves the concurrent design of software (running on general-purpose processors) and hardware components to achieve optimal performance. Despite their advantages, the design process for FPGAs is extremely complex and requires specialized expertise, which can impact their attractiveness for automotive system/application designers [89]. In comparison to GPUs, FPGAs may offer much less programming flexibility [81]. Nevertheless, high-level software frameworks for FPGA programming are emerging, making FPGAs more accessible to developers [86, 89].

While FPGAs are versatile and reconfigurable, making them ideal for prototyping, ASICs are specialized chips designed for specific applications [90]. They provide fixed-function hardware tailored to specific tasks, resulting in higher performance and efficiency for certain applications. In contrast to FPGAs, ASIC development involves a longer and more complex design cycle that includes hardware specification, simulation, verification, and physical fabrication. Despite their higher initial design costs, ASICs are preferred for high-volume production because they offer superior performance and lower unit costs when mass-produced. However, once fabricated, they cannot be reprogrammed, which limits flexibility but ensures maximum efficiency and reliability for their intended function. This makes ASICs a cost-effective solution for large-scale automotive applications where performance and efficiency are critical [4].

1.4 Main contributions

This thesis proposes a novel, cognitively inspired approach for enabling intelligent vehicles to learn and predict individual traveling routines in real-time. The work spans the design, implementation, and evaluation of an embedded system architecture, addressing both scientific and engineering challenges related to online learning, spatio-temporal prediction, real-world data collection, and real-time system implementation.

The main contributions of this thesis are as follows:

- A mobile data acquisition application developed for the longitudinal recording of daily driving routines, including GPS traces, occupants, and transported objects.
- A DNF computational model for learning and predicting spatio-temporal mobility patterns. Inspired by principles of cognitive neuroscience, the model supports continual, online learning, enabling adaptation to evolving behaviors. It provides interpretable internal dynamics through spatially organized neural activation patterns that reflect routine structures.
- A DNF-based approach for multi-entity mobility prediction, which integrates the computational model into a broader framework. This approach supports:
  - Online clustering of GPS data to identify and update regions of interest.
  - Multi-entity modeling, capturing the dynamic involvement of drivers, passengers, and objects in routine traveling events.
  - Autonomous recall and temporal anticipation of traveling events and associated context.
  - Transparent decision-making based on meaningful neural activation dynamics, enabling interpretability and contextual adjustment.
- An integrated software and hardware prototype, implemented using Robot Operating System (ROS) and deployed on an embedded computing platform (NVIDIA Jetson Nano). The prototype demonstrates the practical feasibility of deploying the approach in real-time in-vehicle assistance scenarios.
- An experimental validation, using real-world data collected over 11 consecutive weeks of daily vehicle use. The results illustrate the model's ability to adapt to evolving mobility routines and provide promising indications of its practical applicability when compared with traditional offline prediction approaches.

The work presented in this Thesis has resulted in the following core publications:

- P. Guimarães et al. "A data recording mobile application to create datasets of vehicle users' routines". In: *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE. 2022, pp. 167–172. doi: 10.1109/ICARSC55462.2022.9784770. url: https://ieeexplore.ieee.org/document/9784770.

- P. M. F. Guimarães et al. "A Dynamic Neural Field Approach for Intelligent Cockpits: Online Learning and Prediction of Traveling Routines". In: *IEEE Transactions on Intelligent Transportation Systems* 25.12 (2024-12), pp. 20240–20255. doi: 10.1109/TITS.2024.3463389. url: https://ieeexplore.ieee.org/document/10700688.

Additionally, the author contributed to other publications:

- F. Ferreira et al. "Dynamic Identification of Stop Locations from GPS Trajectories Based on Their Temporal and Spatial Characteristics". In: *Artificial Neural Networks and Machine Learning – ICANN 2021*. Vol. 12894. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 347–359. doi: 10.1007/978-3-030-86380-7_28. url: https://link.springer.com/10.1007/978-3-030-86380-7_28.

- W. Wojtak et al. "Towards Endowing Intelligent Cars with the Ability to Learn the Routines of Multiple Drivers: A Dynamic Neural Field Model". In: *Computational Science and Its Applications – ICCSA 2021*. Vol. 12952. Cham: Springer International Publishing, 2021, pp. 337–349. doi: 10.1007/978-3-030-86973-1_24. url: https://link.springer.com/10.1007/978-3-030-86973-1_24.

- P. Barbosa et al. "Endowing Intelligent Vehicles with the Ability to Learn User's Habits and Preferences with Machine Learning Methods". In: *International Conference on Intelligent Data Engineering and Automated Learning*. Springer. 2022, pp. 157–169. doi: 10.1007/978-3-031-21753-1_16. url: https://link.springer.com/chapter/10.1007/978-3-031-21753-1_16.

1.5 Thesis outline

This thesis is organized into six chapters. Following this introductory chapter, each subsequent chapter contributes to the overall research goal of developing a cognitively inspired, real-time capable system for learning and predicting traveling routines.

Chapter 2. Introduces key concepts that provide the necessary foundation for better understanding the technical content presented in the subsequent chapters.

Chapter 3. Describes the development of a mobile application designed for the acquisition of daily vehicle usage data. *This chapter is based on a core publication of the author and was adapted to fit the context of the thesis.*

Chapter 4. Presents the core scientific contribution of the thesis — a neurodynamics-based approach for multi-entity traveling routine learning and prediction. *This chapter is based on a core publication of the author and was adapted to fit the context of the thesis.*

Chapter 5. Details the system-level implementation of the embedded prototype, showcasing its modular architecture, real-time execution capabilities, and integration of the DNF-based approach into a ROS-based embedded environment.

Chapter 6. Provides a general discussion and conclusion of the thesis, including a synthesis of the main findings, broader implications, acknowledged limitations, and future research directions.

Chapter 2

Fundamentals

2.1 Dynamic Neural Fields

Dynamic Neural Field (DNF) theory is the fundamental tool employed in this thesis for developing the neurodynamics-based assistance system. Although this thesis refers to the tool as DNFs, the broader framework is more accurately termed Dynamic Field Theory (DFT). It has been extensively applied in the domain of cognitive robotics, providing a means to endow autonomous robots with high-level cognitive functions. More recently, DNFs has been explored in the automotive domain [91]. DNFs offer the potential to improve intelligent systems by enabling continuous adaptation and real-time decision-making in dynamic environments. This section begins with an overview of DNFs, introducing key principles and theoretical background. Following this, a detailed explanation of the mathematical framework is provided, along with some cognitive mechanisms that can be implemented using DNFs. Finally, the section concludes by presenting an example of a DNF model architecture capable of endowing a robot with the cognitive and behavioral capabilities to learn and play a musical piece.

2.1.1 Overview

DNFs is a biologically-inspired mathematical framework used to model how neural populations generate cognition and behavior [92]. It integrates principles from dynamical systems theory to describe the time-evolving activity of neural fields, which represent continuous variables such as space or stimulus features (e.g., color, orientation). By modeling these neural dynamics, DNFs provide a formal account of cognitive mechanisms, including detection, learning, memory, and decision-making [93].

DNF theory has its roots in the motor domain, where researchers aimed to explain how movements are generated through dynamical systems principles [94]. Early work, inspired by functional muscle models [95], proposed that human movement could be guided by virtual forces, similar to a pendulum moving toward equilibrium [96]. This dynamical systems approach gained further traction with the discovery of instabilities in movement coordination [97, 98]. The researchers observed stable patterns, such as finger coordination, which maintained stability under perturbations. However, at higher frequencies, some patterns would lose stability, leading to a switch to a more stable pattern. This finding highlighted the critical role of stability in the emergence and maintenance of coordinated behavior, providing a formal mathematical basis for understanding how patterns arise and change over time.

While the dynamical systems approach worked well for motor control — since movement is inherently time-continuous and its resistance to perturbations can be directly observed — it was less clear how it applied to cognitive processes. DNF theory addressed this by incorporating representational states into the dynamical systems framework [94]. In this view, not only motor states but also internal neural states that underlie cognition evolve dynamically, forming stable attractors that enable cognitive processes to persist, resist distractions, and accomplish goals. Since its establishment in the early 2000s, DNFs has become a versatile tool in cognitive sciences, where they have been used to model a wide range of phenomena,

including saccadic motor planning [99], planning of reaching movements [100], development of motor planning [101], perception of motion [102], and word learning [103], to name a few (for an overview of experimental and theoretical studies see [94]).

Interestingly, before its widespread application in cognitive sciences, the foundational ideas of DNF theory emerged from early work in robotics. Schöner and colleagues' work in 1995 [104, 105] became a cornerstone for the development of the theory, where it was first applied to implement cognitive behaviors in autonomous agents. Rather than relying on explicit rules or conditional logic, robots driven by the principles of DNFs utilized neural dynamics to generate behavior. This was also demonstrated in [93], where the dynamic approach (attractor dynamics) to behavior generation was extended with cognitive capabilities, such as memory and decision-making, allowing robots to dynamically adapt to their environments in real time. The application in robots highlights the potential of DNFs to move beyond descriptive models, offering a framework that directly produces behavior. Since the early 2000s, DNF theory has been employed in developing complex control architectures that endow robots with high-level cognitive functions [106–113]. Building DNF models for robotic implementations has enhanced confidence in DNFs principles [94]. Furthermore, robotic models have been observed to be useful for demonstration and to serve as sources of new ideas and questions, contributing to the continuous refinement and expansion of the theory.

2.1.2 Foundations of Dynamic Neural Fields

A dynamic neural field (DNF) is the fundamental building block of DNF theory [92, 94]. It represents the activation of cortical neural populations distributed across a continuous dimension, where information is encoded and processed (see Figure 1). This dimension can represent various types of features, such as spatial location, orientation, color, or movement parameters.

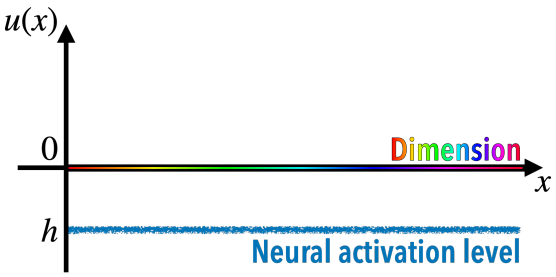


Figure 1: **The fundamental building block of DNF theory: the dynamic neural field.** In this example, the neural population is distributed across a single continuous dimension representing color as a feature. Each neuron in the field is thus tuned to a specific hue value. The activation level of the neural population is shown at its resting state at a particular instant in time.

In a dynamic neural field, each site is tuned to a specific value of the feature space and is characterized by a level of activation. The activation level of a DNF evolves continuously over time, governed by a dynamical system. As a result, the activity across the field changes dynamically in response to external inputs or

interactions within the field. DNFs can maintain or shift their activation patterns depending on the cognitive task, such as focusing on a specific stimulus or retaining information in memory.

Inputs can drive the emergence of activation peaks within a dynamic neural field, which are the basic units of information encoding. Assuming 0 as the activation threshold level, a weak input may generate a peak that remains below this level (refer to Figure 2(a)). In contrast, a sufficiently strong input results in the formation of an activation peak (see Figure 2(b)). These activation peaks represent the presence of specific values along the feature space. Typically, the strength of a peak's activation indicates the degree to which a particular value is present. It will be shown later in this section that, depending on the application, the amplitude of activation can have other meanings. From a dynamical systems perspective, these neural activation peaks function as attractors, representing stable states that the system gravitates toward. Once an attractor state is reached, the field remains in this stable configuration until external inputs or internal interactions drive the system to a different state, such as a resting state. The combination of stability, flexibility, and dynamic adaptation makes DNFs particularly effective for modeling cognitive processes, such as detection, decision-making, and memory.

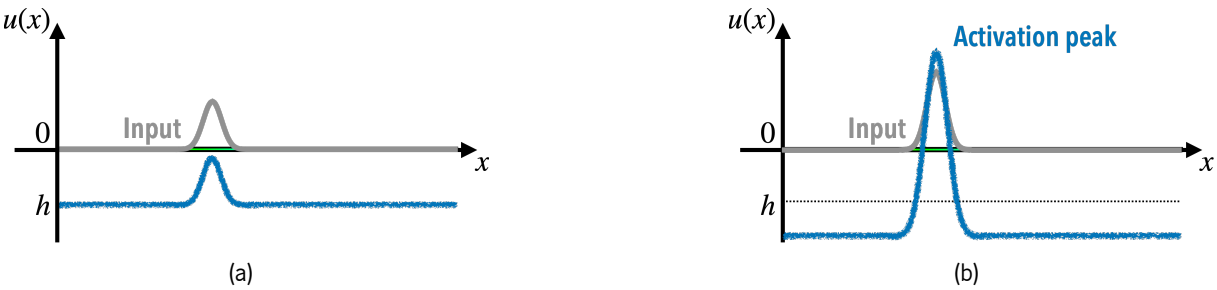


Figure 2: **Activation dynamics in a DNF.** In **(a)**, the applied input is too weak to drive the development of an activation peak, and the field's activation level simply follows the input pattern. **(b)** When a sufficiently strong input is applied, it enables the formation of a stable activation peak, the basic unit of information encoding — in this case, the color green. This self-stabilized peak arises from both the input strength and recurrent interactions within the field.

DNFs can also represent more abstract, non-metric information, such as object features or categories [108]. Furthermore, a DNF's feature space can span multiple dimensions, allowing it to be simultaneously tuned to information from different features (see Figure 3).

Dynamic Neural Field equation

The integro-differential equation proposed by Amari in 1977 [114] was chosen by Schöner et al. [104, 105] to govern the dynamics of DNFs. The mathematical basis governing the fields builds on earlier work in neural and behavioral sciences. It all started in the early 1970s when Grossberg laid the groundwork for neural dynamics by describing how neural networks process information over time [115]. Around the same time, Wilson and Cowan [116] introduced the concept of neural fields, which modeled the activity within

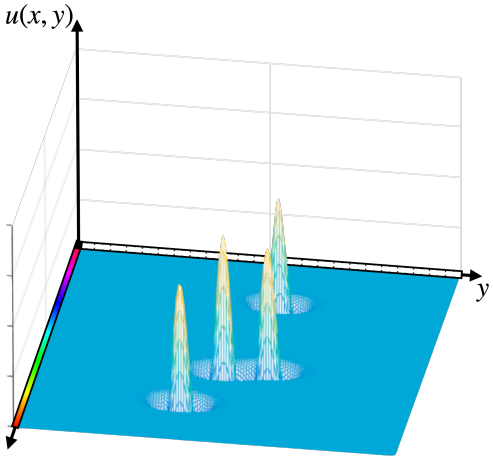


Figure 3: **Example of a two-dimensional dynamic neural field tuned to color and size features.** The neural population is distributed across two continuous dimensions, where the x-axis represents color, and the y-axis represents size. This example shows activation peaks at a specific moment in time, indicating strong neural responses to particular combinations of color and size, while other areas remain at resting activation levels.

cortical regions where neighboring neurons interact. However, the most significant advancement came with Amari [114] later in that decade, who extended these ideas by formalizing the interaction between neural populations. Amari's equation was ultimately chosen because it not only models the dynamic evolution of neural fields but also allows for the analytical study of these dynamics.

The dynamics of a DNF is governed by the following equation based Amari's work [114]:

$$\tau \frac{\partial u(\mathbf{r}, t)}{\partial t} = -u(\mathbf{r}, t) + h + I(\mathbf{r}, t) + \int w(\mathbf{r} - \mathbf{r}') f_\theta(u(\mathbf{r}', t))\, \mathrm{d}\mathbf{r}' \tag{2.1}$$

The equation defines the continuous evolution of the activation level, u, at field location $\mathbf{r}$. To better understand this, consider the field's feature space, $\mathbf{r}$, as a layer of neurons with an infinitely dense distribution (see Figures 1 and 3). In this context, $u(\mathbf{r}, t)$ represents the neural activation of the neuron located at $\mathbf{r}$ at time t. Here, $\mathbf{r} \in \mathbb{R}^d$, with $d = 1, 2$, which can denote either a single dimension, x, or a two-dimensional feature space, (x, y). **For simplicity, the rest of this section will proceed with $\mathbf{r} = x$, focusing on a one-dimensional representation.**

The following description addresses both the linear (2.2) and nonlinear (2.3) components of the equation. The analysis begins with the linear part of the equation as follows:

$$\tau \frac{\partial u(x, t)}{\partial t} = -u(x, t) + h + I(x, t) \tag{2.2}$$

As previously noted, $u(x, t)$ stands for the activation level of the neuron at feature value x at time t. The parameter $h < 0$ corresponds to the resting level to which neural activity converges in the absence

of external input, $I(x,t)$. In the presence of input, the field simply reproduces the input pattern. The parameter τ defines the relaxation rate at which the field converges to stable states.

The nonlinear integral part of the equation represents the interactions between neurons within the neural field, consisting of a convolution operation. These interactions primarily explain how activation patterns can be sustained even when external input is removed, thereby serving as a neural substrate for modeling cognitive mechanisms:

$$\int w(x-x')f_\theta(u(x',t))\,\mathrm{d}x' \tag{2.3}$$

Specifically, for each neuron, the integral sums up the excitatory and inhibitory effects that every other neuron in the field has on it. These effects are governed by an interaction kernel (also referred to as a connectivity function), $w(x-x')$, which defines how strongly the activity at neuron x' influences neuron x based on their distance. The nonlinear firing rate function f_θ ensures that only neurons with a sufficient activation level contribute to the recurrent interactions. This function is typically taken as a Heaviside step function:

$$f_\theta(u) = \begin{cases} 0, & \text{if } u < \theta \\ 1, & \text{if } u \geq \theta \end{cases} \tag{2.4}$$

The activity of each neuron x is thus influenced by all neurons in the field with activation above threshold (suprathreshold level), where $f_\theta(u(x',t)) > \theta$ and typically $\theta = 0$, with the strength of each interaction being determined by the interaction kernel $w(x-x')$.

Revisiting now Equation (2.1), with $\mathbf{r} = x$, one can observe that the behavior of the field depends on both the presence and strength of external input. In the absence of input, the field converges to a resting state level (see Figure 1). When a localized, transient input is applied, the field dynamics vary according to the input strength. With weak input — insufficient to bring activation above threshold θ (activation at subthreshold level) — the field operates in an input-driven regime and reproduces the input pattern without forming sustained activation peaks (see Figure 2(a)). In contrast, strong input that brings local activation above threshold θ generates a localized activation peak, activating the integral term and initiating interactions between neurons governed by the interaction kernel (see Figure 2(b)). The interaction kernel determines whether these activation patterns are transient — returning to the resting state when the input is removed — or self-sustaining, persisting even in the absence of input. In the following, various interaction kernels are examined, highlighting how they can shape the activation patterns within a field.

Type of interaction kernels

Different formulations of the interaction kernel can be used to support specific types of interactions. Figure 4 illustrates the three primary kernel shapes.

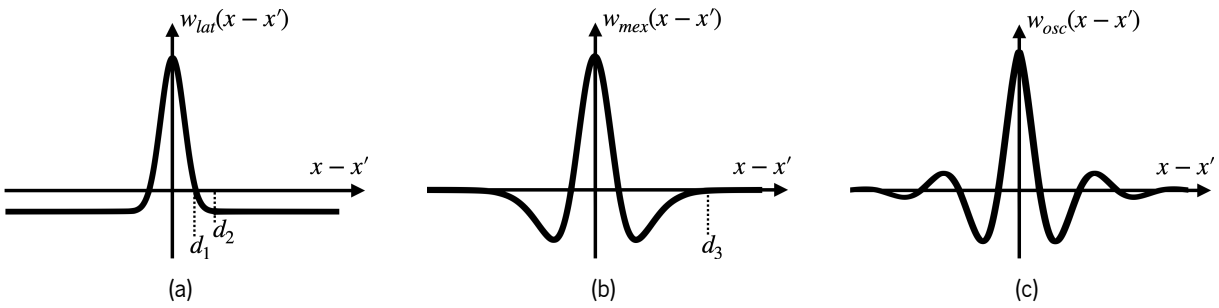


Figure 4: **Types of interaction kernels defining interaction strength based on the distance between two neurons. (a)** Lateral inhibition (or Gaussian) kernel; **(b)** Mexican-hat kernel; and **(c)** Oscillatory kernel.

A lateral inhibition (or Gaussian) kernel (see Figure 4(a)) supports, at most, the evolution of a single activation peak in response to one or multiple localized inputs. In this kernel, excitation dominates at closer distances, while neurons farther apart are subject to homogeneous inhibition. This structure forces competition between sufficiently distant neural subpopulations, allowing only the strongest localized activation to persist. The kernel is defined by the following equation:

$$w_{lat}(x-x') = Ae^{-\frac{(x-x')^2}{2\sigma^2}} - g \tag{2.5}$$

where $A > 0$ represents the amplitude and $\sigma > 0$ the standard deviation of the Gaussian, and $g > 0$ the global inhibition. When interactions are characterized by this lateral inhibition kernel, any neuron at position x' will contribute with an excitatory strength to neuron x if $-d_1 < (x-x') < d_1$ (see Figure 4(a)). Neurons at a distance greater than d_2 from each other (in absolute terms) will contribute (if they are at suprathreshold levels) with an approximately constant inhibitory strength.

The Mexican-hat kernel (see Figure 4(b)), defined by the Equation (2.6), enables the coexistence of multiple activation peaks if they are separated by a distance greater than d_3, as lateral interactions approach zero at those distances.

$$w_{mex}(x-x') = A_{exc}e^{-\frac{(x-x')^2}{2\sigma^2}} - A_{inh}e^{-\frac{(x-x')^2}{2(\gamma\sigma)^2}} \tag{2.6}$$

This kernel is represented by the difference of two Gaussians, where the excitatory Gaussian has an amplitude larger than that of the inhibitory Gaussian ($A_{exc} > A_{inh} > 0$). $\sigma > 0$ determines the lateral excitatory range, and $\gamma\sigma$, with $\gamma > 1$, determines the range of lateral inhibition. This configuration produces a localized excitatory peak with inhibitory side lobes.

The Oscillatory kernel (see Figure 4(c)) introduces an interaction structure that supports the existence and stability of multiple activation peaks within a neural field [117, 118]. Its oscillatory shape, characterized by multiple zero crossings, creates alternating regions of excitatory and inhibitory interactions at increasing

distances. This configuration enables the formation of excitatory interactions at larger distances, though with decreasing strength as the distance increases, allowing for distributed and stable patterns of multiple peaks. The kernel is defined by the following equation:

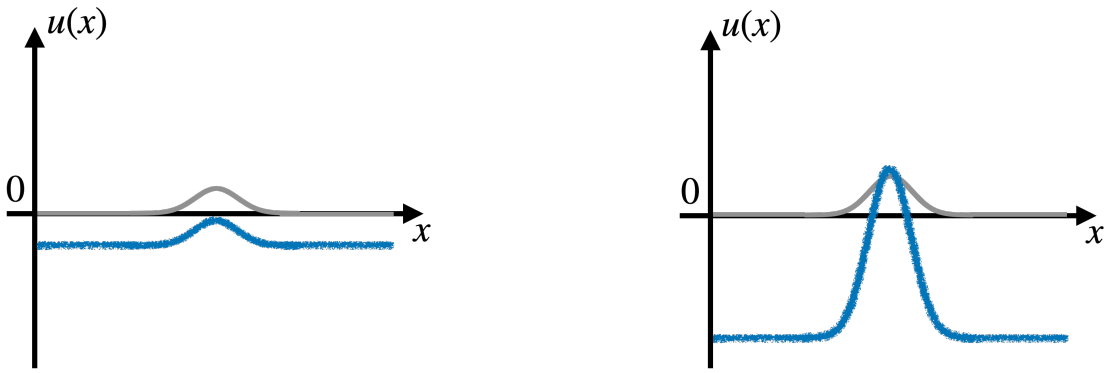
$$w_{osc}(x - x') = Ae^{-b|x-x'|}\left(b \sin|\alpha(x - x')| + \cos(\alpha(x - x'))\right) \tag{2.7}$$

The decay rate of oscillations is controlled by the parameter $b > 0$, which dictates how rapidly the oscillations decrease in amplitude with distance. The parameter $A > 0$ sets the overall amplitude, while $0 < \alpha \leq 1$ regulates the frequency of zero crossings.

Dynamic Neural Fields cognitive mechanisms

As previously mentioned, various cognitive functions can be implemented using principles of DNFs, enabling the development of complex DNF-based cognitive models [94, 107]. The following focuses on examples most relevant to the context of this thesis.

Detection. DNFs can represent both the presence and absence of information. When input $I(x, t)$ is too weak to drive activation above the threshold level, the field remains subthreshold, indicating an absence of detection. In this regime, the field's activation pattern stabilizes in $h + I(x, t)$, not forming an activation peak (see Figure 5(a)). To induce detection, a localized input must be sufficiently strong and applied for a sufficient time interval to ensure that $f_0(u) > 0$, meaning the activation reaches the suprathreshold level at that location. Once reached, recurrent interactions within the field amplify the response, resulting in a stable activation peak reflecting the detection of the input, thus shaped by both the input and the interactions within the field (see Figure 5(b)). When the input is removed, the field relaxes back to the resting state.



(a) (b)

Figure 5: **Detection mechanism in a DNF with a Gaussian interaction kernel.** In **(a)**, the input (gray line) is too weak to drive activation above threshold, resulting in a subthreshold response. In **(b)**, a stronger input (gray line) drives the activation above threshold, forming a stable activation peak that reflects the detection of the input.

Memory and forgetting. One of the key features of DNFs is the ability to maintain an activation peak through recurrent interactions, even after the input that initially generated the peak is removed. This self-sustained activity remains centered around the original input location, effectively serving as a memory of that perceived input. The conditions for this memory mechanism depend on the properties of the interaction kernel and the resting level. As shown by [114], a neural subpopulation will maintain an activation peak without input if the balance of excitation and inhibition is sufficient to keep active neurons above the threshold level. There is thus a limit level for the resting level, $h \geq -W_m = -\max_x \left\{\int_0^x w(x')\,dx'\right\}$, above which the field can sustain a self-stabilized peak (see Figure 6). In some applications, however, it may be necessary to forget a memorized event (such as one that is no longer relevant) by suppressing the corresponding self-sustained peak. This can be achieved by homogeneously lowering the resting level below the limit level and returning it to the original resting state.

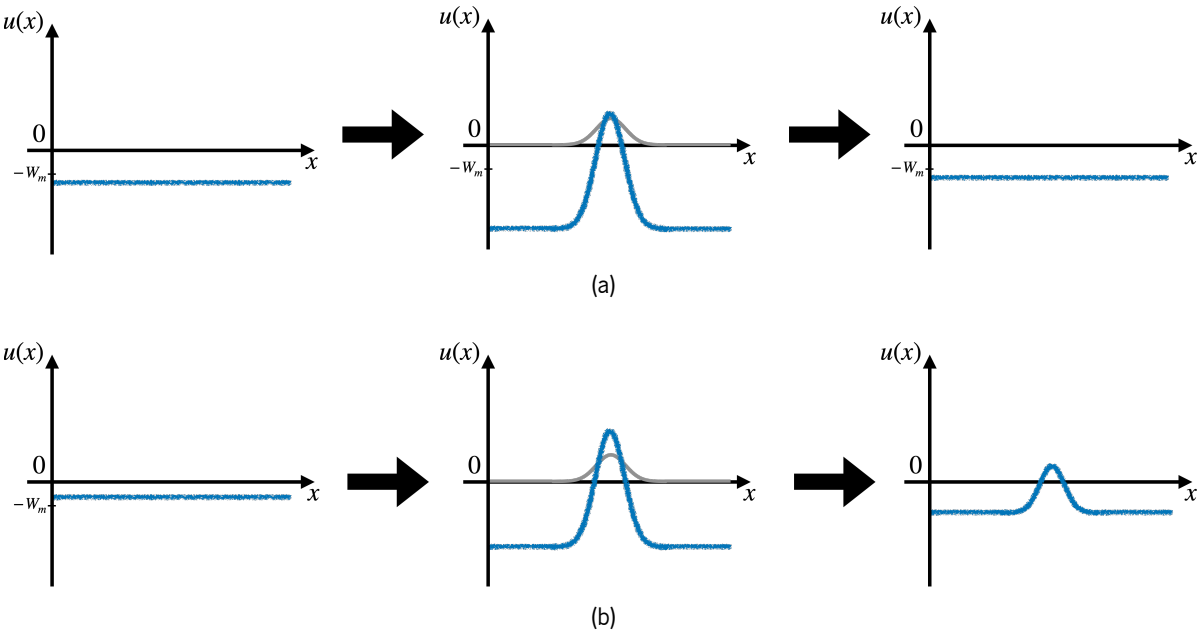


Figure 6: **Memory mechanism in a DNF with a Gaussian interaction kernel.** In **(a)**, with the resting level set below the critical threshold $h < -W_m$, the activation peak generated by the input (gray line) cannot sustain itself once the input is removed. In **(b)**, increasing the resting level above $-W_m$ allows the activation peak to persist even after the input (gray line) ceases, demonstrating a self-sustaining memory effect.

Memory trace. In DNFs, a memory trace operates on a longer timescale, characterized by slow transitions. When localized input, from an external input source or a connected neural field, is provided, respective activation builds gradually, creating a trace that represents a memory of the input (see Figure 7). Due to this slower timescale, the memory trace does not immediately vanish once the input ceases but instead decays slowly over time. This persistent trace can be used to preshape other neural fields, enhancing their sensitivity to future inputs. It can be also useful for generating a representation of the probability of perceived events [100].

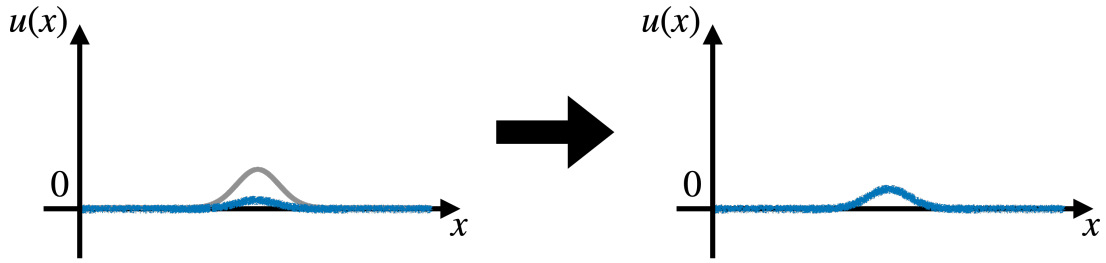


Figure 7: **Memory trace mechanism.** The input (gray line) induces a gradual buildup of activation, which persists as a decaying memory trace even after the input is removed, due to the slower timescale of the field dynamics.

Multi-item memory. The field dynamics can support multi-peak solutions when the interaction kernel is of the oscillatory type (Equation (2.7)), allowing for the stable representation of multiple items simultaneously. This may occur in two main scenarios: either when multiple sufficiently strong localized inputs are presented simultaneously (see Figure 8(a)) or when localized inputs are applied sequentially, with new inputs added to an already active field (see Figure 8(b)). In both cases, the oscillatory coupling enables the field to stabilize a multi-peak pattern, in contrast to fields with lateral inhibition coupling (see Figure 6), which support only a single peak at a time. When necessary, it is possible to suppress a self-sustained activation peak (forgetting mechanism) without globally adjusting the resting level by applying a sufficiently strong, localized, transient inhibitory input. This input disrupts the peak directly, without altering the overall stability conditions.

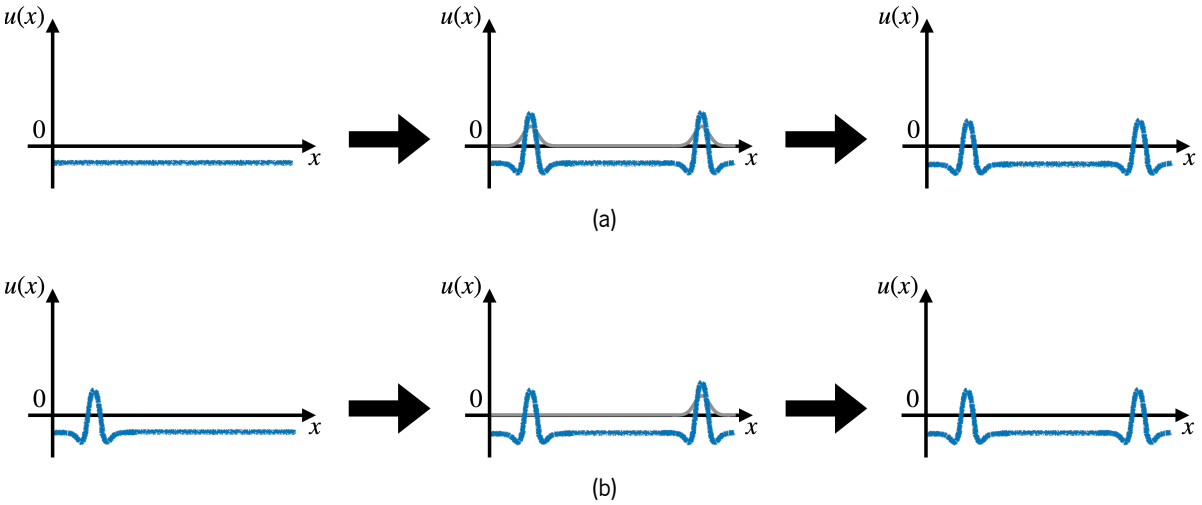


Figure 8: **Multi-item memory mechanism in a DNF with an Oscillatory interaction kernel.** In **(a)**, multiple localized inputs (gray line) are presented simultaneously, leading to the evolution of two self-sustained peaks. In **(b)**, a single input (gray line) is applied to an already active field, adding a new peak without disturbing the existing one, demonstrating the field's ability for sequential memory encoding.

Encoding of the order and timing. While multi-item memory allows for the simultaneous representation of multiple items within the field, encoding the order and timing of perceived events enables DNFs to

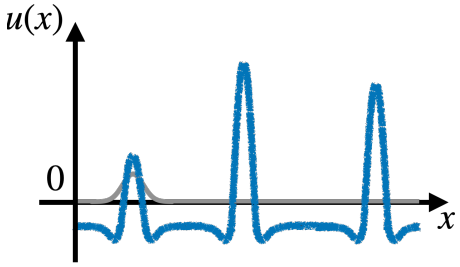


Figure 9: **Encoding of the order and timing of perceived events in a DNF with an Oscillatory interaction kernel.** The activation peaks reflect a sequence of inputs, with peak amplitudes corresponding to the elapsed time since each input was received. The highest peak corresponds to the earliest input received. This encoding allows the field to represent both the order and relative timing of events.

capture the sequential structure and temporal properties of occurrences, which is crucial for tasks requiring event sequence memory. This challenge has been addressed by [111] and [119], who proposed a solution by making the resting level activity both time- and state-dependent, $h = h(x, t)$, allowing it to adapt based on recent inputs. This resting level dynamics results in a steady buildup of peak amplitude at locations where input has been received, with the amplitude reflecting the elapsed time since the input onset. Consequently, when a sequence of perceptual events occurs, the resulting multi-peak pattern captures both the order and relative timing of events, as later peaks exhibit smaller amplitudes compared to those generated earlier in the sequence (see Figure 9).

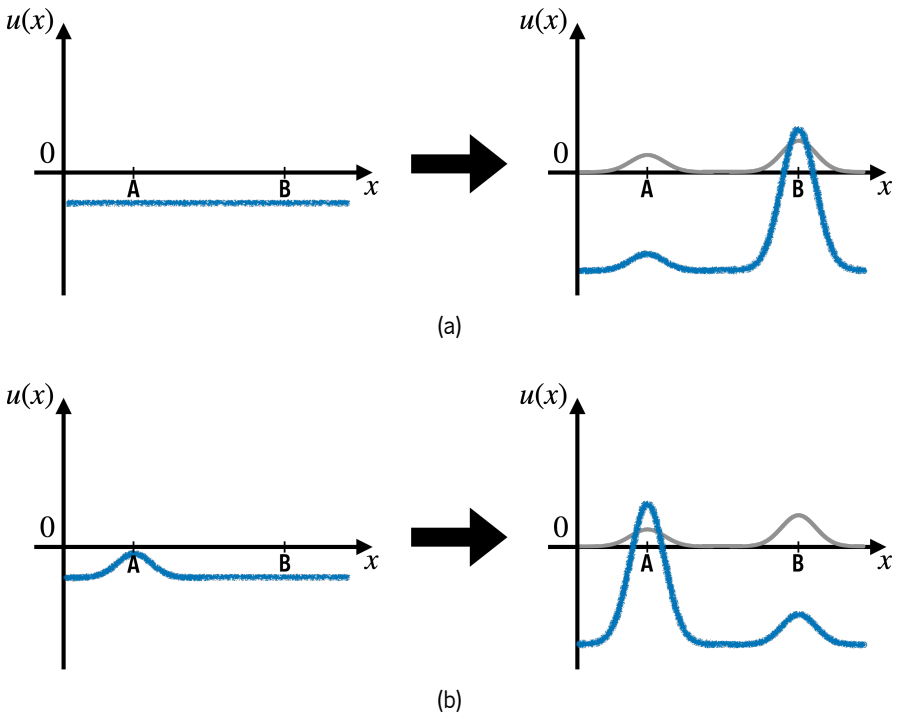


Figure 10: **Decision-making mechanism in a DNF with a Gaussian interaction kernel.** In **(a)**, field sites A and B receive excitatory input simultaneously, but the stronger input at site B drives the development of an activation peak, reflecting the decision outcome. In **(b)**, prior preshaping at site A gives it a competitive advantage, allowing it to dominate even when the input at site B is stronger.

Decision-making. DNFs can implement the fundamental cognitive ability of decision-making. For this to occur, a lateral inhibition kernel with sufficiently strong global inhibition is required (see Equation (2.5)). For example, consider an input pattern with two distinct peaks. If these peaks are sufficiently distant, their interaction will be purely inhibitory, resulting in a competition between them in which one peak ultimately suppresses the other. The outcome of this competition depends on two main factors. The first is the strength of the excitatory input: if two field sites, A and B, begin with similar initial conditions and receive excitatory input for the same duration, the site with the stronger input will drive the development of an activation peak (see Figure 10(a)). As to the second factor, if one of the input peaks is located at a field site that has been preshaped by input from a connected field — such as site A in Figure 10(b) — it gains a competitive advantage. In this case, site A may dominate the competition, even if the external input at site B is initially stronger.

2.1.3 Example of a Dynamic Neural Field model architecture

Complex DNF model architectures are constructed by coupling multiple neural fields to encode task-specific knowledge and generate intelligent behaviors. These large-scale networks are organized as interconnected fields, each modeled using variants of Amari's equation (Equation (2.1)). The stability of attractor solutions within these networks depends on model parameters and current inputs. Coupling is achieved by using the activity of one field as input to another, often with additional processing, such as convolution with an interaction kernel — similar to the nonlinear integral term in (2.3), but computed based on the activation level of the input neural field —, or modulation by the firing rate function (2.4) to ensure that only suprathreshold neural activity contributes as input. In this formulation, the input term $I(x,t)$ represents the combined influence of external sources, such as sensors, and connected fields.

We illustrate inter-field coupling in DNFs using a basic model with three coupled one-dimensional fields (see Figure 11). This example facilitates understanding of more complex DNF architectures, introduced

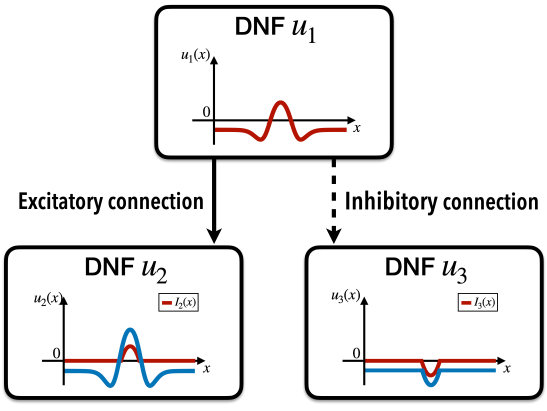


Figure 11: **Basic DNF model architecture illustrating inter-field interactions.** In this coupled configuration, only suprathreshold activity in u_1 provides localized excitatory input to u_2 and inhibitory input to u_3. Intra-field interactions within u_2 may support the formation of a self-sustained activation peak.

next and throughout the remainder of this thesis. In this basic model, the top field, u_1, contains a stable activation pattern, triggered by input from sensors. The activity of u_1 serves as input to the two bottom fields, u_2 and u_3, with excitatory and inhibitory effects, respectively. In this setup, only suprathreshold activity in u_1 affects the other fields. To achieve this, the activation of u_1 is considered for modulation by the nonlinear firing rate function (2.4), with a threshold $\theta = 0$. The inputs to fields u_2 and u_3 are then defined as $I_2(x,t) = u_1(x,t) f_0(u_1(x,t))$ and $I_3(x,t) = -u_1(x,t) f_0(u_1(x,t))$. Scaling factors could also be applied to the inputs to control the influence of u_1 on the dynamics of the coupled fields. The suppression in u_3 ensures, for instance, that excitatory input from other fields cannot initiate an activation peak at the inhibited location. Furthermore, depending on the within-field (or intra-field) recurrent interactions, the activation peak in u_2 resulting from the excitatory input from u_1 may become self-sustained if the input ceases, while in u_3, suppressed localized activity returns to the stable resting state.

In the following, a complex DNF model demonstrates how neural field architectures can be designed for tasks such as sequential learning and recall [111]. In this experiment, a robot learns, by observation, memorizes, and then plays a precisely timed musical sequence (see Figure 12). Each note in the sequence is represented as a colored square on a computer screen, perceived by the robot. The fields' feature space is therefore conceptually divided into slots, each corresponding to a specific color to encode the sequence (see Figure 13).



Figure 12: **Autonomous agent endowed with cognitive and behavioral capabilities to learn and play a musical piece.** A complex DNF model allows the robot to learn and play a musical piece through observation. Image retrieved from [111].

This model consists of two main parts: sequence learning and sequence recall. During the sequence learning phase, homogeneous external input is provided to the corresponding color slot in the Perceptual field u_{per}, driving the development of an activation peak that represents the sensory information from the vision system. Through excitatory connections, this peak triggers the evolution of a localized activation

pattern at the corresponding site in the Present Sequence Memory field u_{mem}. This setup encodes the order and timing of observed events, as previously discussed in this section on *Encoding of the order and timing* within *Dynamic Neural Fields cognitive mechanisms*. A resting level dynamics in u_{mem} ensures a steady buildup of peak amplitudes, where each amplitude reflects the elapsed time since the input onset. The objective is to establish an activation gradient that encompasses all elements of the observed sequence, creating a multi-peak pattern that captures the temporal sequence. Inhibitory feedback from u_{mem} to u_{per} destabilizes the existing peak in the Perceptual field, allowing newly arrived perception input to automatically drive a peak at a different location in the field — even if the color information is repeated later in the sequence. This feedback inhibition helps ensure that each element in the sequence is encoded in a distinct location within a slot.

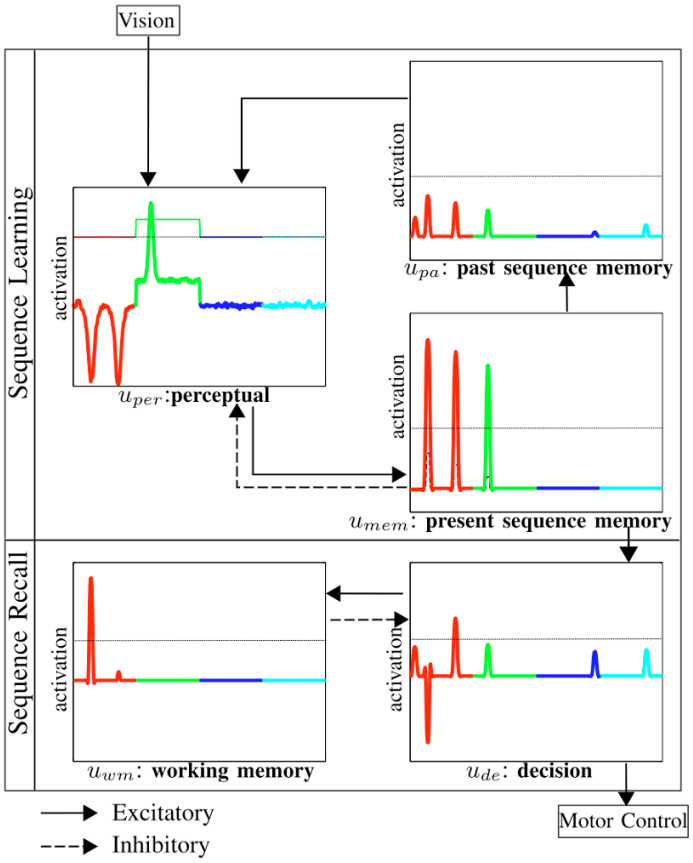


Figure 13: **Architecture of the DNF model for learning a musical piece.** The model consists of interconnected fields implementing sequence learning and sequence recall. Diagram retrieved from [111].

The learned multi-peak pattern serves as subthreshold input to the Decision field u_{de}. The sequence recall phase begins when the resting level in u_{de} steadily increases, homogeneously, bringing the subthreshold pattern closer to threshold. As each element in the Decision field sequentially (and timely) reaches threshold, an activation peak evolves, signaling the recall of that specific element. In the robotic experiment, this threshold crossing triggers the robot to press the corresponding color key on the keyboard, producing a musical note. Each recalled element in u_{de} excites the Working Memory field u_{wm} through excitatory

connections, while in turn, an inhibitory feedback loop suppresses the corresponding active decision in u_{de}. This dynamic process of sequential recall continues until all elements in the sequence are recalled and stored in u_{wm}, representing the full execution of the sequence.

2.2 GPU acceleration

This section delves into using GPUs for offloading intensive computations from CPUs. GPUs are particularly efficient at parallel processing, making them ideal for managing large sets of information in AI applications. This section begins with an introduction to GPU acceleration, followed by a summary of the NVIDIA Jetson Nano board, highlighting its computing capabilities. Finally, a programming tool is introduced that enables developers to use GPUs for general-purpose computing, specifically to accelerate computational processes.

2.2.1 Overview

GPUs have transcended their initial purpose of rendering graphics in gaming and video applications to become pivotal in accelerating general-purpose computing [120]. This paradigm shift, known as GPU acceleration, leverages, in certain contexts, the highly parallel architecture of modern GPUs to perform complex computations faster than traditional CPUs can [121]. Unlike CPUs, which excel at executing sequences of operations and managing a few tens of threads in parallel, GPUs are specialized for computationally intense, highly parallel operations, handling thousands of threads simultaneously. This makes them exceptionally efficient at tasks involving large-scale, complex data computations, such as matrix operations, simulations, and ML [122, 123]. By offloading heavy computational tasks from CPUs to the GPU, systems may achieve higher throughput and efficiency, reducing the time required to process large sets of data [121].

Key to GPU acceleration is the development of programming models and platforms that facilitate the use of GPU resources. Among these, NVIDIA's CUDA stands out as a valuable tool, allowing direct programming of NVIDIA GPUs using familiar languages like C/C++ [124]. CUDA enables developers to harness the parallel processing power of GPUs, optimizing the performance of applications and making sophisticated computations more accessible. The NVIDIA Jetson Nano is a highly accessible and practical embedded board for developers, researchers, and hobbyists to develop software applications that utilize GPU acceleration with CUDA [125]. Despite being the least powerful option in NVIDIA's embedded board lineup [126], it was selected for the development of this thesis due to its affordability and practicality. Moreover, using the Jetson Nano demonstrates that the system can accomplish its purpose even with the most economical and least powerful NVIDIA board, proving the efficiency and scalability of the proposed approach. In [127], the substantial performance and efficiency differences between the NVIDIA Jetson Nano and NVIDIA's latest developer kit, the Jetson Orin Nano, are quickly apparent.

2.2.2 NVIDIA Jetson Nano board

The NVIDIA® Jetson Nano™ [125] is a small, powerful computer designed specifically for AI development, embedded systems, and edge applications. Launched as part of NVIDIA's Jetson family [126], the Jetson Nano brings the power of modern AI to a compact and accessible form factor, making it ideal for developers,

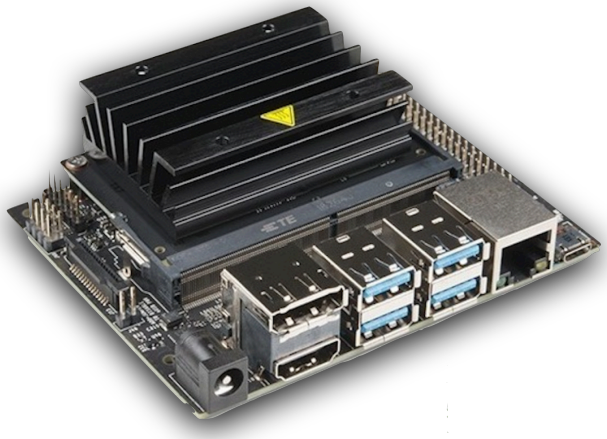


Figure 14: **NVIDIA Jetson Nano Developer Kit.** The developer kit provides a development environment by leveraging the standalone Jetson Nano computing module, and adding necessary peripherals and interfaces to facilitate development and prototyping.

researchers, and hobbyists. Even as the most modestly capable module in NVIDIA's Jetson series, it stands out due to its affordability, low power consumption, and capability to run complex AI solutions.

The technical specifications of the Jetson Nano underscore its suitability for AI, machine learning, and systems involving complex mathematical operations. The board features a 64-bit quad-core ARM Cortex-A57 CPU, which is efficient for general-purpose processing tasks. The integrated 128-core Maxwell GPU is the real highlight, providing the parallel processing power necessary for heavy computations, achieving up to 472 GFLOPS (giga floating point operations per second) of AI performance. It includes 4GB 64-bit LPDDR4 RAM, sufficient for many AI models and applications. Storage is managed via a microSD card slot. The kit offers a variety of connectivity options, including three USB 2.0 ports and one USB 3.0 port, an HDMI 2.0 output, a 12 lanes MIPI CSI-2 camera connector, a 40-pin GPIO header (including UART, SPI, I2S and I2C communication protocols, and other general purpose pins), and it features a Gigabit Ethernet port. While GPUs are generally known for their increased power consumption due to their intensive computational capabilities, the Nano is designed to operate efficiently with a power draw of approximately 5W to 10W. This makes it a highly efficient choice for embedded and edge applications, where power availability might be limited. The Jetson Nano is also supported by NVIDIA JetPack, which includes Board Support Packages (BSP), a Linux Operating System (OS), and, as mentioned, CUDA. It also includes additional software libraries for deep learning, computer vision, and multimedia processing, like cuDNN and TensorRT.

The NVIDIA Jetson Nano handles applications requiring real-time processing and decision-making capabilities effectively. It has been employed in various research studies, demonstrating its versatility and effectiveness in real applications, including in agriculture [128], mobile robotics [129], medical and social applications [130, 131], and the automotive sector [132–135].

2.2.3 CUDA programming

CUDA® [124] is a general-purpose computing platform and programming model for parallel computing that enables developers to use NVIDIA GPUs to execute programs written with C/C++, among many languages [84, 120]. By harnessing the power of such processing units, CUDA allows developers to solve many complex computational problems more efficiently than on a CPU.

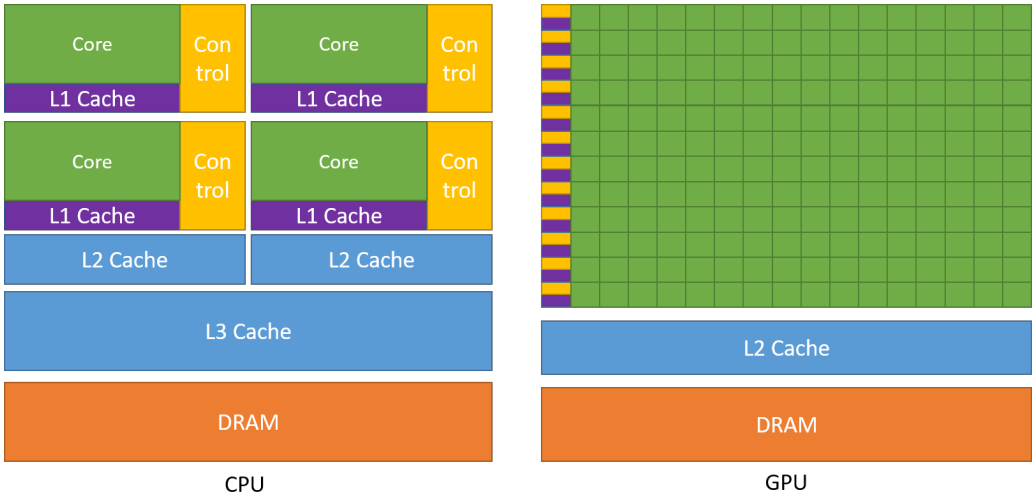


Figure 15: **Simplified architectural differences between a CPU (left) and a GPU (right).** CPUs typically have fewer, more complex cores optimized for sequential tasks along with significant control logic and large caches to manage complex tasks efficiently. In contrast, GPUs are designed with a large number of smaller, simpler cores that are optimized for parallel processing, making them highly effective for tasks involving large-scale data computations. Image retrieved from [81].

First presented in 2006 [84], CUDA was designed to provide a straightforward model for parallel computing, making it accessible for developers accustomed to traditional sequential programming languages [120]. Most programming languages were created for sequential execution. CUDA, however, retains this familiar sequential model while introducing a set of minimalist abstractions that allow developers to express parallelism. This approach enables programmers to concentrate on developing efficient parallel algorithms without needing to learn an entirely new language paradigm. CUDA is designed to create scalable parallel programs that can run across tens of thousands of concurrent threads in hundreds of cores (see Figure 15). This scalability is a key feature, as it allows a compiled CUDA program to execute efficiently on any GPU, leveraging more parallelism on GPUs with higher numbers of processor cores and threads.

A CUDA program is organized into two main parts [120, 136]: a host program, where serial portions are executed on the CPU (host), and one or more kernels, i.e., the parallel portions, that run on the GPU (device). The host program initiates kernel execution on the GPU. A kernel is a function written in CUDA C/C++ that runs multiple instances of itself in parallel, with each instance known as a thread. To better understand the following explanations, please check Figure 16.

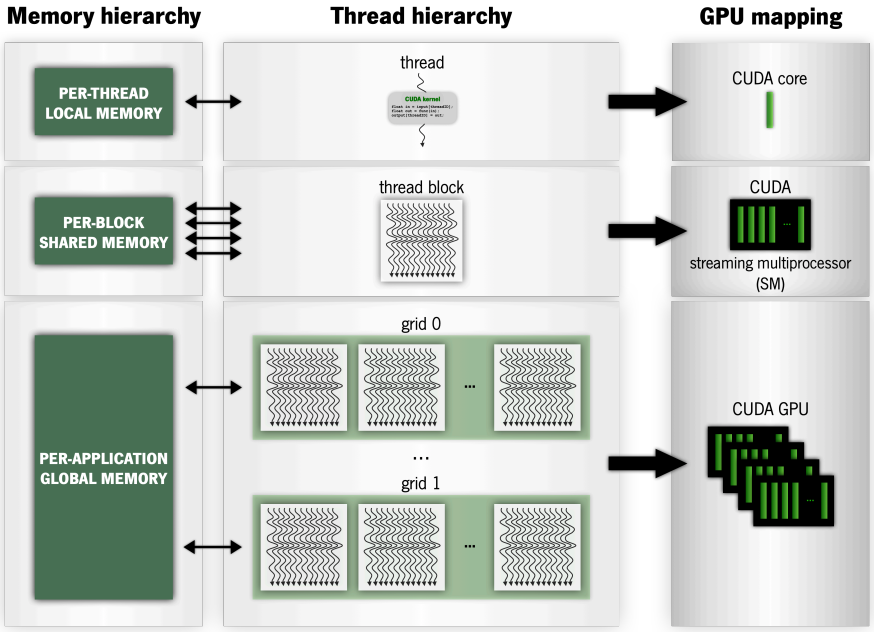


Figure 16: **The CUDA hierarchy, including the memory (left) and thread hierarchy (center), and the GPU mapping (right).** The CUDA hierarchy is comprised of threads (running a kernel) organized into blocks and grids. Grids are thus collections of blocks that execute the same kernel function. Various levels of memory are accessed at different speeds and scopes. The GPU mapping shows how threads are distributed across the GPU. For further details, please refer to the text.

Threads are the smallest execution unit in CUDA, running on CUDA cores. Each thread has private local memory, the fastest form of memory available, albeit in limited supply. There is also the largest and slowest memory, the global memory, which is accessible by all threads and persists for the duration of the application. Threads are grouped into blocks, containing each block a limited number of threads. Typically, the maximum number of threads per block is 1024. Within a block, threads can cooperate via high-speed shared memory and synchronize their execution to coordinate memory accesses. The blocks are further organized into grids, which are collections of blocks that execute the same kernel function and scale across the entire set of data. This hierarchical structure of threads, blocks, and grids allows for efficient parallel processing of large sets of data. Just for reference, the CUDA programming model also offers an optional level of hierarchy, the thread block clusters, consisting of groups of thread blocks able to share memory.

As demonstrated in Figure 16, the CUDA thread hierarchy is closely tied to the hardware architecture of the GPU, specifically the Streaming Multiprocessors (SMs). A GPU is built around an array of SMs, each containing several CUDA cores that execute threads. The more SMs and cores a GPU has, the higher its potential for parallel processing. The concepts of threads, blocks, and grids are thus essential for achieving scalability and efficient resource management. Blocks are designed to be independent of each other, allowing the CUDA program to scale across GPUs with different numbers of cores. This independence ensures that a kernel can utilize as many or as few blocks as available on the hardware, enabling it to scale up or down depending on the capabilities of the GPU. Additionally, the GPU can schedule blocks

independently, distributing them across the available SMs. This independent scheduling helps manage GPU resources more effectively, ensuring optimal hardware utilization.

Next, a CUDA C++ program is introduced to illustrate the basics of CUDA programming (see Listing 2.1). The example demonstrates the element-wise addition of two arrays, showcasing how a computation typically performed sequentially on a CPU can be accelerated by leveraging the parallel processing capabilities of CUDA on a GPU. CUDA files have the .cu file extension and are compiled using the NVIDIA CUDA compiler, the nvcc. As mentioned, a typical CUDA program includes sequential sections running on the CPU (host) and GPU (device) acceleration phases, where a kernel function is launched from the host to perform a specific task — in this case, the element-wise addition of two arrays in parallel. Once the GPU computation is complete, the program returns to the host and continues with sequential execution. This back-and-forth between the host and the device highlights how CUDA allows developers to offload computationally intensive tasks to the GPU while maintaining control flow on the host.

The host code in this CUDA program manages memory allocation, kernel execution, and result handling. It begins by allocating unified memory for the input and result arrays using cudaMallocManaged, which simplifies data management by making the memory accessible to both the CPU and GPU (alternatively, cudaMalloc can be used, which allocates memory only accessible on the device). The input arrays are then initialized on the host. The kernel complexAcceleration is launched with a specified number of blocks and threads per block, as defined in the <<< >>> syntax, to ensure all elements of the input arrays are covered. The cudaDeviceSynchronize ensures that the GPU has finished processing before the host accesses the results. Finally, the allocated memory is freed using cudaFree.

In CUDA, a kernel is a function that runs on the GPU and is executed by many threads in parallel. Kernel functions are declared with the ___global___ specifier, indicating that they are executed on the GPU but called from the host. In this example, the kernel function complexAcceleration performs the element-wise addition of two arrays by calling a ___device___ function, complexOperation, which simply adds two floating-point numbers. In CUDA, ___device___ functions run on the GPU and can only be called from other GPU code. These functions help modularize code, making it more readable and reusable. Although the complexOperation function is not necessary for this simple CUDA program, it is included here for demonstration purposes. Within the kernel, the idea is that each thread is assigned a specific element of the arrays, enabling the operation to be carried out simultaneously across all elements. Thus, the kernel calculates the index for each thread, which determines the specific array element it will process: thread one will process index one of the arrays, thread two will process index two of the arrays, and so on. Concretely, each thread gets its global index by computing the offset to the beginning of its block (blockIdx.x * blockDim.x) plus the thread's index within the block (threadIdx.x). Here, threadIdx.x represents the position of the thread within its block, blockIdx.x represents the position of the block within the grid and blockDim.x is the number of threads per block.

Listing 2.1: **CUDA C++ program, *simple_cuda_example.cu*.** Element-wise addition of two arrays using parallel processing on a GPU.

```
#include <iostream>
#include <cuda.h>
#include <cuda_runtime.h>

/* Device function to perform the sum of two values */
__device__ float complexOperation(float val1, float val2)
    return val1 + val2;

/* Kernel function to add the elements of two arrays */
__global__ void complexAcceleration(int n, float *arr1, float *arr2, float *res)
{
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    if(index < n)
        res[index] = complexOperation(arr1[index], arr2[index]);
}

int main(void)
{
    /* Aprox. 1 million elements */
    int N = 1<<20;
    /* Declare input and output arrays */
    float *in1, *in2, *res;

    /* Allocate memory accessible from CPU and GPU */
    cudaMallocManaged(&in1, N*sizeof(float));
    cudaMallocManaged(&in2, N*sizeof(float));
    cudaMallocManaged(&res, N*sizeof(float));

    /* Initialize input arrays in1 and in2 from the host side */
    for (int i = 0; i < N; i++)
    {
        in1[i] = 1.0f;
        in2[i] = 2.0f;
    }

    /* Define the number of CUDA threads per block,
    and set the number of blocks */
    int blockSize = 256;
    int numBlocks = (N + blockSize - 1) / blockSize;
    /* Launch CUDA kernel */
    complexAcceleration<<<numBlocks, blockSize>>>(N, in1, in2, res);

    /* Wait for GPU device to finish the job (blocking function) */
    cudaDeviceSynchronize();

    /* Print the results (first ten elements) */
    for (int i = 0; i < 10; i++)
        std::cout << in1[i] << "+" << in2[i] << "=" << res[i] << std::endl;

    /* Free memory */
    cudaFree(in1);
    cudaFree(in2);
    cudaFree(res);

    return 0;
}
```

2.3 Robot Operating System 2

This section introduces the ROS, a widely recognized framework within the robotics community. Although initially designed for robotic applications, ROS is also increasingly utilized in automotive contexts [137] due to the growing complexity of intelligent vehicular systems. These systems require robust frameworks for communication, sensor integration, and real-time data processing — capabilities that ROS provides. The section starts with an overview, including the history and motivations behind the creation of ROS, the evolution from ROS to ROS 2, and applications of ROS 2 across multiple domains. Following that, the software architecture of ROS 2 is described, core concepts explained, and key algorithms and utilities outlined.

2.3.1 Overview

Building a robotic system requires the integration of various subsystems, including sensing, actuation, decision-making, and communication. Ideally, each component must function together seamlessly, allowing for modularity, scalability, and code reuse. Traditionally, this integration posed significant challenges, as developers often had to create custom software for each new robotic solution in a non-standardized manner, tailoring systems to their individual preferences. This resulted in unnecessary multiplication of effort, commonly referred to as “reinventing the wheel”, where developers repeatedly wrote similar code for typical tasks such as perception, path planning, and sensor fusion.

To tackle these issues, the Robot Operating System was initially developed at Stanford’s Artificial Intelligence Laboratory and later popularized by the robotics incubator Willow Garage in 2008-2009. As outlined in the foundational paper by Quigley et al. [138], ROS is not an operating system in the traditional sense but rather a middleware that provides a structured communication layer above the host OS. As robotic systems grew in complexity, ROS has evolved into a comprehensive open-source platform, offering the building blocks necessary for developing robotic applications [139, 140]. As a middleware, ROS provides a standardized approach to software integration and development, enabling large-scale collaboration, accelerating research, and simplifying the addition of new functionalities through its distributed design principles. Furthermore, ROS includes an extensive set of software libraries, ranging from device drivers to state-of-the-art algorithms, as well as specialized tools for visualization, debugging, and data logging. The ROS project is also supported by a large, global community, with developers actively sharing algorithms and strategies to continuously enrich the platform [140].

Although ROS played a critical role in the early development of modern robotics, it was primarily designed as a research tool and lacked the robustness needed for production environments. As robotics moved beyond academic research into commercial applications, such as autonomous vehicles and industrial robotics, some limitations — particularly in real-time performance, security, scalability, and flexibility — became more apparent [141].

To address the aforementioned limitations, ROS was redeveloped from the ground up, resulting in ROS 2, the second generation of the framework. ROS 2 is based on the Data Distribution Service (DDS), a communication protocol widely used in critical systems like aerospace and defense. This shift to DDS enabled ROS 2 to provide real-time computing, enhanced security, and improved support for embedded systems [141]. Moreover, ROS 2 aims to support production-ready applications. Another major shift with ROS 2 was the expanded support for platforms beyond Linux, including Windows, macOS, and various embedded platforms (see micro-ROS [142]), making it more versatile for developers in different domains. The ROS community remains strong and diverse, with contributors ranging from students and hobbyists to multinational corporations and government agencies [140]. Similar to its predecessor, ROS 2 is released in distributions (also known as *distros*), which are stable collections of the ROS 2 core libraries, tools, and packages [143]. Each distribution is supported for a set period, allowing developers to work within a reliable and consistent framework.

ROS 2 is a platform designed with four primary architectural principles to meet the demands of modern robotics [141]:

- Distribution: different nodes (software components) can operate on a single device or across multiple devices. This distributed design is crucial for scaling robotic systems in complex environments, allowing various system components to work independently while maintaining communication through a decentralized network.
- Abstraction: ROS 2 emphasizes abstracting implementation details to simplify development and integration. Developers define interface specifications (messages) that govern communication between components, enabling them to focus, for instance, on high-level application behavior rather than hardware specifics. This abstraction encourages code reuse across projects and ensures interoperability within the ROS ecosystem.
- Asynchrony: ROS 2 supports asynchronous communication between components, enabling different parts of a system to function independently of each other's timing. This event-driven approach is essential for systems with components, such as sensors and actuators, that operate at varying frequencies.
- Modularity: ROS 2 is highly modular, enhancing system flexibility by breaking down complex software systems into discrete, reusable software components. Each component can be developed, tested, and deployed independently, enabling easier updates, scalability, and seamless integration.

Since its inception, ROS 2 has been adopted across various industries due to its flexibility, scalability, and support for real-time communication. Its distributed architecture and modular design make it suitable for a wide range of applications, from agriculture to aerospace. Next, some works and case studies across key domains will be presented

In agriculture, ROS 2 is used in autonomous systems, such as in a recent study where ROS 2 controlled a gantry robot for precision weeding. The system integrates vision and machine learning for precise weed detection and removal [144].

In healthcare, ROS 2 has been utilized in the PeTRA project [145], which focuses on autonomous hospital transport systems. Additionally, ROS 2 is applied in the SlicerROS2 module for image-guided robotic interventions, facilitating integration between surgical robots and medical imaging systems for improved procedural precision [146]. In another application, a 6G-enabled telerobotic examination system employs ROS 2 for distributed control, enabling remote real-time robotic surgeries [147].

Space exploration presents unique challenges, including the need for autonomous operations, real-time decision-making, and long-term reliability in extreme environments. NASA has adopted ROS technologies for space robotics, including projects like Robonaut 2 [148] and Astrobee [149, 150], and continues to explore ROS 2 for its autonomous systems in space exploration [151]. SpaceROS, an extension of ROS 2, is being developed to meet the demanding requirements of space robotics, aligning ROS software with aerospace mission and safety assurance standards [152–155].

On Earth, ROS 2 has been deployed across various environments [141], including Ghost Robotics' quadruped robots for rough terrain on land, Mission Robotics' marine robots for underwater exploration, and Auterion's aerial drones operating in unstructured, hazardous spaces.

In the automotive sector, research on autonomous driving is predominantly being conducted using ROS 2 [156]. The authors in [156] propose an architecture for interoperability between ROS 2 and Adaptive AUTOSAR — a flexible, high-performance software platform standard for dynamic, service-oriented applications in modern vehicles, developed by the AUTOSAR Consortium. Another prominent example is the Autoware project [137], which leverages ROS 2 for autonomous driving functionalities, from perception to control, with case studies in autonomous shuttles and valet parking [157]. Additionally, Apex.AI has achieved ISO 26262 certification for its ROS 2-based software, meeting functional safety standards required for autonomous vehicles [141].

2.3.2 Architecture

The ROS 2 software architecture is organized into multiple abstraction layers [141, 158]: the Application Layer, the ROS Client Layer, the ROS Middleware Layer, and the underlying Operating System Layer. The structure is illustrated in Figure 17.

The Application Layer is where developers interact with the ROS 2 framework to implement and test their custom applications. At this level, they develop nodes that work together to perform tasks such as data processing, communication, and control. ROS 2 allows developers to distribute nodes across multiple machines or leverage cloud resources, enabling large-scale deployments. It can also connect to cloud

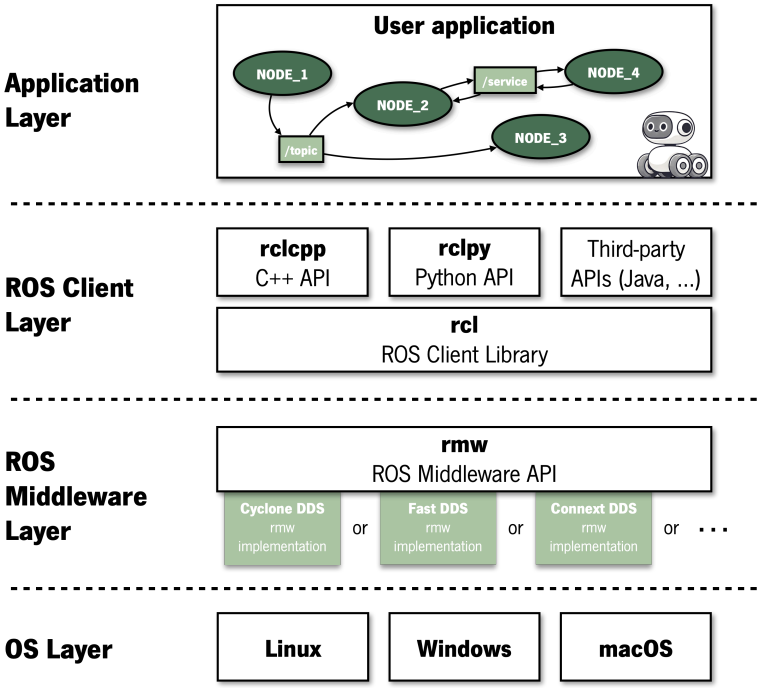


Figure 17: **High-level ROS 2 software architecture.** This diagram illustrates the main abstraction layers: Application Layer, ROS Client Layer, ROS Middleware Layer, and OS Layer.

platforms such as AWS (Amazon Web Services), which further extend the system's capabilities for cloud-based robotics. Below the Application Layer, lies the ROS Client Layer, which provides language-specific APIs, such as rclcpp for C++ and rclpy for Python, enabling developers to write the nodes in various programming languages. This layer relies on the ROS Client Library, the rcl, a C-based library that provides core ROS 2 functionalities (e.g., node creation and management, set communication between nodes) and serves as a middleware-agnostic foundation, allowing language-specific APIs to interact seamlessly with the ROS middleware. In the ROS Middleware Layer lies the ROS Middleware (rmw), which serves as an abstraction layer for different middleware implementations. Middleware options like Cyclone DDS, Fast DDS, and Connext DDS are commonly used for inter-process communication, handling discovery (locating nodes on the network), message serialization, and QoS (quality of service) settings. At the lowest level, the OS Layer supports various platforms, including Linux, Windows, and macOS, ensuring cross-platform compatibility.

2.3.3 Core concepts

The ROS 2 framework is built on several core concepts that enable modularity, reusability, and structured communication between different components of a robotic system [141, 159]. Understanding these foundational elements is essential for developing, configuring, and managing any ROS 2-based solution.

A ROS 2-based system is composed of packages, which are the fundamental organizational units that contain all resources necessary to implement a specific functionality. Each package may include nodes,

configuration files, scripts, and libraries. By grouping related components together, packages make it easy to manage complex projects and share functionality between different systems.

The fundamental computational units are called nodes [160]. Each node is responsible for a specific task, such as reading data from sensors, processing inputs, or controlling actuators. When a node is started, it advertises its presence to other nodes on the network within the same ROS domain, allowing appropriate connections to be established. Nodes communicate using three main interfaces: topics, services, and actions. To optimize performance, ROS 2 supports node composition, which allows multiple nodes to be combined within a single process, reducing inter-process communication overhead. Furthermore, intra-process communication facilitates efficient data exchange between nodes in the same process, minimizing latency and improving resource utilization. Each interface type is suited for different communication patterns depending on the nature of the task. All three interfaces rely on structured message files — respectively, .msg, .srv, and .action files — to define the specific fields for data exchange, ensuring that nodes can interpret and process the communicated information consistently. Refer to the diagram in Figure 18 for a clearer understanding of how communication patterns are utilized in this simplified patrol robot system.

Topics [161] facilitate asynchronous, one-to-many communication, making them ideal for streaming continuous data like sensor readings. In our example, the Intruders Detector Node publishes messages to the /intruder_alert topic whenever an intruder is detected. The UI Node subscribes to this topic,

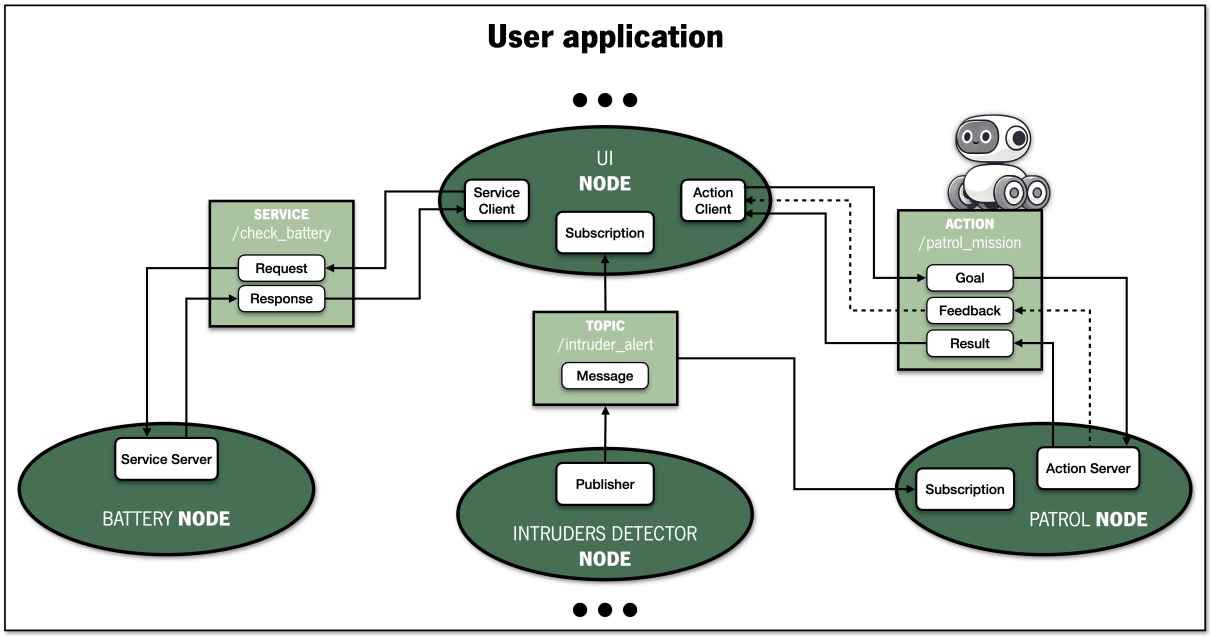


Figure 18: **Example of a simplified system architecture for a patrol robot.** It shows the communication patterns between nodes using topics, services, and actions. The diagram depicts a minimal setup where the User Interface (UI) Node controls the patrol mission and monitors alerts, while additional nodes for sensing and actuation would typically be included in a more comprehensive system.

receiving alerts to react accordingly. This pattern allows multiple nodes to receive updates without directly querying the publisher, making it a non-blocking way to share information.

Services [162] follow a synchronous, request-response pattern, which is useful for tasks that need an immediate reply. For example, the $\mathrm{UI\ Node}$ sends a request to the $\mathrm{Battery\ Node}$ using the $/\mathrm{check_battery}$ service. This synchronous communication ensures that the client node can wait for the result before proceeding with its next operation. Moreover, ROS 2 also provides an option for a non-blocking service call.

Actions [163] are used for long-running tasks that require continuous feedback and may need to be interrupted. In our simplified patrol robot system, the $\mathrm{UI\ Node}$ sends a goal to the $\mathrm{Patrol\ Node}$ using the $/\mathrm{patrol_mission}$ action to start a patrol mission. The action interface enables the $\mathrm{UI\ Node}$ to receive feedback (e.g., current position) and the result of the mission (e.g., mission completed or interrupted). This makes actions particularly suited for operations that involve sequences of steps, such as navigating through a series of waypoints or performing a complex robotic manipulation.

In addition to the main communication interfaces like topics, services, and actions, other core concepts in ROS 2 further enhance node functionality and system configuration. One such concept is timers, which are used to schedule periodic tasks within nodes, such as publishing sensor data or executing control loops at fixed intervals. Timers allow nodes to perform repetitive operations without blocking other activities. Another key concept is the use of parameters, which provide a flexible way to configure node behavior dynamically [164]. Parameters can define settings and can be set at launch or modified during runtime. This ability to adjust parameters without changing the source code enables nodes to adapt to varying conditions or user inputs. Finally, launch files [165] are used to automate the startup of multiple nodes with predefined configurations. Written in XML, YAML, or Python, launch files allow developers to automate node launches, set parameters, and define dependencies. This is particularly useful for complex systems, where a single launch file can bring up an entire system with the necessary settings, ensuring consistent configurations and reducing setup time.

2.3.4 Algorithms and tools

ROS 2 provides a wide range of built-in algorithms commonly used in robotics, covering areas including perception, Simultaneous Localization and Mapping (SLAM), and motion planning [141]. Additionally, ROS 2 offers various tools and utilities that facilitate development, debugging, and visualization [141]. Key tools include RViz, a 3D visualization tool used for displaying sensor data, robot states, and Unified Robot Description Format (URDF) models; rqt, a set of Graphical User Interface (GUI) tools for monitoring and managing a ROS-based system; and rosbag, a utility for recording and replaying ROS messages. Gazebo is a powerful simulation tool that can be integrated with ROS for testing robot behavior in realistic environments, enabling developers to validate algorithms before deployment. Another crucial tool is TF2,

which manages the transformation between different coordinate frames, allowing developers to track the spatial relationships between various components of a robot. To further aid system introspection, ROS 2 includes command-line tools [166] like $\mathrm{ros2\ topic}$, $\mathrm{ros2\ service}$, and $\mathrm{ros2\ node}$, which allow developers to inspect and interact with topics, services, and nodes directly from the terminal.

2.4 Flutter

This section briefly presents Flutter, a versatile framework used to develop a tailor-made mobile application for data collection purposes, specifically, traveling routines information. The section begins with a concise overview, emphasizing its key features. Following this, Firebase, a platform that allows extending the capabilities of mobile application development, is introduced.

2.4.1 Overview

Flutter is an open-source UI toolkit developed by Google, that offers a unified approach to application development across multiple platforms [168].

Launched in 2017, Flutter enables developers to create visually appealing and highly customized applications for mobile (Android and iOS), web, desktop (Linux, macOS, and Windows), and embedded environments from a single codebase [167]. This unified development approach not only simplifies the development process but also reduces maintenance overhead, as developers need to manage only one codebase for deploying applications across various devices with all sorts of screen sizes (see Figure 19). Since its release, the adoption of Flutter has been steadily growing, with increasing usage observed not only in scientific research [169], but also in renowned businesses such as BMW, Toyota, Sony, and eBay [170–172].

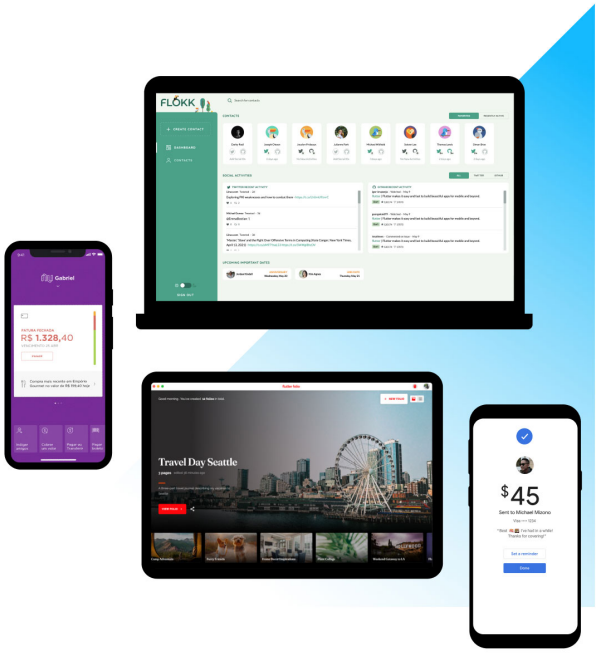


Figure 19: **UI examples developed with the Flutter toolkit.** Applications across various environments and screen sizes. Image retrieved from [167].

The following summarizes the key features and advantages of using Flutter [167]:

- Cross-platform support with consistent UI and application logic: develop applications for multiple platforms (iOS, Android, web, desktop, embedded) using a single codebase. Not only does this make the development process more efficient, but it also ensures that the UI and application logic remain consistent across all platforms.

- Fast development: the Hot Reload feature in Flutter allows developers to instantly view changes made to the code, providing instant feedback and increasing productivity.

- Expressive UI and highly customizable UI design: Flutter provides a highly customizable and creative UI design. It offers a widget system for development with a rich set of customizable widgets, making it easier for developers to create visually engaging and highly interactive user interfaces.

- Native performance and access to native features: Flutter compiles to native code, resulting in high performance and, e.g., smooth animations, across various platforms. Furthermore, it provides access to native features like storage, camera, location, and network, to name a few.

- Built-in Material Design and Cupertino Widgets: Flutter includes built-in widgets that follow Material Design guidelines [173], for Android, and Cupertino design guidelines [174], for iOS, ensuring consistent and platform-specific UI experiences.

- Rich ecosystem: Flutter has a growing ecosystem of packages, plugins, and tools contributed by the community.

2.4.2 Architecture

Flutter's architecture revolves around its layered design, comprising three main layers: the framework, the engine, and the embedder [175].

The framework, written in Dart, provides a comprehensive set of libraries and widgets for building UI components, handling user input, and managing app state. Dart [176] is a versatile, general-purpose programming language developed by Google, which supports many programming paradigms (e.g., object-oriented, declarative, and imperative programming paradigms). It is known for its simplicity and flexibility, making it suitable for a wide range of development tasks. UI developers compose their applications, also in Dart, on top of the Flutter framework. The framework is characterized by its reactive and declarative nature, where the user interface is explicitly decoupled from its underlying state. This means that developers solely implement the UI description, leaving the framework responsible for dynamically creating the user interface as appropriate, and/or updating it at runtime when the application state changes [175].

Beneath the framework lies the Flutter engine layer, a portable runtime (mostly written in C++) for executing Flutter applications, which includes a fast and efficient rendering engine. This engine communicates with the host platform through a platform-specific embedding layer, allowing Flutter to integrate seamlessly with the underlying services of the OS and APIs.

2.4.3 Building user interfaces

The central concept of developing with Flutter is that user interfaces are constructed using widgets [177]. A widget describes the visual properties that will be displayed to the user based on its configuration and any changes in its state. Widgets are therefore regarded as building blocks that can be assembled and customized to create a functional and elegant application.

Flutter offers a vast catalog of visual, structural, platform-specific, and interactive pre-built widgets [173, 174, 178–182]. It is also possible to find Flutter packages, which are open-source libraries of code created by the community that can be incorporated into projects with minimal effort.

As stated, everything inside a Flutter application is a widget. Therefore, when creating a user interface, the developer builds widgets upon widgets, forming what is known as the widget tree [177]. The widget tree represents the anatomy of the application, depicting the hierarchical arrangement of widgets within the application (see Figure 20). This hierarchical organization allows for the composition of complex UIs by nesting widgets within one another (with parent widgets containing child widgets), ultimately culminating in the final user interface displayed to the user. The structure of the code closely reflects the widget tree, with each nested widget in the code corresponding to a node in the widget tree.

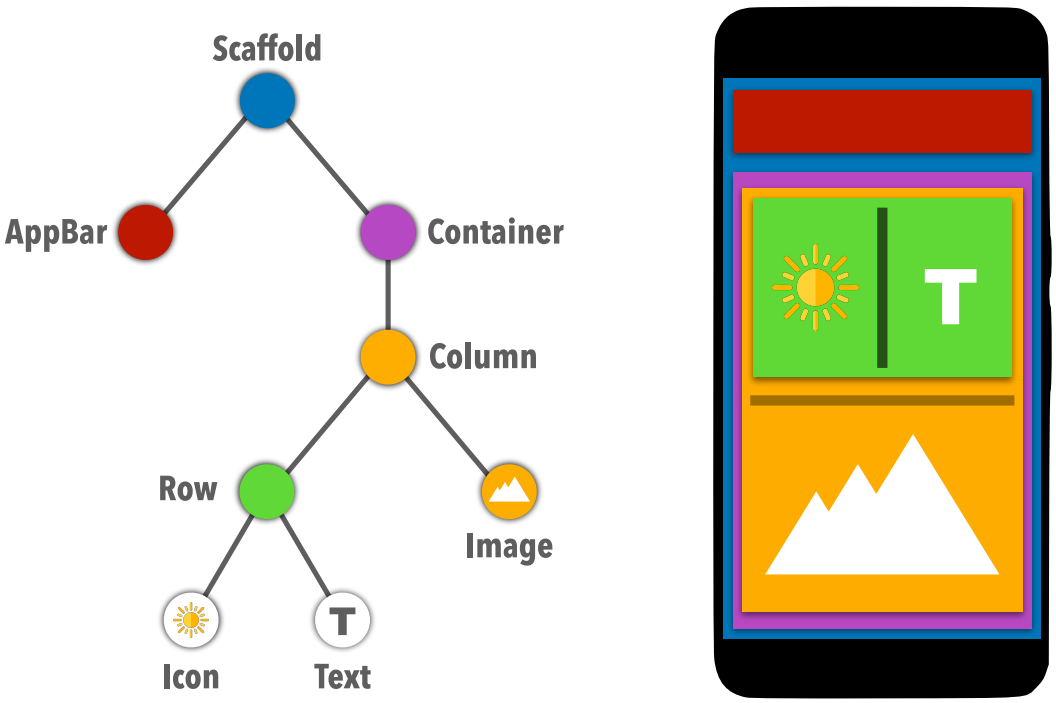


Figure 20: **Anatomy of a Flutter application.** On the left, one can observe the widget tree of the example application. The root widget consists of a black screen, represented by a Scaffold widget. Inside the Scaffold, an AppBar widget and a Container widget are added. The Container contains a Column widget, which, at the top, includes a Row widget with two children: an Icon and Text. At the bottom of the Column widget, an Image widget is added. On the right panel, it is shown the outline of the resulting UI.

Flutter defines two types of widgets [183]: stateless and stateful. Stateless widgets maintain a static appearance, as their graphical properties are determined solely by their initial configuration and do not change over time (e.g., a simple icon). Stateful widgets are dynamically altered and reconfigured based on user interactions or changes in the application's data, allowing for dynamic updates to their appearance and behavior (e.g., a checkbox). When a widget's state changes, its description is rebuilt, and the framework compares the new description with the previous one to determine the minimal rendering changes necessary for transitioning between states.

2.4.4 Integration with Google Firebase

Google Firebase is a versatile platform provided by Google for developing mobile and web applications [184]. It offers developers a unified platform for building, improving, and growing their applications, enabling them to focus on creating innovative user experiences rather than managing infrastructure.

Originally launched as Firebase in 2011 and acquired by Google in 2014, Firebase serves as a backend service provider, offering various solutions and functionalities to support the development of modern applications. Some of the key products provided include [185]:

- Real-time Database: cloud-hosted NoSQL database that allows developers to store and sync data in real time across clients.
- Authentication: authentication services that allow developers to easily integrate secure user authentication into their applications using email/password, phone number, and more.
- Cloud Storage: cloud storage solutions for storing and serving user-generated content, such as images, videos, and files.
- Firebase Extensions: pre-packaged solutions for common backend tasks such as online payments or resizing documents, allowing developers to extend their applications' capabilities with ease.
- Firebase ML: allows developers to easily integrate machine learning capabilities into their applications, enabling features such as image recognition, text recognition, language translation, and more.

Google Firebase can be seamlessly integrated with Flutter to provide backend services for mobile application development. Developers can access Firebase functionalities within their Flutter applications through the various plugins available, each corresponding to a Firebase product mentioned above.

Chapter 3

A data recording mobile application to create datasets of vehicle users' routines

A data recording mobile application to create datasets of vehicle users' routines[1]

Pedro Guimarães, Flora Ferreira, Ana Carolina Silva, Wolfram Erlhagen,
Sérgio Monteiro, and Estela Bicho

Article published in *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*
April 2022
DOI: 10.1109/ICARSC55462.2022.9784770
Link: doi.org/10.1109/ICARSC55462.2022.9784770

Author contribution statement: Pedro Guimarães designed and implemented the mobile application, and wrote the manuscript. Flora Ferreira supervised the work. Ana Carolina Silva studied the data requirements for the application. Wolfram Erlhagen reviewed the manuscript. Sérgio Monteiro supervised the work and provided the resources needed for its development. Estela Bicho provided the resources needed for the development of the work and reviewed the manuscript.

[1] © 2022 IEEE. Reprinted, with permission, from P. Guimaraes, F. Ferreira, A. C. Silva, W. Erlhagen, S. Monteiro, and E. Bicho. A Data Recording Mobile Application to Create Datasets of Vehicle Users' Routines. In *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. April 2022.

A Data Recording Mobile Application to Create Datasets of Vehicle Users' Routines

Pedro Guimarães[1,3], Flora Ferreira[2], Ana Carolina Silva[3], Wolfram Erlhagen[2], Sérgio Monteiro[1], and Estela Bicho[1]

[1]Department of Industrial Electronics, University of Minho
[2]Department of Mathematics, University of Minho
[3]Bosch Car Multimedia Portugal

Abstract

One of today's automotive research focus is the development of vehicles for the future, with their own intelligence, aware of their occupants, able to give support to its users, striving for natural and efficient interaction, and giving rise to the concept of the cognitive vehicle. Furthermore, truly adaptive intelligence can be achieved with assistance systems capable of adapting to different drivers. Vehicles with the potential to learn users' routines and preferences, and make decisions to prepare the next trip (e.g., manage comfort; check if the usual objects are being transported), is a concrete example that has started gaining attention. To accomplish such a challenge, data-driven approaches are required. Hence, datasets that include information on different vehicle occupants' habits and preferences are essential for building cognitive computational models. To the best of our knowledge, there is no tool capable of obtaining these data in a real-world situation. Thus, this work proposes a mobile application capable of collecting real data and creating datasets about: (1) where and when the driver and passengers get in and out of the vehicle; (2) objects brought/taken by the occupants; and (3) vehicle settings preferences. Collected data are internally structured in files that can be uploaded anytime. The developed mobile application can be described as an easy-to-use, flexible, and free-of-charge solution for collecting data on the travel routines of vehicle occupants, to support the development of personalized assistance systems.

Keywords—mobile application, flutter, data acquisition, routines, intelligent vehicles

3.1 Introduction

The future is heading towards the presence of robots on the road [186]. There was a time when vehicles were merely mechanical machines, but progressively, one can witness an increase in the number of sensors, controllers, pre-programmed algorithms, and intelligent software equipped on them [2, 5, 187]. In this sense, a vehicle, much like a robot, can also be characterized as an electro-mechanical entity capable of making autonomous or pre-programmed decisions to assist its occupants.

Recently, there has been much progress in the personalization of assistance systems in vehicles, so that they become more adaptable to their drivers [2]. This means that vehicles may provide more effective and comfortable assistance by implicitly considering their users' individual characteristics and preferences. As an outcome, assistance systems may become more acceptable, trustworthy, and personal [2]. Consistent with this level of personalization, a recent survey [14] found a strong positive public interest in the idea of having a vehicle with cognitive and social behavior, capable of learning and assisting users in their everyday traveling habits. More precisely, provide the vehicle the ability to learn drivers' routines and preferences (such as the usual destination, routine timestamps, regular driver and passengers, and their items), so that it can, autonomously, manage comfort within the cockpit and assist its occupants by issuing suggestions/alerts. In light of this, further research on the subject is promising. It may focus on the development of prototypes with such cognitive aspects, simulator-based studies, and on-road experiments [14]. For such, and even for the entire ITS, data-driven approaches are typically adopted, entailing the use of datasets at various stages of development (e.g., for training, testing, simulating, and tuning) [2, 188, 189].

As far as our knowledge goes, although there is a plethora of datasets featuring a wide range of driving aspects such as trajectory data, traffic, environment perception, driver intention and state, and much more (see [2, 189]), one cannot find datasets incorporating routines of vehicles occupants, covering aspects like routine timestamps, drivers, passengers, and their personal objects. Indeed, according to Yi and colleagues [2], personal data collection is the foundation of personalized services, and unique IDs should be considered to distinguish different drivers. However, how to implement personal data collecting is still a significant problem in personalized driving assistance research [2].

A manual process for creating driving routine datasets, e.g., through questionnaires, appears to be a more straightforward option. Yet, it can be very time-consuming, and the simple act of attempting to recreate different real driving routines, for different days, for different drivers can introduce inconsistencies and incongruences. Simulators are a second alternative, although the artificial nature of the environment can result in datasets that are far from reality. A third possibility is to equip vehicles with recording setups, including, e.g., cameras and GPS. This strategy can be costly and susceptible to users' rejection because the setups would have to be installed on personal vehicles. Considering the previous points, it is known that the GPS technology is a standard feature in current mobile phones. It is also known that smartphones have become one of the essential objects that people carry with them [190]. Such proximity appoints phones as a valuable device to collect data for travel studies through passive acquisition, i.e., data automatically recorded without bothering users, and/or through the use of interactive mobile applications [191, 192].

Several apps that can record the geographic position, export recorded files, and offer more extensive and complex analytics can be found in Google Play and Apple App Store. However, there is no application capable of recording the routines of vehicle occupants. In this regard, the current work tackles this problem

by proposing a mobile application capable of collecting real data and creating datasets comprising journeys' trajectory data, drivers' objects carried in the vehicle, passengers' arrivals and departures, as well as their objects, and vehicle settings preferences.

The mobile software, entitled P29DR, was developed under the scope of the joint project UMinho and Bosch – "Easy Ride: Experience is everything". The project aimed to develop computational models with the ability to understand the routines of the drivers/users (e.g., *where does the driver intend to go? When does the driver intend to arrive at a particular destination? And for how long does the driver intend to stay there? Which passengers and objects enter and leave next?*) and issue predictions that can be useful to plan the trips and assist the driver and/or the passengers [46, 54, 193].

In Section 3.2, the design and interaction flow, the sort of information recorded, and how the data is arranged are presented. Section 3.3 depicts the usage principle of the P29DR mobile application. Lastly, the paper wraps up with conclusions and future work in Section 3.4. Throughout the explanations, safety and data security matters will be covered.

3.2 P29DR application

The developed data recording app is an easy-to-use, flexible, free, and effective tool for collecting data on vehicle occupants' travel routines (see Figure 21). The software enables the compilation of datasets, by recording real information about where and when the driver and passengers get in and out of the vehicle, objects brought/taken by the occupants, and vehicle settings preferences. Datasets with contextual and real data can be valuable for developing, simulating, tuning, training and evaluating personalized computational models and intelligent strategies for in-vehicle assistance systems.



Figure 21: **A driver using the data recording mobile application.**

The execution and interaction flow of the P29DR application, including its many pages and functionalities, will be discussed below. Furthermore, it will also be described the sort of data recorded and how it is arranged. It is worth mentioning that the software behaves the same way on both Android and iOS platforms.

3.2.1 Methods

The mobile application was developed using Flutter [167], an open-source UI Toolkit that makes it easy for developers to design and implement appealing interfaces for all sorts of screen sizes and devices. It comes with a large number of pre-built widgets that allow to easily lay out an application with rows, columns, and margins, for instance, as well as animations and other functionalities. It is also possible to find Flutter packages, open-source libraries of code created by the community that can be incorporated into projects with minimal effort. Widgets can be seen as building pieces that can be plugged together and customized, to create a functioning and elegant application that can be deployed on Android, iOS, Linux, Mac, Windows, and even embedded platforms [168].

Flutter applications are written in Dart, a general-purpose programming language developed by Google. It is considered a client-optimized language whose purpose is to provide the most productive programming language for multi-platform development, along with a flexible execution runtime platform [194].

The developed mobile program also employs the Google Firebase. Firebase is a platform backed by Google that offers services and a backend infrastructure to help accelerate and support application development [185]. More specifically, the application uses the Firebase Authentication – an end-to-end identity service for user authentication, sign-in, and onboarding – and the Firebase Cloud Storage – service for easily storing and handling the created datasets.

3.2.2 Design and user flow

It is essential to mention that the user of the application is always the person driving the vehicle (driver). Moreover, the user should start recording when entering the vehicle (beginning of a journey), finishing the recording when leaving the vehicle. Concerning safety, using a smartphone or similar gadget while driving is dangerous and not encouraged. Yet, the driver is typically required to come to a complete stop if someone or anything enters or leaves the vehicle. So, after the vehicle has been safely stopped, the driver can interact with the application and edit the states of passengers and objects. As to the vehicle settings, the user should record the temperature at the beginning of the journey (if the car is not at the proper temperature) and at the end of the recording session (when the vehicle is, ideally, at the desired temperature). The mobile application comprises nine pages, one of which, the Main page, has two states (nonrecording and recording modes). The user flow of each page is detailed next.

Welcome page. The Welcome page is the first to appear when the application is launched. It ensures that the location service and access to the user's location are enabled. Following the fulfillment of these prerequisites, if the user has already completed a registration, the execution proceeds to the Disclaimer page. Otherwise, the Registration page is displayed.

Registration page. Because a user only registers once in the application, the Registration page only appears once. The registration merely allows the research team to differentiate between app users and their respective recording data. In this regard, to avoid having users with the same username, the Authentication service from Google Firebase is used, granting the authentication and management of the users. To register in this manner, a connection to the internet is needed. Following a successful registration, the Disclaimer page will appear.

Disclaimer page. This page divulges safety information and warnings regarding the use of the application while driving. In that sense, it appears whenever the application starts. If the user agrees with the terms, the execution continues to the Main page (nonrecording mode).

Main page (nonrecording mode). On the Main page, in nonrecording mode, it is possible to add, remove, and check the passengers, as well as verify their current states. When a passenger is added to the list of passengers (or removed), the respective information is saved/updated (see New Passenger page). The passengers added (together with their states) are put into the passengers' list each time the app is launched. Editing the passengers' states is only enabled during recording. In nonrecording mode, it

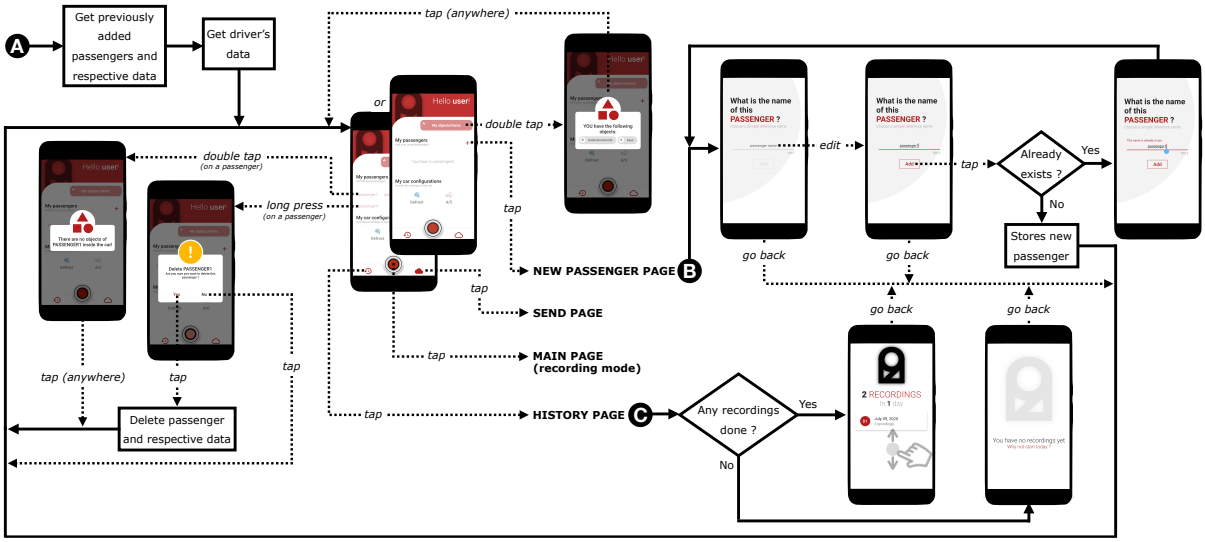


Figure 22: **Interaction flow of (A) the Main page (nonrecording mode), (B) the New Passenger page, and (C) the History page.** The purpose of superposed screens is to show multiple facets of a page at the same flow stage, which is dependent on the users' ongoing interaction with the app. Interactions on the touchscreen are represented by dashed lines.

is also possible to see if there are any objects (both from the driver and the passengers) in the vehicle. Furthermore, users can check their recording history (see History page) and upload recorded material to remote storage (see Send page). The interaction flow of the Main page in nonrecording mode is depicted in Figure 22A.

New Passenger page. This page must be accessed whenever the user intends to add a new person to the passengers list. It is only necessary to enter an arbitrary ID (a simple reference name) and press the *ADD* button. At this stage, the passenger's ID is saved on the device and added to the list of passengers. The application does not permit to addition of passengers using the same ID (nor using the same ID as the driver) as this may cause the user to get confused when recording. The interaction flow of the New Passenger page can be seen Figure 22B.

History page. The total number of recordings, total recording days, and the number of recordings per day, can all be seen on the History page. The interaction flow is depicted in Figure 22C.

Send page. This page allows the user to upload recording material from the device to a remote storage location. For this purpose, an online connection is required. The Google Firebase's services are again utilized, more concretely, the Cloud Storage, an object storage service with Google security, to upload and download files independent of network quality. When the data is successfully uploaded, it is automatically removed from the device to free up memory space. Because the data is maintained in remote storage, the research team can access the recordings data, which are organized by username and recording date (see Subsection 3.2.3, *Recording data description*).

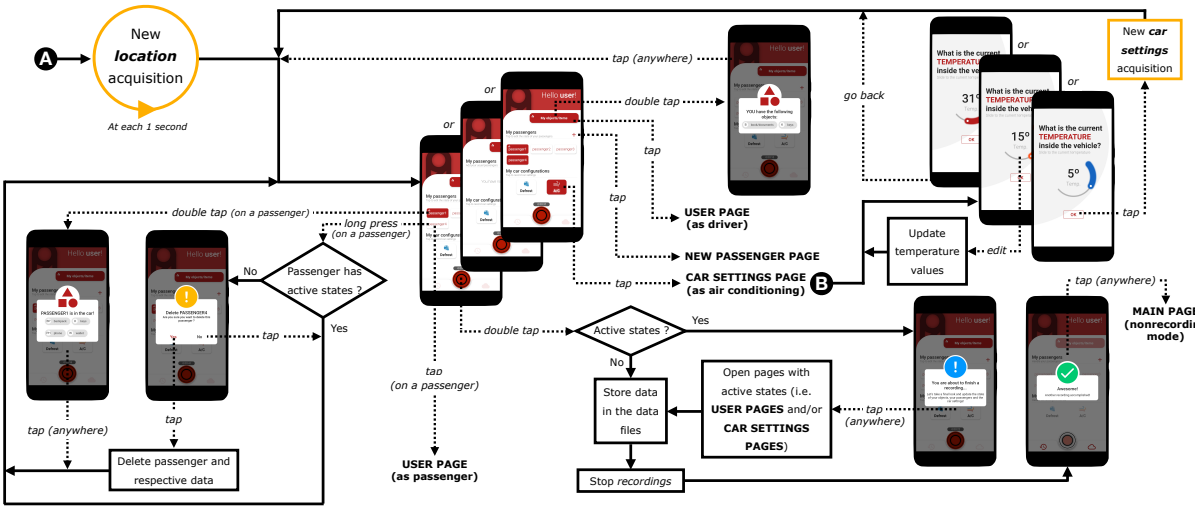


Figure 23: **Interaction flow of (A) the Main page (recording mode), and (B) the Car Settings page.** The purpose of superposed screens is to show multiple facets of a page at the same flow stage, which is dependent on the users' ongoing interaction with the app. Interactions on the touchscreen are represented by dashed lines.

Main page (recording mode). The interaction flow is represented in Figure 23A. When the app is on the Main page and in nonrecording mode (see Figure 22), the user can tap the large circular button at the bottom to start recording. The large circular button turns red when the recording session begins, and trajectory data (see Table 1) is acquired and saved, at a frequency of 1 Hz. By interacting with the mobile application, the user can now record passengers, objects, and vehicle settings data (see Users page and Car Settings page below). It is important to refer that the application can record while running in the background. Other applications can be used simultaneously, as long as the location service is not required. It is also possible to lock the smartphone while the application is recording. When the user arrives at a destination, the recording session can be finished by double-tapping the large circular button. However, if there are still active states (*inside* states) on the application (e.g., passengers or objects inside the car, and/or car settings turned on), the program forces the user to take a final look at them to assess whether the states should be updated. This is a preventive strategy used to mitigate usage errors that can be easily made when, for example, both the driver and the passengers leave the vehicle at the same location.

Car Settings page. The Car Settings page should be used whenever the air conditioning is turned on or off. It only takes the user to select the current temperature inside the vehicle and then press the *OK* button to perform a new acquisition from Table 4. It is possible to observe the interaction flow of the Car Settings page in Figure 23B.

Users page. Figure 24 illustrates the interaction flow of the Users page. This page, only unlocked when recording, is utilized when the driver has to alter the passengers' or the objects' states. Herein, the driver can change the presence of a passenger in the vehicle (getting in or getting out), as well as select or remove objects that enter or leave the vehicle during a journey. Because a recording is being conducted, it implies that the user is inside the vehicle, and so, concerning the driver, only objects can be managed. It is

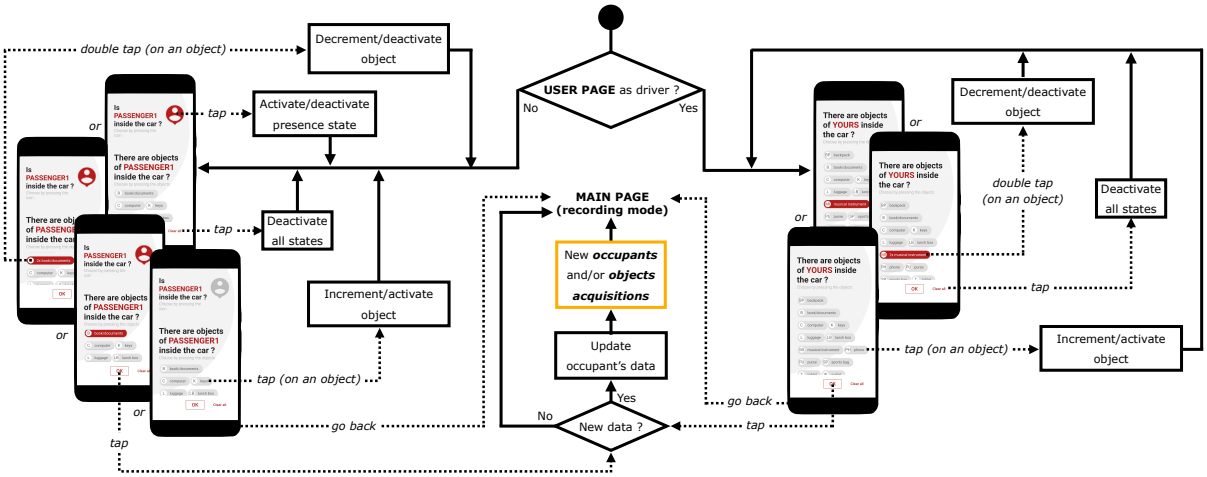


Figure 24: **Interaction flow of the Users page.** The purpose of superposed screens is to show multiple facets of a page at the same flow stage, which is dependent on the users' ongoing interaction with the app. Interactions on the touchscreen are represented by dashed lines.

worth stating that more than one object from the same category (e.g., two backpacks) can be added. When the *OK* button is pressed, new acquisitions are saved according to the alterations made: acquisitions from Table 2 when the states of passengers change; acquisitions from Table 3 when objects' states change; or both.

3.2.3 Recording data description

At the end of each recording session, recorded data is exported in spreadsheet files and saved on the phone, specifically in the application's documents (only accessible by the app). The spreadsheet format was chosen to facilitate data visualization and manipulation *a posteriori*. Thus, the acquired data is separated into four file collections: (a) Trajectory file; (b) Occupants file; (c) Objects file; and (d) Car Settings file. It is important to mention that the four data files, one from each collection, are created for each recording session. The recording files are organized by user ID and recording date.

Tables 1, 2, 3, and 4 depict the type of information that is collected. Each variable represents a column in the spreadsheet, and so, acquisitions yield rows of data from the correspondent collection.

Table 1: **Trajectory collection.** Geographic position data of a journey is acquired passively (no user intervention) at a predefined frequency (1 Hz).

Variables	Description
Driver	Username or ID of the user (driver)
Date	Date of acquisition's timestamp (yyyy-mm-dd)
Time	Time of acquisition's timestamp (hh:mm:ss)
Altitude	Altitude value of the GPS acquisition (meters)
Latitude	Latitude coordinate of the GPS acquisition (decimal degree)
Longitude	Longitude coordinate of the GPS acquisition (decimal degree)
Leg length	Distance between two consecutive GPS acquisitions (meters)
Leg time	Time between two consecutive GPS acquisitions (seconds)
Leg speed	The speed of GPS acquisitions (Km/h)

Table 2: **Occupants collection.** Data about the occupants of the vehicle. When, where, and which occupants entered and left the vehicle, as well as the driver.

Variables	Description
Occupant	Name or ID of the occupant
State	State of the occupant in the vehicle, i.e., inside or out
Total occupants	Current number of occupants in the vehicle
Date	Date of acquisition's timestamp (yyyy-mm-dd)
Time	Time of acquisition's timestamp (hh:mm:ss)
Altitude	Altitude value of the GPS acquisition (meters)
Latitude	Latitude coordinate of the GPS acquisition (decimal degree)
Longitude	Longitude coordinate of the GPS acquisition (decimal degree)

Table 3: **Objects collection.** Data about the occupants' objects. When, where, and which objects entered or left the vehicle and to whom they belong.

Variables	Description
Object	Type of object (e.g., backpack, phone, etc.)
Occupant	Name or ID of the occupant (to whom it belongs to)
State	State of the object in the vehicle, i.e., inside or out
Date	Date of acquisition's timestamp (yyyy-mm-dd)
Time	Time of acquisition's timestamp (hh:mm:ss)
Altitude	Altitude value of the GPS acquisition (meters)
Latitude	Latitude coordinate of the GPS acquisition (decimal degree)
Longitude	Longitude coordinate of the GPS acquisition (decimal degree)

Table 4: **Car settings collection.** Data about the air conditioning state.

Variables	Description
Driver	Username or ID of the user (driver)
Setting	Car setting edited. So far, only the air conditioning setting exists
State	State of the car setting, i.e., on or off
Temperature	Temperature of the car when setting is edited (Celsius)
Date	Date of acquisition's timestamp (yyyy-mm-dd)
Time	Time of acquisition's timestamp (hh:mm:ss)

3.3 Usage principle

An operational mobile application that collects real data and creates datasets of occupants' travel routines was accomplished. In the following points, the application's usage principle is outlined. Moreover, a demonstration video of the usage is available at the following link: https://youtu.be/dqbNTORxLbY.

Considering the moment immediately after the software is installed on a smartphone:

1. When using the app for the first time, it is recommended to launch it and complete the registration process. An Internet connection is necessary for this step.

2. After entering a valid username (or using an ID provided by the research team) and completing a successful registration, the user will be taken to the app's main page. From now on, whenever the app is launched, the user will not be prompted to register; instead, the app will take the user to the Main page (nonrecording mode).

3. After the two previous steps, the application will be ready to record every time the user travels. As stated earlier, the user should begin a recording session when entering the vehicle and finish recording when leaving the vehicle.

4. On the Main page, when not recording, the user can:
    a) add (and delete) their usual passengers from a list of passengers. To avoid adding passengers just before the recording sessions, the user should, preferably, add all possible passengers to the list of passengers during the application's first use.
    b) check to see if the user has any objects or passengers' objects in the vehicle.
    c) review their recording history.
    d) upload recorded data to the remote storage (if there is any data to upload). The recording content can be uploaded at the end of the day, week, or anytime the user prefers.
5. Before driving the vehicle and embarking on a journey, the user should start a recording session by pressing the Main page's large circular button. From that moment on, trajectory data is captured periodically. The user may:
    a) manage objects, i.e., register objects entering or leaving the vehicle.
    b) manage their passengers, i.e., change their state (inside or out, when a passenger enters or exits the vehicle, respectively).
    c) change the state of the air conditioning (on or off). Again, for the sake of safety, it is critical to emphasize that the user should record the temperature at the beginning of the journey (if the car is not at the appropriate temperature) and at the end of the recording session (when the vehicle is, ideally, at the desired temperature).
6. The recording session should be stopped when the user arrives at a destination and intends to leave the vehicle. The collected data is saved and made accessible for upload, and the application returns to the main page in nonrecording mode.

As previously referred, the mobile application continues to collect data even if running in the background. This means that the user can, e.g., have the smartphone locked or safely make calls while driving.

The P29DR application fosters identity protection by engaging in a strategy of anonymous usernames and IDs provided by the study team, instead of information that allows the application's users to be identified (e.g., name, identification number, online identifiers, or video footage). However, since location data is acquired and alone can be used to identify a person, consents under the General Data Protection Regulation (GDPR) must be signed.

3.4 Conclusion

In this work, a data recording mobile application, the P29DR app, with the ability to collect real data and create datasets of people's driving habits, is proposed. The software was developed under the scope of

the joint project UMinho and Bosch – "Easy Ride: Experience is everything". This tool can be a valuable solution for university or corporate research teams working on breakthrough in-vehicle solutions since, to the best of our knowledge, there is no application capable of recording the routines of vehicle occupants.

Essentially, recorded information consists of data on the geographic position of trips, occupants and objects that enter and leave the vehicle, and the temperature within the vehicle. The data are internally organized in files that can be transferred at any time via the app. The created files can then be compiled or filtered by the research team to produce more meaningful datasets and gather the types of variables most relevant to the study or development at hand.

Aside from the fact that no usability study has yet been conducted, a small number of drivers have already expressed positive reactions and feedback on the app's usability and user-friendliness. Furthermore, the software is a flexible and free of charge solution as it can be easily installed on personal devices. Identity protection is promoted to the best extent possible by using anonymous usernames and IDs, instead of identifiable information. Nevertheless, location data alone can be used to identify someone (by finding the places visited), so consents under the GDPR must be considered.

The possibility of conducting a usability study to carefully assess the users' satisfaction with the software is considered for future work. Moreover, the application can be further extended by adding more variables to record, e.g., collect information about other comfort settings. It can be studied alternative strategies that may enhance and ease the interaction between the user and the app even more. Finally, safety techniques can be adopted. For example, algorithms exploiting the mobile device's sensors (e.g., accelerometer) can be developed, to enable interaction with the application only when the vehicle is completely stopped.

Chapter 4

A Dynamic Neural Field approach for intelligent cockpits: online learning and prediction of traveling routines

A Dynamic Neural Field approach for intelligent cockpits: online learning and prediction of traveling routines[1]

Pedro M. F. Guimarães, Flora Ferreira, Weronika Wojtak, Paulo J. S. Barbosa, Sérgio Monteiro, Estela Bicho, and Wolfram Erlhagen

Article published in *IEEE Transactions on Intelligent Transportation Systems*
December 2024
DOI: 10.1109/TITS.2024.3463389
Link: doi.org/10.1109/TITS.2024.3463389

Author contribution statement: Pedro M. F. Guimarães conceived and developed the model and the approach, performed the initial model validation using MATLAB, planned and conducted the experiments, analyzed and interpreted the results, and wrote the manuscript. Flora Ferreira helped conceive the approach and supervised the work. Weronika Wojtak and Paulo J. S. Barbosa performed the initial model validation using MATLAB. Sérgio Monteiro supervised the work and provided essential resources. Estela Bicho provided essential resources and helped review the manuscript. Wolfram Erlhagen contributed to interpreting the results and reviewed the manuscript.

[1] © 2024 IEEE. Reprinted, with permission, from P. M. F. Guimarães, F. Ferreira, W. Wojtak, P. J. S. Barbosa, S. Monteiro, E. Bicho, and W. Erlhagen. A Dynamic Neural Field Approach for Intelligent Cockpits: Online Learning and Prediction of Traveling Routines. In *IEEE Transactions on Intelligent Transportation Systems*. September 2024.

A Dynamic Neural Field Approach for Intelligent Cockpits: Online Learning and Prediction of Traveling Routines

Pedro M. F. Guimarães[1,2,3,4] , Flora Ferreira[2], Weronika Wojtak[1,2,3], Paulo J. S. Barbosa[2], Sérgio Monteiro[1], Estela Bicho[1], and Wolfram Erlhagen[2]

[1]ALGORITMI Research Center, University of Minho
[2]Center of Mathematics, University of Minho
[3]CCG/ZGDV Institute
[4]Bosch Car Multimedia Portugal

Abstract

Empirical studies of human mobility patterns reveal a high degree of spatio-temporal regularity, with individuals frequently traveling to the same destinations at relatively the same time. Moreover, the vehicle frequently transports the same occupants carrying specific objects. Indeed, there is a rich body of work on predicting users' future destinations. However, existing computational approaches primarily rely on offline learning methods that require stationary data distributions, thus not accounting for situations in which traveling habits may change over time. The present research focuses on individual mobility prediction for an advanced driver assistance system within the In-Vehicle Information Systems (IVIS) domain. Specifically, we propose a novel approach based on the framework of Dynamic Neural Fields that learns the traveling routines of a vehicle, including the drivers, passengers, and objects. In our neurodynamics approach, the learning takes place continually and online, allowing the system to accommodate changes in traveling habits over time. The proposed strategy goes beyond learning the mobility routines of a vehicle, enabling predictive assistance for questions like *When is the next departure? Where to go?* but also extended assistance such as predicting *Who is the next driver? Which passengers/objects will enter/leave?* We evaluate the performance of the approach in predicting trip information on 11-week datasets of real traveling routines. A vehicle with such cognitive abilities has the potential to improve drivers' readiness, enhance trip comfort, and prevent objects from being forgotten.

Keywords—individual mobility, intelligent vehicles, neurodynamics, spatial clustering, spatio-temporal prediction.

4.1 Introduction

4.1.1 Motivation

In recent years, the automotive industry has experienced significant advancements in intelligent transportation and vehicle technologies, driven by the growing demand for innovative solutions to enhance the

driving experience and ensure safety [1–3]. Central to this evolution is the development of ADAS, which play a pivotal role in minimizing accidents, optimizing driving performance, and improving user comfort [4]. Notably, the demand for ADAS is expected to continue growing in the coming years due to safety regulations and consumer interest in applications that promote comfort and efficiency [5, 6]. Despite the traditional orientation of ADAS toward generic design principles, there is a growing recognition of the necessity for personalized solutions adaptable to different drivers [2, 7]. This paradigm shift underscores the importance of considering user preferences and characteristics to increase user acceptance and trust. Since personalization in real-life applications cannot be viewed as a static process, the system needs to be updated and improved continuously by driver interactions [8–10].

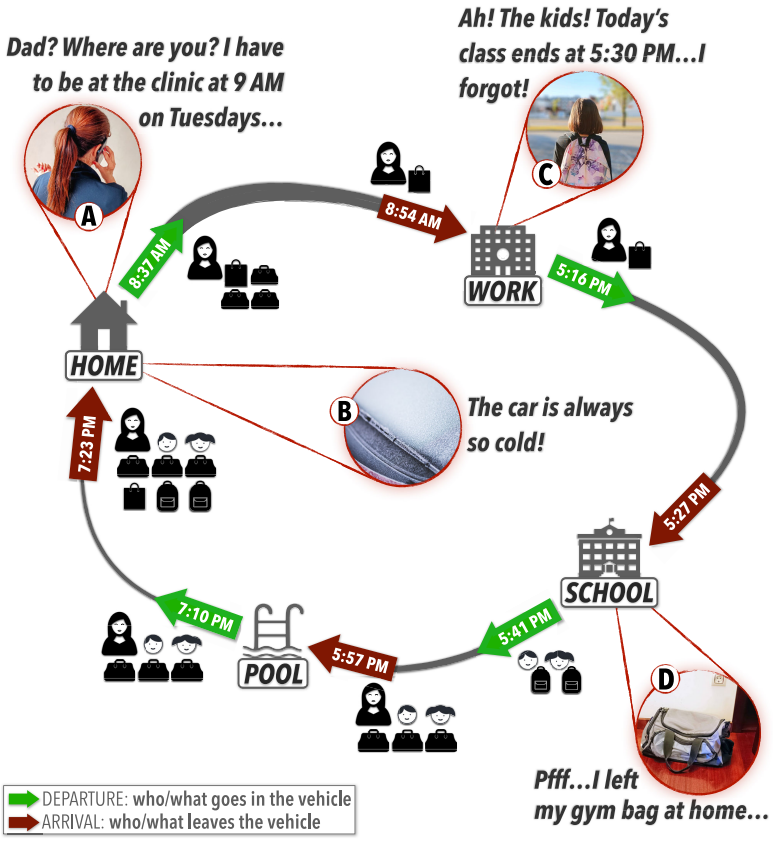


Figure 25: **A common weekday traveling routine of a private vehicle, including drivers, passengers and objects status, which can be learned and recalled by an intelligent driver assistant.** Its cognitive capacities can be used to optimize car sharing (A), adjust comfort settings in advance (B), and alert about upcoming traveling events (C) and missing objects (D).

The focus of the present research is on mobility prediction for a personalized ADAS which belongs to the category of IVIS in the domain of driving assistance [2]. Specifically, we propose a system with cognitive capabilities for online learning and predicting the mobility patterns of vehicle occupants, including departures, arrivals, next driver, passengers, and objects. It responds to a 2020 survey by Patel et al. [14] who collected opinions about desired features of cognitive vehicles. One such feature is the

capacity to anticipate the next travel destination, to timely inform a driver about the predicted departure time, and to alert a user about objects usually transported in the vehicle at a specific time of the day. Furthermore, based on the predicted departure time, the system could enhance comfort and convenience for a subsequent driver by adjusting the AC temperature and the position of mirrors and seats (Figure 25). Besides the application in a private vehicle, such a personalized ADAS has also the potential to improve ride-hailing services like Waymo [15] or community-shared vehicles. Learning the daily traveling routines of individual users could enable vehicles to anticipate pick-up times, improving overall efficiency and user experience, as well as fleet management.

A major challenge for existing mobility prediction models is that they have to capture both the spatial and temporal characteristics of the underlying travel activities. Indeed, several studies suggest that human mobility exhibits a high degree of spatio temporal regularity [17–19], with individuals often visiting the same places at similar times of a specific day. For instance, weekday trajectories of a typical family car may involve dropping the kids off at school, going to work, picking the kids up again at school, and returning home. Specific days of a week may include additional stops to a scheduled gym class or music lessons, and regular trips to the grocery store or the park.

Despite the importance for proactive and personalized applications, limited attention has been paid thus far to individual mobility prediction [21]. Most existing theoretical approaches require massive, stationary datasets to get stable prediction models. They do not account for realistic application scenarios in which a limited set of individual training samples become progressively available, or the individual traveling patterns change over time. Furthermore, the arbitrary discretization of time in most existing models does not capture well the continuous time choices characterizing human mobility patterns. Thus, the main objective is to tackle the problem of individual mobility prediction by developing a neuro-dynamics model that supports the continual learning of next trip information from time-stamped location coordinates. Inspired by computational principles of processing spatio-temporal information in the brain [119], the model represents the location information by stable activity patterns. The continuous temporal evolution of these input-driven patterns can be used to simultaneously measure and store the associated timing information. A dynamic readout of the stored spatio-temporal information is then used for the online prediction of traveling routines.

4.1.2 Contribution

Compared to existing literature (see Section 1.3, *State of the Art*, in Chapter 1, *Introduction*), the significant advances of the proposed DNF approach include:

- Online clustering of geographical locations to identify regions of interest on the topographic map.
- Continual and implicit learning as spatio-temporal GPS records come in, i.e., the driver is not explicitly asked for preferences or habits.

- Individualized learning of the vehicle's mobility routines, including driver, passengers, and objects. This covers the adaptation to changing user habits, including changes in the timing of specific traveling events (arrivals or departures), as well as the forgetting of specific visits that are no longer part of routine behavior.
- Transparency of the learning process and the individual mobility prediction. The activation patterns of the distributed neural network provide meaningful behavioral interpretation, and model parameters can be adjusted depending on the application context.
- Autonomous recall of the learned sequences of traveling events in anticipation of their realization. This allows an ADAS to timely alert the driver about upcoming arrivals or departures and potential changes in the status of passengers and objects.

We evaluate the performance of the neurodynamics approach using datasets of real traveling routines recorded during 11 weeks in the region of Braga (Portugal).

Section 4.2 introduces the fundamentals of the theoretical framework of Dynamic Neural Fields. The proposed DNF approach for learning traveling routines is described in Section 4.3. Section 4.4 reveals the experimental results, and Section 4.5 the discussion of the approach and future work. Finally, the chapter wraps up with conclusions in Section 4.6.

4.2 The Dynamic Neural Field framework

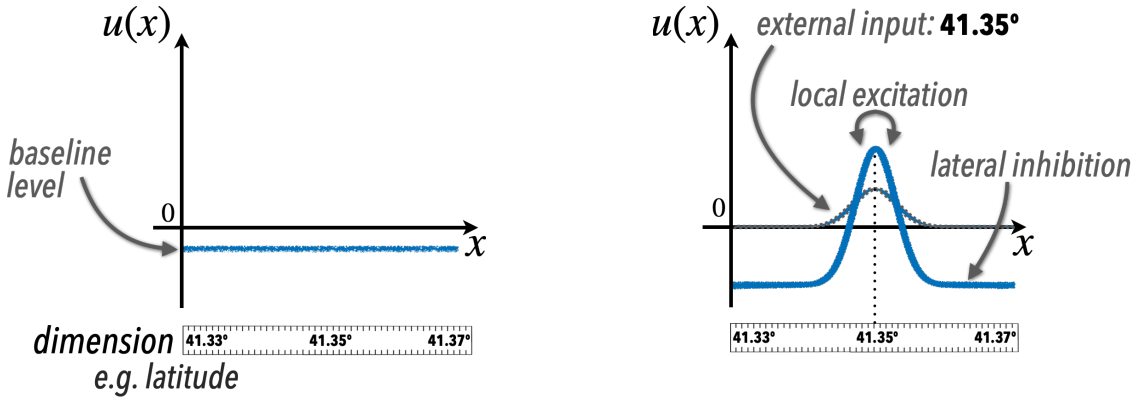
Dynamic Neural Fields represent a theoretical framework for modeling cognitive functions inspired by cortical processing mechanisms. It provides means to implement memory, decision-making, planning, adaptation and learning in artificial agents such as autonomous robots [107, 112, 195]. It has also been widely used in the past to model human cognitive behavior (for an overview of experimental and theoretical studies see [94])

DNF models implement the dynamics of recurrently connected neural populations (Figure 26(a)), which are tuned to continuous input features such as object color, the direction of heading during navigation, or, much like in the present application, spatial coordinates. The recurrent network dynamics may switch in response to a transient input from a homogeneous baseline state into a self-sustained activation peak localized in feature space (Figure 26(b)). Moreover, since the interaction strength between neurons depends only on their distance in parametric space, the network structure is translation invariant. Therefore, the network can hold a continuous family of peaks, each representing the memory of a specific input value.

The continuous temporal evolution of the activity $u(\mathbf{r}, t)$ at position $\mathbf{r} \in \mathbb{R}^d$ (with $d = 1, 2$ corresponding to one- or two-dimensional domains, respectively) is governed by the field equation proposed by Amari [114]:

$$\tau \frac{\partial u(\mathbf{r}, t)}{\partial t} = -u(\mathbf{r}, t) + h + I(\mathbf{r}, t) + \int w(\mathbf{r} - \mathbf{r}') f_\theta(u(\mathbf{r}', t))\, \mathrm{d}\mathbf{r}' \tag{4.1}$$

The parameter τ defines the timescale of the dynamics. The global inhibition $h < 0$ determines the resting (or baseline) level to which the field activation converges without time-dependent external input $I(\mathbf{r}, t)$. The interaction kernel $w(\mathbf{r} - \mathbf{r}')$, also referred to as connectivity function, determines the excitatory-inhibitory connection strength between any two sites $\mathbf{r}$ and $\mathbf{r}'$, which only depends on the distance. The nonlinear firing rate function $f_\theta(u)$ is chosen as a Heaviside step function with a threshold θ. It ensures that only neurons with activation above threshold contribute to the recurrent interactions within the network. To simplify the notation, we use the following definition for the reverse of the Heaviside step function, $\overline{f_\theta(u)} = 1 - f_\theta(u)$.



(a) (b)

Figure 26: **Bi-stable field dynamics example. (a)** 1D neural field ($\mathbf{r} = x$) representing latitude at stable baseline (or resting) state. **(b)** Transient external input drives the evolution of an activation peak centered at the input location 41.35°. The peak is self-sustained due to the recurrent excitation in the network which is counterbalanced by longer-ranged inhibition (kernel of lateral inhibition type).

In the present application, there are fields spanned over two spatial dimensions representing latitude and longitude of the GPS data (Figure 27(a)), where $\mathbf{r} = (x, y) \in \mathbb{R}^2$. In addition, there are one-dimensional fields (Figure 27(b)) where the variable $\mathbf{r} = x \in \mathbb{R}$ represents labeled, categorical data, concretely, stop locations (e.g., home, gym, or office) resulting from a clustering algorithm.

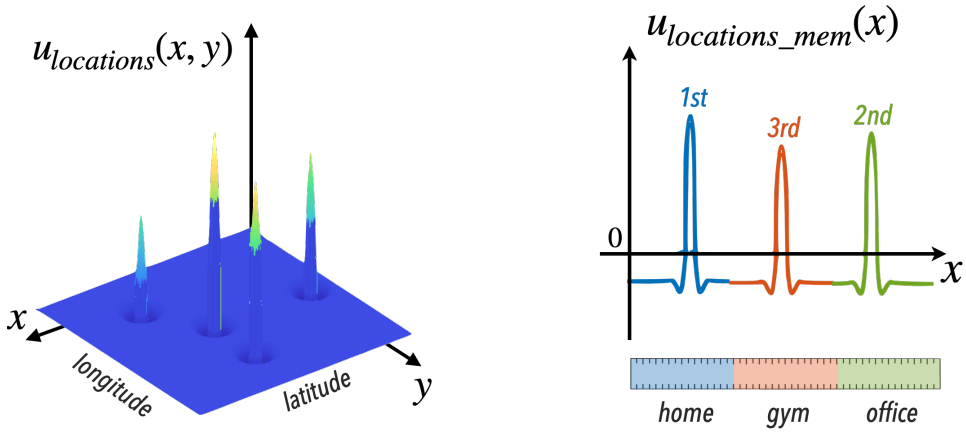
Concerning the recurring interactions, we apply two different connectivity functions for which conditions on the network parameters supporting the existence of stable single- and multi-peak solutions can be rigorously proven (for details see e.g., [114] and [118]). A kernel of lateral inhibition type guarantees that only a single peak may evolve at a time (Figure 26(b)):

$$w_{lat}(\mathbf{r} - \mathbf{r}') = A e^{-\frac{(\mathbf{r}-\mathbf{r}')^2}{2\sigma^2}} - g \tag{4.2}$$

where $A > 0$ defines the amplitude, $\sigma > 0$ the standard deviation of the Gaussian, and $g > 0$ the global inhibition. The second type of kernel has oscillatory decay, which supports the existence of multiple input-induced peaks in the field [118]:

$$w_{osc}(\mathbf{r} - \mathbf{r}') = Ae^{-b|\mathbf{r}-\mathbf{r}'|}\left(b \sin|\alpha(\mathbf{r} - \mathbf{r}')| + \cos(\alpha(\mathbf{r} - \mathbf{r}'))\right) \tag{4.3}$$

where $A > 0$ defines the amplitude, $b > 0$ controls the rate at which the oscillations decay with distance, and $0 < \alpha \leq 1$ the zero crossings of $w_{osc}(\mathbf{r} - \mathbf{r}')$. Combined with a resting state dynamics [119], the oscillatory kernel enables the creation of an activation gradient over neural subpopulations in the field. It represents the memory of the temporal order and the relative timing of a series of traveling events (Figure 27(b)).



(a) (b)

Figure 27: **Different neural field representations. (a)** 2D neural field ($\mathbf{r} = (x, y)$) representing the memory of four stop locations (points of interest). The peak positions encode the corresponding latitude and longitude values of the GPS data. **(b)** 1D neural field ($\mathbf{r} = x$) divided into three discrete, non-overlapping slots representing the sequence memory of three stop locations (points of interest). The information about the temporal order and relative timing in which the locations were visited is stored in the activation gradient.

4.3 DNF approach to learn and predict traveling routines

4.3.1 Preliminaries

Next, we provide some key definitions frequently used throughout the chapter:

Definition 1 (Traveling Events): Occurrence of a departure from or an arrival at a certain location.

Definition 2 (Traveling Routine): Sequence of traveling events that happen regularly around a specific time on a given day.

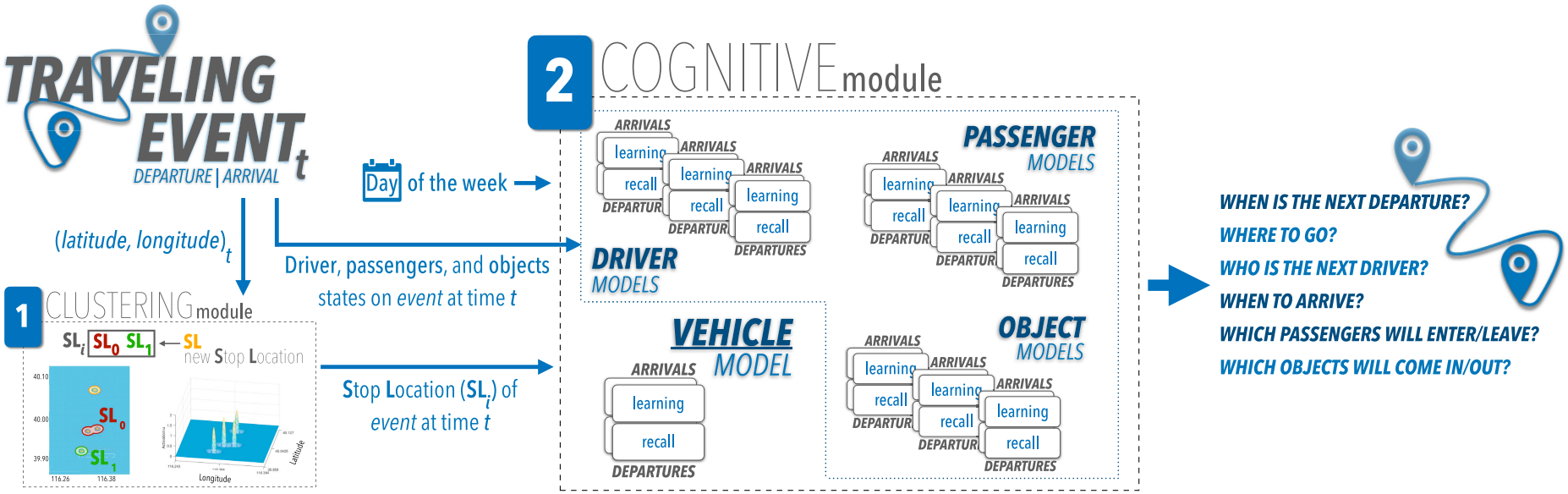


Figure 28: **Schematic overview of the proposed DNF-based approach for learning daily traveling routines of a shared vehicle.** Whenever an event occurs, its geographical location is provided to the clustering module, which returns the ID of a stop location to the cognitive module. The cognitive module implements the neurodynamics approach to learning the spatio-temporal sequences of arrivals and departures for each day of the week. The associated information about driver, passengers and objects is stored in separate sequence representations. At the beginning of a certain day of the week, the corresponding activation patterns from the previous week are used to pre-activate decision fields. The stored spatio-temporal information is then autonomously recalled using a competitive dynamics. This allows an intelligent driver assistant to anticipate the next traveling event including the associated information, and generate predictive assistance.

Definition 3 (Stop Location): Spatial area representing a region of interest. Regions are obtained by a clustering algorithm from repeated events in relatively close geographical locations. A stop location is defined by an associated ID.

Regarding temporal regularities, drivers may have a consistent routine on Mondays but vary their traveling habits significantly on Fridays. In a similar vein, a certain driver may take his/her children to school every day, travel between home and work only on specific workdays, and attend a gym class at a fixed time on specific days. The frequency of recurrent visits at a certain destination and their timing during the day are of fundamental importance for predicting traveling behavior [196]. Drawing on these concepts, we assume that each day of the week is defined by a specific sequence of time-constrained events involving the vehicle, drivers, passengers, and objects. Thus, traveling events that consistently fail to stick to their timings in a narrow time window, are not considered part of routine behavior. While the vehicle takes part in all events, the participation of different drivers, passengers, and objects varies.

4.3.2 Overview

Figure 28 depicts a schematic view of the DNF approach with two main modules covering the clustering and cognitive parts. Since the GPS coordinates will most likely differ to some extent for repeated visits to the same travel destination, the framework is first used for spatial clustering of similar geographic positions. We apply a recently developed DNF model consisting of two coupled two-dimensional field equations of Amari type, Equation (4.1), with $\mathbf{r} = (x, y) \in \mathbb{R}^2$. The mathematical description can be found in Appendix A.1. It supports the continuous integration of inputs over time [46, 197]. The spatial range of the excitatory connections controls the spatial spread of self-sustained peak activity on the GPS map in response to a single input data. When a series of inputs is applied within this excitatory range, the field dynamics develops a broad, connected pattern of persistent activity. This data grouping in the same excited region happens for all travel destinations, resulting in several activity clusters, or stop locations, on the topographic map (Figure 29). The choice of the kernel parameter σcontrols the cluster size and, thus, the spatial resolution for distinct travel destinations (for more details on the algorithm, see [46]). Due to its ability to group similar GPS coordinates together to identify stop locations, the clustering model effectively accommodates fluctuations in GPS data, addressing potential inconsistencies and mitigating the impact of noise.

The clusters provide the spatial information for DNF models in the Cognitive module to learn mobility routines. To simplify the neural computations and reduce the computational cost, the different clusters on the two-dimensional field are mapped onto non-overlapping intervals or slots of a one-dimensional field. Newly arriving GPS input then activates the neural subpopulations representing the corresponding slot. The learned routine of the vehicle on a specific day of the week is memorized as a stable multi-peak pattern, encoding the location and relative timing of events. The stored information is autonomously recalled from

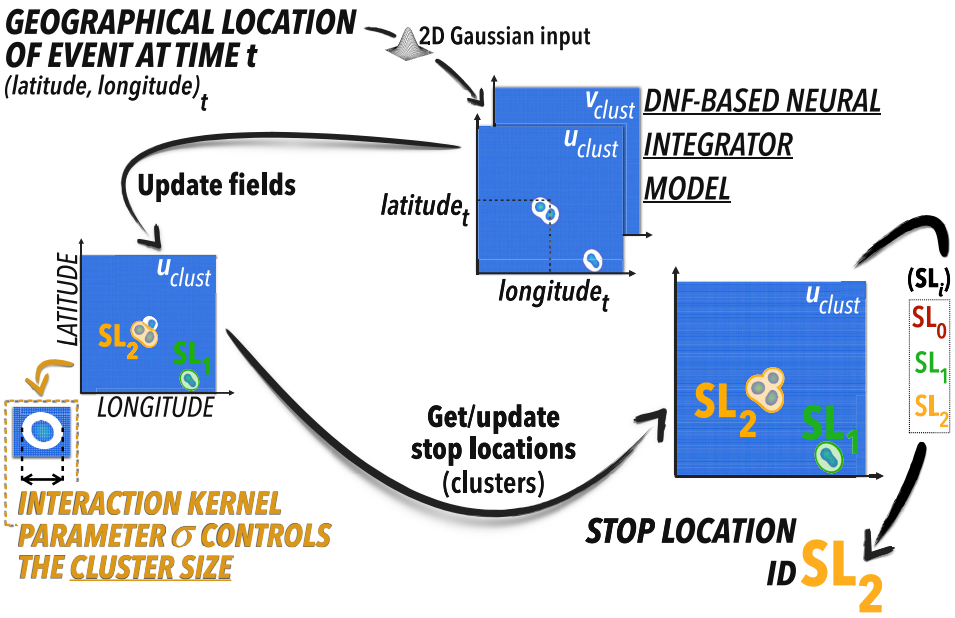


Figure 29: **Clustering module.** A DNF-based neural integrator model, comprising 2D fields, represents accumulated input at specific geographical locations. Data grouping in the same excited region results in different activity clusters (stop locations).

the beginning of the day using a competitive neural dynamics. The result is a sequential activation of the neural representations of all traveling events in the correct order and at the expected time.

Each vehicle and all associated entities have their own instance of a DNF model (Figure 28). Importantly, since the same generic model is used in parallel to learn the sequential information for each driver, passenger, and object taking part in the vehicle's routine, the recall of the respective sequences appears to be synchronized in space and time. This ensures that a readout of simultaneously activated neural populations allows an ADAS to predict the location and timing of the next event together with the status of the driver, passengers, and objects.

4.3.3 Details of the DNF cognitive model

In this subsection, we explain the functioning of the DNF model and provide an illustrative example considering the trajectory of a vehicle during a certain weekday. The same processing principles apply to all associated entities. The mathematical description of the model can be found in Appendix A.2. Additionally, a comprehensive list of model variables is presented in Table 10.

Model description

Figure 30 illustrates the cognitive model architecture consisting of two main parts covering the learning and the recall of routines. When we refer to a traveling event, it can be an arrival or a departure event since the same DNF architecture is used independently for both types. The specific functional role of each

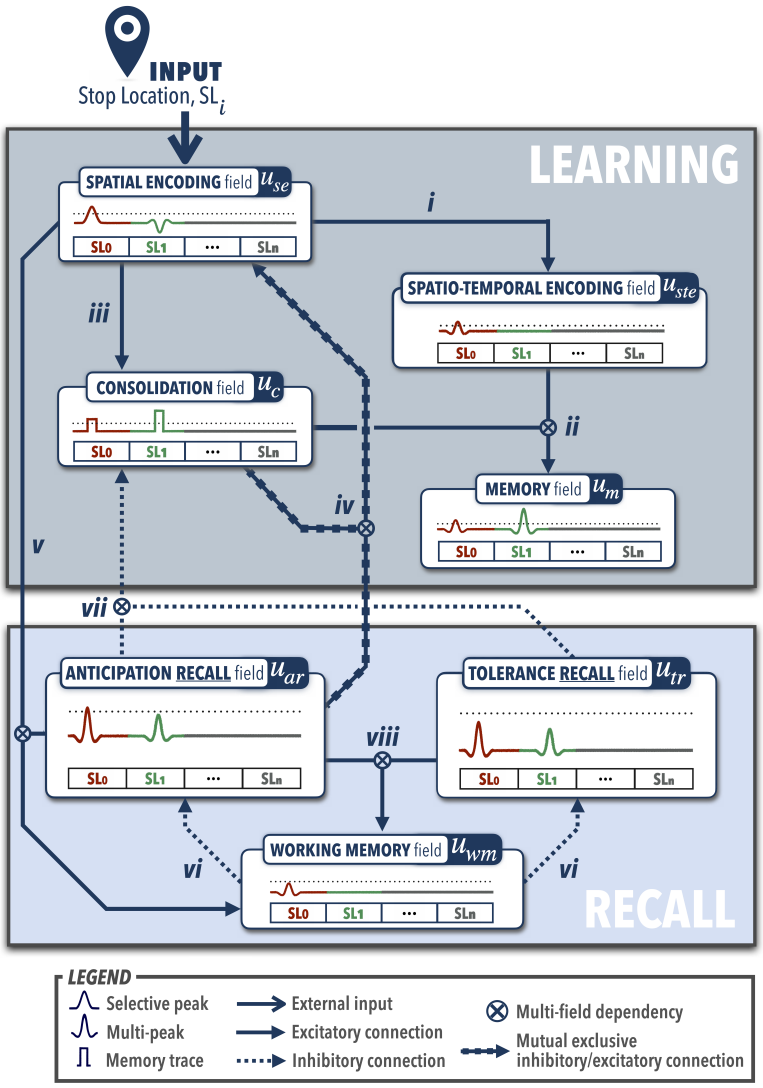


Figure 30: **Architecture of the cognitive model which is used independently for departure and for arrival events.** It is divided into two interconnected parts, Learning and Recall, which run in parallel. connections between fields are depicted using roman numerals. Stop locations are represented using different colors to denote different slots in the neural fields. For more details, see the text.

of the seven neural fields is defined by the excitatory and inhibitory connections to other fields (connections are labeled with Roman numerals). The one-dimensional fields are divided into non-overlapping intervals or slots representing the different stop locations (depicted in different colors in Figure 30). Their number increases whenever the clustering algorithm stabilizes new regions of elevated activity on the topographic map. The interval length is chosen to guarantee that multiple input-induced peaks may coexist in a single slot. Such a multi-peak pattern emerges when a series of GPS data indicates repeated visits to a specific stop location throughout the day.

Learning the daily mobility routine starts with the first departure event. Transitory homogeneous input is applied to the respective slot in the Spatial Encoding field u_{se} (Equation (A.3)), resulting in the evolution of

a stable activation peak. Mediated by excitatory connections (i), this suprathreshold activity pattern drives the evolution of a peak of the corresponding slot subpopulation in the Spatio-Temporal Encoding field u_{ste} (Equation (A.4)). The peak amplitude continuously increases with elapsed time until the end of the day due to a linear resting state dynamics h_{ste} (Equation (A.5)) for field sites with suprathreshold activity [91, 119, 193]. Therefore, in response to a series of traveling events during the course of the day, the field dynamics develops a stable multi-peak pattern in u_{ste} with the strength of activation decreasing from peak to peak. The amplitude difference between successive event memories thus encodes their relative timing.

The Consolidation field u_c (Equation (A.6)) gates the transfer of the spatio-temporal information about events being learned in u_{ste} to the Memory field u_m (Equation (A.7)), via the excitatory connections (ii). A memory trace of a particular traveling event builds up in u_c due to the localized excitatory input (iii) whenever the corresponding suprathreshold activity pattern exists in u_{se}. The accumulated activity reaches the activation level 0, and the combined excitatory input from u_{ste} and u_c drives the evolution of the neural representation in u_m. When, on the other hand, the strength of the memory trace falls below 0(more details below), the peak in u_m becomes destabilized, and the activity in both fields decays to the baseline level (forgetting process). As a second functional role, localized activity above 0 in u_c inhibits through the connections (iv) the corresponding sites in use. Since the feedback inhibition suppresses the activity below the baseline level, a newly arriving (homogeneous) input to the same slot, representing a repeated visit of the same stop location, automatically drives the evolution of a peak at a different position. There is thus no interference with previously learned memories.

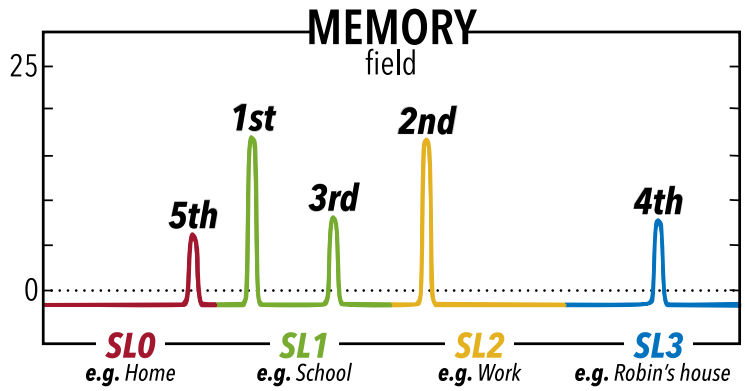


Figure 31: **Example of a multi-peak pattern in the memory field at the end of a day, representing arrivals.** Temporal order and timing of events are encoded in the peak amplitudes.

The autonomous recall of the learned sequential information is performed in parallel in two independent fields (Equation (A.11)), the Anticipation Recall field u_{ar} and the Tolerance Recall field u_{tr}. At the beginning of a day, the memory pattern stored in u_m from the corresponding day of the preceding week (Figure 31) is loaded to both recall fields as subthreshold input. This results in a preactivation of the respective event representations with a relative strength encoding the relative timing of events. The two event

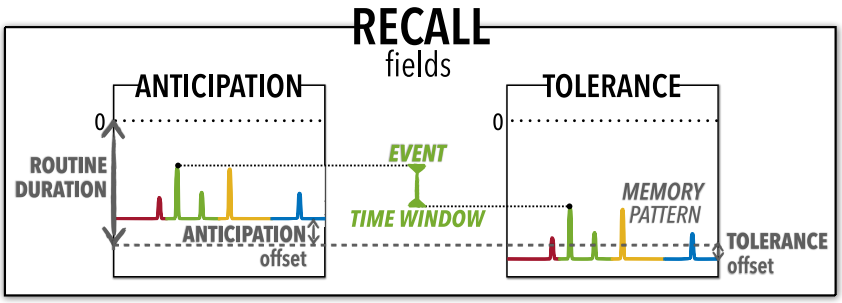


Figure 32: **Time window implemented by the recall dynamics in the Anticipation and Tolerance Recall fields, u_{ar} and u_{tr}.** It is defined by the difference in the preactivation level of the corresponding neural event representations. The baseline levels increase at a constant rate bringing all preactivated neural subpopulations closer to the threshold 0. For example, with a 30min offset for both fields, an event memorized at 3 PM will be recalled (that is, the event representation reaches threshold) by u_{ar} and u_{tr} at 2:30 PM and 3:30 PM, respectively.

representations evolve with a temporal offset, defining a time window around the expected event timing (Figure 32). In u_{ar}, the events are predicted ahead of time, whereas in u_{tr}, a temporal delay is introduced. The recall process starts at the beginning of a specific day by increasing the baseline level h_r at a constant rate (Equation (A.12)). Such ramping brings all preactivated neural subpopulations closer to the threshold 0 for the evolution of a self-stabilized peak. The moment at which the threshold is reached is taken as an internally generated prediction for the timing of the encoded event. The time elapsed since the start of the recall process depends on the distance between the level of preactivation and the threshold. The absolute event timing can thus be calibrated by adapting the baseline level, h_r, of the recall fields, accordingly. To this end, a value h_{r_0} corresponding to the duration of the daily routine is taken as reference. It corresponds to the amplitude of a duration peak (not shown) representing the accumulation of elapsed time until the end of the day (i.e., the gap between the homogeneous resting state and the threshold 0 represents 24h). Then, temporal anticipation and tolerance are achieved in the two recall fields, u_{ar} and u_{tr}, by selecting slightly higher and lower baseline levels, $h_{tr} < h_{r_0} < h_{ar}$, respectively (Figure 32).

When the first peak emerges in u_{ar}, the corresponding neural subpopulation in u_{se} receives subthreshold input through the excitatory connections (iv). This can be seen as creating an expectation about the event. If the expectation has been satisfied, i.e., the specific event has been realized within its time window, the activation peaks in u_{se} and u_{ar} drive through excitatory connections (v) the evolution of a peak in the Working Memory field u_{wm} (Equation (A.13)). This activation pattern, in turn, suppresses through the inhibitory feedback connections (vi) the peaks in the recall fields. To account for the natural variability or track a potential systematic drift in the timing of events, we apply an Exponential Moving Average (EMA) [198] (Equation (A.10)) between the past and the current experience for adapting the peak amplitudes in the Memory field u_m. The EMA places a greater weight on the current pattern in u_{ste} than on the previous pattern stored in u_m. The autonomous recall process continues until the last event if all events have been realized within the respective time window. If this is not the case for one or more of the recalled events,

the respective activation patterns in u_{tr} will eventually reach the threshold 0 (time window "closes") and a peak develops. The inhibitory connections (vii) from the recall fields to the Consolidation field u_c partially weaken the memory trace of the respective event. It is important to stress that this mechanism implements a process based on the concept of active forgetting (for a discussion in the context of cognitive science, see [199] and [200]) whenever the temporal expectation has been violated. The excitatory and inhibitory connections (viii and vi) between the two recall fields, u_{ar} and u_{tr}, and u_{wm} lead to a peak in u_{wm}, which subsequently suppresses the corresponding activity in the recall fields.

In the model applications, an ADAS may communicate to the driver a prediction about an upcoming traveling event realized in u_{ar} only if it is considered part of a routine behavior. This happens when the accumulated activity of the memory trace in u_c is above a threshold θ_h. The choice of buildup and decay rates of the memory trace thus defines how many consecutive repetitions, n, of a new event are needed to call it a routine behavior and the number of times, m, the temporal expectation must be violated for a traveling event to no longer be seen as a habit.

Illustrative example

Figure 33 illustrates the functioning of the cognitive model for an example set of departure events on Mondays. For this example, a temporal anticipation of 30min and a tolerance of 15min was employed. Moreover, we have assumed $m = n = 2$. The snapshots of activity patterns taken at 6 PM on the first Monday show that information about the morning departure is being encoded in the red slot of the Spatio-Temporal Encoding field u_{ste}. The buildup of the memory trace in the Consolidation field u_c resulted in a localized inhibition of the initial activation peak in u_{se}. Since $u_c(x) > 0$, the information encoded in u_{ste} about the morning event is being transferred to the Memory field u_m. The green slot in u_{se} is receiving homogeneous input indicating the departure from *Work1*. At the start of the learning process, there is obviously no recall of departures in the two recall fields, u_{ar} and u_{tr}. The difference in their resting states defines the recall time window.

On Monday 2, the fields u_c and u_m start with the pattern acquired on Monday 1, whereas all other fields start from the baseline level. The snapshots taken at 6:08 PM of the Working Memory field u_{wm} and the Consolidation field u_c show that the morning event has been realized within the expected time window. The memory trace is now above threshold θ_h, meaning that the system considers the event as being part of a routine. Furthermore, by examining the red slot more closely, one can see that the peak in u_m has a larger amplitude than the one in u_{ste}. This happens because temporal information is being once again encoded in the u_{ste}, triggering the adjustment of the peak in u_m through the EMA mechanism. The evolving peak in the green slot of u_{ar} indicates that a departure from *Work1* is being expected. However, the GPS data drives instead the evolution of a peak in the yellow slot of u_{se}, representing a departure from *Work2*. Consequently, both the memory trace in u_c and the neural representation in u_m of the departure at *Work1* converge to the baseline state, as revealed by the activity patterns on Monday 3.

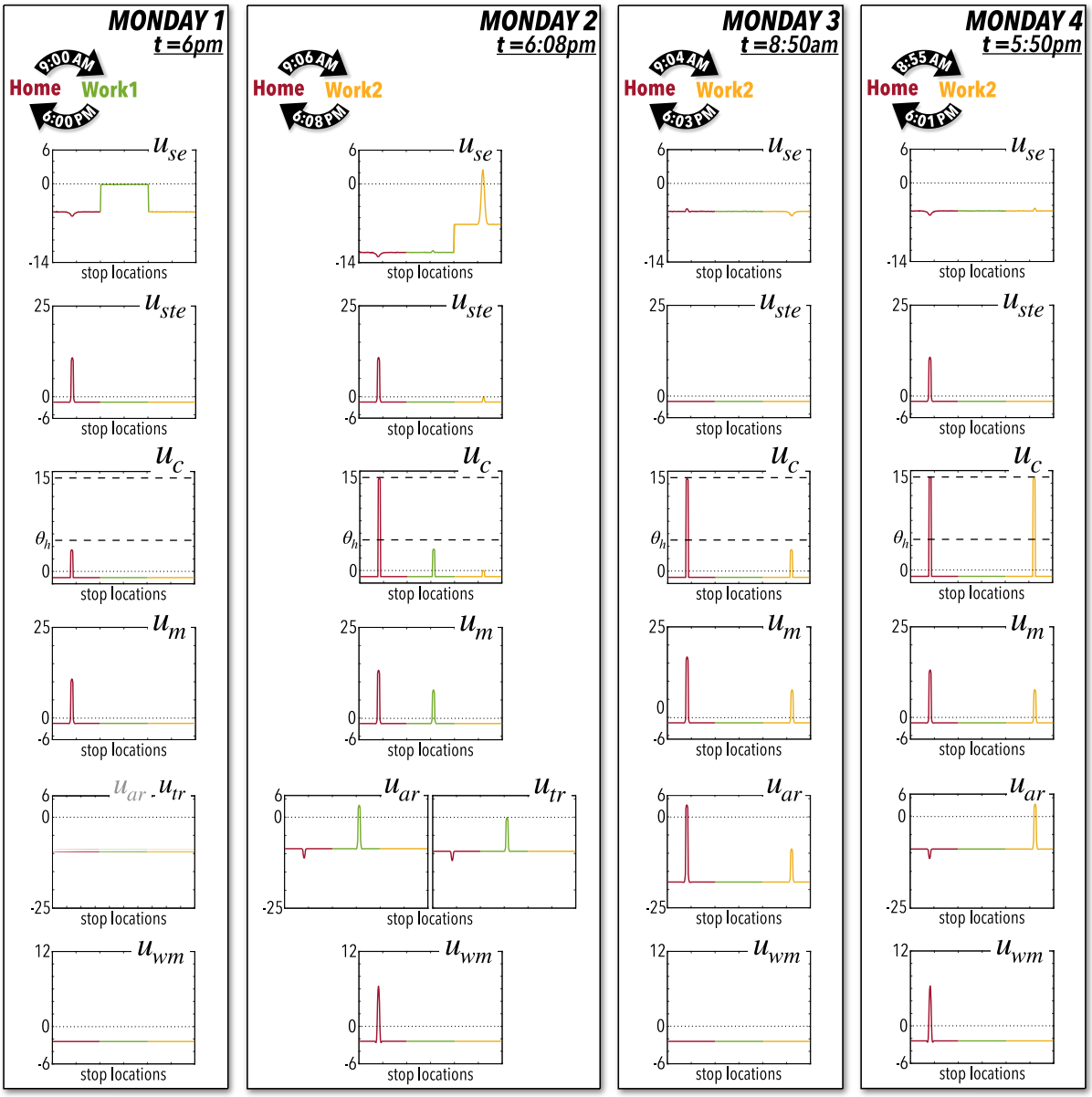


Figure 33: **Snapshots of the activity patterns in the various fields of the vehicle cognitive model for the example of departure events on four consecutive Mondays.** For details see the text.

At 8:50 AM on Monday 3, the first departure event of the day is being anticipated since there is a suprathreshold peak in u_{ar}. The time window for realizing the event is "open", and the neural event representation in the red slot of u_{se} appears to be preshaped. Moreover, since the anticipated event is considered a habit, such information can be used by the ADAS to alert the driver about the upcoming departure event. As indicated by the absence of suprathreshold activation in the yellow slot of u_{ar}, the time window for the evening departure from *Work2* has not opened yet. Moreover, the inhibitory feedback from its memory trace leads to an active suppression of the corresponding neural population in u_{se}. This guarantees that an additional departure from *Work2*, which might occur during the day, will be represented

by a peak evolving at a different slot position.

Lastly, on Monday 4, the strengthening of the memory trace in the yellow slot indicates that the event has been realized within the expected time window on Monday 3. Moreover, the departure from *Home* already took place in the morning. At 5:50 PM, the departure from *Work2* is already being anticipated because it usually happens around 6 PM. Since it is considered part of the routine, the ADAS can use this information for predictive assistance. If the departure eventually happens within the expected time window, its memory trace will be strengthened (or persist).

4.4 Experimental results

4.4.1 Datasets

To test and evaluate the proposed DNF approach, we conducted numerical model simulations on four real-world datasets. We asked drivers of four vehicles to record their daily traveling routines, including passengers and objects, for 11 consecutive weeks using a tailor-made mobile application [201]. Multiple drivers may drive the same vehicle. The four vehicles were chosen from a larger collection of datasets since they show a variety of spatio-temporal travel activities, including both regular and irregular patterns, that challenge the few-shot learning of individual mobility implemented in our neurodynamics approach. Table 5 depicts the information stored in the datasets for the example of a vehicle's traveling trajectory on a certain Friday.

In our study, we prioritized user privacy by implementing a robust approach to identity protection and anonymization within our tailor-made mobile app. Users undergo a single registration process, choosing an anonymous username/ID or alias instead of providing identifiable personal information like names or emails. We are aware of the potential privacy implications of location data, which can indirectly be used to identify individuals. However, in our study, users voluntarily agreed to this condition.

Table 5: **Sample of a Friday traveling routine from the datasets.**

Latitude	**Longitude**	**Type**[1]	**Time**[2]	**Day**	**N. Day**	**Driver**	**Pass.**[3]	**Objects**[3,4]
				...				
41.42489°	-8.29608°	1	1057	Fri	47	Sofia	Pedro	bp2
41.42923°	-8.31896°	0	1074	Fri	47	Sofia	-Pedro	-gb1, -gb2
41.42923°	-8.31896°	1	1143	Fri	47	Sofia	Pedro	gb1, gb2
41.41545°	-8.30388°	0	1156	Fri	47	Sofia	-Pedro	-bp1, -bp2, -gb1, -gb2
				...				

Notes: [1] Departure events labelled with 1 and arrivals with 0.
[2] Time (in minutes) of day when event occurred.
[3] Passengers and objects get in on departures and may leave on arrivals.
[4] "bp"refers to backpack and "gb"to gym bag.

4.4.2 Simulation process

The proposed DNF approach is employed for each vehicle, and all days are successively simulated. The simulation parameters are given in Appendix B. The Clustering module is updated whenever a traveling event occurs, while the models of the Cognitive module are updated at each simulation step. When a new entity appears, e.g., a new passenger, a dedicated learning and recall model is dynamically initiated. Thus, the Cognitive module keeps track of all entities that participate in the mobility pattern of the vehicle at each day.

The beginning of a specific day starts the autonomous recall process of all associated models. It is worth noting that the Vehicle model is always running. At each simulation minute, the u_{ar} and u_c fields of the Vehicle model are read out. If at time t the conditions for predicting the next event are fulfilled ($u_c(x,t) > \theta_h$ and $u_{ar}(x,t) > 0$), the spatio-temporal information of the expected event is gathered. Additionally, all activation patterns of associated models that reach threshold at the same time are read out to determine which entities, if any, are expected to participate in the predicted traveling event. The resulting predictions are saved in spreadsheet files to facilitate comparisons with the datasets and to evaluate the performance of the approach.

We have implemented the numerical simulations of the neurodynamics approach using C++ on the NVIDIA Jetson Nano single-board computer (see Chapter 5, *A neurodynamics-based system for traveling routine assistance in intelligent vehicle cockpits* for a detailed description of the implemented system). To enable parallel computation for updating the field equations, we utilized the NVIDIA CUDA toolkit.

4.4.3 Specifications of the modules

The Clustering module aims to identify stop locations from the beginning on, regardless of how frequently the events occur. This is accomplished by choosing a sufficiently large amplitude of the Gaussian input function such that a suprathreshold peak emerges whenever input is applied. Further details on the parameters used in the Clustering module can be found in Appendix B.1.

With regard to the Cognitive module, we have configured the length of the one-dimensional fields to operate with a capacity of 20 stop location slots. We have inferred this capacity limit by running the Clustering module over all datasets and observing the maximum number of stop locations. The number of neural sites per slot has been chosen to allow the existence of six stable activation peaks memorizing up to six visits of the same travel destination per day. It is important to stress that the number of slots and the maximum number of peaks per slot can be easily increased by choosing a larger field size. Since the computational cost for a one-dimensional convolution gets reduced, a larger field size will not significantly affect the processing speed. For the recall process, we have specified a temporal anticipation of 30min and tolerance of 15min, resulting in a fixed time window of 45min. Additionally, we have assumed that new

events are considered routine if they happen twice in consecutive weeks ($m = n = 2$). Appendixes A.2 and B.2 provide a full list of model parameters.

4.4.4 Performance

Since each day of the week is characterized by a specific sequence of traveling events, we evaluate the performance of the cognitive model for each day of the week on:

(P1) The Vehicle model, predicting the departure and arrival events.

(P2) Associated models, predicting the status of the driver, passengers, and objects on correctly predicted events.

Since $n = 2$ is chosen to define routine behavior, we begin evaluating the performance from the third week on.

To evaluate (P1), we take for each D = {Mondays, Tuesdays, Wednesdays, Thursdays, Fridays, Saturdays, Sundays} the precision, recall, and F1-scores as evaluation metrics. The precision allows us to judge how trustful the model is, i.e., how many of the predictions are actually true. It is defined as:

$$P@\mathrm{D} = \frac{\textit{Correct predictions on } \mathrm{D}}{\textit{Total predictions on } \mathrm{D}} \times 100. \tag{4.4}$$

The recall metric allows us to assess how many real events the model was able to predict out of the total number of real events. It is defined as:

$$R@\mathrm{D} = \frac{\textit{Correct predictions on } \mathrm{D}}{\textit{Real events on } \mathrm{D}} \times 100. \tag{4.5}$$

The F1-score describes the harmonic mean of precision and recall. It is defined as:

$$F_1@D = 2 \times \frac{P@D \times R@D}{P@D + R@D}. \tag{4.6}$$

A prediction is considered correct if the predicted event and a real event satisfy all the following criteria:

- The geographic coordinates of the real event belong to the predicted stop location (cluster).
- The real event time happens within the predicted event time window.
- The events happen on the same day.
- The event type matches (departure or arrival).

To evaluate (P2), we take the correct predictions from (P1) and verify, for each of them, whether the predicted passenger status fully matches the actual one. Then, for each D, the ratio of correct status

predictions to the total number of correct event predictions is calculated. The same procedure is applied to the status of driver and objects.

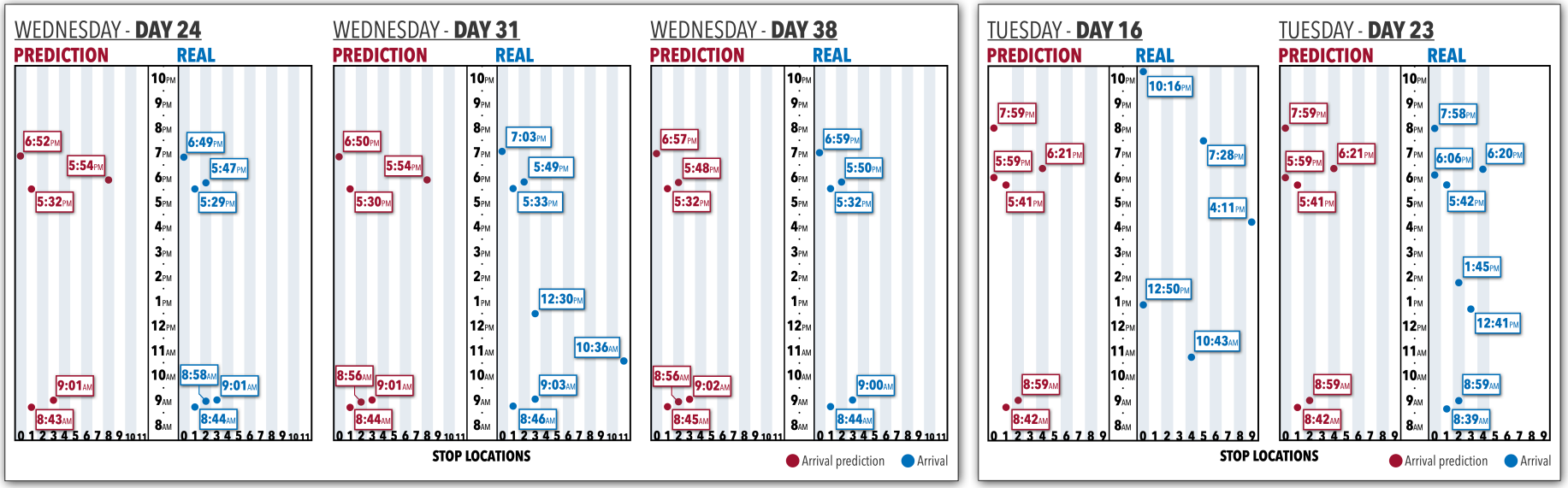
Tables 6 and 7 depict the performance results for (P1) and (P2), respectively. Comparing the different scores in Table 6 reveals that the model detects different levels of regularity in the traveling behavior depending on the day. For vehicles 2 and 4, the recurrence of traveling patterns on weekdays is generally high, resulting in F1-scores in the range of 83.3% to 92.0% for dataset 2 and 87.5% to 95.7% for dataset 4. In contrast, vehicles 1 and 3 exhibit higher variability in their traveling patterns from week to week, making their trajectories less predictable. This is particularly true for the Tuesday routines, where F1-scores of 57.1% and 43.1%, respectively, were obtained. Figure 34 illustrates, for the example of arrival events, two major sources of prediction errors associated with persistent or exceptional changes in the routines. Panel (a) shows that starting from day 24, the arrival event at about 5:50 PM has changed from stop location 8 to stop location 2. The model predicts the old location on days 24 and 31 but successfully adapts to the new habit, as shown by the correct prediction on day 38. Panel (b) demonstrates the behavior of the model when a complete break in the traveling routine occurs on day 16, resulting in 100% wrong predictions. Nevertheless, the accumulated information from the preceding weeks about the recurrence of arrival events is not entirely lost. This is evident from the correct predictions on day 23, except for two new events taking place at lunchtime.

Table 6: **Results for (P1).** Predicting the traveling events of four vehicles.

	Dataset1			**Dataset2**			**Dataset3**			**Dataset4**		
Days	*P*	*R*	F_1	*P*	*R*	F_1	*P*	*R*	F_1	*P*	*R*	F_1
MON	88.7	71.6	**79.2**	83.0	84.9	**83.9**	88.2	64.3	**74.4**	97.8	93.6	**95.7**
TUE	56.6	57.7	**57.1**	91.3	92.7	**92.0**	51.9	36.8	**43.1**	91.8	96.3	**94.0**
WED	93.7	75.5	**83.6**	89.1	78.8	**83.7**	87.1	64.9	**74.4**	81.6	94.3	**87.5**
THU	90.1	68.1	**77.6**	86.0	80.7	**83.3**	78.2	51.2	**61.9**	83.1	92.5	**87.6**
FRI	90.5	65.7	**76.1**	88.9	94.9	**91.8**	91.4	63.1	**74.6**	95.7	89.3	**92.4**
SAT	73.4	63.0	**67.8**	0	0	**0**	75.9	65.2	**70.2**	73.9	70.7	**72.2**
SUN	56.4	30.6	**39.6**	44.4	44.4	**44.4**	61.1	30.6	**40.7**	86.4	68.9	**76.7**

Notes: Values are expressed as percentages (%).
P, *R* and F_1 represent precision, recall and F_1-score, respectively.

Perhaps not surprisingly, the recorded data show that vehicle trajectories during weekends exhibit a larger variability in destinations compared to working days, likely due to the less temporal constraints and looser routines on weekends in general. The Saturday trajectories of vehicle 2 represent the extreme case in which no match in space and time of traveling events in two consecutive weeks was observed at all, resulting in a F1-score of 0%.



(a) (b)

Figure 34: **Predicted and real arrival events. (a)** Three consecutive Wednesdays from dataset 2 are shown. The model predictions track a routine change for an event that usually occurs around 5:50 PM, where the arrival location changes from stop location 8 to 2. **(b)** Two consecutive Tuesdays from dataset 1 are shown. A learned routine is not instantaneously forgotten even when it is interrupted by a completely different traveling pattern on day 16. The model accurately predicts in space and time most of the real events on day 23. Note that the anticipation offset of 30min for the predictions was discounted to facilitate a direct comparison.

The results in Table 7 show that the DNF model is able to successfully predict the status of driver, passengers, and objects when an event prediction is correct. The driver prediction yields the best results, indicating that regular events are strongly associated with specific drivers, and driver changes during a day do not happen frequently. The predictions for passengers and objects are less accurate but still highly successful. Failures occur, e.g., when a regular passenger with the associated objects does not travel due to unforeseen circumstances or when a new passenger unexpectedly joins a trip.

Table 7: **Results for (P2).** Predicting the status of driver, passengers, and objects.

Days		**Dataset1**	**Dataset2**	**Dataset3**	**Dataset4**
MON	*Driver*	100	100	100	100
	Passengers	100	86.3	93.3	100
	Objects	93.7	87.7	77.8	93.2
TUE	*Driver*	100	100	100	100
	Passengers	100	98.3	92.9	99.4
	Objects	88.3	98.3	78.6	94.9
WED	*Driver*	100	100	100	100
	Passengers	95.9	85.4	100	99.4
	Objects	97.3	86.6	80.3	90.2
THU	*Driver*	100	100	100	100
	Passengers	100	92.4	100	100
	Objects	87.5	93.5	81.4	95.3
FRI	*Driver*	100	100	100	100
	Passengers	98.5	98.2	100	98.5
	Objects	98.5	96.4	94.3	95.0
SAT	*Driver*	100	-	100	81.5
	Passengers	95.0	-	85.0	73.8
	Objects	80.0	-	43.3	92.3
SUN	*Driver*	100	100	100	100
	Passengers	45.8	100	50.0	94.1
	Objects	95.8	87.5	95.5	84.3

Notes: Values are expressed as percentages (%).

4.5 Discussion

We have developed a neurodynamics approach for learning daily traveling routines from GPS data. It is inspired by computations in the brain that show how self-sustained activity patterns of neural populations may encode both the spatial and temporal information of perceived events [119]. The geographical

coordinates of traveling events are represented by stable activation peaks in the two spatial coordinate fields, resulting in spatial clusters. Moreover, a series of traveling events experienced during the day are represented by stable multi-peak patterns with an activation gradient over neural subpopulations. During the sequential recall of the stored spatio-temporal information, the population activity evolves continuously in time from a preactivated state to an attractor state. Flexible event timing is achieved by adjusting the initial conditions of the neural trajectories [119].

An intelligent driver assistant should not only be able to predict the location and timing of the vehicle's next stop, but should also continuously adapt to changing driver habits including regularly transported passengers and objects. The DNF approach implements this capacity by assuming the parallel learning of the events sequences for each element associated with the vehicle's trip during a specific day of the week. Since the same resting level dynamics starts the simultaneous recall of all sequences, the neural patterns evolve synchronously in space and time. The readout of these patterns thus guarantees that the prediction about the next departure or arrival event includes the respective status of the driver, passenger(s), and object(s).

A routine can be roughly defined as a series of frequently repeated behaviors. The concept of a memory trace with activation threshold is used to automatically track habitual events. For the test data recorded over 11 weeks, we applied a growth and decay rates to the memory trace such that two consecutive weeks of occurrence establish a habitual event, while two consecutive weeks of non-occurrence lead to the event no longer being considered a habit (compare Figure 34). The rationale behind this choice is that events that are perceived only once should not be predicted in the next week, while the missing of a routine event due to exceptional circumstances (e.g., school closed or day off) should not lead to its complete forgetting. The focus on the predictive power of the DNF approach in the present study does not tell us which features characterize a daily traveling routine. For larger data sets, it would be interesting to apply statistical or neural network-based learning techniques to identify typical sequential patterns which may allow, for instance, to group users with similar habits or to distinguish the trajectories of family and business cars.

The performance evaluation summarized in Table 6, with F1-scores in the range of 70% to 90% for most days, reveals that the online learning approach is quite efficient in capturing individual traveling routines. Prediction errors, which negatively affect the scores, are unavoidable whenever exceptional or persistent changes in traveling patterns occur. When an unexpected event happens, the ADAS does not know whether this new event is part of a routine change or not. It might thus be worthwhile to associate a confidence level with any prediction communicated to the driver. The degree of confidence can be associated with the activation level of the memory trace representing the accumulated evidence of event recurrence over preceding weeks. The higher the activation, the more likely it is that the past predicts the future.

There are also potential sources of prediction errors that can be addressed in future implementations.

In the current implementation, the arrival and departure events are stored and recalled as independent sequences. A prediction about the next departure event should be only issued if its location matches the location of the preceding arrival event.

A further caveat is that the time window, in which an expected traveling event should occur, is predefined and fixed. An event that happens outside this time window will be treated by the system as a new event, which has to be confirmed at least once to be considered part of a routine (if $n = 2$). Of course, this affects the prediction capacity of the system whenever larger, apparently random fluctuations in the timing pattern of the event sequence are observed. The Saturday trajectories of dataset 2 represents an example. The vehicle frequently traveled to the same locations, but the model was unable to find a regular routine since the real event time often missed the expected time window by a few minutes. Ideally, the time windows should be event-specific by considering the significant differences in the variability of the timing patterns. To achieve this, the system could keep track of the minimum and maximum amplitudes of peaks, which represent the shortest and longest observed timestamps, and use them to tune (with some error margin) the anticipation-tolerance time windows for the recall fields.

In a similar vein, the choice of a fixed spatial scale, defined by the kernel parameter σ, for clustering similar GPS coordinates represents a limitation. While a growing cluster that covers a few hundred meters to identify a single stop location might suffice for a family car driving in a town, the same may not hold true for a professional service car. Its routine trajectory may include distinct destinations at shorter distances, for which the variability of GPS coordinates is small. We plan to address the issue of task-dependent clustering in future work using the neural integrator model described in [197] and [202]. This includes an extension to the temporal domain to distinguish traveling events at the same spatial location but at different times of the day.

Our approach to learning traveling routines based on attractor dynamics differs fundamentally from weight-based learning in deep neural network models, which have become increasingly popular due to their success in several application areas [203]. Of particular interest are recurrent LSTM architectures [57], which have been successfully applied to predict sequential data in both spatial and temporal domains, such as travel time [204], traffic flow [205], and next trip [54]. Deep learning represents a data-driven approach that requires human experts to define the hyperparameter settings of the network, the learning rule, and the cost function. However, DL models are challenged by applications like individual mobility prediction due the heterogeneity and sparsity of available data [59]. The requirement of massive training data excluded, in the present study, a direct comparison of the predictive power of DL models with the DNF approach. We believe, however, that with the increasing availability of mobility data, both approaches may play a complementary role in future studies of individual mobility prediction. The flexibility and interpretability of the few-shot learning process of the DNF model can be combined with the capacity of DL models to learn longer-term dependencies in the traveling patterns to further increase prediction performance.

4.6 Conclusion

The proposed DNF approach represents a data-efficient solution supporting few-shot learning of the spatio-temporal information characterizing traveling routines (including drivers, passengers, and objects). The dynamic essence of our approach and the way it is designed, allows the system to smoothly track routine changes, enhancing its utility and effectiveness in real-world applications. This opens new avenues for research and development in intelligent transportation systems.

The implementation of the neuro-computational principles in one-dimensional fields has a reduced computational cost which does not require large resources. The approach thus provides the basis for an integrated personal ADAS, able to timely notify users about the next arrival and departure time, anticipate comfort settings for the next driver, or remember which objects to take to the next destination or which ones are being forgotten. This has broad implications for enhancing the overall driving experience and convenience by providing personalized assistance. Furthermore, the approach holds the potential to assist also in the context of new public transportation services like ride-hailing and community-shared vehicles, via centralized computing, with the goal to optimize fleet management.

The present study focused on implicit learning without any intervention of users, meaning that the system does not question the drivers about their traveling routines. A bidirectional communication between the two agents could further improve the learning/forgetting process for the assistant without significantly increasing the cognitive workload for the driver.

In real-world applications, assistive predictions could be sent to the user through various channels such as email, SMS, or dedicated mobile apps, accessible through different devices like laptops, smartphones, or smartwatches. In addition, assistive cues could also be displayed on the vehicle's infotainment system or communicated through a voice interface.

Chapter 5

A neurodynamics-based system for traveling routine assistance in intelligent vehicle cockpits

5.1 Introduction

In this chapter, we detail the implementation of the proposed approach within a prototype assistance system, built using ROS and designed to operate on an embedded platform. As presented in the previous chapter (see Chapter 4, *A Dynamic Neural Field Approach for Intelligent Cockpits: Online Learning and Prediction of Traveling Routines*), the neurodynamics-based approach is capable of learning and predicting where and when a vehicle, along with its drivers, passengers, and objects, arrives or departs, forming the foundation of an assistance system for traveling routines. However, it relies on solving a large number of neural field equations, a computationally demanding task that can be alleviated through high-performance parallel processing. To meet these computational demands, the system is implemented on the NVIDIA Jetson Nano, a compact and power-efficient embedded platform equipped with GPU acceleration.

The prototype demonstrates the practical feasibility of the neurodynamics-based approach, showcasing its ability to function in both simulation (as presented in Section 4.4, *Experimental Results* from the previous chapter) and real-time environments. Moreover, the prototype represents a crucial step toward increasing the Technology Readiness Level (TRL) of the proposed approach. Although not yet tested in a real vehicle, its implementation on the cost-effective NVIDIA Jetson Nano ensures compatibility with resource-constrained embedded environments. Furthermore, the system features a modular design comprising distinct nodes for input handling, computation, prediction management, and neural pattern visualization for analysis and debugging. This modularity provides scalability, enabling the development of additional nodes to implement diverse assistive applications tailored to specific user needs. It also ensures interoperability for potential integration and future testing in relevant scenarios.

This chapter is organized as follows: Section 5.2 presents the overall system architecture, detailing the modular design, interactions between nodes, and the dual operational modes of the system: Real-Time and Simulation. Section 5.3 delves into the implementation details of the proposed system, focusing on the internal structure and computational processes of its core component, the Engine Node. The section also explores the ROS interfaces that enable seamless communication between nodes, along with parameter configurations and the organized filesystem. Section 5.4 evaluates the system's performance, emphasizing computational efficiency and scalability across varying configurations. Finally, Section 5.5 summarizes the key findings and discusses the limitations encountered.

5.2 ROS-based system architecture

5.2.1 Overview

The system architecture is designed with modularity and scalability to support two distinct operational modes: Real-Time and Simulation. This modular design enabled the seamless adaptation of the system to

both modes without altering its computational foundation. Four main components compose the system, each with a distinct role (see Figure 35):

- Inputs Node: Processes and manages traveling input data from external (real-time) sources or handles pre-recorded datasets. It remains ready to operate in both modes, adapting based on the active mode.
- Engine Node: Serves as the core computational module, implementing the DNF approach to dynamically learn and predict traveling routines.
- Assistant Node: Consumes raw predictions of traveling routines to generate outputs tailored to assistive tasks.
- Visualization Node: Provides a GUI for real-time visualization of the DNF models running on the Engine Node.

These ROS nodes operate in both modes, albeit with slight variations. In Real-Time mode, the system processes live streams of information from external sources, with an additional node, the Watch Node, introduced to track the current time and day. Conversely, in Simulation mode, pre-recorded datasets emulate traveling routines, allowing for controlled testing and validation of the system's predictions.

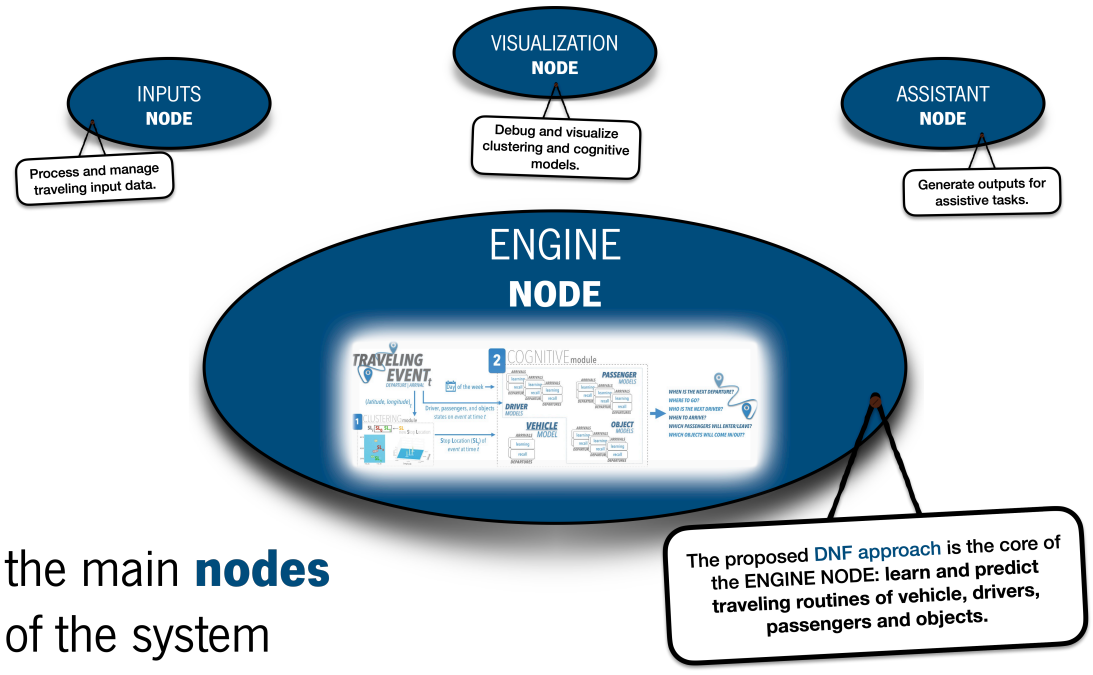


Figure 35: **The main ROS nodes of the system and their roles.** The system consists of four primary nodes: Inputs Node, Engine Node, Assistant Node, and Visualization Node, working cohesively to support both Real-Time and Simulation modes. The Engine Node implements the DNF approach, serving as its computational core.

Despite the operational differences, the core functionality of the Engine Node remains unchanged across both modes. Both share a common computational core, implemented within the Engine Node, which houses the DNF approach (see Figure 28 presented in Chapter 4, *A Dynamic Neural Field approach for intelligent cockpits: online learning and prediction of traveling routines*). The DNF approach processes spatio-temporal data to dynamically learn and predict traveling routines for vehicles, drivers, passengers, and objects. Regardless of the mode, the Engine Node executes the same DNF computations, generating predictions utilized by other components. This modular strategy simplifies adaptability and facilitates future expansions and customizations. Furthermore, since both modes share the same computational core within the Engine Node, the Simulation mode was implemented to enable controlled testing and validation of the system's predictions using pre-recorded datasets. The success of the DNF approach in Simulation mode provides confidence in its theoretical performance under Real-Time mode, ensuring the system's accuracy and robustness while minimizing development costs and risks.

To meet the computational demands of the DNF approach while maintaining compatibility with resource-constrained environments, the system is implemented on the NVIDIA Jetson Nano [125], running Ubuntu OS. Despite being the least powerful option in NVIDIA's embedded board lineup [126], the Jetson Nano was selected due to its affordability and practicality. Its GPU acceleration capabilities provide the computational power needed for the DNF approach, ensuring real-time and enhanced performance. Using the Jetson Nano also demonstrates that the system can achieve its purpose even with the most economical and least powerful NVIDIA board, which implies that more powerful embedded boards would further improve execution speed and enable support for additional functionalities or larger data volumes.

5.2.2 Real-Time mode

In Real-Time mode, the system processes live data streams from external systems, ensuring timely predictions and assistive outputs that align with ongoing scenarios. Unlike Simulation mode, which is optimized for speed, Real-Time mode utilizes real temporal data to synchronize operations with real-world events. The modular architecture, where each node communicates using predefined ROS topics and services, is depicted in Figure 36. Nodes are implemented using a combination of C++ and Python, chosen to balance computational efficiency and development flexibility.

Inputs Node. Implemented in Python, it manages external information of traveling events and serves as the gateway for real-time inputs. It handles information about entities (drivers, passengers, or objects) entering or leaving the vehicle, received through the /entity_in and /entity_out topics. These updates allow the system to dynamically track and maintain an accurate state of the vehicle's occupants and objects. Additionally, the node processes geographic location updates for the vehicle's arrivals and departures, received via /vehicle_in and /vehicle_out topics. When a traveling event occurs, the Inputs Node

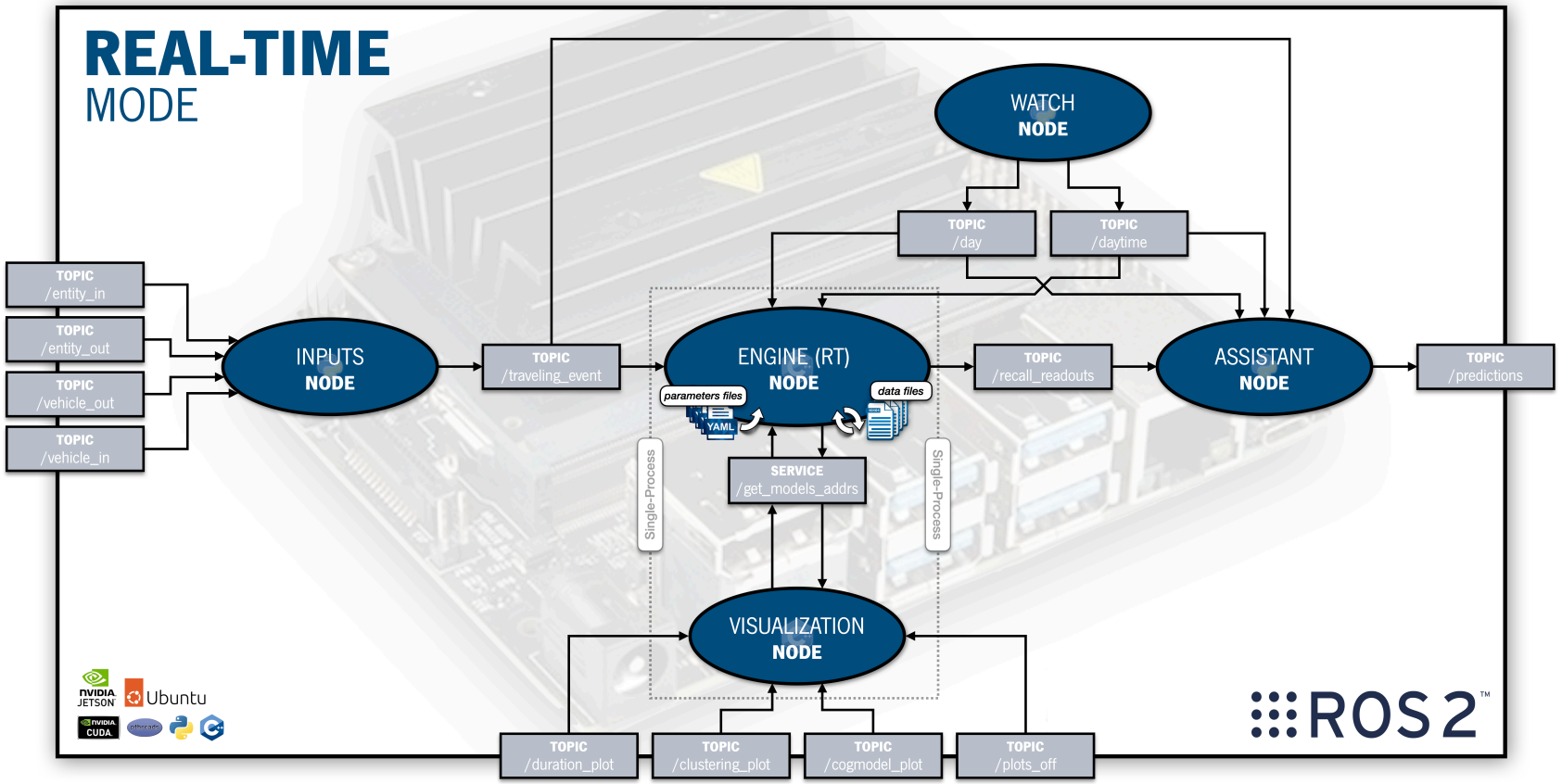


Figure 36: **Real-Time mode system architecture.** It supports the processing of live data streams to generate predictions in real time. The Inputs Node acts as the entry point, subscribing to external topics such as /vehicle_in, /vehicle_out, /entity_in, and /entity_out to gather information about traveling events. This data is consolidated and published to the /traveling_event topic. The Engine Node (RT) processes the structured information from the /traveling_event topic along with temporal information provided by the Watch Node through the /day and /daytime topics. Such node applies this combined input to the DNF approach, which is running continuously, to learn and predict traveling routines. These predictions are published to the /recall_readouts topic, making them available to downstream nodes (currently comprising a single consumer but designed to accommodate additional ones as needed). The Assistant Node consumes these predictions and contextual data from the /traveling_event topic to generate structured outputs, which are then published to the /predictions topic for use by external systems.

consolidates the relevant state information into a structured message (detailed in Subsection 5.3.2, *System interfaces*), and publishes it to the /traveling_event topic for downstream nodes to process.

Engine Node (RT). Implemented in C++, the Engine Node (RT) serves as the central component of the system, within which the DNF approach for learning and predicting traveling routines is running continuously. To optimize performance, the node leverages CUDA for GPU acceleration and *pthreads* for efficient parallel processing. It subscribes to the /traveling_event topic to receive structured event data when traveling events occur, as well as to the /day and /daytime topics to incorporate temporal context. The Engine Node (RT) uses YAML files for configuration, including parameters that define the behavior of DNF models and operational settings. It also uses data files to store or load neural patterns at the end or beginning of a day, respectively. These memory patterns are maintained for each day of the week and are specific to the different drivers, passengers, or objects, as well as the vehicle itself. At predefined intervals, the Engine Node (RT) performs a readout of all active cognitive models to evaluate their predictions. If predictions are detected, the information is structured and published to the /recall_readouts topic, making it available for downstream nodes (currently with one consumer but designed to accommodate more). Additionally, the Engine Node (RT) acts as a service provider, sharing pointers to models running on the Engine Node (RT) through the /get_models_addrs service, enabling other nodes to access specific methods of the models being processed.

Visualization Node. Implemented in C++, the Visualization Node provides real-time insights into the DNF models, particularly the cognitive models running within the Engine Node (RT). The Visualization Node and Engine Node (RT) operate within a shared process, enabling direct (but limited) access to the cognitive models, and thus, the neural field data for visualization and debugging. This design minimizes latency and eliminates communication overhead, ensuring efficient visualization performance. The Visualization Node interacts with the system through dedicated topics. The /clustering_plot topic signals the Visualization Node to open a GUI for visualizing the clustering model, aiding in the analysis of identified stop locations. The /duration_plot topic triggers the visualization of the duration model, which represents the length of daily routines as neural patterns. The /cogmodel_plot topic is used to visualize a specific cognitive model, with messages including an identifier for the model, such as an entity type and name. The /plots_off topic signals the Visualization Node to close any active GUIs associated with ongoing visualizations. These functionalities enable the Visualization Node to support both debugging and in-depth analysis of the DNF models, enhancing the transparency and interpretability of the system's operations.

Watch Node. Implemented in Python, the Watch Node provides essential temporal context for the system. It publishes real-time temporal information to two dedicated topics. The /day topic signals transitions between days, indicating when a new day starts or ends, and providing the current day of the week. The /daytime topic publishes the current minute of the day, updating with each new minute.

Assistant Node. Designed to transform raw predictions into actionable assistive outputs tailored to specific user needs. It subscribes to the /recall_readouts topic to receive individual predictions from the cognitive models in the Engine Node (RT). It also gets contextual information from the /traveling_event topic. By leveraging predictions of traveling events and contextual information about real-time activities, the node provides a foundation for developing various assistive functionalities. Currently, the Assistant Node, implemented in Python, gathers and structures information about predicted traveling events, if any, and publishes this structured data to the /predictions topic. While the node does not yet generate assistive outputs, its modular design ensures extensibility, enabling developers to expand its functionality and create diverse assistive applications in the future.

5.2.3 Simulation mode

The Simulation mode enables the system to process pre-recorded datasets for testing and validating predictions in a controlled environment. Unlike Real-Time mode, which synchronizes with real-world events, Simulation mode operates as fast as computational resources allow, maximizing efficiency for evaluating the system's theoretical performance. The modular architecture is illustrated in Figure 37.

Inputs Node. Implemented in Python, the Inputs Node acts as the entry point for the Simulation Mode. Upon receiving a request through the /get_traveling_routines service, it opens the specified dataset file (spreadsheet) and processes the data to create an array of traveling events. This structured array is returned as the service response.

Engine Node (SIM). Implemented in C++, the Engine Node (SIM) drives the simulation process in this mode. Integrating the simulation handler (responsible for iterating a dataset, day by day, simulating traveling events at predefined time steps) directly within the Engine Node (SIM) minimizes inter-node communication overhead, avoiding unnecessary latency and maximizing efficiency. After requesting and receiving the array of traveling events from the Inputs Node via the /get_traveling_routines service, it iterates through the dataset day by day. For each day, it progresses through time in fixed steps, checking for any scheduled traveling events. When an event matches the current simulated time, the Engine Node (SIM) processes it using the DNF approach in the same manner as in Real-Time mode. Similarly, the resulting predictions are published to the /recall_readouts topic. In addition to processing traveling events, the Engine Node (SIM) assumes the responsibilities of the Watch Node from Real-Time mode, publishing minute transitions to the /daytime topic and signaling day transitions via the /day topic. It also publishes the processed traveling events to the /traveling_event topic, ensuring that other nodes, such as the Assistant Node, receive the necessary contextual information. The /get_models_addrs service remains available, enabling the Visualization Node to access DNF models for debugging and analysis.

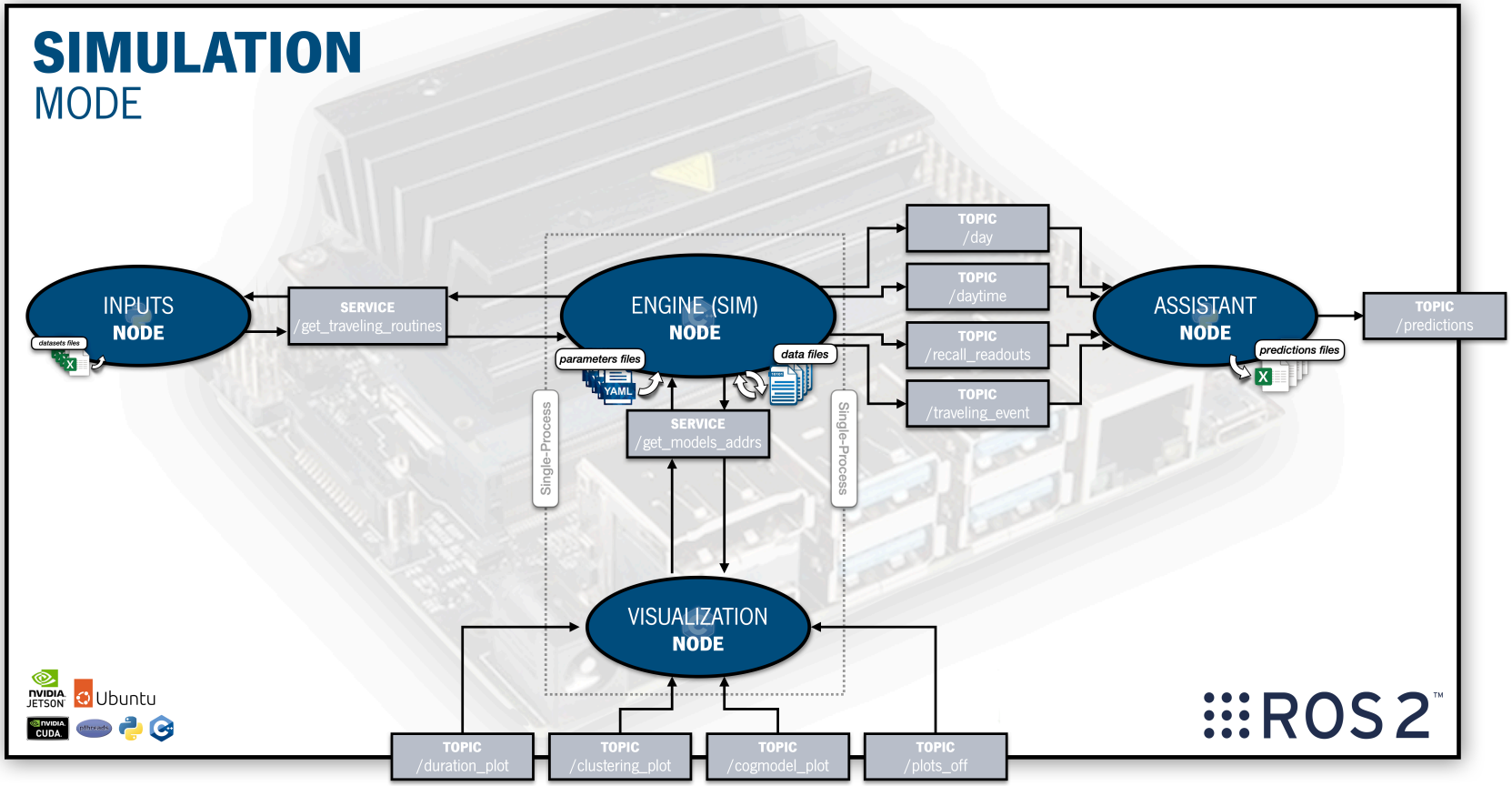


Figure 37: **Simulation mode system architecture.** It supports the processing of pre-recorded datasets to simulate and validate predictions in a controlled environment. The Inputs Node acts as the entry point, providing the Engine Node (SIM) with structured traveling event data upon request via the /get_traveling_routines service. The Engine Node (SIM) conducts the simulation by iterating through the dataset, day by day, simulating events at predefined time steps. It processes traveling events using the DNF approach to learn and predict traveling routines, publishing the resulting predictions to the /recall_readouts topic. Additionally, the Engine Node (SIM) publishes temporal information, such as minute and day transitions, to the /daytime and /day topics. The Assistant Node consumes the predictions from the /recall_readouts topic and contextual data from the /traveling_event topic to generate structured outputs, which are published to the /predictions topic and stored in spreadsheet files for detailed analysis and comparison.

Visualization Node. Implemented in C++, the Visualization Node operates in the same way as in Real-Time mode, providing real-time insights into the system's DNF models. Running within the same process as the Engine Node (SIM), it has direct but controlled access to neural field data, enabling efficient visualization and debugging without additional communication overhead.

Assistant Node. Implemented in Python, the Assistant Node operates in the same way as in Real-Time mode. It subscribes to the /recall_readouts topic to gather raw predictions and to the /traveling_event topic for contextual information about simulated events. The Assistant Node structures this data and publishes it to the /predictions topic for further use. In Simulation mode, the structured predictions are also stored in spreadsheet files, following the same format as the input datasets. This consistency facilitates analysis and enables deep comparisons between predictions and the original datasets. Although the Assistant Node currently does not generate assistive outputs, its modular design allows for easy expansion to develop diverse assistive functionalities in the future.

5.3 System implementation details

This section delves into the implementation details of the proposed system, encapsulated within the routines_adas_system ROS package, designed to support real-time operation and simulation. Each node in the system is implemented as a modular collection of classes, ensuring encapsulation, scalability, and adaptability. Among these nodes, the Engine Node serves as the system's core component, and this section focuses primarily on its internal structure and computational processes. Additionally, this section explores the ROS interfaces that enable seamless communication between nodes, along with key configuration details, including parameter files and the organized filesystem structure.

5.3.1 Engine Node implementation

The Engine Node is the central component of the system, leveraging, at its core, the DNF approach to dynamically learn and predict traveling routines (see Chapter 4, *A Dynamic Neural Field approach for intelligent cockpits: online learning and prediction of traveling routines*, for details on the approach). The Engine's implementation combines a modular class-based design with CUDA-accelerated operations, ensuring scalability and computational efficiency. Two variants of the Engine Node exist: the Engine Node (RT) for real-time operation and the Engine Node (SIM) tailored for simulation. This section focuses on the Engine's internal structure, presenting its class hierarchy and the design choices that support its functionality while addressing both implementation variants. Additionally, the CUDA kernel prototypes, which enable high-performance computations on the neural fields, are introduced and described.

Figure 38 presents the class diagram of the Engine Node, highlighting the relationships between the key entities. The entry points for the two Engine Node variants are the EngineDNFRealTime and

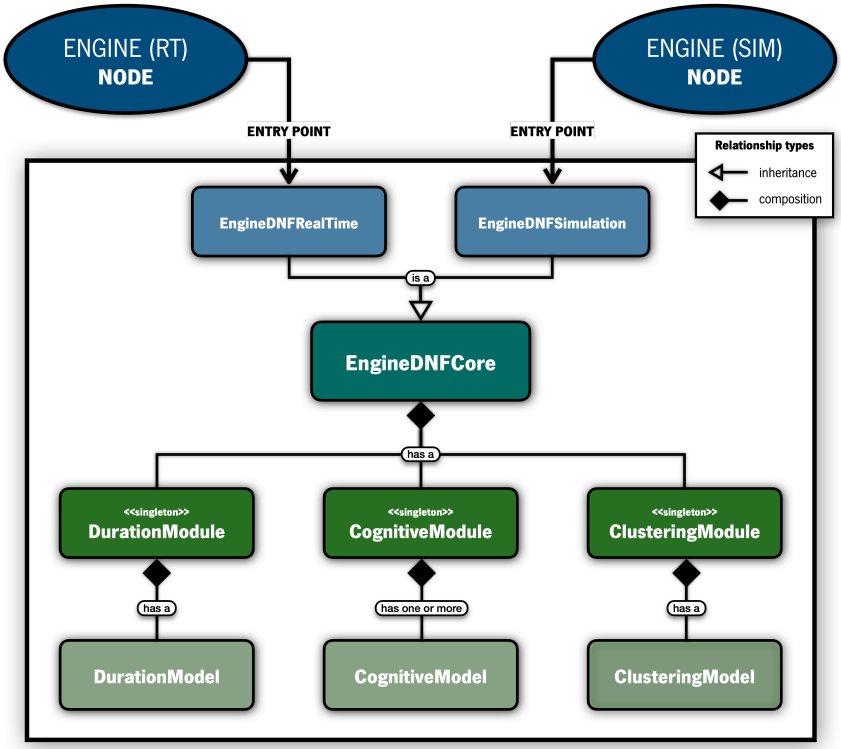


Figure 38: **Class diagram of the** Engine Node**.** The diagram illustrates the modular structure of the Engine Node, highlighting its two variants, EngineDNFRealTime (RT) and EngineDNFSimulation (SIM), which inherit from the shared core class EngineDNFCore. The EngineDNFCore coordinates three main modules: the CognitiveModule, the ClusteringModule, and the DurationModule. Each module is responsible for managing the respective DNF models.

EngineDNFSimulation classes. These classes are responsible for initializing the Engine in their respective operational modes. The EngineDNFRealTime class oversees real-time operation, managing the periodic update of models based on a pre-defined update time step and ensuring that the system promptly handles incoming inputs during real-time execution. Conversely, the EngineDNFSimulation class facilitates the simulation of traveling routines, iterating over the content of pre-recorded datasets while allowing control over simulation aspects such as the starting day, time range, or specific days of the week. Both classes inherit from the shared core class EngineDNFCore, which encapsulates the common functionality, i.e., the DNF approach for learning and predicting traveling routines.

The EngineDNFCore is composed of three main modules:

- The CognitiveModule contains one or more CognitiveModel instances, with each model learning and predicting the traveling routines of a specific entity (i.e., vehicle, drivers, passengers, and objects). It is thus responsible for creating, destroying, and managing these cognitive models for each day. The CognitiveModel class implements the cognitive model (see model architecture in Figure 30), handling the update of its neural fields and interactions, the input to the model, and performing model readouts.

- The ClusteringModule manages the clustering model, implemented in the ClusteringModel class. Its primary function is to process incoming 2D spatial information and reduce it to a lower-dimensional representation, effectively identifying stop locations.
- The DurationModule manages an instance of the DurationModel class, which encodes the daily duration of traveling routines.

The Model classes (e.g., CognitiveModel, ClusteringModel, and DurationModel) leverage GPU acceleration to perform neural field updates efficiently, employing CUDA kernels to parallelize the operations.

While the class diagram in Figure 38 illustrates the structural relationships and modular design of the Engine Node, sequence diagrams are next presented to depict the main functional workflows within the node, specifically, the primary functions of the EngineDNFRealTime class (representing the Engine Node in Real-Time mode). It is important to note that although EngineDNFRealTime inherits from EngineDNFCore, the sequence diagrams explicitly represent method calls (arrows) from EngineDNFRealTime to EngineDNFCore. This approach emphasizes the methods exposed by EngineDNFCore to the concrete classes, showcasing how these functionalities are utilized. While the focus is on the real-time implementation, the same methods (underlined in the diagrams) are called by EngineDNFSimulation in the context of dataset simulations, demonstrating the shared functionality across both operational modes.

The sequence diagram in Figure 39 illustrates the workflow for handling the beginning and end of each day within the Engine Node. At the start of a new day, the newDay(...) method is triggered upon receiving a message from the /day topic. The CognitiveModule dynamically manages CognitiveModel instances for each participating entity (e.g., vehicle, drivers, passengers, or objects), creating new models as needed and loading memory patterns specific to the current day. Notably, the *vehicle* model is always active. During the day, the current minute is updated based on messages received from the /daytime topic. At the end of the day, the closeDay(...) method is invoked to finalize operations. This includes halting the periodic model updates, saving memory patterns to designated data files, resetting the *vehicle*'s cognitive model, and destroying the remaining active CognitiveModel instances.

The sequence diagram in Figure 40 illustrates how the Engine Node processes traveling events and updates the cognitive models accordingly. When a new traveling event is received from the /traveling_event topic, the newEvent(...) function is triggered. The event's spatial information (longitude and latitude) is mapped to a stop location ID via the ClusteringModule. The CognitiveModule then processes the event by providing it as input to the cognitive models of all participating entities (vehicle, drivers, passengers, and objects). If an entity is participating for the first time on that day of the week, a new CognitiveModel instance is created dynamically, ensuring that only relevant models are instantiated. This approach optimizes memory usage and computational efficiency. Additionally, the updateEngine() is periodically triggered at a predefined time step to advance the cognitive models' neural field updates.

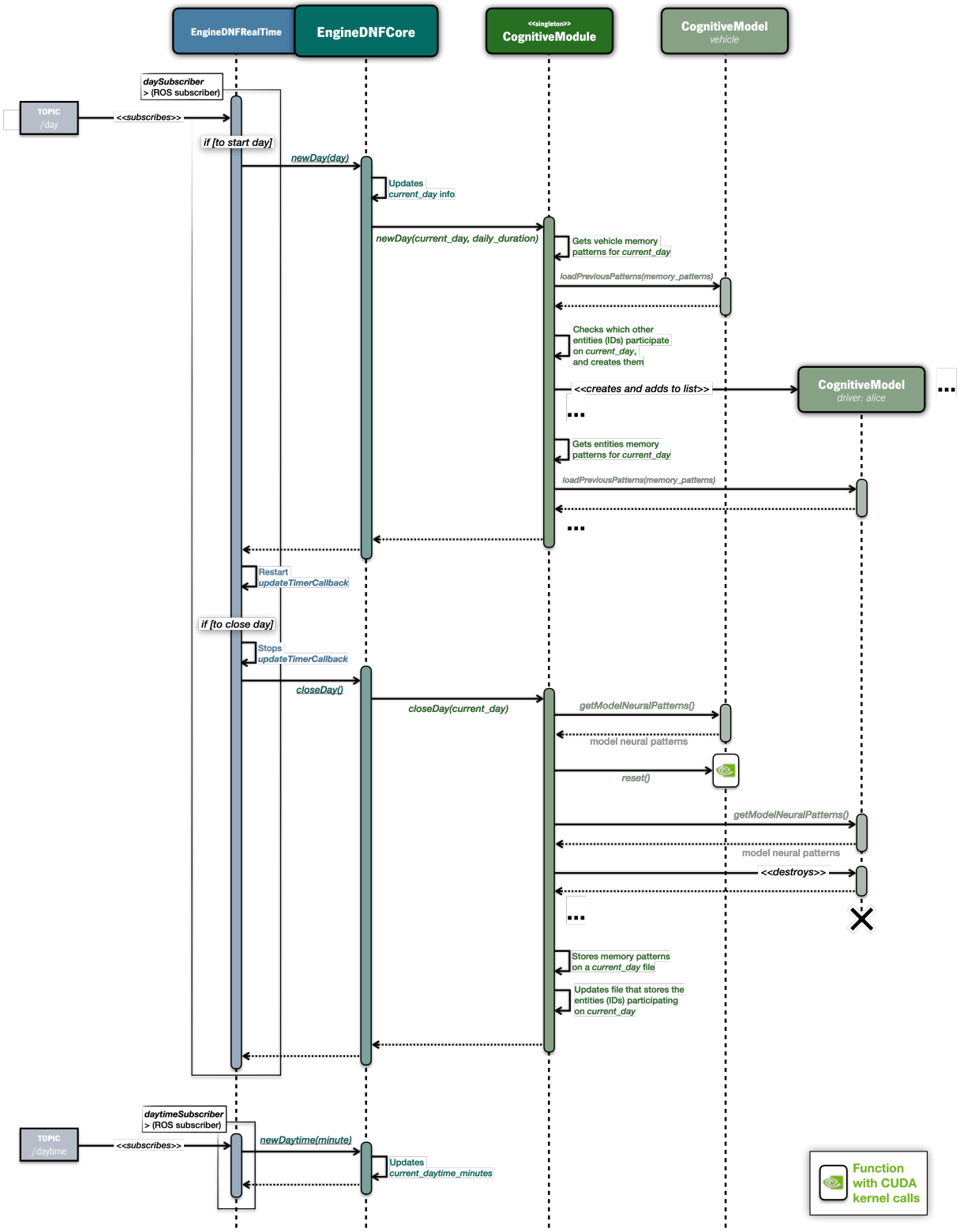


Figure 39: **Sequence diagram outlining the workflow within the** Engine Node **when a day begins and ends.** Functions underlined in the diagram correspond to methods defined in EngineDNFCore, available to EngineDNFRealTime and EngineDNFSimulation.

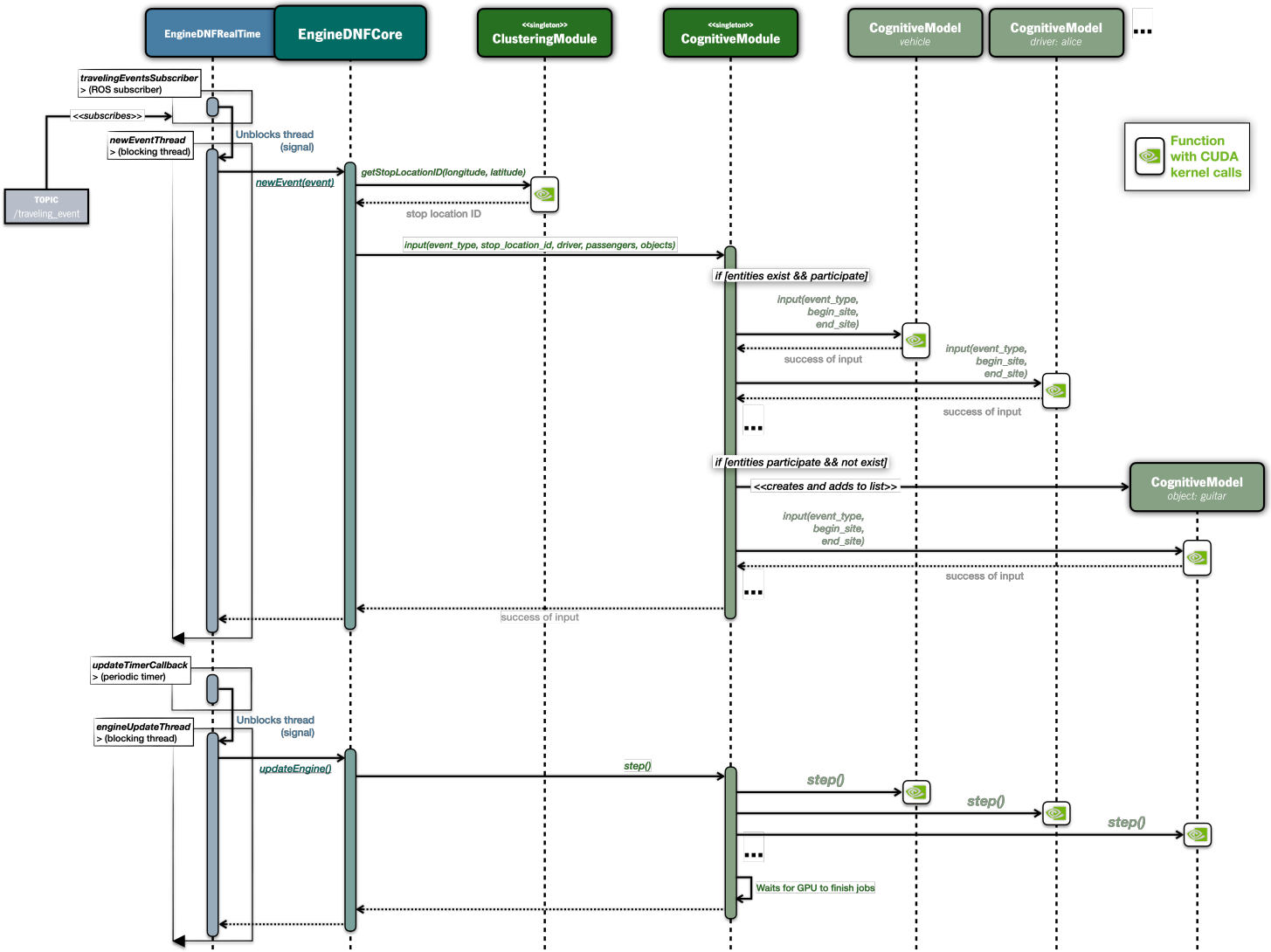


Figure 40: **Sequence diagram illustrating the workflow within the** Engine Node **for processing traveling events and the periodic model updates.**

Within each $\mathrm{CognitiveModel}$, the $\mathrm{step}()$ method is executed asynchronously, leveraging CUDA kernels for parallel computation. Since these kernel calls are non-blocking, the function returns immediately while the GPU continues processing. Consequently, at the end of the $\mathrm{step}()$ execution in $\mathrm{CognitiveModule}$, the system explicitly waits for all CUDA kernel jobs to complete before proceeding, ensuring synchronized updates across all active cognitive models.

The sequence diagram in Figure 41 illustrates the periodic retrieval and publication of recall readouts (predictions) from active cognitive models. At the end of each minute, the $\mathrm{cogmodelReadoutThread}$ is unblocked, triggering the $\mathrm{cognitiveModuleReadout}()$ function within $\mathrm{EngineDNFCore}$. This function calls $\mathrm{modelsReadouts}()$ from the $\mathrm{CognitiveModule}$, which iterates through all active cognitive models to extract peaks from their respective Anticipation Recall fields (see Figure 30). The recall peaks, representing predicted stop locations and expected times, are retrieved via $\mathrm{getRecallPeaks}()$ and subsequently published on the $/\mathrm{recall_readouts}$ topic, making them available to other system components requiring cognitive model predictions.

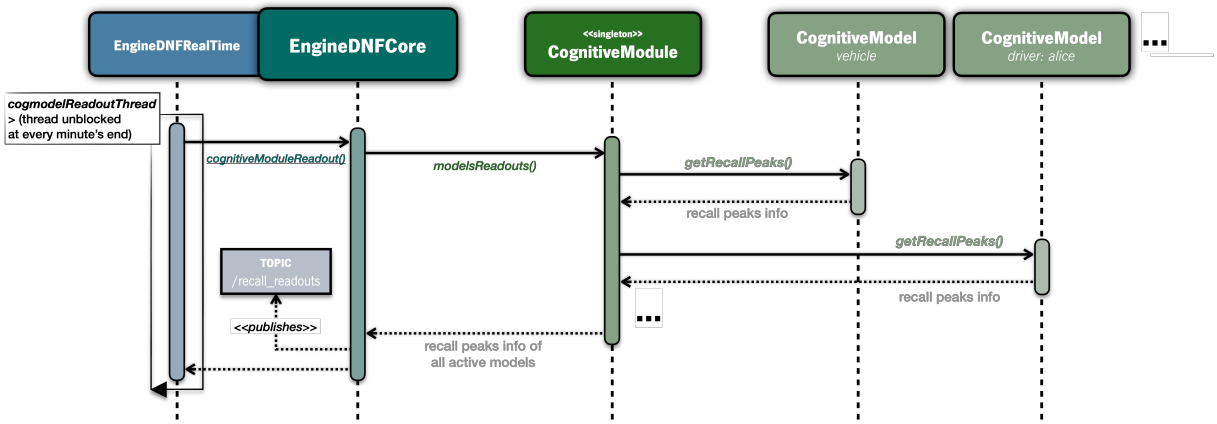


Figure 41: **Sequence diagram illustrating the periodic readout process of the cognitive models.**

A fundamental component of the $\mathrm{CognitiveModel}$ class is the $\mathrm{step}()$ method, which executes a single update step of the neural fields. GPU acceleration, through CUDA, is employed to efficiently handle large-scale parallel computations. The following breakdown details the execution flow of the $\mathrm{step}()$ method, emphasizing key computations and GPU-accelerated operations:

1. **Compute GPU execution parameters:** The number of CUDA blocks required for parallel execution is determined based on the fields' size.

2. **Transform field responses (e.g., field firing rates) to the frequency domain:** A Fast Fourier Transform (FFT) is applied to the field firing rate data, converting them to the frequency domain for efficient convolution operations. This transformation is performed using the $\mathrm{cufftExecC2C}(\ldots)$

function from cuFFT (CUDA Fast Fourier Transform library), which computes the forward complex-to-complex Fourier transform. Since the neural field responses are stored in complex form, this ensures seamless execution of both forward and inverse FFT computations.

3. **Compute interaction effects using convolution in the frequency domain:** Neural field interactions are computed using frequency-domain element-wise multiplication of field responses with precomputed interaction kernels, significantly reducing computational complexity compared to direct convolutions over neural field activations in their original representation. This operation is executed in parallel using the developed CUDA kernel $\mathrm{modelConvolutionsGPU}$ (see Listing 5.1). The CUDA kernel processes the interaction responses of each neural field site independently, performing frequency-domain multiplications and storing the results for subsequent inverse transformation.

4. **Transform interaction results back to the original domain:** The Inverse Fast Fourier Transform (IFFT) is applied to convert interaction results from the frequency domain back to their original representation, allowing the next processing step to operate on the reconstructed neural field interactions. This transformation is performed using the cuFFT function $\mathrm{cufftExecC2C}(\ldots)$, now computing the inverse complex-to-complex Fourier transform.

5. **Update neural fields in parallel:** The CUDA kernel $\mathrm{modelStepGPU}$ (see Listing 5.1) executes a parallelized update of all neural field sites by applying the governing equations of the model. Note that the update step incorporates the precomputed convolution results for the interactions. Each CUDA thread is responsible for processing a single site within a cognitive model's neural fields, simultaneously updating both departure and arrival fields. For example, if a cognitive model has neural fields with 5000 sites, CUDA will assign one thread per site (or process multiple sites across threads in blocks, depending on hardware configuration), so each thread only updates the equations for one specific index across all 14 fields (7 for departure, 7 for arrival). This allows the model to process thousands of neural sites in parallel, significantly accelerating computation. Once updated, the corresponding firing rates (for each neural site) are computed and stored, ensuring that the processed responses are available for the next update step.

6. **Check input state:** When a traveling event occurs, a homogeneous temporary input is applied to the cognitive models, concretely to the Spatial Encoding field. So, at each time step, if the input is active, a verification process determines whether it is time to deactivate the applied input based on predefined temporal constraints.

All responses, interaction results, and neural fields in the $\mathrm{CognitiveModel}$ are stored as flattened 1D arrays rather than multi-dimensional matrices. Each field type (i.e., Spatial Encoding, Spatial-Temporal Encoding, Consolidation, Memory, Anticipation Recall, Tolerance Recall, and Recall Working Memory) is stored sequentially within a single contiguous array. Similarly, interaction results and responses are stored

in dedicated 1D arrays to maintain structured and predictable memory access patterns. To retrieve or modify a specific neural field site, the appropriate index is computed based on: (a) the field type; (b) the event type (arrival or departure); and (c) the site position within the field. For example, to access the activation value of a specific site in the Spatial Encoding field, the index is computed using the field type offset, the total number of fields, and the event type offset within the _fields_ array. A similar indexing scheme is applied to interaction results.

Listing 5.1: **Prototypes of CUDA kernels and auxiliary CUDA functions for neural field updates and interactions.**

```
/* |CUDA kernel| Performs convolutions (frequency domain) for the model interactions. */
___global___ void modelConvolutionsGPU(
        uint32_t interactions_size,
        float* constants,
        float2* interactions_result,
        float2* response_interactions,
        float2* kernels);

/* |CUDA function| Sigmoid function to get the response of a neuron ("firing" or not "firing") */
___device___ float sigmoidFunction(
    float beta,
    float activation,
    float theta = 0.000000);

/* |CUDA function| Performs a complex multiplication (complex numbers). */
___device___ float2 complexConvolutionMultiplication(
        float2 signal1,
        float2 signal2,
        float scale,
        uint8_t index);

/* |CUDA kernel| Performs an update step of the model. */
___global___ void modelStepGPU(
        curandState *state,
        uint16_t fields_size,
        uint32_t interactions_size,
        float* constants,
        float* fields,
        float* dyn_resting_levels,
        float* previous_memory_pattern,
        float* input,
        bool just_temporal_correction,
        float2* interactions_result,
        float2* response_interactions,
        int* is_encoding_event);
```

5.3.2 System interfaces

The system's nodes communicate through well-defined ROS interfaces, including topics and services. Table 8 summarizes the key ROS interfaces, detailing their types and roles.

Table 8: **Summary of the ROS interfaces.**

Interface Type	Name	Data Type	Description	Modes RT	Modes SIM
Topic	/traveling_event	TravelingEvent.msg	Publishes structured data about traveling events (departures/arrivals).	×	×
Topic	/vehicle_in	Location.msg	Publishes the vehicle's arrival location.	×	
Topic	/vehicle_out	Location.msg	Publishes the vehicle's departure location.	×	
Topic	/entity_in	Entity.msg	Publishes information about entities (e.g., drivers, passengers, or objects) entering the vehicle.	×	
Topic	/entity_out	Entity.msg	Publishes information about entities leaving the vehicle.	×	
Topic	/daytime	std_msgs/UInt16.msg	Publishes the current minute of the day.	×	×
Topic	/day	Day.msg	Publishes transitions between days of the week.	×	×
Topic	/recall_readouts	RecallReadouts.msg	Publishes raw predictions generated by the cognitive models.	×	×
Topic	/predictions	TravelingEvent.msg	Publishes structured information about predicted traveling events.	×	×
Topic	/duration_plot	std_msgs/Empty.msg	Triggers the visualization of the duration model's fields in a GUI.	×	×
Topic	/clustering_plot	std_msgs/Int8.msg	Triggers the visualization of the clustering model's fields around a specific stop location in a GUI.	×	×
Topic	/cogmodel_plot	PlotCogModel.msg	Triggers the visualization of a specific cognitive model's fields in a GUI.	×	×
Topic	/plots_off	std_msgs/Empty.msg	Signals active plots to be turned off.	×	×
Service	/get_models_addrs	GetModel.srv	Shares memory addresses of cognitive models, enabling access to their neural field data.	×	×
Service	/get_traveling_routines	GetTravelingRoutine.srv	Retrieves a list of traveling events from a specific dataset.		×

Several custom message (.msg) and service (.srv) definitions are employed to standardize the structure of data exchanged between nodes. These include key custom messages such as TravelingEvent.msg, Location.msg, Entity.msg, RecallReadouts.msg, Day.msg, and PlotCogModel.msg, as well as foundational messages like EventType.msg, RecallPeak.msg, and RecallPeaks.msg, which are embedded within other custom messages. For instance, TravelingEvent.msg integrates Day.msg and EventType.msg as fields, while RecallReadouts.msg references RecallPeaks.msg to structure prediction-related data. This hierarchical organization ensures a modular and consistent approach to data representation. Additionally, the system includes custom services like GetTravelingRoutine.srv and GetModel.srv. For brevity, this section details one representative example from each category: the TravelingEvent.msg message type (Listing 5.2) and the GetTravelingRoutine.srv service type (Listing 5.3).

Listing 5.2: **The *TravelingEvent.msg* message type.** This message structure encapsulates information about a single traveling event, including location, time, associated entities, and event type.

```
# This message represents a traveling event, containing information about the location, time,
# day, and associated entities (driver, passengers, objects).

float64 latitude          # Latitude of the location
float64 longitude         # Longitude of the location
string location_name      # Location name

EventType event_type      # Type of the traveling event (e.g., arrival, departure)

uint16 time               # Time of the event (minute of the day)
Day day_info              # Day information (day id, day name, day counter)
uint16 week_num           # Week number

string driver             # Name of the driver
string[] passengers       # List of passengers going in/out
string[] objects          # List of objects going in/out
```

Listing 5.3: **The *GetTravelingRoutine.srv* service type.** This service facilitates the retrieval of traveling events for simulation purposes, including details about the dataset file and a response containing the number and list of events.

```
# This service retrieves a list of traveling events from a specific dataset for simulation purposes.
# The request specifies the dataset file; the response returns the events.

# Request part
string routine_filename     # Name of the dataset file containing the traveling routines
---
# Response part
uint32 n_events             # Number of traveling events returned
TravelingEvent[] events     # Array of traveling events
```

5.3.3 System launching and parameters

The system is launched manually to allow precise control over its configuration and execution. Nodes are initialized using direct commands, with some accepting runtime arguments to configure their execution. While this manual launch process ensures fine-grained control during development, it can be easily automated with two launch files — one for each mode of operation — streamlining the deployment for real-time and simulation scenarios.

Inputs Node

- Launch command: `ros2 run routines_adas_system inputs`
- It does not require any runtime arguments.

- Launched in both Real-Time and Simulation modes.

Assistant Node

- Launch command: ros2 run routines_adas_system assistant
- It does not require any runtime arguments. Future updates may include arguments to enable or configure specific types of assistive functionalities.
- Launched in both Real-Time and Simulation modes.

Watch Node

- Launch command: ros2 run routines_adas_system watch --start_minute [START_MINUTE] --end_minute [END_MINUTE]
- Runtime arguments:

  --start_minute: (Optional) Specifies the minute of the day when the system begins tracking routines (default: 360, i.e., 6:00 AM).

  --end_minute: (Optional) Specifies the minute of the day when the system stops tracking routines (default: 1440, i.e., midnight).

- Launched exclusively in Real-Time mode.

Engine RT

- Launch command: ros2 run engine_rt [UPDATE_PERIOD]
- Runtime argument:

  [UPDATE_PERIOD]: (Mandatory) Specifies the engine's update period (or time step) in seconds (e.g., 0.2).

- This executable initializes two nodes: the engine_dnf_rt node and the visualization node.
- Launched exclusively in Real-Time mode.

Engine SIM

- Launch command: ros2 run engine_sim [DATASET_FILENAME] [UPDATE_PERIOD] --daynum [DAY_NUM] --day [DAY] --daybegin [STARTING_MINUTE] --dayend [ENDING_MINUTE]
- Runtime arguments:

[DATASET_FILENAME]: (Mandatory) Specifies the routines dataset file to simulate (e.g., *RoutineDataset1*).

[UPDATE_PERIOD]: (Mandatory) Defines the simulation time step in seconds (e.g., 0.5).

--daynum [DAY_NUM]: (Optional) Specifies the starting day number for the simulation. The system will iterate through subsequent days starting from this one.

--day [DAY]: (Optional) Specifies the day of the week to simulate (e.g., if set to *monday* only Mondays will be simulated).

--daybegin [STARTING_MINUTE]: (Optional) Specifies the starting minute of the day for the simulation (default: 360, i.e., 6:00 AM).

--dayend [ENDING_MINUTE]: (Optional) Specifies the ending minute of the day for the simulation (default: 1440, i.e., 24:00).

- This executable initializes two nodes: the engine_dnf_sim node and the visualization node.
- Launched exclusively in Simulation mode.

Beyond the runtime arguments detailed above, the system utilizes a set of YAML-based parameter files to define key settings and properties, including neural field properties, interaction kernels, and engine-level configurations. This modular parameterization enables fine-tuning and scalability, ensuring the system can adapt to diverse use cases and computational environments. Below is an overview of the parameter files used in the system:

- main_parameters.yml: Defines high-level system configurations, such as the number of consecutive occurrences required for an event to be classified as a habit.
- engine_dnf_core_parameters.yml: Focuses on settings specific to the engine's core, such as the maximum number of stop locations and sites per stop location.

- clustering_field_parameters.yml: Defines neural field parameters for the clustering model, such as the time constant and resting level.
- clustering_kernel_parameters.yml: Specifies the interaction kernel parameters for the clustering model, including attributes like widths and amplitudes.
- cogmodel_field_parameters.yml: Defines the parameters for each neural field of the cognitive model, including properties like resting levels and buildup rates.
- cogmodel_kernel_parameters.yml: Specifies the interaction kernel parameters for the cognitive model, such as widths and amplitudes.

- duration_field_parameters.yml: Defines neural field parameters for the duration model.
- duration_kernel_parameters.yml: Specifies the interaction kernel parameters for the duration model.

5.3.4 Filesystem organization

The routines_adas_system ROS package is structured to ensure maintainability and accessibility. As illustrated in Figure 42, the directory is divided into distinct folders for system components, datasets, and data storage, providing a clear organization for efficient navigation and development.

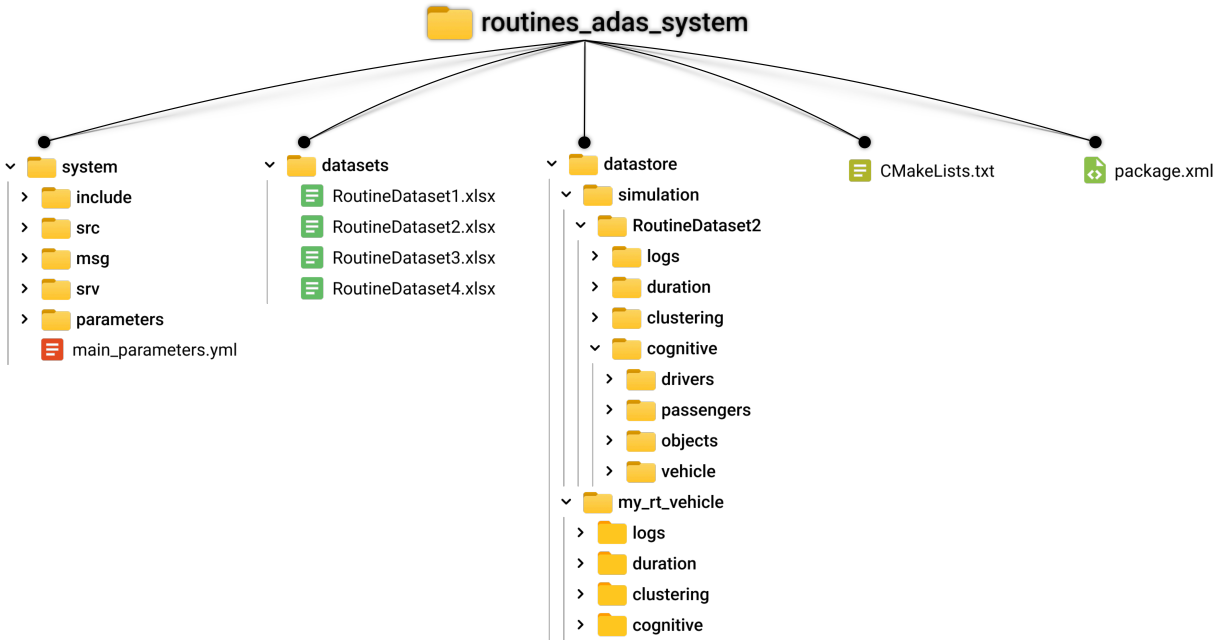


Figure 42: **Overview of the routines_adas_system ROS package directory structure.**

The datasets folder is where traveling routine datasets (for simulations) are placed. The datastore folder, initially empty, is dynamically populated during system execution in both Simulation and Real-Time modes. During simulations, a subfolder corresponding to the dataset being processed is created (e.g., RoutineDataset2 in this case). Whether in Simulation or Real-Time mode, the system stores neural patterns of the clustering model and neural patterns of cognitive models for various entities, as well as perception and prediction logs.

Figure 43 provides a detailed view of the system subfolder within the routines_adas_system ROS package. This subfolder is structured to encapsulate all core components, custom message and service definitions, and parameter configurations of the system. The include/ directory contains the header files for the engine's C++ implementation, while the src/ directory contains the source code files for all the system nodes.

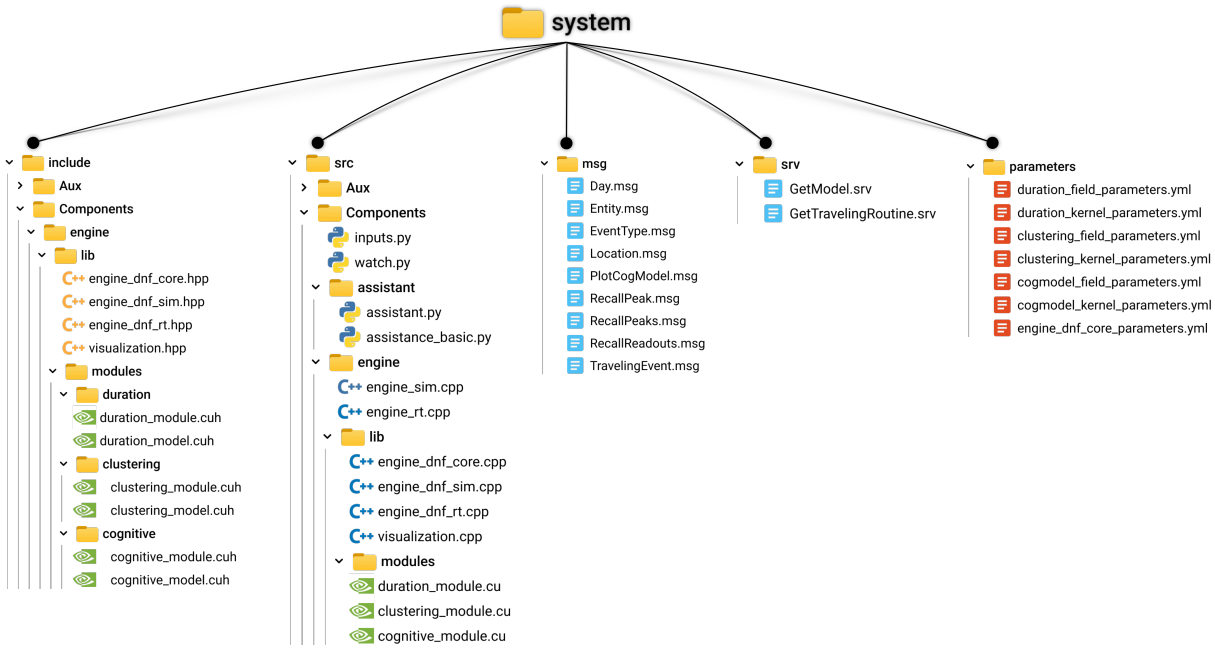


Figure 43: **Detailed directory structure of the** system **subfolder within the** routines_adas_system **ROS package.**

5.4 Performance insights

Understanding the computational performance of the proposed system is critical to ensure it meets real-time demands and remains scalable across varying workloads. To this end, the purpose of these tests is to evaluate the system's ability to process multiple cognitive models and larger field sizes efficiently, leveraging GPU acceleration through CUDA for these computations. Multiple instances of the cognitive model correspond to multiple individual entities, such as the vehicle, drivers, passengers, or objects. Larger field sizes represent the capacity to encode a greater number of stop locations, with each stop location represented as a non-overlapping slot within the cognitive model's neural fields. It is worth noting that a single cognitive model is composed of two independent sets of neural fields — one set dedicated to departures and the other to arrivals — resulting in a total of 14 fields (see model architecture in Figure 30).

The experiments were conducted using a simulated environment representing a single day's morning period (from 8:00 AM to 12:00 PM) with a fixed simulation time step of 0.5 seconds, resulting in a total of 28,800 steps per simulation. For each combination of configurations, the system was tested over this 4-hour window. The simulation parameters are provided in Appendix B. These tests were performed using the Simulation Mode, as it allows controlled testing and validation of the approach using pre-recorded datasets. Since in both Simulation and Real-Time modes the computational core is the same (EngineDNFCore), the results are indicative of the system's expected performance under real-time conditions. The number of models was selected to ensure a gradual progression, beginning with smaller workloads (1, 2, 3 models) and incrementally increasing to represent higher demands (5, 8, 13, 21, 34, 55, 89, 100 models). This

progression provides a balanced distribution of test points across a wide range of computational loads. Field sizes corresponded to 20, 40, and 60 stop locations, each with a fixed slot size of 100 sites (capable of accommodating six stable activation peaks to memorize up to six visits per stop location per day), resulting in cognitive models with 14 fields of size 2000, 4000, and 6000 sites, respectively.

For each combination of configurations, execution time was measured as the duration required to process all active cognitive models during a single time step. High-resolution timing was employed to ensure precision, with the timer starting immediately before calling the method $\mathrm{updateEngine}()$ from the $\mathrm{EngineDNFCore}$ class and stopping immediately after its completion. The measured execution times for each iteration were logged into a structured format, with entries including the time step, the number of models, the field size, and the measured execution time. These results were saved into spreadsheet files to facilitate further analysis and visualization. From the logged execution times, the average execution time for each configuration was calculated. The first 10% of iterations were discarded to account for system warm-up effects, ensuring the results accurately reflected steady-state performance.

The results are summarized in Figure 44 and Table 9, with the table serving as a detailed complement to the figure, providing exact values of the averages and standard deviations for better clarity and precision.

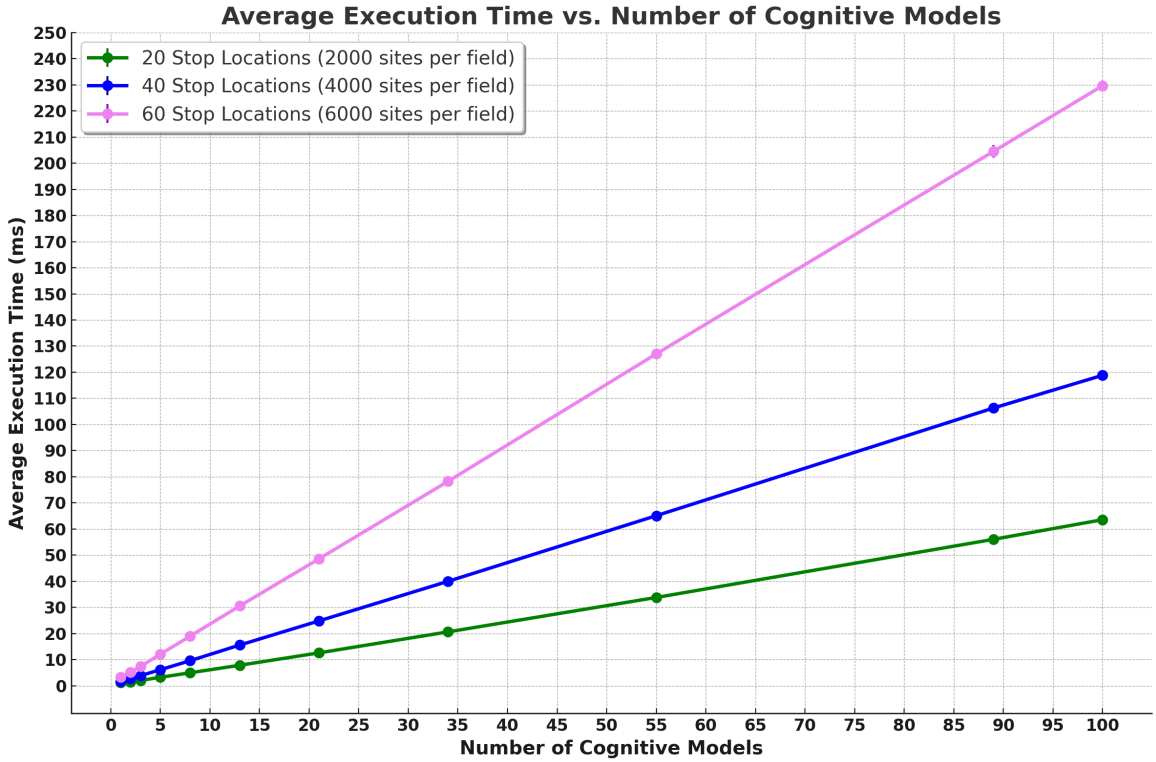


Figure 44: **Average execution time per time step versus the number of cognitive models for different field sizes.** Each cognitive model comprises 14 neural fields (seven for departures and seven for arrivals), with field sizes representing 20, 40, and 60 stop locations (2000, 4000, and 6000 sites per field, respectively). The system leverages CUDA-based GPU acceleration to efficiently handle the increasing computational load as the number of cognitive models and field sizes grow.

Table 9: **Average execution time (in milliseconds) per single update (time) step, along with the standard deviation, for different configurations of the number of cognitive models and stop locations.** The reported execution times represent the mean over a 4-hour simulated period, after discarding the initial 10% of iterations to account for system warm-up effects.

	20 Stop Locations	**40 Stop Locations**	**60 Stop Locations**
N. Models	*Avg. ± Std. Dev.*	*Avg. ± Std. Dev.*	*Avg. ± Std. Dev.*
1	1.18 ± 0.20	1.58 ± 0.19	3.25 ± 1.05
2	1.52 ± 0.20	2.80 ± 0.39	5.14 ± 0.45
3	2.06 ± 0.22	3.89 ± 0.38	7.38 ± 0.53
5	3.24 ± 0.38	6.14 ± 0.52	12.07 ± 0.75
8	4.97 ± 0.48	9.59 ± 0.64	18.99 ± 1.08
13	7.85 ± 0.62	15.58 ± 1.03	30.54 ± 0.57
21	12.60 ± 0.76	24.84 ± 1.11	48.59 ± 0.91
34	20.63 ± 0.83	39.97 ± 1.27	78.24 ± 1.07
55	33.80 ± 0.62	65.09 ± 0.95	127.00 ± 1.39
89	56.05 ± 1.31	106.30 ± 1.71	204.50 ± 2.41
100	63.57 ± 1.23	118.82 ± 1.75	229.65 ± 1.88

From the results, a clear linear relationship is observed between the number of cognitive models and the average execution time for all tested field sizes. This scalability reflects the consistent computational demands of processing individual cognitive models, where the system's architecture efficiently handles incremental workloads. For instance, in the case of 20 stop locations (2000 sites per field), the execution time starts at 1.18±0.20 ms for 1 model and increases to 63.57±1.23 ms for 100 models. Similarly, for 40 stop locations (4000 sites per field) and 60 stop locations (6000 sites per field), the execution times for 100 models rise to 118.82±1.75 ms and 229.65±1.88 ms, respectively. Comparing across field sizes, larger fields introduce proportionally higher execution times due to the increased computational complexity associated with processing more neural sites. This trend is particularly evident when comparing the execution times for the same number of models across field sizes. For example, at 21 models, the execution time is 12.60±0.76 ms for 20 stop locations, 24.84±1.11 ms for 40 stop locations, and 48.59±0.91 ms for 60 stop locations.

The linearity and stability of the results, as indicated by relatively low standard deviations, demonstrate the effectiveness of the proposed system's CUDA-accelerated implementation in maintaining predictable performance under varying computational loads. Furthermore, the results highlight the scalability of the system, which can handle both increasing numbers of cognitive models and larger field sizes while maintaining execution times suitable for real-time applications and facilitating high-speed simulations for testing and validation. While configurations with up to 100 cognitive models exceed typical real-world scenarios, they serve as critical stress tests to evaluate the system's robustness and scalability under

extreme workloads. Additionally, empirical studies of human mobility [206] suggest that individuals typically maintain a set of approximately 25 familiar locations. This finding aligns well with the chosen field sizes, such as 20 and 40 stop locations, which represent realistic configurations for everyday scenarios. Larger field sizes, such as 60 stop locations, were included to assess the system's scalability and adaptability to more complex or evolving mobility patterns. These scalability tests provide confidence that the system can support not only current real-world applications but also future scenarios requiring greater computational demands.

To further evaluate the performance benefits of the CUDA-accelerated implementation, a comparative analysis was conducted by replacing, in the $\mathrm{step}()$ method of the $\mathrm{CognitiveModel}$ class, the GPU-based $\mathrm{modelStepGPU}(...)$ method with a sequential CPU-based implementation, $\mathrm{modelStepCPU}(...)$. The $\mathrm{step}()$ method encapsulates the core computational workload for updating the neural fields of the cognitive model during each time step. As a reminder, it involves two main parts:

(a) the convolution of the neural fields' firing rates with interaction kernels to get fields interactions, executed using the $\mathrm{modelConvolutionsGPU}(...)$ CUDA kernel;

(b) the fields update step performed by the $\mathrm{modelStepGPU}(...)$ CUDA kernel, which updates each field site in parallel.

With this change, whenever the $\mathrm{updateEngine}()$ method of the $\mathrm{EngineDNFCore}$ class is called, it runs the $\mathrm{modelStepCPU}(...)$ function within the $\mathrm{step}()$ method of every active cognitive model. The $\mathrm{modelStepCPU}(...)$ function replaces the parallel processing of CUDA threads in $\mathrm{modelStepGPU}(...)$ with a loop iterating over all neural field sites, updating them sequentially. While the convolution process remains GPU-accelerated — utilizing FFT operations via $\mathrm{cufftExecC2C}(...)$ and frequency-domain element-wise multiplication via $\mathrm{modelConvolutionsGPU}(...)$ — this adjustment isolates the performance impact of GPU parallelization on the fields update process, providing insights into the scalability and efficiency of the proposed system under varying configurations. Specifically, the analysis compared execution times for 1 and 13 cognitive models across field sizes corresponding to 20, 40, and 60 stop locations.

The comparative results, highlighting the execution times for both GPU and CPU implementations across varying configurations, are presented in Figure 45. The results presented clearly illustrate the performance benefits of the CUDA-accelerated implementation compared to the sequential CPU-based approach. For all configurations, the GPU implementation demonstrates significantly lower execution times, with the gap between GPU and CPU performance widening as the number of stop locations and cognitive models increases. For a single cognitive model, the execution time for the GPU implementation remains consistently low, increasing only slightly with larger field sizes (e.g., 1.18 ms for 20 stop locations and 3.25 ms for 60 stop locations). In contrast, the CPU implementation exhibits a pronounced increase in execution time, reaching 10.10 ms for 20 stop locations — a nearly 10-fold difference compared to the GPU implementation.

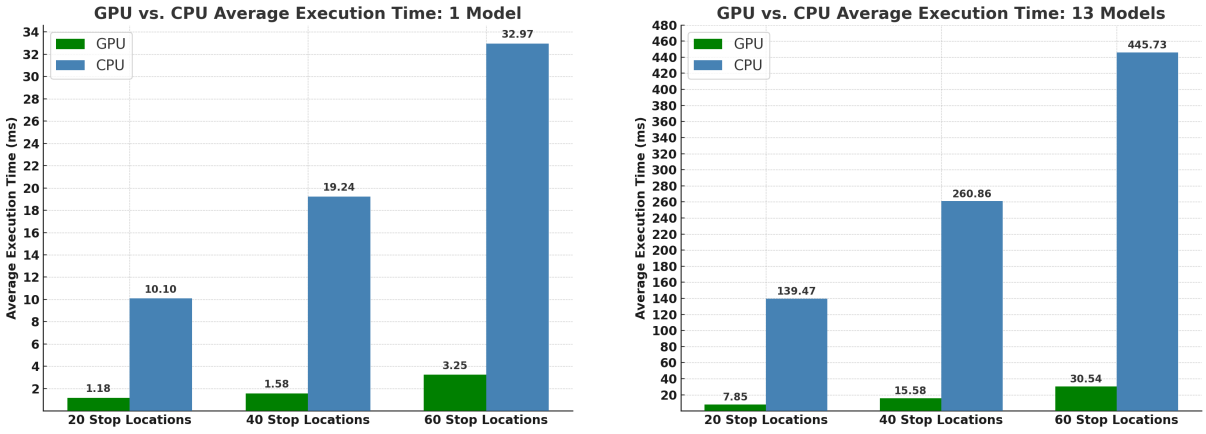


Figure 45: **GPU vs. CPU execution time comparison across varying configurations.** Average execution time per time step (in milliseconds) is compared between the CUDA-accelerated GPU implementation and the sequential CPU-based approach for 1 and 13 cognitive models with field sizes corresponding to 20, 40, and 60 stop locations.

To provide a concrete example of this performance disparity, consider the time required to simulate 4 weeks (28 days) of morning routines (8:00 AM to 12:00 PM) with a fixed time step of 0.5 seconds, resulting in 28,800 iterations per day. Using the GPU implementation with 1 cognitive model and 20 stop locations, the execution time per iteration is 1.18 ms, leading to a total simulation time of approximately:

$$\frac{28\,\text{days} \times 28800\,\text{iterations/day} \times 1.18\,\text{ms/iteration}}{1000\,\text{ms/s} \times 60\,\text{s/min} \times 60\,\text{min/hour}} \approx 0.27\,\text{hours} \approx 16\,\text{minutes}.$$

In contrast, using the CPU implementation for the same configuration, with an execution time of 10.10 ms per iteration, the total simulation time would be:

$$\frac{28\,\text{days} \times 28800\,\text{iterations/day} \times 10.10\,\text{ms/iteration}}{1000\,\text{ms/s} \times 60\,\text{s/min} \times 60\,\text{min/hour}} \approx 2.26\,\text{hours} \approx 2\,\text{hours and } 16\,\text{minutes}.$$

This example highlights the reduction in simulation time enabled by GPU acceleration, reducing the total time from over 2 hours to just 16 minutes. Such efficiency is crucial for accelerating the testing and validation process, particularly when iterating over extended simulation periods. Furthermore, the significant increase in execution time with the CPU implementation, particularly as the number of models and stop locations increases, demonstrates a potential constraint when scaling to more demanding workloads. While it may still meet real-time requirements under limited configurations, the observed performance suggests that GPU acceleration provides a more scalable and reliable solution for maintaining consistent timing under varied operational demands. It is important to acknowledge, however, that the modification from $\text{modelStepGPU}(...)$ to $\text{modelStepCPU}(...)$ results in each active cognitive model being updated sequentially. The CPU implementation could be further optimized by employing multithreading to update

cognitive models concurrently, or even finer-grained multithreading within each cognitive model, which could reduce execution times. However, these approaches would add considerable complexity to the implementation.

Still, it is also worth noting that the heaviest computational component — the convolution operations for field interactions — remains GPU-accelerated in both implementations. A full CPU-based implementation that includes convolution would result in significantly longer execution times, further emphasizing the computational challenges of sequential processing.

5.5 Conclusion

This chapter details the ROS-based implementation of the neurodynamic-based assistance system for learning and predicting traveling routines, designed to operate in both Real-Time and Simulation modes on the NVIDIA Jetson Nano. The $\mathrm{Engine\ Node}$ serves as the system's core component, integrating the DNF approach for learning and predicting traveling routines proposed in this thesis. Its implementation leverages GPU acceleration to ensure computational efficiency, enabling the system to process multiple cognitive models in parallel.

The system was implemented using ROS 2 to leverage its modular architecture, which enables encapsulation, scalability, and structured inter-node communication. The use of custom message and service definitions ensures efficient and well-defined data exchange. The modularity not only simplifies system integration but also enhances adaptability, allowing seamless operation across both Real-Time and Simulation modes.

The NVIDIA Jetson Nano was selected for its affordability and efficiency, demonstrating that the system can achieve real-time performance within a resource-constrained environment. While more powerful NVIDIA platforms exist, such as the Jetson Orin or Jetson Xavier series, the Nano was specifically chosen to validate the system's computational efficiency and scalability on the most cost-effective hardware in the NVIDIA embedded lineup.

The results in this chapter demonstrate that GPU-accelerated computations enable the system to maintain real-time execution, even as the fields' size and number of active cognitive models increased. These performance insights were obtained using the Simulation mode, which shares the same computational core ($\mathrm{EngineDNFCore}$) as the Real-Time mode. This unified design ensures that findings from Simulation mode are directly applicable to real-time execution. Notably, in Section 4.4, *Experimental Results* from Chapter 4, *A Dynamic Neural Field Approach for Intelligent Cockpits: Online Learning and Prediction of Traveling Routines*, the Simulation mode validated its functionality by simulating the traveling routines of four vehicles (and their respective entities). This facilitated an evaluation of the DNF approach's prediction performance, highlighting its ability to learn and anticipate traveling routines under realistic conditions.

A comparative analysis with a CPU-based implementation further validated the necessity of GPU acceleration. While the CPU implementation provided a functional benchmark, it was unable to scale efficiently with an increasing number of cognitive models and larger neural fields. This underscores the importance of leveraging GPU parallelization for maintaining system performance under computationally intensive workloads. Additionally, the computational efficiency gained through GPU acceleration enables significantly faster simulations, allowing extensive testing and validation of the system across large datasets and multiple scenarios within a reduced timeframe.

Although the $\mathrm{Assistant\ Node}$ does not yet generate assistive outputs, it serves as a structured framework for future development. At present, it organizes raw cognitive model predictions into structured traveling event predictions — including the type, location, and time of the event, as well as the involved driver, passengers, and objects — before publishing them. Additionally, predictions are stored in spreadsheet files, facilitating further analysis and system validation. The node is designed with extensibility in mind, allowing for the future integration of diverse assistive functionalities. These could include personalized in-vehicle notifications, contextual reminders (e.g., forgotten objects), or adaptive comfort settings based on expected occupants.

Chapter 6

Conclusion

6.1 Thesis summary and highlights

What. This thesis proposes an intelligent system designed to learn and predict traveling routines in cognitive vehicles. It is composed of a neurodynamics-based approach based on Dynamic Neural Fields (DNFs) that enables continuous adaptation to evolving mobility patterns. It allows a vehicle to learn, memorize, and predict when and where it departs and arrives, while also capturing the mobility habits of its drivers, passengers, and transported objects. This learning process occurs implicitly, all without explicit user feedback.

To assess its feasibility, the approach was implemented as a ROS-based prototype system with two operational modes: Real-Time and Simulation. This prototype not only evaluated the predictive capabilities of the DNF approach but also demonstrated the potential for running multiple cognitive models efficiently on embedded hardware. Furthermore, the system provides a structured foundation for future development, paving the way for the implementation and testing of various assistive features, such as proactive reminders for forgotten objects or automatic comfort adjustments based on expected occupants.

This thesis presents notable implications for both the field of intelligent transportation systems and Dynamic Neural Fields studies:

1. Demonstrates the effectiveness of leveraging neurodynamics in the development of intelligent vehicle systems, enhancing its utility in real-world applications. Moreover, the dynamic adaptation capability of DNFs opens new avenues for research and development in intelligent transportation systems (e.g., learning driver behavior, urban environmental perception and decision-making).

2. Proposes a type of assistance system with the potential to enhance the overall driving experience, convenience, and personalization for users.

3. Expands the scope of DNF research and highlights its versatility across different domains.

This thesis was carried out as an industrial PhD in collaboration with Bosch Car Multimedia S.A. Portugal. The research was also developed within the scope of the joint project “Easy Ride: Experience is Everything”, a partnership between the University of Minho and Bosch focused, among others, on advancing user-centered in-vehicle technologies. The topic explored in this work aligned with Bosch’s strategic interest in enhancing personalized mobility experiences. While the work was also explored within an academic environment, the industrial collaboration provided valuable insights and helped align the proposed approach with real-world challenges and potential application scenarios.

Why. Intelligent in-vehicle assistants are emerging as a key component of modern personalized ADAS, offering predictive and adaptive capabilities that enhance the driving experience [2, 3, 11, 14]. Among these, recommendation systems have gained attention for their ability to learn user preferences and

provide proactive suggestions. For instance, commercial systems such as Mercedes' MBUX Hyperscreen [7, 12] generate destination recommendations once a driver profile is selected, while TOMTOM's predictive system [13] focuses on the habitual destinations of the vehicle rather than adapting to individual user behaviors.

Beyond predicting destinations, learning and predicting full traveling habits — including when and where departures and arrivals occur — expands the scope of possible assistive features. Indeed, studies on human mobility patterns suggest that people tend to revisit the same locations in structured, predictable ways [16–18], making trip prediction an increasingly valuable feature for intelligent vehicle systems. While next-destination prediction has been widely explored using probabilistic and machine learning models, most of these methods rely on large, stationary datasets to develop stable predictions, not accounting for real-world scenarios where mobility habits evolve over time or where data is incrementally acquired. Furthermore, significantly less attention has been given to predicting full trip information [20, 21], which requires forecasting not just where to go next but also when to depart or arrive. While challenging, this added complexity is crucial for anticipatory assistance and context-aware automation in intelligent vehicles.

Addressing these challenges benefits from a computational framework capable of learning, adapting, and predicting continuously. DNFs provide a biologically inspired solution that models activity as evolving peaks of neural activation, allowing the system to incrementally integrate new information while dynamically fading outdated patterns. Unlike most deep learning models, which primarily store knowledge in static synaptic weights and often require retraining to integrate new patterns, the activation-based learning mechanisms of DNFs naturally support online learning, gradual forgetting, and continuous adaptation to new information. This capacity for dynamic reconfiguration and self-organization, paired with their scalability in multi-field architectures, makes DNFs particularly well-suited for modeling human-like cognitive processes in intelligent vehicle systems. By leveraging DNFs, this thesis establishes a framework for a system that can not only predict when and where a vehicle will travel but also adapt to new users, changing routines, and real-world uncertainties — all without explicit retraining.

How. To develop an intelligent system capable of learning and predicting traveling routines, the research was conducted in incremental phases, each building upon the previous one. This structured approach ensured that the proposed solution evolved systematically — from data collection to the approach development and finally to system implementation and evaluation:

PHASE I: Data collection

As presented in Chapter 3, this thesis developed a tailored mobile application designed to record real-world traveling routines and generate structured datasets of vehicle users' mobility habits. Unlike surveys, which rely on self-reporting, or vehicle sensor installations, which are costly and complex, a mobile-based approach offers scalability, accessibility, and privacy-conscious data collection. Built using Flutter and

Google Firebase, the application records key trip details, including departure and arrival locations, vehicle occupancy (drivers, passengers, and objects), and in-vehicle settings. The collected data is stored in structured files, allowing researchers to compile and filter datasets for intelligent vehicle applications. Crucially, the goal of these datasets was not to train models but to validate and simulate the DNF-based approach. Although the application employs anonymous user IDs instead of personally identifiable information, it records location data, which alone can be sufficient to infer a person's identity. To address this, strict data privacy measures were enforced, including explicit informed consent. Users are explicitly informed of these considerations every time they use the app, ensuring ethical and transparent usage.

PHASE II: DNF-based approach

As presented in Chapter 4, a DNF-based approach was developed to continuously learn and predict the spatio-temporal properties of traveling routines in intelligent vehicles. Instead of designing separate models for specific entities (e.g., vehicle, driver, passengers, objects), a generic cognitive model was created and later instantiated for multiple entities, forming the DNF-based approach. This allows the system to autonomously learn when and where a vehicle departs and arrives, along with the participation of different drivers, passengers, and transported objects.

The development process followed an incremental validation strategy. Initially, mental simulations were conducted to define the expected behavior of the cognitive model, drawing inspiration from previous work [91, 119]. The model emulates computations in the brain, where self-sustained neural activation peaks encode both spatial and temporal information of perceived events. So, a series of traveling events experienced throughout the day is represented by stable multi-peak patterns, with an activation gradient distributed over neural subpopulations encoding time. The sequential recall of these stored spatio-temporal patterns then yields predictions of future traveling events. Building on these insights, a cognitive model was subsequently implemented in MATLAB for parameter tuning and initial validation. Finally, the full DNF-based approach was implemented and integrated into an embedded ROS-based system (PHASE III). Since all cognitive models share the same underlying dynamics, the system not only learns and predicts the vehicle's departures and arrivals but also maintains synchronized sequences for drivers, passengers, and objects, ensuring that all elements associated with a trip are anticipated simultaneously. To structure the spatial representation of traveling routines, the DNF approach also incorporates a clustering model, which groups geographical locations into stop locations that serve as reference points for the cognitive models. This ensures that spatial information is not encoded as raw GPS coordinates but rather as meaningful place representations, enabling consistent and structured learning of traveling patterns while also reducing computational costs.

The approach continuously tracks daily traveling routines, reinforcing frequent events while naturally forgetting infrequent ones. A memory trace with activation thresholds ensures that routine events consolidate when repeated over consecutive weeks, while extended absences lead to their decay. Additionally,

the approach prioritizes sequence learning with strong time constraints, meaning that events deviating significantly from their typical timings are not considered part of routine behavior. This design choice highlights one of the advantages of the dynamical systems approach over existing methods in representing the continuous nature of human choices and prediction times.

Experimental evaluations, conducted through the embedded prototype's Simulation Mode, confirmed the predictive capabilities of the cognitive model, achieving F1-scores between 70% and 90% for most days of the week in predicting trip information. The system effectively captures consistent mobility patterns, though unexpected or infrequent events introduce prediction errors. These results validate the potential of the DNF framework for learning and predicting mobility patterns without requiring retraining or explicit user feedback. Unlike deep learning approaches, which demand frequent retraining to accommodate evolving mobility patterns, DNFs enable seamless, on-the-fly adaptation. This allows new destinations, drivers, passengers, and objects to be dynamically integrated into the system without costly retraining or dependence on large datasets. Additionally, DNFs offer an interpretable learning process, where activation patterns directly reflect encoded knowledge. This allows model parameters to be adjusted dynamically based on the application context, ensuring adaptability while maintaining transparency. In contrast, deep learning models function as black-box systems [20], where knowledge is embedded in distributed weight matrices, making their decision-making process inherently opaque.

PHASE III: ROS-based system

To advance the proposed DNF-based approach from conceptual validation to real-world feasibility, this thesis developed a modular embedded prototype, as presented in Chapter 5. The primary objective was to establish a functional proof-of-concept system that not only validated the predictive capabilities of the DNF approach but also paved the way for future advancements toward higher TRLs.

A key design principle was the implementation of a shared computational core, ensuring that the same cognitive model management, learning dynamics, and prediction mechanisms were employed in both Real-Time and Simulation modes. This unified architecture allowed the DNF approach to be tested in Simulation mode, where structured datasets enabled controlled evaluation, while also ensuring its theoretical applicability in real-world deployment, given that the core computational logic remains unchanged.

The system was implemented as a ROS-based prototype, utilizing ROS 2 primarily for inter-node communication, modularity, and structured data handling. ROS 2 facilitated the separation of system components into distinct nodes, ensuring scalability, extensibility, and maintainability. The developed prototype can thus serve as a foundation for future assistive applications. The implementation was deployed on the NVIDIA Jetson Nano, selected for its affordability, power efficiency, and CUDA support, enabling parallelized neural field updates through GPU acceleration. Performance evaluations demonstrated that the system could handle increasing cognitive loads while maintaining execution times suitable for real-time operation.

The decision to employ GPU over FPGA is based on the complexity and dynamic requirements of the proposed approach. A recent FPGA-based work that implemented a simplified cognitive model [207], revealed that even reduced models face restrictive resource demands and a loss of robustness and flexibility in FPGA implementations. In contrast, this thesis proposes a cognitive model with 14 neural fields (covering both departures and arrivals) to capture spatial and temporal patterns of events, and further instantiates this model for multiple entities — namely the vehicle, drivers, passengers, and objects. This multi-instance architecture involves continuous adaptation, iterative updates, and irregular, state-dependent memory accesses, all of which pose significant challenges for FPGA-based implementations. Moreover, while neuromorphic computing approaches that combine DNFs with spiking neurons have been explored [208], these systems still face challenges such as stochastic behavior and limitations in scalability and stability guarantees, despite their promise for improved real-world applicability [207, 208]. Moreover, transitioning to neuromorphic hardware from conventional DNF models often requires significant architectural changes to fully capitalize on the benefits of spiking systems.

GPUs with CUDA support offer the necessary flexibility and parallel processing capabilities to efficiently manage the dynamic operations in real time. The use of cuFFT enables efficient frequency-domain convolutions, while CUDA kernels provide optimized field updates across thousands of neural field sites, ensuring not only real-time feasibility but also the ability to conduct high-speed simulations for validation and testing without extensive hardware reconfiguration. Therefore, given the increased complexity, multi-instance nature, and resource demands of the approach, GPU acceleration was selected as the optimal platform for achieving both scalability and computational efficiency.

6.2 Limitations and future work

While the proposed system demonstrates strong capabilities in learning and predicting spatio-temporal mobility patterns in intelligent vehicles, some challenges and open questions remain. The following segments discuss key limitations and outline promising directions for future research.

DNF-based approach

1. Prediction errors occur when unexpected or persistent changes in traveling patterns arise. The current cognitive model cannot differentiate between one-off anomalies and genuine routine changes. Future work could associate a confidence level with predictions, linked to the activation level of the memory trace (Consolidation field, u_c) representing accumulated evidence of event recurrence over time. Additionally, collecting driver feedback to filter out unexpected, non-routine events could provide a clearer evaluation of the model's true predictive capabilities. Alternatively, establishing a benchmark for the best theoretically achievable performance — accounting for the unpredictability of such events — would offer a more realistic comparison than aiming for 100%.

2. The approach uses predefined, fixed time windows for expected events. Events occurring outside these windows are treated as new, which affects predictions when irregular fluctuations arise, such as those observed in Dataset 2 on Saturdays (for more detail, see *Performance* within Section 4.4, *Experimental results*). In this case, the vehicle frequently traveled to the same locations, but the model failed to identify a routine because the actual event times often missed the expected window by just a few minutes. Future work could implement event-specific time windows by tracking the minimum and maximum peak amplitudes, and dynamically adjusting anticipation-tolerance offsets for the recall fields. This would help balance the need to accommodate variability in event timing while maintaining accurate routine predictions for effective assistance.

3. The spatial clustering parameter σ is fixed, which may not suit all scenarios. For instance, a family vehicle may tolerate larger clustering radii, while a professional service vehicle may require finer spatial clusters. The issue of task-dependent clustering could be addressed using the neural integrator model described in [197, 202]. This includes an extension to the temporal domain to distinguish traveling events occurring at the same spatial location but at different times of the day.

4. The cognitive model uses a memory trace with activation thresholds to track habitual events, parameterized by growth and decay rates. In the experimental evaluations using data recorded over 11 weeks, a traveling event is considered routine if it occurs for two consecutive weeks ($n = 2$) and is no longer considered routine after two weeks of non-occurrence ($m = 2$). While this choice balances sensitivity and stability — ensuring that one-off events are not prematurely classified as habits, and occasional absences do not lead to immediate forgetting — it raises questions about the generalizability of these fixed thresholds. Future research could explore the impact of varying n and m values (e.g., $n = 3$, $m = 4$) to optimize habit formation and forgetting dynamics for different contexts. Additionally, implementing dynamic, event-specific thresholds could allow the model to adapt based on the nature of each event. For instance, frequently recurring events like daily commutes may require lower thresholds, while sporadic activities might demand stricter criteria.

5. The current DNF approach operates on a daily time scale, meaning that traveling routines are learned and recalled for each specific day. At the end of each day, the memory activation patterns are stored, and the patterns for the upcoming day are loaded. While this enables effective tracking of daily routines, it poses limitations in recognizing longer-period routines, such as biweekly or monthly events. For example, a biweekly event (e.g., going to a specific destination every two weeks) would never be classified as a routine, as its occurrences do not meet the requirement for consecutive activations within a daily-based framework. A possible solution is to incorporate cognitive models whose learning and recall processes operate on a weekly time scale, storing and loading memory patterns at the end of each week rather than each day. This would allow the models to capture event periodicities beyond strict daily cycles, improving their ability to recognize routines that extend over longer time frames. Additionally, integrating dynamic, event-specific thresholds for habit formation

and forgetting would further enhance the system's adaptability, ensuring that event classification aligns with the true periodicity of events rather than being constrained by fixed thresholds.

6. While the current approach emphasizes implicit learning without explicit user feedback, future work could explore integrating the user into the learning loop. Direct, bidirectional interaction between the system and the user could enhance its ability to accurately identify and adapt to routine behaviors. For instance, users could confirm or reject predicted routines, helping the system differentiate between one-off anomalies and genuine routine changes. This feedback mechanism could also dynamically adjust habit formation and forgetting thresholds, complementing the event-specific parameterization and time-scale adjustments discussed in the two previous points. By involving users in this process, the model could achieve more personalized and accurate predictions. Additionally, user-in-the-loop feedback could accelerate the learning of new routines or adjust for exceptions (e.g., holidays, temporary schedule changes) without compromising overall system stability.

ROS-based system

1. In the current implementation, arrival and departure events are stored and recalled as independent sequences, which may lead to inconsistencies. Future implementations should ensure that a departure prediction is only considered if its location matches the preceding arrival event.

2. While the current implementation relies exclusively on DNF models for learning and predicting traveling routines, future work could explore an assistance system with hybrid approaches, integrating, e.g., DL methods. The flexibility and interpretability of the few-shot learning process inherent to DNF models could complement the ability of DL models to capture long-term dependencies in mobility patterns, especially as more mobility data becomes available. This hybrid approach could enhance prediction performance by enabling the system to adapt quickly to new routines while maintaining robust knowledge of established, complex mobility behaviors. However, integrating such hybrid models within the ROS pipeline would require optimizing computational efficiency to ensure real-time performance. This includes exploring techniques such as lightweight neural architectures, hardware acceleration (e.g., GPU-based inference), and distributed processing.

3. The ROS-based system is designed for modularity and extendibility, allowing the $\mathrm{Assistant\ Node}$ (or nodes) to subscribe to the $\mathrm{Engine\ Node}$'s spatio-temporal predictions. These raw predictions, obtained through the recall process of activity patterns and published at a specific frequency, provide insights into upcoming traveling events for each active cognitive model, including departure and arrival times, locations, and whether events are considered routine. Future research could explore assistive functionalities that leverage these predictions. For instance, in private vehicles, anticipating departure times could allow notifying users in advance, helping them avoid delays and ensuring they carry essential objects; optimizing vehicle sharing among multiple drivers; and pre-adjusting

vehicle settings for enhanced comfort. In autonomous ride-hailing services, such as Waymo [15], predicting user routines could automate pickups and improve fleet management efficiency. To enhance accessibility, assistive predictions could be delivered via mobile apps, SMS, emails, or directly through the vehicle's infotainment or voice interface. Future work can refine how predictions are utilized, making intelligent mobility assistance more intuitive, pertinent, and adaptive.

4. While the proposed assistance system has been evaluated in simulation with real data, future work should focus on conducting thorough real-world testing in actual vehicles. This includes assessing system performance, usability, and human factors to ensure seamless integration into daily mobility routines. Key evaluation aspects could involve analyzing system responsiveness to dynamic environmental changes, and evaluating user acceptance and interaction with assistive features. Validating the system in real-world settings is crucial for improving robustness, optimizing human-machine interaction, and ensuring that assistive functionalities enhance — not disrupt — the user experience.

Appendix A

Mathematical formulation of the approach

A.1 Clustering model

The clustering of nearby GPS locations in stop locations is performed by a neural integrator model [46, 197], formalized by two coupled two-dimensional field equations:

$$\tau_{clust}\frac{\partial u_{clust}(x,y,t)}{\partial t} = -u_{clust}(x,y,t) + v_{clust}(x,y,t) + I_{clust}(x,y,t) + \int\int w_{mex}(x-x',y-y')f_\theta(u_{clust}(x',y',t))\,\mathrm{d}x'\,\mathrm{d}y', \tag{A.1}$$

$$\tau_{clust}\frac{\partial v_{clust}(x,y,t)}{\partial t} = -v_{clust}(x,y,t) + u_{clust}(x,y,t) - \int\int w_{mex}(x-x',y-y')f_\theta(u_{clust}(x',y',t))\,\mathrm{d}x'\,\mathrm{d}y', \tag{A.2}$$

where $\mathbf{r} = (x, y) \in \mathbb{R}^2$ represent longitude and latitude.

A.2 Cognitive models

The model architecture consists of several coupled DNFs with specific functionalities. The mathematical formulation is explained in detail below. The fundamental processing principle of representing sequential, spatio-temporal information in an activation gradient over neural subpopulations in u_{ste} (Equations (A.4) and (A.5)) and u_m (Equations (A.7), (A.8) and (A.9)) has been proposed in [119] and tested in several sequential

Table 10: **Table of variables of the cognitive model.**

VARIABLE	DESCRIPTION
x	Metric dimension, $x \in \mathbb{R}$, divided into non-overlapping intervals or slots representing the different stop locations
Spatial Encoding	
u_{se}	Activation of the Spatial Encoding field
τ_{se}	Time constant of the Spatial Encoding field (constant)
h_{se}	Resting (or baseline) level of the Spatial Encoding field (negative constant)
I_{se}	Transient and homogeneous input to the Spatial Encoding field
c	Noise strength (constant)
ξ	Gaussian white noise
Spatio-Temporal Encoding	
u_{ste}	Activation of the Spatio-Temporal Encoding field
τ_{ste}	Time constant of the Spatio-Temporal Encoding field (constant)
h_{ste}	Resting (or baseline) level dynamics of the Spatio-Temporal Encoding field
k_{ste}	Activity growth rate for temporal encoding (constant)
h_{ste_0}	Default resting level of the Spatio-Temporal Encoding field (negative constant)
Consolidation	
u_c	Activation of the Consolidation field
$\lambda_{strength}$	Memory strength buildup rate (constant)
$\lambda_{strength_h}$	Additional memory strength buildup rate for habitual events (constant)
λ_{weak}	Memory weakening decay rate (constant)
λ_{forget}	Memory forgetting decay rate (constant)
a	Maximum memory strength (constant)
h_c	Resting (or baseline) level of the Consolidation field (negative constant)
θ_h	Habit threshold level (constant)
Memory	
u_m	Activation of the Memory field
τ_m	Time constant of the Memory field (constant)
h_m	Resting (or baseline) level dynamics of the Memory field
h_{m_0}	Default resting level of the Memory field (negative constant)
$h_{m_{inh}}$	Resting level to destabilize (forget) existing memory (negative constant)
$h_{m_{prev}}$	Previous (day of the week) resting level pattern
λ_{EMA}	EMA weight (constant)
Anticipation/Tolerance Recall	
u_{ar}/u_{tr}	Activation of the Anticipation/Tolerance Recall fields
τ_{ar}/τ_{tr}	Time constant of the Anticipation/Tolerance Recall fields (constant)
h_{ar}/h_{tr}	Resting (or baseline) level dynamics of the Anticipation/Tolerance Recall fields
$u_{m_{prev}}$	Previous (same day of the week) routine memory pattern
u_{day_start}	Zero-dimensional dynamic node representing the beginning of a new day
k_r	Activity growth rate defining the recall speed (constant)
h_{r_0}	Default resting level of the Anticipation/Tolerance Recall fields, which represents the duration of the daily routine, e.g., 24h (constant)
off_{ar}/off_{tr}	Recall temporal offset of the Anticipation Recall field (constant) and Tolerance Recall field (negative constant)
Working Memory	
u_{wm}	Activation of the Working Memory field
τ_{wm}	Time constant of the Working Memory field (constant)
h_{wm}	Resting (or baseline) level of the Working Memory field (negative constant)
Interactions	
w_{lat}	Connectivity function (or interaction kernel) of lateral inhibition
w_{osc}	Connectivity function (or interaction kernel) with oscillatory decay
f_θ	Heaviside step function with a threshold θ ($\overline{f_\theta}$ is the reverse of the step function, and defined by $1 - f_\theta$)

tasks including mobility data [91, 193]. Major extensions compared to previous work include the concept of a consolidation field representing a dynamic memory trace, u_c (Equation (A.6)), the implementation of a time window for the predictions in the recall fields, u_{ar} and u_{tr} (Equations (A.11)), the active forgetting principle, and the EMA mechanism (Equation (A.10)).

The one-dimensional neural fields are spanned over the metric dimension $\mathbf{r} = x \in \mathbb{R}$, which is divided into non-overlapping intervals or slots representing the different stop locations. Table 10 exhibits a comprehensive list of model variables.

The Spatial Encoding field, u_{se}, is given by

$$\begin{aligned}\tau_{se}\frac{\partial u_{se}(x,t)}{\partial t} &= -u_{se}(x,t) + h_{se} + I_{se}(x,t) + c\xi(x,t) \\ &+ \int w_{lat}(x-x')f_0(u_{se}(x',t))\,\mathrm{d}x' \\ &- \int w_{osc_1}(x-x')f_0(u_c(x',t))\overline{f_0(u_{ar}(x',t))}\,\mathrm{d}x' \\ &+ \int w_{osc_2}(x-x')f_0(u_c(x',t))f_0(u_{ar}(x',t))\,\mathrm{d}x' \\ &+ \int 0.5w_{osc_2}(x-x')f_{\theta_h}(u_c(x',t))f_0(u_{ar}(x',t))\,\mathrm{d}x' \\ &- \int w_{osc_3}(x-x')\overline{f_0(u_c(x',t))}f_0(u_{wm}(x',t))\,\mathrm{d}x'. \end{aligned} \tag{A.3}$$

Neural subpopulations representing a specific stop location (slot) receive a transient, homogeneous input $I_{se}(x,t)$. Gaussian white noise, $\xi(x,t)$, with strength $c > 0$, breaks the symmetry, leading to the evolution of a single peak shaped by the lateral interaction kernel w_{lat} (see Equation (4.2)). The mutually exclusive inputs from u_c and u_{ar}, shaped by w_{osc_1} and w_{osc_2} (see Equation (4.3)), cause, respectively, a localized inhibition when an event is not being recalled (by u_{ar}) and a preactivation when it is. A stronger preactivation is applied when the corresponding memory trace is above θ_h.

To enable stable multi-peak solutions in the Spatio-Temporal Encoding field, u_{ste}, a kernel with oscillatory decay w_{osc} (see Equation (4.3)), is used:

$$\begin{aligned}\tau_{ste}\frac{\partial u_{ste}(x,t)}{\partial t} &= -u_{ste}(x,t) + h_{ste}(x,t) \\ &+ u_{se}(x,t)f_0(u_{se}(x,t)) \\ &+ \int w_{osc}(x-x')f_0(u_{ste}(x',t))\,\mathrm{d}x'. \end{aligned} \tag{A.4}$$

A state-dependent resting level dynamics [91, 119, 193] guarantees that an evolving peak, initially triggered by localized activity from u_{se}, continuously ramps up at a constant rate $k_{ste} > 0$ when the activity reached threshold, otherwise it converges to $h_{ste_0} < 0$:

$$\begin{aligned}\frac{\partial h_{ste}(x,t)}{\partial t} &= k_{ste} f_0(u_{ste}(x,t)) \\ &\quad + \big(-h_{ste}(x,t) + h_{ste_0}\big)\,\overline{f_0(u_{ste}(x,t))}.\end{aligned} \tag{A.5}$$

The Consolidation field, u_c, is governed by

$$\begin{aligned}\frac{\partial u_c(x,t)}{\partial t} &= \lambda_{strength}\big(-u_c(x,t) + a\big) f_0(u_{se}(x,t)) \\ &\quad + \lambda_{strength_h}\big(-u_c(x,t) + a\big) f_0(u_{se}(x,t)) f_{\theta_h}(u_c(x,t)) \\ &\quad + \lambda_{weak}\big(-u_c(x,t) - a\big) f_0(u_{ar}(x,t)) f_0(u_{tr}(x,t)) \\ &\quad + \lambda_{forget}\big(-u_c(x,t) + h_c\big)\overline{f_0(u_c(x,t))}\,\overline{f_0(u_{se}(x,t))}.\end{aligned} \tag{A.6}$$

The memory trace builds up with the rate $\lambda_{strength} > 0$ towards its maximum strength $a > 0$ whenever suprathreshold activity representing an event in u_{se} exists. An additional buildup with rate $\lambda_{strength_h} > 0$ is applied if an event is considered habit, that is, $u_c(x,t) > \theta_h$. It decays with rate $\lambda_{weak} > 0$ whenever there is suprathreshold activity in both recall fields, meaning that the expected event has not occurred. Finally, the memory trace decays back to baseline level with the rate $\lambda_{forget} > 0$ when its activity falls below 0.

The equation of the Memory field, u_m, given by

$$\begin{aligned}\tau_m \frac{\partial u_m(x,t)}{\partial t} &= -u_m(x,t) + h_m(x,t) \\ &\quad + \int w_{osc}(x - x') f_0(u_m(x',t))\,\mathrm{d}x' \\ &\quad + \int w_{osc}(x - x') f_0(u_c(x',t))\,\mathrm{d}x',\end{aligned} \tag{A.7}$$

where the state-dependent dynamics of h_m is governed by

$$\begin{aligned}\frac{\partial h_m(x,t)}{\partial t} &= \big(-h_m(x,t) + h_{m_0}\big)\overline{f_0(u_m(x,t))} \\ &\quad + \big(-h_m(x,t) + h_{m_T}(x,t)\big) f_0(u_{ste}(x,t)) f_0(u_c(x,t)) \\ &\quad + \big(-h_m(x,t) + h_{m_{inh}}\big) f_0(u_m(x,t))\overline{f_0(u_c(x,t))},\end{aligned} \tag{A.8}$$

where

$$h_{m_T}(x,t) = h_{ste}(x,t)\overline{f_0(u_{wm}(x,t))} + h_{EMA}(x,t)f_0(u_{wm}(x,t)), \tag{A.9}$$

and

$$h_{EMA}(x,t) = h_{m_{prev}}(x) + \lambda_{EMA}\big(h_{ste}(x,t) - h_{m_{prev}}(x)\big). \tag{A.10}$$

Excitatory input shaped by w_{osc} (see Equation (4.3)) from u_c at field positions with $u_c(x) \geq 0$ triggers the evolution of peaks in u_m at corresponding sites. They become self-sustained (when $u_m(x) \geq 0$) due to the lateral interactions also described by a kernel w_{osc} (see Equation (4.3)). The temporal information of an event gets stored in the activation patterns of u_m through the h_m dynamics (Equation (A.8) and (A.9)). When a new peak emerges, h_m tracks the h_{ste} dynamics (Equation (A.5)) from u_{ste}. On the other hand, when a memory representation already exists, the local baseline level h_m (respectively the peak amplitude) is adjusted using the EMA mechanism (Equation (A.10)) [198]. It considers the previous values of temporal information $h_{m_{prev}}$ and the current values h_{ste} from u_{ste} with a weight $0 < \lambda_{EMA} < 1$ controlling the influence of the past. A peak in the Working Memory field, u_{wm}, indicates that an event is not new but has been recalled from previous experiences. When a memory trace in u_c reaches the baseline state, h_m converges to $h_{m_{inh}} < 0$. As a result, an existing peak in u_m becomes destabilized (forgotten) and h_m decays back to baseline $h_{m_0} < 0$.

The dynamics of the recall fields, u_{ar} and u_{tr}, are identical, except they start from different baseline levels. In the following, we use the common notation u_r for simplicity:

$$\begin{aligned}\tau_{_r}\frac{\partial u_{_r}(x,t)}{\partial t} = &-u_{_r}(x,t) \\ &+ \big(h_{_r}(t) + u_{m_{prev}}(x)\overline{f_0(u_{wm}(x,t))}\big)\overline{f_0(u_{_r}(x,t))} \\ &+ \int w_{osc_1}(x-x')f_0(u_{_r}(x',t))\,\mathrm{d}x' \\ &- \int w_{osc_2}(x-x')f_0(u_{wm}(x',t))\,\mathrm{d}x',\end{aligned} \tag{A.11}$$

where

$$\frac{\partial h_{_r}(t)}{\partial t} = k_r f_0(u_{day_start}(t)), \qquad \text{with } h_{_r}(t_0) = -h_{r_0} + off_r. \tag{A.12}$$

The initial conditions at time $t_0 = 0$ for the recall fields are shaped by two contributions. First, the baseline activity is defined by the amplitude of a peak reflecting the routine duration (e.g., 24h), $h_{r_0} > 0$, plus the respective offsets, $off_{ar} > 0$ and $off_{tr} < 0$. Second, the stable memory pattern from the previous routine experience, $u_{m_{prev}}(x)$, is loaded to the recall fields as subthreshold input. The recall process starts at the beginning of the day with a linear increase of the baseline level which stops at the end of the day (see Equation (A.12)). The constant k_r controls the speed of sequence recall. The fields receive suppressive input, shaped by w_{osc_2} (see Equation (4.3)), from the Working Memory field u_{wm} indicating that the recall of the specific event has been finished. This destabilizes the peaks in the recall fields.

The dynamics of the Working Memory field, u_{wm}, is governed by

$$\begin{aligned}\tau_{wm}\frac{\partial u_{wm}(x,t)}{\partial t} = &-u_{wm}(x,t) + h_{wm} \\ &+ \int w_{osc_1}(x-x')f_0(u_{wm}(x',t))\,\mathrm{d}x' \\ &+ \int w_{osc_2}(x-x')f_0(u_{ar}(x',t))f_0(u_{tr}(x',t))\,\mathrm{d}x' \\ &+ \int 7w_{osc_2}(x-x')f_0(u_{ar}(x',t))f_0(u_{se}(x',t))\,\mathrm{d}x'. \end{aligned} \tag{A.13}$$

Excitatory input to u_{wm} drives the evolution of a peak which denotes the end of an event recall either because the time window for recall is closed or because the event actually happened.

Appendix B

Simulation parameters

For each vehicle, each day from the datasets ($\approx$ 77 days) is successively simulated with a time step $\Delta t_{sim} = 0.5$s.

B.1 Clustering model

We specified a spatial discretization step of 0.0002° ($\approx$ 20m per step) for the two-dimensional clustering fields (for model details see [46]). Using the following parameters, a single cluster emerges with a radius of $\approx$ 100m:

Table 11: **Parameters and values for the fields of the clustering model.**

Clustering fields, u_{clust} and v_{clust}	
$\mathrm{d}x = \mathrm{d}y$	1
τ_{clust}	3
I_{clust}	$A = 1; \sigma = 1$
w_{mex}	$A_{exc} = 0.75; A_{inh} = 0.4; \sigma = 1.6; \gamma = 1.33313$
θ	0.5

B.2 Cognitive models

For a capacity of 20 stop locations, the one-dimensional fields of length $L = 2000$ are divided in 20 subintervals (slots) of equal length, each containing 100 neurons ($\mathrm{d}x = 1$). A safety margin is added to the field length to prevent from boundary effects when using the FFT for the convolutions of the fields.

For the recall process, we have specified a temporal $anticipation$ of 30min and $tolerance$ of 15min, resulting in a fixed time window of 45min. The parameter controlling the growth rate of an evolving peak (k_{ste}) and the recall speed (k_r) are chosen as $k_{ste} = k_r = 1/3600 = 0.000278$ hours, corresponding to one second (in hours unit) of growth and recall per second of simulation time. The threshold level for an event to be considered routine is set as $\theta_h = 5$.

Table 12: **Parameters and values for the Spatial Encoding field of the cognitive model.**

Spatial Encoding, u_{se}	
τ_{se}	3
h_{se}	-5
I_{se}	$A = 5$
w_{lat}	$A = 3; \sigma = 3; g = 2$
w_{osc_1}	$A = 0.2; b = 0.25; \alpha = 0.052$
w_{osc_2}	$A = 0.12; b = 0.5; \alpha = 0.8$
w_{osc_3}	$A = 0.2; b = 0.25; \alpha = 0.0052$
c	0.05

Table 13: **Parameters and values for the Spatio-Temporal Encoding field of the cognitive model.**

Spatio-Temporal Encoding, u_{ste}	
τ_{ste}	3
h_{ste}	$h_{ste_0} = -1.5$
w_{osc}	$A_{ste} = 1.2; b_{ste} = 0.72; \alpha_{ste} = 0.52$

Table 14: **Parameters and values for the Consolidation field of the cognitive model.**

Consolidation, u_c	
h_c	-1
$\lambda_{strength}$	$1/100$
$\lambda_{strength_h}$	$1/5$
λ_{weak}	$1/145$
λ_{forget}	$1/5$
a	15

Table 15: **Parameters and values for the Memory field of the cognitive model.**

Memory, u_m	
τ_m	3
h_m	$h_{m_0} = -1.5; h_{m_{inh}} = -5; \lambda_{EMA} = 0.65$
w_{osc}	$A = A_{ste}/2; b = b_{ste}; \alpha = \alpha_{ste}$

Table 16: **Parameters and values for the Recall fields of the cognitive model.**

Recall fields, u_{ar} and u_{tr}	
τ_r	3
h_{ar}	$off_{ar} = anticipation/60$
h_{tr}	$off_{tr} = -(tolerance/60)$
w_{osc_1}	$A = 1.2; b = 0.72; \alpha = 0.52$
w_{osc_2}	$A = 1.5; b = 0.72; \alpha = 0.52$

Table 17: **Parameters and values for the Working Memory field of the cognitive model.**

Working Memory, u_{wm}	
τ_{wm}	30
h_{wm}	-2.4
w_{osc_1}	$A = 3.7; b = 0.9; \alpha = 0.8$
w_{osc_2}	$A = 1.4; b = 0.9; \alpha = 0.5$

Bibliography

[1] H. Zhang et al. “Emerging Trends in Intelligent Vehicles: The IEEE TIV Perspective”. In: *IEEE Trans. Intell. Veh.* 8.8 (2023-08), pp. 3983–3995. doi: 10.1109/TIV.2023.3291457. url: https://ieeexplore.ieee.org/document/10171427/ (cit. on pp. 2, 62).

[2] D. Yi et al. “Implicit Personalization in Driving Assistance: State-of-the-Art and Open Issues”. In: *IEEE Trans. Intell. Veh.* 5.3 (2020-09), pp. 397–413. doi: 10.1109/TIV.2019.2960935. url: https://ieeexplore.ieee.org/document/8936867/ (cit. on pp. 2, 11, 49, 50, 62, 113).

[3] S. Wang et al. “ChatGPT as Your Vehicle Co-Pilot: An Initial Attempt”. In: *IEEE Trans. Intell. Veh.* 8.12 (2023-12), pp. 4706–4721. doi: 10.1109/TIV.2023.3325300. url: https://ieeexplore.ieee.org/document/10286969/ (cit. on pp. 2, 3, 62, 113).

[4] F. J. Belmonte et al. “Overview of Embedded Systems to Build Reliable and Safe ADAS and AD Systems”. In: *IEEE Intell. Transport. Syst. Mag.* 13.4 (2021), pp. 239–250. doi: 10.1109/MITS.2019.2953543. url: https://ieeexplore.ieee.org/document/8995493/ (cit. on pp. 2, 11, 12, 62).

[5] S. Choi et al. *Advanced Driver-Assistance Systems: Challenges and Opportunities Ahead*. Tech. rep. 2016. url: https://www.mckinsey.com/industries/semiconductors/our-insights/advanced-driver-assistance-systems-challenges-and-opportunities-ahead (visited on 2024-07-22) (cit. on pp. 2, 11, 49, 62).

[6] S. Gospodinova and F. Miccoli. *New rules to improve road safety and enable fully driverless vehicles in the EU*. 2022-07-06. url: https://ec.europa.eu/commission/presscorner/detail/en/IP_22_4312 (visited on 2024-07-06) (cit. on pp. 2, 62).

[7] T. Tarnowski et al. “Mercedes-Benz MBUX Hyperscreen Merges Technologies into Digital Dashboard Application”. In: *Information Display* 38.3 (2022-05), pp. 12–17. doi: 10.1002/msid.1300. url: https://sid.onlinelibrary.wiley.com/doi/10.1002/msid.1300 (cit. on pp. 2, 62, 114).

[8] S. Hoch et al. “The BMW SURF Project: A Contribution to the Research on Cognitive Vehicles”. In: *2007 IEEE Intelligent Vehicles Symposium*. Istanbul, Turkey: IEEE, 2007-06, pp. 692–697. doi: 10.1109/IVS.2007.4290197. url: http://ieeexplore.ieee.org/document/4290197/ (cit. on pp. 2, 62).

[9] L. Li et al. “Cognitive Cars: A New Frontier for ADAS Research”. In: *IEEE Trans. Intell. Transport. Syst.* 13.1 (2012-03), pp. 395–407. doi: 10.1109/TITS.2011.2159493. url: http://ieeexplore.ieee.org/document/5941006/ (cit. on pp. 2, 62).

[10] G. Vitello et al. “A Bio-Inspired Cognitive Agent for Autonomous Urban Vehicles Routing Optimization”. In: *IEEE Trans. Cogn. Dev. Syst.* 9.1 (2017-03), pp. 5–15. doi: 10.1109/TCDS.2016.2608500. url: http://ieeexplore.ieee.org/document/7564464/ (cit. on pp. 2, 62).

[11] P. K. Murali, M. Kaboli, and R. Dahiya. “Intelligent In-Vehicle Interaction Technologies”. In: *Advanced Intelligent Systems* 4.2 (2022-02). doi: 10.1002/aisy.202100122. url: https://onlinelibrary.wiley.com/doi/10.1002/aisy.202100122 (cit. on pp. 2, 3, 113).

[12] Mercedes-Benz Group AG. *The MBUX Hyperscreen*. 2021-01-07. url: https://group.mercedes-benz.com/innovation/digitalisation/connectivity/mbux-hyperscreen.html (visited on 2024-07-10) (cit. on pp. 2, 3, 114).

[13] TomTom International BV. *About Destination Prediction - TomTom Support*. 2024. url: https://help.tomtom.com/hc/en-us/articles/360013901080-About-Destination-Prediction (visited on 2024-07-10) (cit. on pp. 2, 3, 114).

[14] A. R. Patel et al. “Prediction of attitudes towards human-centred cognitive vehicles aware of their users’ routines and preferences”. In: *2021 IEEE 6th International Forum on Research and Technology for Society and Industry (RTSI)*. Naples, Italy: IEEE, 2021-09-06, pp. 194–199. doi: 10.1109/RTSI50628.2021.9597265. url: https://ieeexplore.ieee.org/document/9597265/ (cit. on pp. 2, 50, 62, 113).

[15] Waymo LLC. *Waymo - Self-Driving Cars - Autonomous Vehicles - Ride-Hail*. 2024. url: https://waymo.com (visited on 2024-07-07) (cit. on pp. 3, 63, 120).

[16] M. C. González, C. A. Hidalgo, and A.-L. Barabási. “Understanding individual human mobility patterns”. In: *Nature* 453.7196 (2008-06), pp. 779–782. issn: 0028-0836, 1476-4687. doi: 10.1038/nature06958. url: https://www.nature.com/articles/nature06958 (cit. on pp. 3, 4, 114).

[17] C. Song et al. “Limits of Predictability in Human Mobility”. In: *Science* 327.5968 (2010-02), pp. 1018–1021. issn: 0036-8075, 1095-9203. doi: 10.1126/science.1177170. url: https://www.science.org/doi/10.1126/science.1177170 (cit. on pp. 3, 4, 8, 63, 114).

[18] Y. Li et al. “Limits of Predictability for Large-Scale Urban Vehicular Mobility”. In: *IEEE Trans. Intell. Transport. Syst.* 15.6 (2014-12), pp. 2671–2682. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2014.2325395. url: https://ieeexplore.ieee.org/document/6837495 (cit. on pp. 3–5, 63, 114).

[19] M. Bieler, R. R. Mukkamala, and T.-M. Gronli. “A Context- and Trajectory-Based Destination Prediction of Public Transportation Users”. In: *IEEE Intell. Transport. Syst. Mag.* 15.1 (2023-01), pp. 300–317. issn: 1939-1390, 1941-1197. doi: 10.1109/MITS.2021.3132772. url: https://ieeexplore.ieee.org/document/9676700/ (cit. on pp. 3, 5, 8, 9, 63).

[20] B. Mo et al. “Individual Mobility Prediction in Mass Transit Systems Using Smart Card Data: An Interpretable Activity-Based Hidden Markov Approach”. In: *IEEE Trans. Intell. Transport. Syst.* 23.8 (2022-08), pp. 12014–12026. doi: 10.1109/TITS.2021.3109428. url: https://ieeexplore.ieee.org/document/9537634/ (cit. on pp. 3, 7, 9, 114, 116).

[21] Z. Ma and P. Zhang. “Individual mobility prediction review: Data, problem, method and application”. In: *Multimodal Transportation* 1.1 (2022-03), p. 100002. issn: 27725863. doi: 10.1016/j.multra.2022.100002. url: https://linkinghub.elsevier.com/retrieve/pii/S2772586322000028 (cit. on pp. 5–9, 63, 114).

[22] P. Sadeghian et al. “Testing feasibility of using a hidden Markov model on predicting human mobility based on GPS tracking data”. In: *Transportmetrica B: Transport Dynamics* 12.1 (2024-12), p. 2336037. issn: 2168-0566, 2168-0582. doi: 10.1080/21680566.2024.2336037. url: https://www.tandfonline.com/doi/full/10.1080/21680566.2024.2336037 (cit. on pp. 5, 8, 9).

[23] UN DESA. *68% of the World Population Projected to Live in Urban Areas by 2050, Says UN*. 2018-05. url: https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html (visited on 2024-07-10) (cit. on p. 5).

[24] X. Ma et al. “Large-Scale Transportation Network Congestion Evolution Prediction Using Deep Learning Theory”. In: *PLoS ONE* 10.3 (2015-03), e0119044. issn: 1932-6203. doi: 10.1371/journal.pone.0119044. url: https://dx.plos.org/10.1371/journal.pone.0119044 (cit. on pp. 5, 6).

[25] M. Talebkhah et al. “Task offloading for edge-IoV networks in the industry 4.0 era and beyond: A high-level view”. In: *Engineering Science and Technology, an International Journal* 54 (2024-06), p. 101699. issn: 22150986. doi: 10.1016/j.jestch.2024.101699. url: https://linkinghub.elsevier.com/retrieve/pii/S2215098624000855 (cit. on p. 5).

[26] M. Veres and M. Moussa. “Deep Learning for Intelligent Transportation Systems: A Survey of Emerging Trends”. In: *IEEE Trans. Intell. Transport. Syst.* 21.8 (2020-08), pp. 3152–3168. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2019.2929020. url: https://ieeexplore.ieee.org/document/8771378/ (cit. on pp. 5, 7, 9).

[27] X. Zhang et al. “Federated Representation Learning With Data Heterogeneity for Human Mobility Prediction”. In: *IEEE Trans. Intell. Transport. Syst.* 24.6 (2023-06), pp. 6111–6122. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2023.3252029. url: https://ieeexplore.ieee.org/document/10074979/ (cit. on pp. 5, 6, 10).

[28] R. Imai et al. “Early Destination Prediction with Spatio-temporal User Behavior Patterns”. In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1.4 (2018-01), pp. 1–19. issn: 2474-9567. doi: 10.1145/3161197. url: https://dl.acm.org/doi/10.1145/3161197 (cit. on pp. 5, 9).

[29] W. Long et al. “Location Prediction for Individual Vehicles via Exploiting Travel Regularity and Preference”. In: *IEEE Trans. Veh. Technol.* 71.5 (2022-05), pp. 4718–4732. doi: 10.1109/TVT.2022.3151762. url: https://ieeexplore.ieee.org/document/9714182/ (cit. on pp. 5, 9, 10).

[30] L. Irio et al. “An Adaptive Learning-Based Approach for Vehicle Mobility Prediction”. In: *IEEE Access* 9 (2021), pp. 13671–13682. issn: 2169-3536. doi: 10.1109/ACCESS.2021.3052071. url: https://ieeexplore.ieee.org/document/9326302/ (cit. on p. 5).

[31] X. Geng et al. “Spatiotemporal Multi-Graph Convolution Network for Ride-Hailing Demand Forecasting”. In: *AAAI* 33.01 (2019-07), pp. 3656–3663. issn: 2374-3468, 2159-5399. doi: 10.1609/aaai.v33i01.33013656. url: https://ojs.aaai.org/index.php/AAAI/article/view/4247 (cit. on pp. 5, 6).

[32] M. Chen, X. Yu, and Y. Liu. “Mining moving patterns for predicting next location”. In: *Information Systems* 54 (2015-12), pp. 156–168. issn: 03064379. doi: 10.1016/j.is.2015.07.001. url: https://linkinghub.elsevier.com/retrieve/pii/S0306437915001295 (cit. on p. 6).

[33] P. Zhang, H. N. Koutsopoulos, and Z. Ma. “DeepTrip: A Deep Learning Model for the Individual Next Trip Prediction With Arbitrary Prediction Times”. In: *IEEE Trans. Intell. Transport. Syst.* 24.6 (2023-06), pp. 5842–5855. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2023.3252043. url: https://ieeexplore.ieee.org/document/10074989/ (cit. on pp. 6, 9, 10).

[34] W. Wang et al. “An Attention-Based Deep Learning Framework for Trip Destination Prediction of Sharing Bike”. In: *IEEE Trans. Intell. Transport. Syst.* 22.7 (2021-07), pp. 4601–4610. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2020.3008935. url: https://ieeexplore.ieee.org/document/9152145/ (cit. on p. 6).

[35] Q. Li, D. Zou, and Y. Xu. “Combining individual travel behaviour and collective preferences for next location prediction”. In: *Transportmetrica A: Transport Science* 18.3 (2022-12), pp. 1754–1776. issn: 2324-9935, 2324-9943. doi: 10.1080/23249935.2021.1968066. url: https://www.tandfonline.com/doi/full/10.1080/23249935.2021.1968066 (cit. on p. 6).

[36] Y. Hong et al. “Context-aware multi-head self-attentional neural network model for next location prediction”. In: *Transportation Research Part C: Emerging Technologies* 156 (2023-11), p. 104315. issn: 0968090X. doi: 10.1016/j.trc.2023.104315. url: https://linkinghub.elsevier.com/retrieve/pii/S0968090X23003042 (cit. on pp. 6, 9).

[37] S. Gambs, M.-O. Killijian, and M. N. Del Prado Cortez. “Next place prediction using mobility Markov chains”. In: *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*. Bern Switzerland: ACM, 2012-04, pp. 1–6. isbn: 978-1-4503-1163-2. doi: 10.1145/2181196.2181199. url: https://dl.acm.org/doi/10.1145/2181196.2181199 (cit. on p. 7).

[38] Z. Zhao, H. N. Koutsopoulos, and J. Zhao. “Individual mobility prediction using transit smart card data”. In: *Transportation Research Part C: Emerging Technologies* 89 (2018-04), pp. 19–34. issn: 0968090X. doi: 10.1016/j.trc.2018.01.022. url: https://linkinghub.elsevier.com/retrieve/pii/S0968090X18300676 (cit. on pp. 7, 9).

[39] D. J. Phillips, T. A. Wheeler, and M. J. Kochenderfer. “Generalizable intention prediction of human drivers at intersections”. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. Los Angeles, CA, USA: IEEE, 2017-06, pp. 1665–1670. isbn: 978-1-5090-4804-5. doi: 10.1109/IVS.2017.7995948. url: http://ieeexplore.ieee.org/document/7995948/ (cit. on p. 7).

[40] D. Cheng et al. “Vehicle Trajectory Prediction With Interaction Regions and Spatial–Temporal Attention”. In: *IEEE Access* 11 (2023), pp. 130850–130859. issn: 2169-3536. doi: 10.1109/ACCESS.2023.3335091. url: https://ieeexplore.ieee.org/document/10323462/ (cit. on p. 7).

[41] Y. Liang and Z. Zhao. “NetTraj: A Network-Based Vehicle Trajectory Prediction Model With Directional Representation and Spatiotemporal Attention Mechanisms”. In: *IEEE Trans. Intell. Transport. Syst.* 23.9 (2022-09), pp. 14470–14481. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2021.3129588. url: https://ieeexplore.ieee.org/document/9629362/ (cit. on p. 7).

[42] Y. Lv et al. “Traffic Flow Prediction With Big Data: A Deep Learning Approach”. In: *IEEE Trans. Intell. Transport. Syst.* (2014), pp. 1–9. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2014.2345663. url: https://ieeexplore.ieee.org/document/6894591 (cit. on p. 7).

[43] B. Yu, H. Yin, and Z. Zhu. “Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization, 2018-07, pp. 3634–3640. isbn: 978-0-9992411-2-7. doi: 10.24963/ijcai.2018/505. url: https://www.ijcai.org/proceedings/2018/505 (cit. on p. 7).

[44] X. Liang et al. “A Deep Learning Model for Transportation Mode Detection Based on Smartphone Sensing Data”. In: *IEEE Trans. Intell. Transport. Syst.* 21.12 (2020-12), pp. 5223–5235. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2019.2951165. url: https://ieeexplore.ieee.org/document/8913719/ (cit. on p. 8).

[45] D. Ashbrook and T. Starner. “Using GPS to learn significant locations and predict movement across multiple users”. In: *Personal and Ubiquitous Computing* 7.5 (2003-10), pp. 275–286. issn: 1617-4909, 1617-4917. doi: 10.1007/s00779-003-0240-0. url: http://link.springer.com/10.1007/s00779-003-0240-0 (cit. on p. 8).

[46] F. Ferreira et al. “Dynamic Identification of Stop Locations from GPS Trajectories Based on Their Temporal and Spatial Characteristics”. In: *Artificial Neural Networks and Machine Learning – ICANN 2021*. Vol. 12894. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 347–359. isbn: 978-3-030-86379-1 978-3-030-86380-7. doi: 10.1007/978-3-030-86380-7_28. url: https://link.springer.com/10.1007/978-3-030-86380-7_28 (cit. on pp. 8, 51, 68, 121, 127).

[47] C. Yu et al. “Modeling User Activity Patterns for Next-Place Prediction”. In: *IEEE Systems Journal* 11.2 (2017-06), pp. 1060–1071. issn: 1932-8184, 1937-9234, 2373-7816. doi: 10.1109/JSYST.2015.2445919. url: http://ieeexplore.ieee.org/document/7150318/ (cit. on p. 9).

[48] A. Rossi et al. “Modelling Taxi Drivers’ Behaviour for the Next Destination Prediction”. In: *IEEE Trans. Intell. Transport. Syst.* 21.7 (2020-07), pp. 2980–2989. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2019.2922002. url: https://ieeexplore.ieee.org/document/8741193/ (cit. on p. 9).

[49] Q. Lv et al. “Big data driven hidden Markov model based individual mobility prediction at points of interest”. In: *IEEE Transactions on Vehicular Technology* 66.6 (2016), pp. 5204–5216. doi: 10.1109/TVT.2016.2611654. url: https://ieeexplore.ieee.org/abstract/document/7572081 (cit. on p. 9).

[50] Y. Qiao et al. “A hybrid Markov-based model for human mobility prediction”. In: *Neurocomputing* 278 (2018), pp. 99–109. doi: 10.1016/j.neucom.2017.05.101. url: https://www.sciencedirect.com/science/article/abs/pii/S0925231217314455 (cit. on p. 9).

[51] F. Zong et al. “Trip destination prediction based on multi-day GPS data”. In: *Physica A: Statistical Mechanics and its Applications* 515 (2019), pp. 258–269. doi: 10.1016/j.physa.2018.09.090. url: https://www.sciencedirect.com/science/article/abs/pii/S0378437118312226 (cit. on p. 9).

[52] J. Zhang et al. “Predicting Individual Mobility Behavior of Ride-Hailing Service Users considering Heterogeneity of Trip Purposes”. In: *Data Science for Transportation* 6.3 (2024), p. 29. doi: 10.1007/s42421-024-00113-1. url: https://link.springer.com/article/10.1007/s42421-024-00113-1 (cit. on p. 9).

[53] B. Kažič et al. “Predicting users’ mobility using Monte Carlo simulations”. In: *IEEE Access* 5 (2017), pp. 27400–27420. doi: 10.1109/ACCESS.2017.2768125. url: https://ieeexplore.ieee.org/abstract/document/8089732 (cit. on p. 9).

[54] C. Fernandes et al. “A Deep Learning Approach for Intelligent Cockpits: Learning Drivers Routines”. In: *Intelligent Data Engineering and Automated Learning – IDEAL 2020*. Vol. 12490. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 173–183. isbn: 978-3-030-62364-7 978-3-030-62365-4. doi: 10.1007/978-3-030-62365-4_17. url: https://link.springer.com/10.1007/978-3-030-62365-4_17 (cit. on pp. 9, 51, 82).

[55] F. Li et al. “A hierarchical temporal attention-based LSTM encoder-decoder model for individual mobility prediction”. In: *Neurocomputing* 403 (2020-08), pp. 153–166. issn: 09252312. doi: 10.1016/j.neucom.2020.03.080. url: https://linkinghub.elsevier.com/retrieve/pii/S0925231220304677 (cit. on pp. 9, 10).

[56] J. Zhao et al. “MDLF: A Multi-View-Based Deep Learning Framework for Individual Trip Destination Prediction in Public Transportation Systems”. In: *IEEE Trans. Intell. Transport. Syst.* 23.8 (2022-08), pp. 13316–13329. issn: 1524-9050, 1558-0016. doi: 10.1109/TITS.2021.3123342. url: https://ieeexplore.ieee.org/document/9605211/ (cit. on p. 9).

[57] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997-11), pp. 1735–1780. issn: 0899-7667, 1530-888X. doi: 10.1162/neco.1997.9.8.1735. url: https://direct.mit.edu/neco/article/9/8/1735-1780/6109 (cit. on pp. 9, 82).

[58] P. Barbosa et al. “Endowing Intelligent Vehicles with the Ability to Learn User’s Habits and Preferences with Machine Learning Methods”. In: *International Conference on Intelligent Data Engineering and Automated Learning*. Springer. 2022, pp. 157–169. doi: 10.1007/978-3-031-21753-1_16. url: https://link.springer.com/chapter/10.1007/978-3-031-21753-1_16 (cit. on p. 10).

[59] J. Feng et al. “DeepMove: Predicting Human Mobility with Attentional Recurrent Networks”. In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW ’18*. Lyon, France: ACM Press, 2018, pp. 1459–1468. isbn: 978-1-4503-5639-8. doi: 10.1145/3178876.3186058. url: http://dl.acm.org/citation.cfm?doid=3178876.3186058 (cit. on pp. 10, 82).

[60] G. I. Parisi et al. “Continual lifelong learning with neural networks: A review”. In: *Neural Networks* 113 (2019-05), pp. 54–71. issn: 08936080. doi: 10.1016/j.neunet.2019.01.012. url: https://linkinghub.elsevier.com/retrieve/pii/S0893608019300231 (cit. on p. 10).

[61] M. Boukhechba et al. “Prediction of next destinations from irregular patterns”. In: *J Ambient Intell Human Comput* 9.5 (2018-10), pp. 1345–1357. issn: 1868-5137, 1868-5145. doi: 10.1007/s12652-017-0519-z. url: http://link.springer.com/10.1007/s12652-017-0519-z (cit. on p. 10).

[62] V. Kulkarni, A. Moro, and B. Garbinato. “MobiDict: a mobility prediction system leveraging realtime location data streams”. In: *Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming*. Burlingame California: ACM, 2016-10, pp. 1–10. isbn: 978-1-4503-4579-8. doi: 10.1145/3003421.3003424. url: https://dl.acm.org/doi/10.1145/3003421.3003424 (cit. on p. 10).

[63] Erik Eckermann. *World History of the Automobile*. 2001. doi: 10.4271/r-272 (cit. on p. 10).

[64] J. P. Trovao. “Trends in Automotive Electronics [Automotive Electronics]”. In: *IEEE Veh. Technol. Mag.* 14.4 (2019-12), pp. 100–109. issn: 1556-6072, 1556-6080. doi: 10.1109/MVT.2019.2939757. url: https://ieeexplore.ieee.org/document/8907379/ (cit. on pp. 10, 11).

[65] J. Bereisa. “Applications of Microcomputers in Automotive Electronics”. In: *IEEE Trans. Ind. Electron.* IE-30.2 (1983-05), pp. 87–96. issn: 0278-0046, 1557-9948. doi: 10.1109/TIE.1983.356715. url: http://ieeexplore.ieee.org/document/4180448/ (cit. on p. 10).

[66] H. Kopetz. “Automotive electronics”. In: *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS’99*. York, UK: IEEE Comput. Soc, 1999, pp. 132–140. isbn: 978-0-7695-0240-3. doi: 10.1109/EMRTS.1999.777459. url: http://ieeexplore.ieee.org/document/777459/ (cit. on pp. 10, 11).

[67] R. Fletcher et al. “The case for an end-to-end automotive-software platform”. In: (2020) (cit. on p. 11).

[68] K. Cho et al. “An FPGA-Based ECU for Remote Reconfiguration in Automotive Systems”. In: *Micromachines* 12.11 (2021-10), p. 1309. issn: 2072-666X. doi: 10.3390/mi12111309. url: https://www.mdpi.com/2072-666X/12/11/1309 (cit. on p. 11).

[69] C. Park, C. Cui, and S. Park. “Analysis of E2E Delay and Wiring Harness in In-Vehicle Network with Zonal Architecture”. In: *Sensors* 24.10 (2024-05), p. 3248. issn: 1424-8220. doi: 10.3390/s24103248. url: https://www.mdpi.com/1424-8220/24/10/3248 (cit. on p. 11).

[70] J. P. Trovão. “Automotive Electronics: Key System Components [Automotive Electronics]”. In: *IEEE Veh. Technol. Mag.* 18.2 (2023-06), pp. 113–121. issn: 1556-6072, 1556-6080. doi: 10.1109/MVT.2023.3260544. url: https://ieeexplore.ieee.org/document/10132315/ (cit. on p. 11).

[71] S. Lu and W. Shi. “Vehicle Computing: Vision and challenges”. In: *Journal of Information and Intelligence* 1.1 (2023-05), pp. 23–35. issn: 29497159. doi: 10.1016/j.jiixd.2022.10.001. url: https://linkinghub.elsevier.com/retrieve/pii/S2949715922000038 (cit. on pp. 11, 12).

[72] H. Kondo et al. “A 28-nm Automotive Flash Microcontroller With Virtualization-Assisted Processor Supporting ISO26262 ASIL D”. In: *IEEE J. Solid-State Circuits* 55.1 (2020-01), pp. 133–144. issn: 0018-9200, 1558-173X. doi: 10.1109/JSSC.2019.2953826. url: https://ieeexplore.ieee.org/document/8917654/ (cit. on p. 11).

[73] S. Van Singel. “The Renesas Automotive Story in the History of the Microprocessor”. In: *IEEE Micro* 41.6 (2021-11), pp. 107–108. issn: 0272-1732, 1937-4143. doi: 10.1109/MM.2021.3113821. url: https://ieeexplore.ieee.org/document/9623427/ (cit. on p. 11).

[74] F. Baumann and R. Woolley. *RTOS and Linux Are Defining In-Vehicle Computing – Together*. White Paper. 2024-06. url: https://www.aptiv.com/docs/default-source/white-papers/2024-aptiv-whitepaper-linux-vs-rtos.pdf?sfvrsn=184dac37_1 (visited on 2024-07-14) (cit. on p. 11).

[75] L. Sharara et al. “An Intelligent Sensor-Based Back Seat Monitoring System for Preventing Pediatric Vehicular Heatstroke”. In: *IEEE Trans. Intell. Veh.* (2024), pp. 1–17. issn: 2379-8904, 2379-8858. doi: 10.1109/TIV.2024.3408842. url: https://ieeexplore.ieee.org/document/10547271/ (cit. on p. 11).

[76] Z. Nikolic, A. Saxena, and G. Agarwal. *TDA3x SoC processor delivers cost-effective ADAS solutions for front, rear and surround view applications*. Tech. rep. 2018-01. url: https://www.ti.com/lit/wp/spry272a/spry272a.pdf (visited on 2024-07-14) (cit. on p. 11).

[77] Y. Yamada and K. Kimura. “Efficient System-on-Chip (SOC) for Automated Driving with High Safety”. In: *NANO-CHIPS 2030*. Cham: Springer International Publishing, 2020, pp. 563–575. isbn: 978-3-030-18337-0 978-3-030-18338-7. doi: 10.1007/978-3-030-18338-7_29. url: http://link.springer.com/10.1007/978-3-030-18338-7_29 (cit. on p. 11).

[78] Mobileye. *The Evolution of EyeQ*. 2024. url: https://www.mobileye.com/technology/eyeq-chip/ (visited on 2024-07-15) (cit. on p. 11).

[79] Qualcomm Technologies. *Automated Driving | Snapdragon Ride*. 2024. url: https://www.qualcomm.com/products/automotive/automated-driving (visited on 2024-07-15) (cit. on p. 11).

[80] J. Park et al. “An Optimized DNN Model for Real-Time Inferencing on an Embedded Device”. In: *Sensors* 23.8 (2023-04), p. 3992. issn: 1424-8220. doi: 10.3390/s23083992. url: https://www.mdpi.com/1424-8220/23/8/3992 (cit. on p. 12).

[81] NVIDIA Corporation. *The Benefits of Using GPUs*. 2024-07. url: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#the-benefits-of-using-gpus (visited on 2024-07-15) (cit. on pp. 12, 33).

[82] H. Tabani et al. “Performance Analysis and Optimization Opportunities for NVIDIA Automotive GPUs”. In: *Journal of Parallel and Distributed Computing* 152 (2021-06), pp. 21–32. issn: 07437315. doi: 10.1016/j.jpdc.2021.02.008. url: https://linkinghub.elsevier.com/retrieve/pii/S0743731521000290 (cit. on p. 12).

[83] NVIDIA Corporation. *In-Vehicle Computing for AI-Defined Cars*. 2024. url: https://www.nvidia.com/en-eu/self-driving-cars/in-vehicle-computing/ (visited on 2024-07-15) (cit. on p. 12).

[84] NVIDIA Corporation. *CUDA®: A General-Purpose Parallel Computing Platform and Programming Model*. 2024-07. url: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-a-general-purpose-parallel-computing-platform-and-programming-model (visited on 2024-07-15) (cit. on pp. 12, 33).

[85] Ó. Mata-Carballeira et al. “An FPGA-Based Neuro-Fuzzy Sensor for Personalized Driving Assistance”. In: *Sensors* 19.18 (2019-09), p. 4011. issn: 1424-8220. doi: 10.3390/s19184011. url: https://www.mdpi.com/1424-8220/19/18/4011 (cit. on p. 12).

[86] G. Tatar and S. Bayar. “Real-Time Multi-Task ADAS Implementation on Reconfigurable Heterogeneous MPSoC Architecture”. In: *IEEE Access* 11 (2023), pp. 80741–80760. issn: 2169-3536. doi: 10.1109/ACCESS.2023.3300379. url: https://ieeexplore.ieee.org/document/10198234/ (cit. on p. 12).

[87] J. Kim, J.-K. Kang, and Y. Kim. “A Low-Cost Fully Integer-Based CNN Accelerator on FPGA for Real-Time Traffic Sign Recognition”. In: *IEEE Access* 10 (2022), pp. 84626–84634. issn: 2169-3536. doi: 10.1109/ACCESS.2022.3197906. url: https://ieeexplore.ieee.org/document/9853508/ (cit. on p. 12).

[88] L. Cunha et al. “Hardware-Accelerated Data Decoding and Reconstruction for Automotive LiDAR Sensors”. In: *IEEE Trans. Veh. Technol.* 72.4 (2023-04), pp. 4267–4276. issn: 0018-9545, 1939-9359. doi: 10.1109/TVT.2022.3223231. url: https://ieeexplore.ieee.org/document/9954909/ (cit. on p. 12).

[89] S. Shanker. “Enhancing automotive embedded systems with FPGAs”. PhD thesis. Nanyang Technological University, 2016. doi: 10.32657/10356/67064. url: http://hdl.handle.net/10356/67064 (cit. on p. 12).

[90] S. Luthra, M. A. Khalid, and M. A. Moin Oninda. “FPGA-Based Evaluation and Implementation of an Automotive RADAR Signal Processing System using High-Level Synthesis”. In: *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. London, ON, Canada: IEEE, 2020-08, pp. 1–6. isbn: 978-1-72815-442-8. doi: 10.1109/CCECE47787.2020.9255725. url: https://ieeexplore.ieee.org/document/9255725/ (cit. on p. 12).

[91] F. Ferreira et al. “A dynamic neural model for endowing intelligent cars with the ability to learn driver routines: where to go, when to arrive and how long to stay there”. In: *Towards Cognitive Vehicles Workshop (TCV2019), IROS2019*. 2019, pp. 15–18 (cit. on pp. 17, 71, 115, 123, 124).

[92] G. Schöner. “Dynamical Systems Approaches to Cognition”. In: *The Cambridge Handbook of Computational Psychology*. Cambridge University Press, 2008, pp. 101–126. isbn: 978-0-511-81677-2 978-0-521-85741-3 978-0-521-67410-2. doi: 10.1017/CBO9780511816772.007. url: https://www.cambridge.org/core/product/identifier/9780511816772%23c85741-ch4/type/book_part (cit. on pp. 17, 18).

[93] E. Bicho, P. Mallet, and G. Schöner. “Target Representation on an Autonomous Vehicle with Low-Level Sensors”. In: *The International Journal of Robotics Research* 19.5 (2000-05), pp. 424–447. issn: 0278-3649, 1741-3176. doi: 10.1177/02783640022066950. url: https://journals.sagepub.com/doi/10.1177/02783640022066950 (cit. on pp. 17, 18).

[94] G. Schöner. “Dynamic Thinking: A Primer on Dynamic Field Theory”. In: Oxford University Press, 2016. doi: 10.1093/acprof:oso/9780199300563.001.0001 (cit. on pp. 17, 18, 23, 64).

[95] A. G. Feldman. “Functional tuning of the nervous system during control of movement or maintenance of a steady posture. III. Mechanographic analysis of the execution by man of the simplest motor tasks”. In: *Biofizika* (1966), pp. 667–675 (cit. on p. 17).

[96] J. A. S. Kelso et al. “On the Concept of Coordinative Structures as Dissipative Structures: II. Empirical Lines of Convergence”. In: *Advances in Psychology*. Vol. 1. Elsevier, 1980, pp. 49–70. isbn: 978-0-444-85466-7. doi: 10.1016/S0166-4115(08)61937-8. url: https://linkinghub.elsevier.com/retrieve/pii/S0166411508619378 (cit. on p. 17).

[97] J. A. S. Kelso. “Phase transitions and critical behavior in human bimanual coordination”. In: *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology* 246.6 (1984-06), R1000–R1004. issn: 0363-6119, 1522-1490. doi: 10.1152/ajpregu.1984.246.6.R1000. url: https://www.physiology.org/doi/10.1152/ajpregu.1984.246.6.R1000 (cit. on p. 17).

[98] G. Schöner and J. A. S. Kelso. “Dynamic Pattern Generation in Behavioral and Neural Systems”. In: *Science* 239.4847 (1988-03), pp. 1513–1520. issn: 0036-8075, 1095-9203. doi: 10.1126/science.3281253. url: https://www.science.org/doi/10.1126/science.3281253 (cit. on p. 17).

[99] K. Kopecz and G. Schöner. “Saccadic motor planning by integrating visual information and pre-information on neural dynamic fields”. In: *Biol. Cybern.* 73.1 (1995-06), pp. 49–60. issn: 0340-1200, 1432-0770. doi: 10.1007/BF00199055. url: http://link.springer.com/10.1007/BF00199055 (cit. on p. 18).

[100] W. Erlhagen and G. Schöner. "Dynamic field theory of movement preparation". In: *Psychological Review* 109.3 (2002), pp. 545–572. issn: 1939-1471, 0033-295X. doi: 10.1037/0033-295X.109.3.545. url: http://doi.apa.org/getdoi.cfm?doi=10.1037/0033-295X.109.3.545 (cit. on pp. 18, 24).

[101] E. Thelen et al. "The dynamics of embodiment: A field theory of infant perseverative reaching". In: *Behav Brain Sci* 24.1 (2001-02), pp. 1–34. issn: 0140-525X, 1469-1825. doi: 10.1017/S0140525X01003910. url: https://www.cambridge.org/core/product/identifier/S0140525X01003910/type/journal_article (cit. on p. 18).

[102] D. Jancke et al. "Shorter latencies for motion trajectories than for flashes in population responses of cat primary visual cortex". In: *The Journal of Physiology* 556.3 (2004-05), pp. 971–982. issn: 0022-3751, 1469-7793. doi: 10.1113/jphysiol.2003.058941. url: https://physoc.onlinelibrary.wiley.com/doi/10.1113/jphysiol.2003.058941 (cit. on p. 18).

[103] L. K. Samuelson et al. "Grounding Word Learning in Space". In: *PLoS ONE* 6.12 (2011-12), e28095. issn: 1932-6203. doi: 10.1371/journal.pone.0028095. url: https://dx.plos.org/10.1371/journal.pone.0028095 (cit. on p. 18).

[104] C. Engels and G. Schöner. "Dynamic fields endow behavior-based robots with representations". In: *Robotics and Autonomous Systems* 14.1 (1995-02), pp. 55–77. issn: 09218890. doi: 10.1016/0921-8890(94)00020-3. url: https://linkinghub.elsevier.com/retrieve/pii/0921889094000203 (cit. on pp. 18, 19).

[105] G. Schöner, M. Dose, and C. Engels. "Dynamics of behavior: Theory and applications for autonomous robot architectures". In: *Robotics and Autonomous Systems* 16.2-4 (1995-12), pp. 213–245. issn: 09218890. doi: 10.1016/0921-8890(95)00049-6. url: https://linkinghub.elsevier.com/retrieve/pii/0921889095000496 (cit. on pp. 18, 19).

[106] W. Erlhagen, A. Mukovskiy, and E. Bicho. "A dynamic model for action understanding and goal-directed imitation". In: *Brain Research* 1083.1 (2006-04), pp. 174–188. issn: 00068993. doi: 10.1016/j.brainres.2006.01.114. url: https://linkinghub.elsevier.com/retrieve/pii/S0006899306003143 (cit. on p. 18).

[107] W. Erlhagen and E. Bicho. "The dynamic neural field approach to cognitive robotics". In: *J. Neural Eng.* 3.3 (2006-09), R36–R54. issn: 1741-2560, 1741-2552. doi: 10.1088/1741-2560/3/3/R02. url: https://iopscience.iop.org/article/10.1088/1741-2560/3/3/R02 (cit. on pp. 18, 23, 64).

[108] C. Faubel and G. Schöner. "Learning to recognize objects on the fly: A neurally based dynamic field approach". In: *Neural Networks* 21.4 (2008-05), pp. 562–576. issn: 08936080. doi: 10.1016/j.neunet.2008.03.007. url: https://linkinghub.elsevier.com/retrieve/pii/S0893608008000634 (cit. on pp. 18, 19).

[109] Y. Sandamirskaya and G. Schoner. "Dynamic field theory of sequential action: A model and its implementation on an embodied agent". In: *2008 7th IEEE International Conference on Development and Learning*. Monterey, CA: IEEE, 2008-08, pp. 133–138. isbn: 978-1-4244-2661-4. doi: 10.1109/DEVLRN.2008.4640818. url: http://ieeexplore.ieee.org/document/4640818/ (cit. on p. 18).

[110] E. Bicho et al. "Neuro-cognitive mechanisms of decision making in joint action: A human–robot interaction study". In: *Human Movement Science* 30.5 (2011-10), pp. 846–868. issn: 01679457. doi: 10.1016/j.humov.2010.08.012. url: https://linkinghub.elsevier.com/retrieve/pii/S0167945710001508 (cit. on p. 18).

[111] F. Ferreira et al. "Learning a musical sequence by observation: A robotics implementation of a dynamic neural field model". In: *4th International Conference on Development and Learning and on Epigenetic Robotics*. Genoa, Italy: IEEE, 2014-10, pp. 157–162. isbn: 978-1-4799-7540-2. doi: 10.1109/DEVLRN.2014.6982973. url: https://ieeexplore.ieee.org/document/6982973 (cit. on pp. 18, 26, 28, 29).

[112] E. Sousa et al. "Off-line simulation inspires insight: A neurodynamics approach to efficient robot task learning". In: *Neural Networks* 72 (2015-12), pp. 123–139. issn: 08936080. doi: 10.1016/j.neunet.2015.09.002. url: https://linkinghub.elsevier.com/retrieve/pii/S08936080 1500177X (cit. on pp. 18, 64).

[113] W. Wojtak et al. "Adaptive timing in a dynamic field architecture for natural human–robot interactions". In: *Cognitive Systems Research* 82 (2023-12), p. 101148. issn: 13890417. doi: 10.1016/j.cogsys.2023.101148. url: https://linkinghub.elsevier.com/retrieve/pii/S1389041723000761 (cit. on p. 18).

[114] S. Amari. "Dynamics of pattern formation in lateral-inhibition type neural fields". In: *Biol. Cybern.* 27.2 (1977), pp. 77–87. issn: 0340-1200, 1432-0770. doi: 10.1007/BF00337259. url: http://link.springer.com/10.1007/BF00337259 (cit. on pp. 19, 20, 24, 64, 65).

[115] S. Grossberg. "Some Networks that can Learn, Remember, and Reproduce any Number of Complicated Space-time". In: *Stud Appl Math* 49.2 (1970-06), pp. 135–166. issn: 0022-2526, 1467-9590. doi: 10.1002/sapm1970492135. url: https://onlinelibrary.wiley.com/doi/10.1002/sapm1970492135 (cit. on p. 19).

[116] H. R. Wilson and J. D. Cowan. "A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue". In: *Kybernetik* 13.2 (1973-09), pp. 55–80. issn: 0023-5946. doi: 10.1007/BF00288786. url: https://link.springer.com/10.1007/BF00288786 (cit. on p. 19).

[117] C. R. Laing et al. "Multiple Bumps in a Neuronal Model of Working Memory". In: *SIAM J. Appl. Math.* 63.1 (2002-01), pp. 62–97. issn: 0036-1399, 1095-712X. doi: 10.1137/S0036139901389495. url: http://epubs.siam.org/doi/10.1137/S0036139901389495 (cit. on p. 22).

[118] F. Ferreira, W. Erlhagen, and E. Bicho. “Multi-bump solutions in a neural field model with external inputs”. In: *Physica D: Nonlinear Phenomena* 326 (2016-07), pp. 32–51. issn: 01672789. doi: 10.1016/j.physd.2016.01.009. url: https://linkinghub.elsevier.com/retrieve/pii/S016727891530035X (cit. on pp. 22, 65, 66).

[119] F. Ferreira et al. “Rapid Learning of Complex Sequences With Time Constraints: A Dynamic Neural Field Model”. In: *IEEE Trans. Cogn. Dev. Syst.* 13.4 (2021-12), pp. 853–864. issn: 2379-8920, 2379-8939. doi: 10.1109/TCDS.2020.2991789. url: https://ieeexplore.ieee.org/document/9085956/ (cit. on pp. 26, 63, 66, 71, 80, 81, 115, 121, 124).

[120] J. Nickolls and W. J. Dally. “The GPU Computing Era”. In: *IEEE Micro* 30.2 (2010-03), pp. 56–69. issn: 0272-1732. doi: 10.1109/MM.2010.41. url: http://ieeexplore.ieee.org/document/5446251/ (cit. on pp. 31, 33).

[121] L. M. Ang and K. P. Seng. “GPU-Based Embedded Intelligence Architectures and Applications”. In: *Electronics* 10.8 (2021-04), p. 952. issn: 2079-9292. doi: 10.3390/electronics10080952. url: https://www.mdpi.com/2079-9292/10/8/952 (cit. on p. 31).

[122] M. Jorda, P. Valero-Lara, and A. J. Pena. “Performance Evaluation of cuDNN Convolution Algorithms on NVIDIA Volta GPUs”. In: *IEEE Access* 7 (2019), pp. 70461–70473. issn: 2169-3536. doi: 10.1109/ACCESS.2019.2918851. url: https://ieeexplore.ieee.org/document/8721631/ (cit. on p. 31).

[123] S. Mittal and S. Vaishay. “A survey of techniques for optimizing deep learning on GPUs”. In: *Journal of Systems Architecture* 99 (2019-10), p. 101635. issn: 13837621. doi: 10.1016/j.sysarc.2019.101635. url: https://linkinghub.elsevier.com/retrieve/pii/S1383762119302656 (cit. on p. 31).

[124] NVIDIA Corporation. *CUDA Toolkit*. 2024. url: https://developer.nvidia.com/cuda-toolkit (visited on 2024-08-06) (cit. on pp. 31, 33).

[125] NVIDIA Corporation. *NVIDIA Jetson Nano*. 2024. url: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/ (visited on 2024-08-06) (cit. on pp. 31, 87).

[126] NVIDIA Corporation. *Jetson Modules*. 2024. url: https://developer.nvidia.com/embedded/jetson-modules (visited on 2024-08-06) (cit. on pp. 31, 87).

[127] L. S. Karumbunathan and S. H. Sheshadri. *Develop AI-Powered Robots, Smart Vision Systems, and More with NVIDIA Jetson Orin Nano Developer Kit | NVIDIA Technical Blog*. 2023-03. url: https://developer.nvidia.com/blog/develop-ai-powered-robots-smart-vision-systems-and-more-with-nvidia-jetson-orin-nano-developer-kit/ (visited on 2024-08-06) (cit. on p. 31).

[128] V. Mazzia et al. “Real-Time Apple Detection System Using Embedded Systems With Hardware Accelerators: An Edge AI Application”. In: *IEEE Access* 8 (2020), pp. 9102–9114. issn: 2169-3536. doi: 10.1109/ACCESS.2020.2964608. url: https://ieeexplore.ieee.org/document/8951147/ (cit. on p. 32).

[129] E. Sihite et al. “Multi-Modal Mobility Morphobot (M4) with appendage repurposing for locomotion plasticity enhancement”. In: *Nat Commun* 14.1 (2023-06), p. 3323. issn: 2041-1723. doi: 10.1038/s41467-023-39018-y. url: https://www.nature.com/articles/s41467-023-39018-y (cit. on p. 32).

[130] D. Kolosov et al. “Anatomy of Deep Learning Image Classification and Object Detection on Commercial Edge Devices: A Case Study on Face Mask Detection”. In: *IEEE Access* 10 (2022), pp. 109167–109186. issn: 2169-3536. doi: 10.1109/ACCESS.2022.3214214. url: https://ieeexplore.ieee.org/document/9918067/ (cit. on p. 32).

[131] Z. Zhao et al. “PIS-YOLO: Real-Time Detection for Medical Mask Specification in an Edge Device”. In: *Computational Intelligence and Neuroscience* 2022 (2022-11), pp. 1–11. issn: 1687-5273, 1687-5265. doi: 10.1155/2022/6170245. url: https://www.hindawi.com/journals/cin/2022/6170245/ (cit. on p. 32).

[132] E. Guney, C. Bayilmis, and B. Cakan. “An Implementation of Real-Time Traffic Signs and Road Objects Detection Based on Mobile GPU Platforms”. In: *IEEE Access* 10 (2022), pp. 86191–86203. issn: 2169-3536. doi: 10.1109/ACCESS.2022.3198954. url: https://ieeexplore.ieee.org/document/9857826/ (cit. on p. 32).

[133] M. A. Farooq, W. Shariff, and P. Corcoran. “Evaluation of Thermal Imaging on Embedded GPU Platforms for Application in Vehicular Assistance Systems”. In: *IEEE Trans. Intell. Veh.* 8.2 (2023-02), pp. 1130–1144. issn: 2379-8904, 2379-8858. doi: 10.1109/TIV.2022.3158094. url: https://ieeexplore.ieee.org/document/9732195/ (cit. on p. 32).

[134] K. Sarvajcz, L. Ari, and J. Menyhart. “AI on the Road: NVIDIA Jetson Nano-Powered Computer Vision-Based System for Real-Time Pedestrian and Priority Sign Detection”. In: *Applied Sciences* 14.4 (2024-02), p. 1440. issn: 2076-3417. doi: 10.3390/app14041440. url: https://www.mdpi.com/2076-3417/14/4/1440 (cit. on p. 32).

[135] B. Ma et al. “Distracted Driving Behavior and Driver’s Emotion Detection Based on Improved YOLOv8 With Attention Mechanism”. In: *IEEE Access* 12 (2024), pp. 37983–37994. issn: 2169-3536. doi: 10.1109/ACCESS.2024.3374726. url: https://ieeexplore.ieee.org/document/10463042/ (cit. on p. 32).

[136] NVIDIA Corporation. *Programming Model*. 2024. url: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model (visited on 2024-08-08) (cit. on p. 33).

[137] Autoware Foundation. *Home page - Autoware*. 2023. url: https://autoware.org (visited on 2024-09-23) (cit. on pp. 37, 39).

[138] M. Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software* 3 (2009), p. 5 (cit. on p. 37).

[139] Open Robotics. *ROS: Why ROS?* url: https://www.ros.org/blog/why-ros/ (visited on 2024-09-24) (cit. on p. 37).

[140] Open Robotics. *ROS: The ROS Ecosystem*. url: https://www.ros.org/blog/ecosystem/ (visited on 2024-09-24) (cit. on pp. 37, 38).

[141] S. Macenski et al. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Sci. Robot.* 7.66 (2022-05), eabm6074. issn: 2470-9476. doi: 10.1126/scirobotics.abm6074. url: https://www.science.org/doi/10.1126/scirobotics.abm6074 (cit. on pp. 37–40, 42).

[142] micro-ROS. *micro-ROS | ROS 2 for microcontrollers*. 2024. url: https://micro.ros.org (visited on 2024-09-24) (cit. on p. 38).

[143] Open Robotics. *ROS: Getting Started*. url: https://www.ros.org/blog/getting-started/ (visited on 2024-09-24) (cit. on p. 38).

[144] F. Visentin et al. "A mixed-autonomous robotic platform for intra-row and inter-row weed removal for precision agriculture". In: *Computers and Electronics in Agriculture* 214 (2023-11), p. 108270. issn: 01681699. doi: 10.1016/j.compag.2023.108270. url: https://linkinghub.elsevier.com/retrieve/pii/S0168169923006580 (cit. on p. 39).

[145] A. Zachariae et al. "Human-robot interactions in autonomous hospital transports". In: *Robotics and Autonomous Systems* 179 (2024-09), p. 104755. issn: 09218890. doi: 10.1016/j.robot.2024.104755. url: https://linkinghub.elsevier.com/retrieve/pii/S0921889024001398 (cit. on p. 39).

[146] L. Connolly et al. "SlicerROS2: A Research and Development Module for Image-Guided Robotic Interventions". In: *IEEE Trans. Med. Robot. Bionics* (2024), pp. 1–1. issn: 2576-3202. doi: 10.1109/TMRB.2024.3464683. url: https://ieeexplore.ieee.org/document/10684721/ (cit. on p. 39).

[147] S. Kolb et al. "6G in medical robotics: development of network allocation strategies for a telerobotic examination system". In: *Int J CARS* (2024-09). issn: 1861-6429. doi: 10.1007/s11548-024-03260-6. url: https://link.springer.com/10.1007/s11548-024-03260-6 (cit. on p. 39).

[148] J. Badger et al. "ROS in Space: A Case Study on Robonaut 2". In: *Robot Operating System (ROS)*. Vol. 625. Cham: Springer International Publishing, 2016, pp. 343–373. isbn: 978-3-319-26052-5 978-3-319-26054-9. doi: 10.1007/978-3-319-26054-9_13. url: http://link.springer.com/10.1007/978-3-319-26054-9_13 (cit. on p. 39).

[149] L. Flückiger et al. “Astrobee Robot Software: A Modern Software System for Space”. In: (2018) (cit. on p. 39).

[150] NASA. *Astrobee - NASA*. 2024. url: https://www.nasa.gov/astrobee/ (visited on 2024-09-26) (cit. on p. 39).

[151] NASA. *Robotics - NASA*. 2024. url: https://www.nasa.gov/reference/jsc-robotics/ (visited on 2024-09-26) (cit. on p. 39).

[152] B. Gerkey. *ROS...in...space!* 2022-02. url: https://www.openrobotics.org/blog/2022/2/2/rosinspace (visited on 2024-09-26) (cit. on p. 39).

[153] A. Probe et al. “Space ROS: An Open-Source Framework for Space Robotics and Flight Software”. In: *AIAA SCITECH 2023 Forum*. National Harbor, MD & Online: American Institute of Aeronautics and Astronautics, 2023-01. isbn: 978-1-62410-699-6. doi: 10.2514/6.2023-2709. url: https://arc.aiaa.org/doi/10.2514/6.2023-2709 (cit. on p. 39).

[154] Open Robotics. *Announcing the OSRF Project Committee for Space ROS*. 2023-02. url: https://www.openrobotics.org/blog/2023/2/7/announcing-the-osrf-project-committee-for-space-ros (visited on 2024-09-26) (cit. on p. 39).

[155] Open Robotics. *SpaceROS: Home*. url: https://space.ros.org (visited on 2024-09-26) (cit. on p. 39).

[156] D. Hong and C. Moon. “Autonomous Driving System Architecture with Integrated ROS2 and Adaptive AUTOSAR”. In: *Electronics* 13.7 (2024-03), p. 1303. issn: 2079-9292. doi: 10.3390/electronics13071303. url: https://www.mdpi.com/2079-9292/13/7/1303 (cit. on p. 39).

[157] Autoware Foundation. *Case Studies - Autoware*. 2023. url: https://autoware.org/case-studies/ (visited on 2024-09-26) (cit. on p. 39).

[158] Open Robotics. *Internal ROS 2 interfaces - ROS 2 Documentation: Jazzy documentation*. 2024. url: https://docs.ros.org/en/jazzy/Concepts/Advanced/About-Internal-Interfaces.html (visited on 2024-09-27) (cit. on p. 39).

[159] Open Robotics. *Basic Concepts - ROS 2 Documentation: Jazzy documentation*. 2024. url: https://docs.ros.org/en/jazzy/Concepts/Basic.html (visited on 2024-09-27) (cit. on p. 40).

[160] Open Robotics. *Nodes - ROS 2 Documentation: Jazzy documentation*. 2024. url: http://docs.ros.org/en/jazzy/Concepts/Basic/About-Nodes.html (visited on 2024-09-27) (cit. on p. 41).

[161] Open Robotics. *Topics - ROS 2 Documentation: Jazzy documentation*. 2024. url: http://docs.ros.org/en/jazzy/Concepts/Basic/About-Topics.html (visited on 2024-09-27) (cit. on p. 41).

[162] Open Robotics. *Services - ROS 2 Documentation: Jazzy documentation*. 2024. url: http://docs.ros.org/en/jazzy/Concepts/Basic/About-Services.html (visited on 2024-09-27) (cit. on p. 42).

[163] Open Robotics. *Actions - ROS 2 Documentation: Jazzy documentation*. 2024. url: http://docs.ros.org/en/jazzy/Concepts/Basic/About-Actions.html (visited on 2024-09-27) (cit. on p. 42).

[164] Open Robotics. *Parameters - ROS 2 Documentation: Jazzy documentation*. 2024. url: http://docs.ros.org/en/jazzy/Concepts/Basic/About-Parameters.html (visited on 2024-09-27) (cit. on p. 42).

[165] Open Robotics. *Creating a launch file - ROS 2 Documentation: Jazzy documentation*. 2024. url: https://docs.ros.org/en/jazzy/Tutorials/Intermediate/Launch/Creating-Launch-Files.html (visited on 2024-09-27) (cit. on p. 42).

[166] Open Robotics. *Introspection with command line tools - ROS 2 Documentation: Jazzy documentation*. 2024. url: https://docs.ros.org/en/jazzy/Concepts/Basic/About-Command-Line-Tools.html (visited on 2024-09-27) (cit. on p. 43).

[167] Flutter. *Flutter - Build apps for any screen*. 2024. url: https://flutter.dev (visited on 2024-04-11) (cit. on pp. 44, 52).

[168] Flutter. *Multi-Platform*. 2024. url: https://flutter.dev/multi-platform (visited on 2024-04-11) (cit. on pp. 44, 52).

[169] E. Lee and C.-Y. Lee. "PPG-Based Smart Wearable Device With Energy-Efficient Computing for Mobile Health-Care Applications". In: *IEEE Sensors J.* 21.12 (2021-06), pp. 13564–13573. issn: 1530-437X, 1558-1748, 2379-9153. doi: 10.1109/JSEN.2021.3069460. url: https://ieeexplore.ieee.org/document/9389545/ (cit. on p. 44).

[170] Flutter. *Showcase - Flutter apps in production*. 2024. url: https://flutter.dev/showcase (visited on 2024-04-11) (cit. on p. 44).

[171] Flutter. *Flutter Showcase | BMW*. 2024. url: https://flutter.dev/showcase/bmw (visited on 2024-04-11) (cit. on p. 44).

[172] Flutter. *Flutter Showcase | Toyota*. 2024. url: https://flutter.dev/showcase/toyota (visited on 2024-04-11) (cit. on p. 44).

[173] Flutter. *Material 2 Components widgets*. 2024. url: https://docs.flutter.dev/ui/widgets/material2 (visited on 2024-04-11) (cit. on pp. 45, 46).

[174] Flutter. *Cupertino (iOS-style) widgets*. 2024. url: https://docs.flutter.dev/ui/widgets/cupertino (visited on 2024-04-11) (cit. on pp. 45, 46).

[175] Flutter. *Flutter architectural overview*. 2024. url: https://docs.flutter.dev/resources/architectural-overview#anatomy-of-an-app (visited on 2024-04-11) (cit. on p. 45).

[176] Dart. *Dart programming language*. 2024. url: https://dart.dev (visited on 2024-04-11) (cit. on p. 45).

[177] Flutter. *Building user interfaces with Flutter*. 2024. url: https://docs.flutter.dev/ui (visited on 2024-04-11) (cit. on p. 46).

[178] Flutter. *Widget catalog*. 2024. url: https://docs.flutter.dev/ui/widgets (visited on 2024-04-11) (cit. on p. 46).

[179] Flutter. *Text class - widgets library - Dart API*. 2024. url: https://api.flutter.dev/flutter/widgets/Text-class.html (visited on 2024-04-11) (cit. on p. 46).

[180] Flutter. *Container class - widgets library - Dart API*. 2024. url: https://api.flutter.dev/flutter/widgets/Container-class.html (visited on 2024-04-11) (cit. on p. 46).

[181] Flutter. *ListView class - widgets library - Dart API*. 2024. url: https://api.flutter.dev/flutter/widgets/ListView-class.html (visited on 2024-04-11) (cit. on p. 46).

[182] Flutter. *AnimatedOpacity class - widgets library - Dart API*. 2024. url: https://api.flutter.dev/flutter/widgets/AnimatedOpacity-class.html (visited on 2024-04-11) (cit. on p. 46).

[183] L. Perinello and O. Gaggi. "Accessibility of Mobile User Interfaces using Flutter and React Native". In: *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*. Las Vegas, NV, USA: IEEE, 2024-01, pp. 1–6. isbn: 9798350304572. doi: 10.1109/CCNC51664.2024.10454681. url: https://ieeexplore.ieee.org/document/10454681/ (cit. on p. 47).

[184] Firebase. *Firebase*. 2024. url: https://firebase.google.com (visited on 2024-04-11) (cit. on p. 47).

[185] Firebase. *Firebase Products*. 2024. url: https://firebase.google.com/products-build (visited on 2024-04-11) (cit. on pp. 47, 52).

[186] J. Rhim et al. "A Deeper Look at Autonomous Vehicle Ethics: An Integrative Ethical Decision-Making Framework to Explain Moral Pluralism". In: *Front. Robot. AI* 8 (2021-05), p. 632394. issn: 2296-9144. doi: 10.3389/frobt.2021.632394. url: https://www.frontiersin.org/articles/10.3389/frobt.2021.632394/full (cit. on p. 49).

[187] K. Bengler et al. "Three Decades of Driver Assistance Systems: Review and Future Perspectives". In: *IEEE Intell. Transport. Syst. Mag.* 6.4 (2014), pp. 6–22. issn: 1939-1390. doi: 10.1109/MITS.2014.2336271. url: http://ieeexplore.ieee.org/document/6936444/ (cit. on p. 49).

[188] J. Zhang et al. "Data-Driven Intelligent Transportation Systems: A Survey". In: *IEEE Trans. Intell. Transport. Syst.* 12.4 (2011-12), pp. 1624–1639. issn: 1524-9050. doi: 10.1109/TITS.2011.2158001. url: http://ieeexplore.ieee.org/document/5959985/ (cit. on p. 50).

[189] Y. Kang, H. Yin, and C. Berger. “Test Your Self-Driving Algorithm: An Overview of Publicly Available Driving Datasets and Virtual Testing Environments”. In: *IEEE Trans. Intell. Veh.* 4.2 (2019-06), pp. 171–185. issn: 2379-8904, 2379-8858. doi: 10.1109/TIV.2018.2886678. url: https://ieeexplore.ieee.org/document/8667012/ (cit. on p. 50).

[190] J. Gadzi□ski. “Perspectives of the use of smartphones in travel behaviour studies: Findings from a literature review and a pilot study”. In: *Transportation Research Part C: Emerging Technologies* 88 (2018-03), pp. 74–86. issn: 0968090X. doi: 10.1016/j.trc.2018.01.011. url: https://linkinghub.elsevier.com/retrieve/pii/S0968090X18300366 (cit. on p. 50).

[191] L. Shen and P. R. Stopher. “Review of GPS Travel Survey and GPS Data-Processing Methods”. In: *Transport Reviews* 34.3 (2014-05), pp. 316–334. issn: 0144-1647, 1464-5327. doi: 10.1080/01441647.2014.903530. url: http://www.tandfonline.com/doi/abs/10.1080/01441647.2014.903530 (cit. on p. 50).

[192] Z. Wang, S. Y. He, and Y. Leung. “Applying mobile phone data to travel behaviour research: A literature review”. In: *Travel Behaviour and Society* 11 (2018-04), pp. 141–155. issn: 2214367X. doi: 10.1016/j.tbs.2017.02.005. url: https://linkinghub.elsevier.com/retrieve/pii/S2214367X17300224 (cit. on p. 50).

[193] W. Wojtak et al. “Towards Endowing Intelligent Cars with the Ability to Learn the Routines of Multiple Drivers: A Dynamic Neural Field Model”. In: *Computational Science and Its Applications – ICCSA 2021*. Vol. 12952. Cham: Springer International Publishing, 2021, pp. 337–349. isbn: 978-3-030-86972-4 978-3-030-86973-1. doi: 10.1007/978-3-030-86973-1_24. url: https://link.springer.com/10.1007/978-3-030-86973-1_24 (cit. on pp. 51, 71, 123, 124).

[194] Dart. *Dart overview*. 2024. url: https://dart.dev/overview (visited on 2024-04-11) (cit. on p. 52).

[195] Y. Sandamirskaya. “Dynamic neural fields as a step toward cognitive neuromorphic architectures”. In: *Front. Neurosci.* 7 (2014). issn: 1662-453X. doi: 10.3389/fnins.2013.00276. url: http://journal.frontiersin.org/article/10.3389/fnins.2013.00276/abstract (cit. on p. 64).

[196] J. Minnen, I. Glorieux, and T. P. van Tienoven. “Transportation habits: evidence from time diary data”. In: *Transportation Research Part A: Policy and Practice* 76 (2015), pp. 25–37. doi: 10.1016/j.tra.2014.12.013. url: https://www.sciencedirect.com/science/article/pii/S0965856415000105 (cit. on p. 68).

[197] W. Wojtak et al. “Robust working memory in a two-dimensional continuous attractor network”. In: *Cognitive Neurodynamics* (2023), pp. 1–17. doi: 10.1007/s11571-023-09979-3. url: https://link.springer.com/article/10.1007/s11571-023-09979-3 (cit. on pp. 68, 82, 118, 121).

[198] J. S. Hunter. "The exponentially weighted moving average". In: *Journal of quality technology* 18.4 (1986), pp. 203–210. doi: 10.1080/00224065.1986.11979014. url: https://www.tandfonline.com/doi/abs/10.1080/00224065.1986.11979014 (cit. on pp. 72, 125).

[199] R. L. Davis and Y. Zhong. "The biology of forgetting—a perspective". In: *Neuron* 95.3 (2017), pp. 490–503. doi: 10.1016/j.neuron.2017.05.039. url: https://www.sciencedirect.com/science/article/pii/S0896627317304981 (cit. on p. 73).

[200] M. C. Anderson and J. C. Hulbert. "Active forgetting: Adaptation of memory by prefrontal control". In: *annual review of psychology* 72.1 (2021), pp. 1–36. doi: 10.1146/annurev-psych-072720-094140. url: https://www.annualreviews.org/content/journals/10.1146/annurev-psych-072720-094140 (cit. on p. 73).

[201] P. Guimarães et al. "A data recording mobile application to create datasets of vehicle users' routines". In: *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE. 2022, pp. 167–172. doi: 10.1109/ICARSC55462.2022.9784770. url: https://ieeexplore.ieee.org/document/9784770 (cit. on p. 75).

[202] W. Wojtak et al. "A neural integrator model for planning and value-based decision making of a robotics assistant". In: *Neural Computing and Applications* 33.8 (2021), pp. 3737–3756. doi: 10.1007/s00521-020-05224-8. url: https://link.springer.com/article/10.1007/s00521-020-05224-8 (cit. on pp. 82, 118).

[203] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *Nature* 521.7553 (2015-05), pp. 436–444. issn: 0028-0836, 1476-4687. doi: 10.1038/nature14539. url: https://www.nature.com/articles/nature14539 (cit. on p. 82).

[204] Y. Duan, Y. L.V., and F.-Y. Wang. "Travel time prediction with LSTM neural network". In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. Rio de Janeiro: IEEE, 2016-11, pp. 1053–1058. isbn: 978-1-5090-1889-5. doi: 10.1109/ITSC.2016.7795686. url: https://ieeexplore.ieee.org/document/7795686/ (cit. on p. 82).

[205] X. Luo et al. "Spatiotemporal Traffic Flow Prediction with KNN and LSTM". In: *Journal of Advanced Transportation* 2019 (2019-02), pp. 1–10. issn: 0197-6729, 2042-3195. doi: 10.1155/2019/4145353. url: https://www.hindawi.com/journals/jat/2019/4145353/ (cit. on p. 82).

[206] L. Alessandretti et al. "Evidence for a conserved quantity in human mobility". In: *Nature human behaviour* 2.7 (2018), pp. 485–491. doi: 10.1038/s41562-018-0364-x. url: https://www.nature.com/articles/s41562-018-0364-x (cit. on p. 108).

[207] J. Carvalho. "Neuromorphic Cognition: an approach toward robotic systems that learn, adapt and predict sequential tasks with time constraints". MA thesis. Universidade do Minho (Portugal), 2022 (cit. on p. 117).

[208] M. Davies et al. “Advancing neuromorphic computing with Loihi: A survey of results and outlook”. In: *Proceedings of the IEEE* 109.5 (2021-05), pp. 911–934 (cit. on p. 117).