Horizon 2020 Program (2014-2020)

A computing toolkit for building efficient autonomous applications leveraging humanistic intelligence (TEACHING)

D4.1: Initial report on the AIaaS system[†]

Contractual Date of Delivery	31/10/2020
Actual Date of Delivery	31/12/2020
Deliverable Security Class	Public
Editor	*Claudio Gallicchio (UNIPI)*
Contributors	UNIPI: Claudio Gallicchio, Davide Bacciu, Daniele Di Sarli, Pouria Faraji, Andrea Cossu, Matteo Montalbetti HUA: Christos Sardianos, Iraklis Varlamis, Konstantinos Tserpes, Dimitrios Michail, Charalampos Davalas. CNR: Emanuele Carlini, Massimo Coppola M: Salvatore Petroni
Quality Assurance	*Reviewer Pietro Cassarà (CNR)*

[†] *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871385.*

The TEACHING Consortium

University of Pisa (UNIPI)	Coordinator	Italy
Harokopio University of Athens (HUA)	Principal Contractor	Greece
Consiglio Nazionale delle Ricerche (CNR)	Principal Contractor	Italy
Graz University of Technology (TUG)	Principal Contractor	Austria
AVL List GmbH	Principal Contractor	Austria
Marelli Europe S.p.A.	Principal Contractor	Italy
Ideas & Motion	Principal Contractor	Italy
Thales Research & Technology	Principal Contractor	France
Information Technology for Market Leadership	Principal Contractor	Greece
Infineon Technologies AG	Principal Contractor	Germany

Document Revisions & Quality Assurance

Internal Reviewers

Pietro Cassarà (CNR)

Revisions

Version	Date	By	Overview
1.0	30/12/2020	Claudio Gallicchio (UNIPI), Daniele Di Sarli (UNIPI), Christos Sardianos (HUA), Salvatore Petroni (M), Emanuele Carlini (CNR), Massimo Coppola (CNR)	Final, including addressed comments.
0.95	28/12/2020	Reviewer: Pietro Cassarà (CNR)	Comments on draft
0.9	27/12/2020	Claudio Gallicchio (UNIPI), Daniele Di Sarli (UNIPI), Christos Sardianos (HUA), Salvatore Petroni (M)	Introduction and conclusions. Added Sections 2.7 and 2.8.
0.8	22/12/2020	Claudio Gallicchio (UNIPI), Daniele Di Sarli (UNIPI), Christos Sardianos (HUA), Iraklis Varlamis (HUA), Emanuele Carlini (CNR), Massimo Coppola (CNR), Salvatore Petroni (M)	Executive summary and refinements in Sections 2, 4, 5.
0.7	11/12/2020	Claudio Gallicchio (UNIPI), Daniele Di Sarli (UNIPI), Pouria Faraji (UNIPI), Andrea Cossu (UNIPI), Christos Sardianos (HUA), Iraklis Varlamis (HUA), Konstantinos Tserpes (HUA), Dimitrios Michail (HUA), Charalampos Davalas (HUA), Emanuele Carlini (CNR), Massimo Coppola (CNR), Salvatore Petroni (M),	Major refinements in all the Sections.
0.5	07/12/2020	Claudio Gallicchio (UNIPI), Daniele Di Sarli (UNIPI), Pouria Faraji (UNIPI), Christos Sardianos (HUA), Iraklis Varlamis (HUA), Konstantinos Tserpes (HUA) Dimitrios Michail (HUA), Charalampos Davalas (HUA), Emanuele Carlini (CNR), Massimo Coppola (CNR), Salvatore Petroni (M),	First draft of the document including major editing in all the Sections
0.3	13/11/2020	Claudio Gallicchio (UNIPI), Daniele Di Sarli (UNIPI), Salvatore Petroni (M), Christos Sardianos (HUA), Emanuele Carlini (CNR), Massimo Coppola (CNR)	Outline of the Sections.
0.2	05/11/2020	Claudio Gallicchio (UNIPI)	Outline Structure of all the Sections
0.1	15/10/2020	Claudio Gallicchio (UNIPI)	ToC.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

EC	European Commission
WP	Work Package
CPSoS	Cyber-Physical Systems of Systems
ML	Machine Learning
DL	Deep Learning
NN	Neural Network
RNN	Recurrent Neural Network
ESN	Echo State Network
RC	Reservoir Computing
HSM	Human State Monitoring
ER	Emotion Recognition
HAR	Human Activity Recognition
HA	Healthcare Applications
HDC	Heart Diseases Classification
SD	Stress Detection
WLD	Workload Detection
HR	Heart Rate
ECG	Electrocardiography
EEG	Electroencephalography
EMG	Electromyography
EDA	Electrodermal Activity
BVP	Blood Volume Pulse
ST	Skin Temperature
RSP	Respiration
GD	Gaze Detection
ADAS	Advanced driver assistance systems
PID	Proportion Integration Differentiation
MPC	Model Predictive Control
LM	Learning Module
RSU	Roadside Unit
DMU	Decision Making Unit
UGV	Unmanned Ground Vehicle

Executive Summary

The Deliverable D4.1, titled “Initial report on the AIaaS system” provides the first report on the state-of-the-art analysis, preliminary requirement specifications, and design of the Artificial Intelligence as a Service (AIaaS) system developed as part of the WP4. As such, it describes the necessary preliminary work on AI and ML algorithms that are intended to support the human-centric intelligence and adaptivity in TEACHING CPSoS applications.

In particular, this document describes the suite of computational methodologies adopted from an AI and ML perspective, focusing on dynamically driven Recurrent Neural Networks and efficiently trained Reservoir Computing, metrics for Neural Networks dependability and safety, Human State Monitoring and personalization in autonomous vehicles. It also presents the preliminary requirement specifications and the design, including the first version of the architecture of the AIaaS system. In addition to this, as a plus in anticipation of the work to be conducted in Y2, this document introduces some preliminary results in Federated Reservoir Computing, stress estimation using Reservoir Computing, and personalization in autonomous vehicles.

This document is structured as follows. In Section 1 we introduce the scopes of this document and the relations with the other deliverables delivered at M12. Then, in Section 2 we give an extensive state-of-the-art analysis to introduce the core AI/ML methodology bricks that will be central for the development of the TEACHING AIaaS software infrastructure. This background analysis is followed in Section 3 by a description of the identified requirements for the development of the AIaaS system. In Section 4, we present the architectural design, illustrating how the TEACHING platform is intended to support the AIaaS applications. In Section 5 we illustrate preliminary results of the work expected in Y2. Finally, in Section 6 conclude the document.

1. Introduction

The ***TEACHING project*** aims at designing a computing platform and the associated software toolkit to support the development and the deployment of ***adaptive and dependable Cyber-Physical Systems of Systems (CPSoS) applications***, enabling them to exploit ***sustainable human feedback*** to drive, optimize and personalize the provisioning of the offered services.

Central to the project is the concept of ***Humanistic Intelligence*** in autonomous CPSoS, where the human and the cybernetic components cooperate in a process of mutual empowerment. The concept of distributed, efficient, and dependable AI, leveraging edge computing support, is hence fundamental to achieve the goals of the project.

The essential goal of WP4 is to design methodologies and tools to create the TEACHING ***Artificial Intelligence as a Service (AIaaS)*** software infrastructure for CPSoS applications, developing human-centric personalization mechanisms. The activities of this WP are organized accordingly: 1) develop a toolkit that implements the core methodological components of the TEACHING AIaaS; 2) develop AI methodologies that are specialized in the recognition and characterization of the human physiological, emotional, and cognitive state from streams of data gathered from heterogeneous sensors; 3) develop the necessary functionalities to self-adapt and personalize AI models, implicitly using the human state information; 4) develop privacy-aware AI methodologies that can be bundled within the AIaaS toolkit. The above-mentioned activities are respectively mapped into the four WP4 tasks T4.1-4. Of these, T4.1 "AI as a service" started at M1, T4.2 "AI for human monitoring" and T4.3 "AI models for human-centric personalization" started together at M7, while T4.4 "Privacy-aware AI models" starts at M12.

According to the overall organization of the TEACHING project development, the first year of work (Phase 1) in WP4 focused on the necessary preliminary research in AI/ML-related algorithms to support humanistic intelligence in CPSoS applications, as well as in the definition of the AIaaS architecture. In this context, *the objectives of this deliverable are the following*:

- Provide an in-depth state-of-the-art analysis of the AI and ML-based methodologies that have been identified as fundamental to the realization of the TEACHING AIaaS system infrastructure, given the nature of the learning problems involved;
- Report the preliminary requirements of the TEACHING AIaaS system;
- Develop a preliminary design of the TEACHING AIaaS system, indicating how the TEACHING platform supports AI applications.

Moreover, in anticipation of the work expected in Y2, in this document, we also provide – as a plus to the deliverable objectives - a report on the preliminary results of the technical analysis on relevant AI/ML techniques performed within the aims of WP4.

The rest of this deliverable is structured as follows. We start in Section 2, presenting a state-of-the-art analysis of the major techniques in AI and ML that have been identified as crucial to the development of the core learning functionalities of the AIaaS. In particular, we focus on dynamical Neural Networks (NNs) models under the umbrella of Recurrent Neural Networks (RNNs) and efficiently trained Reservoir Computing models. The analysis is there complemented by a thorough review of the fundamental concepts in Federated and Continual Learning, assessment of NNs dependability, human-state monitoring approaches, and personalization in autonomous vehicle applications. The state-of-the-art analysis is followed by a description of the preliminary TEACHING AIaaS requirements, in Section 3. The system requirements are analyzed and put forward at different levels (including functional, non-functional, architectural, safety, performance, and structural), and the aim is to give a detailed

breakdown originating from the WP4 technological perspective, within the general project's context (as described in Deliverable D5.1). On these bases, we introduce the preliminary design of the TEACHING AIaaS system in Section 4, where we detail the basic elements of the software architecture and the TEACHING AI application model. The fundamental idea behind our proposal is to enable the developer to construct an AI application by composing a pool of functional units implementing the TEACHING learning-related functionalities (illustrated in Section 2). These represent a sort of Lego blocks that can be chained together exploiting automatic routing of the data that is necessary for training and inference. We also take the chance to introduce some preliminary results on the development work that has been conducted in the active tasks T4.1-3, in Section 5. Finally, we delineate the conclusions in Section 6.

Before that, in the next Section 1.1, we briefly recall the links to the other deliverables.

1.1. Relationship with other deliverables

In compliance with its intended purpose within the scopes of the TEACHING project, this document (D4.1) presents the fundamental ML methodology bricks that will realize the learning-related functionalities of the TEACHING AIaaS, together with its preliminary requirements and design. This document is delivered within a group of related project deliverables, namely D1.1, D2.1, D3.1, D4.1, and D5.1 (listed in Table 1), all of which serve as a mean of verification for milestone MS1, entitled *Release of the TEACHING design (requirements, specification, and architecture)*.

Table 1 Deliverable grouping for verification of TEACHING Milestone 1

D1.1	Report on TEACHING related technologies SoA and derived CPSoS requirements
D2.1	State-of-the-art analysis and preliminary requirement specifications for the computing and communication platform
D3.1	Initial Report on Engineering Methods and Architecture Patterns of Dependable CPSoS
D4.1	Initial report on the AIaaS system
D5.1	Initial use case specifications

Below, we briefly report the major connections of D4.1 with the other deliverables in Table 1.

Relations with D1.1 - Within the conceptual architecture of the TEACHING platform described in D1.1 (Section 6) the work described in D4.1 focuses on the necessary preliminary analysis for the development of the AI Toolkit in the TEACHING SDK, linked to the ML Libraries in the Execution/Management Environment layer. Details on specific technologies and tools of interest for D4.1 can be found in D1.1, including libraries for Machine Learning and Deep Learning (D1.1, Section 5.3.2), and TEACHING selected sensors for software/hardware monitoring (D1.1, Section 5.2.1) and for human monitoring (D1.1, Section 5.2.1), which represent major input sources for the learning modules described in this deliverable.

Relations with D2.1 - D2.1 discusses the preliminary work conducted about the TEACHING computing platform that intends to support the AI/ML modules and methodologies that are described in this document. The High-Performance Computing and Communication Infrastructure (HPC^2I, whose preliminary design is offered in Section 4 of D2.1) provides the necessary computational support to the learning modules that constitute the core bricks of the ML capabilities in TEACHING. The properties of the selected AI/ML techniques and the involved requirements (as described herein D4.1) have an impact on D2.1, especially

concerning communication and data transfer for federated learning modules and functionalities envisaged in the AIaaS system. Moreover, D2.1 (Section 5.3) presents a discussion on the frameworks for ML that are related to the content of this document, presenting the available alternatives from the perspective of the computing platform.

Relations with D3.1 - D3.1 informs the activities of the other WPs from the dependability point of view. Regarding WP4, this interaction mainly regards the development of human-centric adaptivity in CPSoS in a dependable manner. In particular, the implementation of the learning-based functionalities described in D4.1 will be explored within the dependability architectural perspective illustrated in D3.1 Section 4. Besides, Section 6 in D3.1 addresses ML-related topics involved by distributed intelligence on heterogeneous devices from the dependability perspective. Finally, the methods described as part of the state of the art in D4.1 can naturally apply also to problems in cybersecurity identified in D3.1 Section 5.3.

Relations with D5.1 - D5.1 gives the initial specifications of the TEACHING use cases in which the ML-related core technologies described in D4.1 will be adopted to enhance the CPSoS functionalities. Specifically, the Avionics and the Automotive use cases are described respectively in Sections 4 and 5 of D5.1. Moreover, Section 5 in D5.1 gives an overview of the hardware platform chosen for demonstration of both the use cases, therefore being of interest for the ML algorithms described herein D4.1.

The mapping of the viewpoints of the technical WPs, as well as the integration intentions of the TEACHING technology bricks in domain use cases, is depicted in Figure 1.

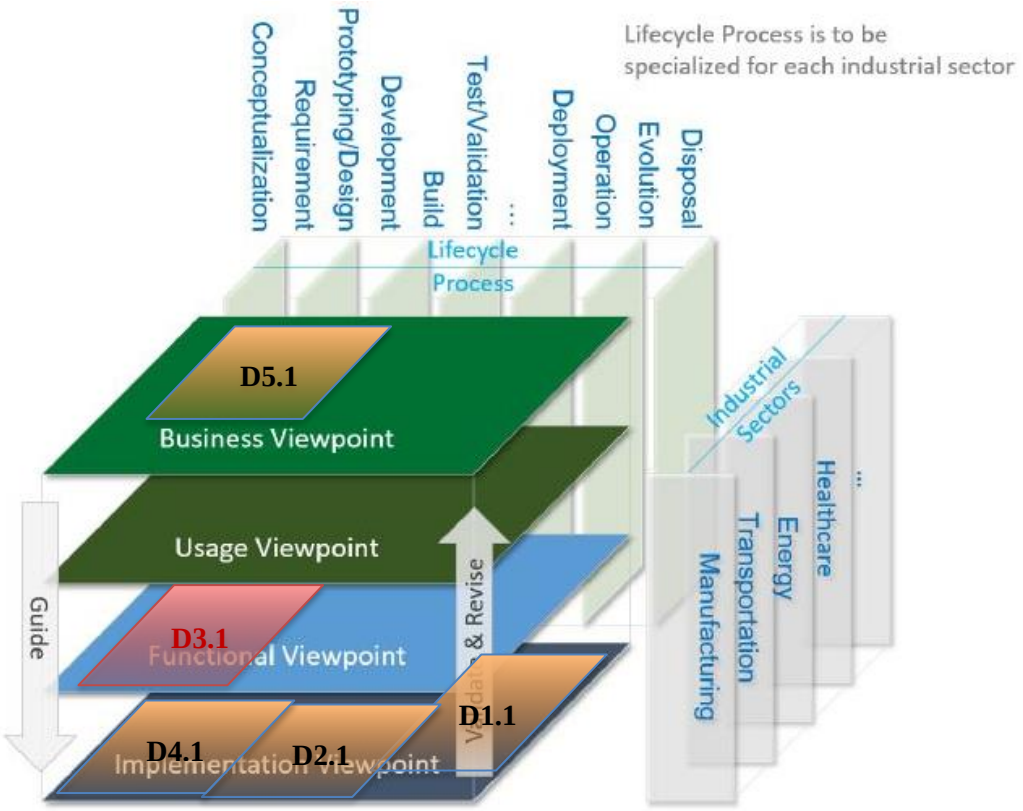

Figure 1 Depiction of the IIRA Viewpoints from [1] and mapping of focus of TEACHING Deliverables MS1**.**

[1] https://iiot-world.com/industrial-iot/connected-industry/iic-industrial-iot-reference-architecture/

2. State-of-the-art Analysis

This section aims at providing an overview of the core Machine Learning (ML) methodologies that are of special interest in the design and development of the TEACHING AIaaS software infrastructure.

AI, and particularly ML, nowadays play a fundamental role in many industrial applications, as a key enabler to infer salient information from huge amounts of data, possibly gathered from heterogeneous sensors, and especially in scenarios where the human interacts with pervasive computing environments. Moreover, AI and ML are increasingly being used as a way to effectively monitor human status conditions, and, more in general, to detect physiological patterns that might impact the way in which humans interact in the environments where they live or work. The relevance of this kind of monitoring activity becomes fundamental especially when the human factor is considered in the context of safety-critical systems operations. Taking a physiological computing perspective, human physiological data streams are considered as input information that enables the creation of a user-state representation of interest, giving implicit feedback that can drive the personalization of the system's services.

From a methodological perspective, the success of Deep Learning (DL) [1] [2] models is now apparent to everyone. Deep NNs hold state-of-the-art results in a large variety of (mainly cognitive-related) applications, including multimedia and natural language processing. However, common DL approaches typically require considerable computational, communication, and storage resources to produce effective results. At the same time, is more and more frequent the case in which the success of an AI application is not defined solely in terms of accuracy, and other constraints such as time and (computational) efficiency of the learning algorithms are of key importance. This is the case, for instance, of embedded and distributed AI, where the objective is to enable the AI application running on the edge device to be able not only to do inference (e.g., using pre-trained learning modules under factory conditions), but to also perform adaptation and personalization. In this context, the field of Deep Randomized NNs is recently hitting the literature [3] [4] [5] [6] [7] [8], showing a great trade-off between accuracy and complexity. The basic idea is to maximally exploit the biases of Deep NN architectures while reducing the amount of trainable internal connections as much as possible to minimize the complexity of training algorithms. This has intriguing mathematical and theoretical grounds, nicely summarized by a famous quote by Rahimi and Recht [7] "randomization is computationally cheaper than optimization", as well as important roots in computational neuroscience [9].

Summing up, approaching learning problems in the field of human-centric personalization for CPSoS applications requires taking into account ML methodologies that are able to easily treat the inherent temporal structure of the sensors' data streams, ensuring flexibility, noise tolerance, and efficiency to enable implementations in low-powerful edge devices. In light of these considerations, while we keep an open perspective in considering a large candidate pool of ML methods, we find that the class of Recurrent NNs, especially implemented according to the dictates of Deep Randomized NNs and Reservoir Computing, is particularly appropriate to constitute the basic ML methodology bricks (and the core reference) in our applications. For our purposes, this class of learning models needs to be integrated with a number of advanced ML techniques, including Federated and Continual Learning, controlled by suitable metrics for NNs dependability, and considered as parts of human-centric personalization systems.

The rest of this section is structured as follows. We start in Section 2.1 by giving an overview of the fundamental concepts in Recurrent and Reservoir Computing NN models, ranging from basic approaches to recent advances in the field (including Deep Reservoir Computing NNs). Then, in Sections 2.2 and 2.3 we respectively discuss the topics of Federated Learning and Continual Learning, which will be exploited in synergy with the NN concepts to provide local, decentralized, and federated adaptation/personalization. When it comes to putting algorithms involving learning modules in production for safety-critical applications, dependability and safety identify important desirable features. In Section 2.4, we present a number of relevant computable metrics for assessing the dependability and safety of NN algorithms. The current state-of-the-art in ML approaches for Human State Monitoring is summarized, along with the TEACHING view, in Section 2.5, while Section 2.6 covers the aspects of personalization in autonomous vehicles applications. In Section 2.7 we contextualize the analyzed methodologies in terms of modern software frameworks for ML. Finally, in Section 2.8 we briefly discuss how the state-of-the-art described here will enable the TEACHING use cases.

2.1. Recurrent and Reservoir Computing Neural Networks

Recurrent Neural Networks (RNNs) [10] are a computing system loosely inspired by the biological neural networks in the brain. The recurrent connections between the neurons, or nodes, allow RNNs to exhibit a memory of the recent past, thus enabling an effective processing of temporal data. Formally, the discrete-time evolution of the internal state of a RNN can be defined as:

$$\mathbf{h}(t) = tanh\big(\mathbf{V}\mathbf{x}(t) + \mathbf{W}\mathbf{h}(t-1)\big),$$

where $\mathbf{h}(t) \in \mathbb{R}^{N_H}$ is the state of the network at time t, $\mathbf{x}(t) \in \mathbb{R}^{N_X}$ is the input at time t, and $\mathbf{V} \in \mathbb{R}^{N_H \times N_X}$, and $\mathbf{W} \in \mathbb{R}^{N_H \times N_H}$ are the parameters which can be adjusted by the training process. From a given state $\mathbf{h}(t)$, the network can compute a prediction $\mathbf{y}(t) \in \mathbb{R}^{N_Y}$ as:

$$\mathbf{y}(t) = \mathbf{U}\mathbf{h}(t),$$

where $\mathbf{U} \in \mathbb{R}^{\mathbf{N_Y} \times \mathbf{N_H}}$ is another matrix of parameters that are adjusted by a training algorithm. Graphically, a RNN can be represented as in Figure 2, where it is illustrated a RNN with an input layer of 3 neurons, one recurrent layer of 7 neurons, and an output layer of 2 neurons. Notice the self-connections within the recurrent layer. Also notice the self-connections within the recurrent layer.

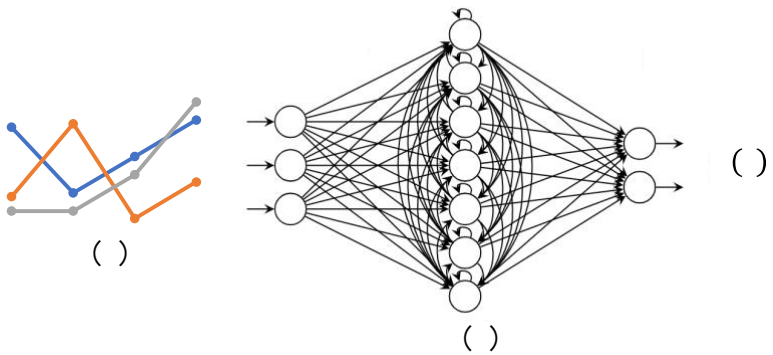

Figure 2 A recurrent neural network.

2.1.1. Backpropagation Training of Recurrent Neural Networks

In the case of supervised learning, training a RNN involves adjusting the parameters of the network on the basis of a set of input-output examples. The examples are provided to the network, processed, and the produced output is compared with the expected one. The parameters are updated to reduce the mismatch between the actual and expected outputs.

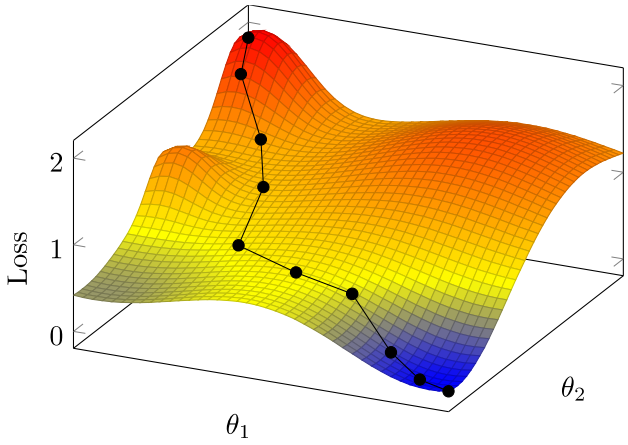

Figure 3 The iterative training process for RNNs.

In particular, first a *loss function* $\mathcal{L}$ is chosen as a scalar measure of the discrepancy between actual and expected outputs. Then, finding the optimal values for the parameters **V**, **W**, and **U** boils down to solving a minimization problem:

$$\underset{\{\mathbf{V},\mathbf{W},\mathbf{U}\}}{\arg\min} \sum_{i=1}^{N_D} \sum_{t=1}^{N} \mathcal{L}\left(\bar{\mathbf{y}}^{(i)}(t), \mathbf{y}^{(i)}(t)\right) = \underset{\{\mathbf{V},\mathbf{W},\mathbf{U}\}}{\arg\min}\, E,$$

in which we have denoted with $\bar{\mathbf{y}}^{(i)}(t)$ and $\mathbf{y}^{(i)}(t)$ respectively the expected and actual output at time t for the i-th example in the dataset.

The minimization problem, which in general involves a non-convex surface, can be solved iteratively by computing the gradients of the error E with respect to all parameters and taking steps in the parameter space by a method known as Gradient Descent.

The process of Gradient Descent is illustrated in Figure 3: the surface represents the value of the loss function for different parametrizations of the network (here, only two generic parameters θ_1 and θ_2 are shown). The gradient descent algorithm iteratively computes the gradient of the loss function and adjusts the parameters towards a direction which reduces the error. A single update of a generic parameter θ is described by:

$$\Delta\theta = -\eta \frac{\partial E}{\partial \theta},$$

where η is a scalar *learning rate*, a hyperparameter whose value is selected to support the learning process.

The gradients in a RNN trained by Gradient Descent are typically computed by a technique named Backpropagation Through Time (BPTT). The name comes from the fact that the computation of the gradients, which is performed by decomposition via the *chain rule*, must propagate from the final time step back to the beginning of the computation.

While often very effective, training a network by BPTT is quite expensive both in terms of time and computational resources. In fact, hardware accelerators (such as GPUs or TPUs) must be employed in many practical situations in order to obtain reasonable training times. This makes it impractical to adopt BPTT with Gradient Descent in embedded application on low-power devices. As such, in these contexts, alternative, more efficient training algorithms must be employed (see, for example, the reservoir computing paradigm described in Section 2.1.2).

Moreover, BPTT does not suit well to applications involving distributed learning scenarios: also in these contexts, alternative training algorithms or techniques should be taken into account. Finally, when training RNNs via BPTT and Gradient Descent it must be kept into consideration the potential issue of gradient vanishing or explosion [11], which may make training difficult over long input sequences.

2.1.2. Reservoir Computing

Reservoir Computing (RC) [6] is a paradigm for learning models in which the input sequences are mapped into a high-dimensional, non-linear system that is called *reservoir*. The key characteristic of the approach is that the reservoir is fixed, i.e., its parameters are not adjusted by training. After a sequence has been mapped into a vector by the reservoir, a *readout* component reads it to produce a prediction. The parameters of the readout are the only ones in the model to be subject to training, and this allows to have a relatively simple readout component that does not depend on temporal dynamics, thus avoiding many of the associated challenges. These include the general problem of credit assignment (i.e., which activations were responsible for the error or success of the network?), and the issues of gradient vanishing or explosion.

The reservoir in RC systems can be implemented in a multitude of ways, from developing artificial dynamical systems to exploiting the intrinsic computational power of physical systems or substrates [12]. For example, a reservoir can be implemented by a grid of semiconductor optical amplifiers and photodetectors for reading the light signals by the readout, with advantages in terms of low power consumption and extremely fast computation. Finally, the most popular option is to employ a RNN as a reservoir: in this case we use the term *Echo State Networks*.

2.1.3. Echo State Networks

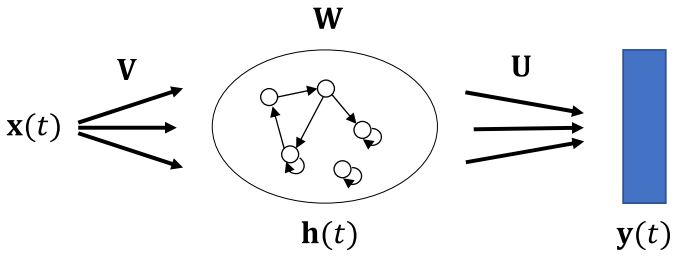

Figure 4 Schematic representation of an Echo State Network.

Echo State Networks (ESNs) [13] [14] are a RC model in which a properly initialized RNN is used as the reservoir (see Figure 4). In fact, the evolution of the internal state of the reservoir in an ESN is described as for a RNN, i.e.:

$$\mathbf{h}(t) = tanh\big(\mathbf{V}\mathbf{x}(t) + \mathbf{W}\mathbf{h}(t-1)\big).$$

Here, however, the matrices **V** and **W** are randomly initialized and then kept fixed, without training.

In ESNs, the readout is also typically implemented as a linear layer:

$$\mathbf{y}(t) = \mathbf{U}\mathbf{h}(t).$$

As such, the readout can be trained with efficient direct methods like Moore-Penrose pseudoinversion or Ridge Regression. While the equations describing the computations of the model are identical to those of a RNN, avoiding training the recurrent part of the model offers significant advantages in terms of efficiency.

In Figure 4 it is illustrated a schematic representation of an ESN. On the left, the input $\mathbf{x}(t)$ is fed to the reservoir, whose activations are denoted with $\mathbf{h}(t)$. Finally, the activations of the reservoir are fed to the readout layer.

UNIPI has made available several implementations of ESNs (and of deep variants, see Section 2.1.4 below), developed with different frameworks, including NumPy[2] and TensorFlow[3].

Initialization

The recurrent hidden layer of an ESN must be initialized in such a way to satisfy a stability property. In particular, the reservoir must meet the so-called Echo State Property (ESP) [13], which ensures that the reservoir will asymptotically wash out any information from the initial conditions of the system. As a stability property, the ESP makes the internal representation developed by the network robust to perturbations. In practice, we have both a sufficient and a necessary condition for the ESP, which have been proven in the context of an ESN with tanh as the activation function [13]:

$$\|\mathbf{W}\|_2 < 1 \implies \text{ESP} \implies \rho(\mathbf{W}) < 1,$$

where $\rho(\mathbf{W})$ indicates the spectral radius of matrix **W** (i.e., its largest eigenvalue in absolute value).

The input-to-reservoir matrix **V** can be initialized by generating a random matrix from a uniform distribution between $-\omega_{in}$ and ω_{in}, where $\omega_{in} \in \mathbb{R}^+$ is a hyperparameter. For the reservoir matrix **W**, in order to satisfy the ESP it is sufficient to randomly generate a matrix and then rescale its norm such that it is less than unity. Moreover, the reservoir matrix **W** is typically sparse: it is not uncommon to have less than 20% of connectivity.

Leaky ESN

A notable variant of a basic ESN is denoted as leaky ESN. The reservoir of a leaky ESN uses leaky-integrator neurons [15], which act as a lowpass filter whose leaking rate is determined and fixed at model selection time. In this case, the state transition function is modified as:

$$\mathbf{h}(\mathrm{t}) = (1-a)\mathbf{h}(t-1) + a\, tanh\big(\mathbf{V}\mathbf{x}(t) + \mathbf{W}\mathbf{h}(t-1)\big),$$

where a $\in \mathbb{R}$ is the leaking rate, under the constraint $0 < a \leq 1$.

In practical applications, leaky-integrator neurons are the most common choice for the reservoir of an ESN, as it enables dealing with fast-changing sensor data which may be collected at very high frequencies.

[2] https://github.com/lucapedrelli/DeepESN

[3] https://github.com/gallicch/DeepRC-TF

Training

In an ESN, the only parameters that are subject to training are those in **U** and thus a closed-form solution can be obtained by extremely fast algorithms such as ridge regression. More in detail, first the input sequences are fed to the reservoir. Then, the states of interest (e.g., all of them for sequence-to-sequence transductions, or just the last state of each sequence for classification tasks) are collected column-wise into a matrix $\mathbf{H} \in \mathbb{R}^{N_H \times N_D}$. At this point the readout can be trained by finding a solution to the following least squares problem:

$$\min_{U} \|\mathbf{UH} - \bar{\boldsymbol{Y}}\|_2^2.$$

Here, $\bar{\boldsymbol{Y}} \in \mathbb{R}^{N_Y \times N_D}$ indicates the column-wise concatenation of the target vectors. A solution to the minimization problem can be computed in closed-form as follows:

$$\mathbf{U} = \bar{\mathbf{Y}}\mathbf{H}^T(\mathbf{H}\mathbf{H}^T + \lambda_r \mathbf{I})^{-1}$$

where $\boldsymbol{I}$ is the identity matrix and $\lambda_r \in \mathbb{R}^+$ is a Tikhonov regularizer parameter which can be chosen for example by random search along with ω_{in} and $\rho(\mathbf{W})$ (and a in case of a leaky ESN).

Training can also be performed in an incremental fashion, or in batches. In this case, if the batches are indexed by k, it is sufficient to compute the following summation of matrices:

$$\mathbf{A} = \bar{\mathbf{Y}}\mathbf{H}^{\mathrm{T}} = \sum_k \mathbf{A}_{(k)} = \sum_k \bar{\mathbf{Y}}_{(\mathrm{k})}\mathbf{H}_{(\mathrm{k})}^{\mathrm{T}}$$

$$\mathbf{B} = \mathbf{H}\mathbf{H}^T = \sum_k \mathbf{B}_{(k)} = \sum_k \mathbf{H}_{(k)}\mathbf{H}_{(k)}^{\mathrm{T}}$$

Then, the readout weights can be computed as:

$$\mathbf{U} = \mathbf{A}\mathbf{B}^{-1}.$$

In addition to a more practical training procedure in the presence of large quantities of data, an incremental training also enables convenient aggregation strategies in Federated Learning contexts. In fact, transmitting the local matrices $\mathbf{A}_{(k)}$ and $\mathbf{B}_{(k)}$ to the centralized server in the federation allows it to train a readout as if all data was available locally.

2.1.4. Advances in Reservoir Computing

The research in the area of ESNs is quite active. One of the most attractive directions is towards reservoir topologies which can lead to better dynamics than those that can be obtained from a random reservoir. The connectivity pattern within the reservoir is defined by **W**, which in its most simple form is a sparse random matrix. However, research has shown that one can engineer specific connectivity patterns with the aim of reducing the stochasticity in the process, increasing efficiency, or reducing memory usage [16]. In the following we will review some advanced reservoir topologies.

Permutation

An important and interesting feature of orthogonal reservoir matrices is that they lead to networks with high memory capacity [17]. Several ways exist to build orthogonal reservoirs, one particularly simple of which is to use a permutation connectivity pattern. In the context of ESNs, this kind of topology has been empirically studied in [18], where it was shown to achieve good memorization skills at the same time improving the performance of randomly initialized reservoirs in tasks involving non-linear mappings. Interestingly, the permutation topology has

been investigated in [19] as a way to implement orthogonal reservoir matrix structures, under the name of Critical ESNs.

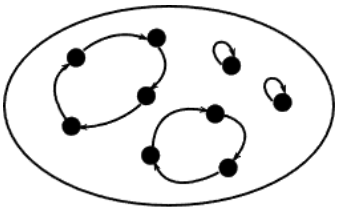

Figure 5 A "permutation" reservoir.

In the *permutation* topology pattern (Figure 5) we take $\mathbf{W} := v\mathbf{P}$, where $\mathbf{P} \in \{0,1\}^{N_H \times N_H}$ is a randomly generated permutation matrix and $v \in \mathbb{R}^+$ determines the spectral radius of $\mathbf{W}$, i.e. $\rho(\mathbf{W}) = v$ (since $\mathbf{W}$ as defined is orthogonal). The permutation pattern produces isolated cycles of various lengths in the connectivity. With this scheme, the space complexity of the reservoir matrix shrinks to $O(N_H)$. The reduced space complexity has positive repercussions also when the reservoir needs to be transmitted over the network, for example in a federated learning scenario. The time complexity for computing the matrix-vector product $\mathbf{W}\mathbf{h}(t-1)$ in the state transition function also becomes linear in the number of recurrent units. Moreover, computing $\rho(\mathbf{W})$ for a generic $\mathbf{W}$ in order to rescale it to meet the ESP is a costly operation. With a permutation reservoir, $\mathbf{W}$ can be directly initialized with the desired spectral radius with zero additional cost.

Ring

Unlike in a permutation reservoir, in a ring reservoir (Figure 6) there is just one cycle that includes all units. The matrix $\mathbf{W}$ is defined as for a permutation reservoir, but $\mathbf{P}$ has a special structure (it is the identity matrix whose first row or column is displaced to the last position). This particular topology is a special case of the permutation topology, and as such it is also orthogonal. Thus, in addition to the advantages of a permutation reservoir, with this pattern the amount of stochasticity is reduced and the space complexity is constant.

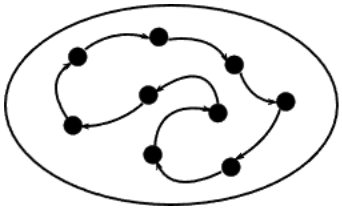

Figure 6 A "ring" reservoir.

Reservoirs following this architectural organization have been subject of several studies in RC literature. Notable instances in this regard are given by the work in [16], in which the ring topology is studied in the context of orthogonal reservoir structures, and by the work in [20], where the study is carried out under the perspective of architectural design simplification for minimum complexity ESN construction. One interesting outcome of previous analysis on the ring topology is that, compared to randomly initialized reservoirs, it shows superior memory capacity that, at least in the linear case, approaches the optimal value [20]. While this optimal memory characterization has been extensively analyzed in literature for the more general class of orthogonal recurrent weight matrices (see e.g. [17] [21] [22]), the ring topology presents the advantage of a strikingly simple (and sparse) dynamical network construction.

Deep Echo State Networks

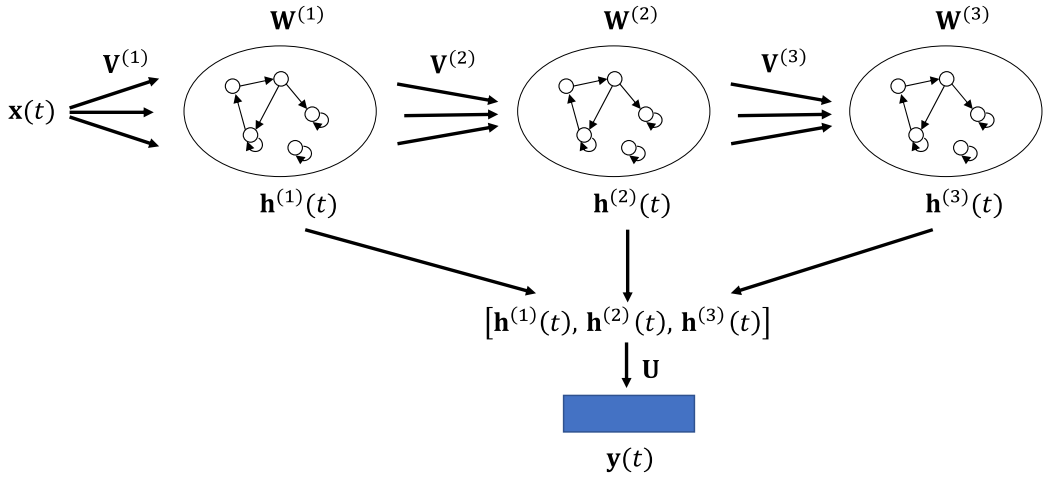

Figure 7 A Deep Echo State Network.

A Deep Echo State Network (DeepESN) [23] [24] extend the concept of reservoir processing of time-series data to the case of deep architectures. Like for the ESN, we can identify two main components (the reservoir and the readout) and the training procedure is basically unchanged. The difference is that the reservoir of a DeepESN is organized hierarchically, with multiple recurrent layers as shown in Figure 7. The first layer is driven by the external input, but then each subsequent layer takes as input the activation state of the previous one. Omitting the bias term for the ease of notation, the state transition function of the first layer is defined as follows:

$$\mathbf{h}^{(1)}(\mathrm{t}) = \left(1 - \mathrm{a}^{(1)}\right)\mathbf{h}^{(1)}(\mathrm{t}-1) + \mathrm{a}^{(1)} \tanh\left(\mathbf{V}^{(1)}\mathbf{x}(\mathrm{t}) + \mathbf{W}^{(1)}\mathbf{h}^{(1)}(\mathrm{t}-1)\right),$$

while for every layer $l > 1$ it is defined as:

$$\mathbf{h}^{(\mathrm{l})}(t) = \left(1 - a^{(\mathrm{l})}\right)\mathbf{h}^{(\mathrm{l})}(t-1) + a^{(\mathrm{l})} \tanh\left(\mathbf{V}^{(\mathrm{l})}\mathbf{h}^{(l-1)}(t) + \mathbf{W}^{(\mathrm{l})}\mathbf{h}^{(\mathrm{l})}(t-1)\right).$$

The states from all the layers can be concatenated into a single, fixed-size vector to be used as input to the readout layer.

The layered organization of the reservoir has been observed to determine an enrichment of the neural representations developed in the dynamical component, exploiting the inherent positive bias of depth in RNN architectural design. For instance, it has been shown that DeepESNs are able to produce reservoir dynamics at multiple time-scales [24] and at multiple frequencies [25]. This effect arises simply thanks to the architectural configuration of the reservoir: no training is involved, and only asymptotic stability of the deep reservoir needs to be controlled at initialization stage [26]. Thus, DeepESNs represent an efficient approach for learning over complex temporal data that exhibits different levels of time granularity. Moreover, the architecture of a DeepESN can be interpreted as a constrained version of a shallow ESN with the same total number of reservoir units, in which some of the connections are removed. Then, the DeepESN can represent both a simplification of the corresponding shallow ESN, and a convenient generalization of the whole RC approach.

Preliminary studies on Gated Reservoir Computing Neural Networks

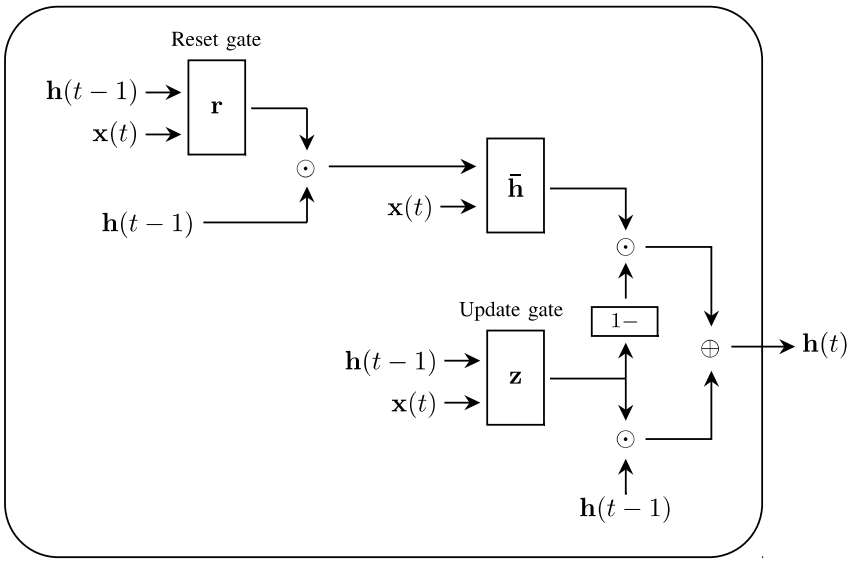

Figure 8 Graphical representation of the recurrent cell of a Gated ESN for a generic time step $\boldsymbol{t}$.

Reservoir Computing approaches like ESNs are extremely efficient in terms of training time and resources thanks to their use of randomly initialized parameters that do not need to be trained. Unfortunately, basic ESNs are also unable to effectively deal with complex long-term dependencies in the data, which are present whenever the target output at time step t depends on inputs that were fed to the network several time steps before. This weakness is closely linked to the concepts of fading memory and stability, thus inherently implied by the ESP in the case of ESNs, but in general all RNNs are affected by the problem.

Luckily, by the use of gating mechanisms it has been shown that this problem can be alleviated for fully trained RNNs. For this reason, we have started investigating the problem of equipping ESNs with gating mechanisms [27]. This approach takes inspiration from the gated architectures which are popular in the context of fully trained RNNs, and in particular from the Gated Recurrent Unit [28].

In a Gated ESN, the state transition function of the reservoir is extended with gating mechanisms as follows (the bias terms are omitted for the ease of notation):

$$\mathbf{r}(t) = \sigma\left(\mathbf{V}^{(r)}\mathbf{x}(t) + \mathbf{W}^{(r)}\mathbf{h}(t-1)\right)$$
$$\mathbf{z}(t) = \sigma\left(\mathbf{V}^{(z)}\mathbf{x}(t) + \mathbf{W}^{(z)}\mathbf{h}(t-1)\right)$$
$$\bar{\mathbf{h}}(t) = tanh\left(\mathbf{V}\mathbf{x}(t) + \mathbf{W}\big(\mathbf{r}(t) \odot \mathbf{h}(t-1)\big)\right)$$
$$\mathbf{h}(t) = \mathbf{z}(t) \odot \mathbf{h}(t-1) + \big(1 - \mathbf{z}(t)\big) \odot \bar{\mathbf{h}}(t)$$

Here, $\mathbf{r}(t)$ and $\mathbf{z}(t)$ represent the activations at time t of respectively the *reset gate* and the *update gate*. The behavior of the gates is regulated by the respective matrices ($\mathbf{V}^{(r)}$, $\mathbf{W}^{(r)}$, $\mathbf{V}^{(z)}$, and $\mathbf{W}^{(z)}$). The activations of the gates are applied to the previous state $\mathbf{h}(t-1)$ or to the current candidate state $\bar{\mathbf{h}}(t)$ by means of the Hadamard product (i.e., the element-wise product of vectors indicated by the $\odot$ operator). A graphical representation of the network is illustrated in Figure 8.

The architecture of the cell is identical to the one of a GRU, however in Gated ESN the parameters controlling the activations of $\mathbf{r}(t)$, $\mathbf{z}(t)$, and $\bar{\mathbf{h}}(t)$ are not trained. In the second variant that has been tested, the architecture is still unchanged, but the parameters controlling $\mathbf{r}(t)$ and $\mathbf{z}(t)$ are trained while the dynamics of $\bar{\mathbf{h}}(t)$ remain untrained.

When all the matrices in the reservoir are left untrained, the model shows improvements in predictive performance with respect to a basic, non-leaky ESN, while the training cost remains low. On the other hand, adjusting the matrices in the gates ($\mathbf{V}^{(r)}$, $\mathbf{W}^{(r)}$, $\mathbf{V}^{(z)}$, and $\mathbf{W}^{(z)}$) via BPTT while keeping the rest untrained allows the model to reach significantly higher predictive performance. In this case, however, the training cost also increases.

2.1.5. Applications of ESNs to sensor data processing in intelligent sensors

As a state-of-the-art approach in the context of efficient learning in temporal domains, the efficacy of Echo State Networks has been tested and successfully demonstrated in numerous practical problems. For some examples of ESN applications see [29], [30] and references therein, as well as more recent works e.g. in [31] [32] [33]). The DeepESN approach also proved effective in a variety of domains, including Ambient Assisted Living (AAL) [34], medical diagnosis [35], speech and polyphonic music processing [25] [36], meteorological forecasting [37] [38], solar irradiance prediction [39], energy consumption and wind power generation prediction [40], destination prediction [41], car parking and bike-sharing in urban computing [38], financial market predictions [38], and industrial applications (for blast furnace off-gas) [42] [43].

More specifically, ESNs are particularly suited to applications involving low-power intelligent sensors and devices. For example, in clinical settings, a learning system based on ESNs has been developed for the automatic assessment of balance abilities in elderly people [44]. The model was able to perform accurate assessments from data coming from just a single balancing exercise.

In the context of ambient assisted living, ESNs have been used as part of integrated and self-organizing solutions that reduce the need of costly pre-programming and maintenance of robot ecologies [45] [46]. ESNs have also been successfully employed for the prediction, from noisy radio signal strength data, of indoor user movement [33] [47] [48] or activity [31].

Since the dimensionality of the reservoir has high impact on both the network performance and efficiency, in the context of low-power devices it is important to adopt solutions to mitigate the computational cost. The techniques in Section 2.1.4 lend themselves well to this purpose. For example, ring topologies allow to significantly reduce not only the time required for training and for evaluation, but also the storage and transmission cost of the reservoir matrix. This is in contrast with what happens in a basic ESN (and, in general, in conventional RNNs), where one would have to store transmit N_H^2 floating-point values. Additional techniques include using a small finite weight alphabet for the non-zero weight values [33], which reduces the number of bits needed for the representation. Similarly, constraining the reservoir weights to n-bit integers leads to drastic effects on the computational performance of the ESN without impacting the predictive performance [49].

2.2. Federated Learning

Traditional machine learning is considered centralized, meaning that there is a single server which hosts both the data and the model in one place. However, there are some cases in which centralized machine learning is not the solution. Federated Learning (FL) in contrast, is an approach in which the data and the main model are separated and the model does not know any

specific knowledge about the data. In this approach, the data is spread among different clients or devices and each client has its own model [50]. Client's current model is downloaded and computes an updated model at the device itself using local data. In FL, collaboratively a shared prediction model is learnt while all the data is kept on devices. Therefore, in this approach there is no need to store the data in the same place as the main server in the cloud.

The general procedure for implementing the FL are as follows:

1. Each client downloads the current model from the main server. Configurations and parameters of the model for all clients are the same.
2. Each local model in the clients is improved by learning from its own local data stored in the local devices.
3. Updates to the local model, such as model parameters, are sent back to the main server in the cloud. Only this update to the model is sent to the cloud, e.g., using encrypted communication.
4. Sent updates are immediately averaged with other user updates to improve the shared model. The training data are kept in the devices, and no individual updates are stored in the cloud.

The main benefit of FL is ensuring privacy. Not only the main server, but also each client does not have any knowledge about individual training data, therefore, privacy is ensured to be preserved. Second benefit is that models become smarter, since they do not have to rely on only local data which are much less and unbalanced with respect to all other client's data. There are some aspects that must be taken into account for designing such a mechanism:

- the number of parameters of each local model should be small, since clients are limited in terms of computational power;
- the number of parameters effects amount of information transmitted server and clients;
- the goal is not to achieve better performance compared to centralized models, but rather to show it can achieve similar performance.

Traditional FL approaches rely on simple averages if the models' parameters transmitted from the clients to the server. For our purposes, we find it particularly intriguing that in the case of ESNs, where training is restricted to a simple linear layer in the neural network, it is possible to rely on the compositionality/incrementality of the training process (as illustrated in Section 2.1.3) to compute an *exact* configuration of the trainable weights as if all of the training data were available in one single place. In practice, each client *k* can locally compute its own $\boldsymbol{A_k}$ and $\boldsymbol{B_k}$ matrices based on the locally available data. These matrices get summed in the server, which can finally compute the output weights in closed-form. This approach is further described in Section 5.1, along with preliminary results in human state monitoring applications.

2.3. Continual Learning

Data coming from real world experience is generated by processes which change dynamically over time [51]. Dealing with non-stationary distributions means being able to adapt to drifts that may occur at any time, either abruptly or gradually. The effect of a drift in the generating process may be immediately evident in the observable data or it may require some time. Drift can last for a long time, thus establishing a new normality, or it can vanish quickly after it appeared. Finally, the same change occurred at a certain point in time may reoccur later (recurring concepts), thus requiring recall of previous knowledge. The field of Continual Learning (CL) [52], also called Lifelong learning, focuses on learning a potentially infinite

sequence of different tasks without forgetting previous knowledge. The Catastrophic Forgetting (CF) phenomenon is one of the main challenges faced by Continual Learning.

Real world applications require to deal with a continuous stream of mutable information, whose nature is not known a priori. Continual Learning is of utmost importance for such applications, since it equips predictive models with the ability to adjust to changes in the input stream and, at the same time, to consolidate already acquired knowledge. Without CL capabilities, even a subtle drift in the input would require retraining the entire predictive model with both new and old data. The training procedure, however, is often very costly and time consuming: until the new model is ready to be deployed, incoming drifts could cause a large drop of the performances in the old model, with potentially serious consequences. Moreover, retraining is unnecessary, since most of the information needed is already inside the model. Therefore, the best solution is to let the model learn continuously by keeping forgetting under control with CL techniques, thus avoiding the need of retraining.

A standard supervised CL setting is the class of incremental setting [53] [54], where the dataset is split into multiple tasks, each of which introduces new classes. In the Domain Incremental setting [53] the number of classes is fixed but the underlying distribution generating each class is subjected to drifts.

To monitor a CL experiment, different metrics can be computed. The simplest one is the average accuracy over all the encountered tasks. If CF effects are in place, average accuracy should decrease as learning progresses. Accuracy on the current task can be used to monitor the orthogonal requirement of learning new tasks. In fact, CF could be easily prevented by freezing the model to a previous state. Although capable of perfect recall, such model would not learn anything new.

CL does not only focus on CF. Since the model is learning a sequence of tasks, it is also useful to monitor the effect of learning on previous and, also, future tasks [55] [56].

The Backward Transfer [56] measures to what extent learning the current task improves performances on previous ones. Correspondingly, Forward Transfer [56] measures to what extent learning the current task affects future ones.

Computational efficiency is also an important factor in the evaluation of CL experiments. In particular, training time and model size or, more in general, memory occupancy characterize in which conditions a CL algorithm can scale and adapt to real world.

2.3.1. Continual Learning Strategies

The main objective of Continual Learning (CL) is to prevent catastrophic forgetting of previous knowledge when learning a sequence of different tasks [57] [58] [59]. This requires finding a trade-off between model's stability (the ability to preserve existing knowledge) and plasticity (the ability to acquire new information) [60] [61] [62].

Regularization strategies try to enforce stability on model's parameters by adding a penalization term to the standard loss function [63] [64] [65] [66]. Elastic Weight Consolidation (EWC) [67] computes the importance of each parameter for the current task by using the gradient of the loss averaged over each input sample. At training time, the penalty term keeps parameters close to their previous value in proportion to their importance. Online, more efficient variants of EWC have been proposed in [68] [69]. Learning without Forgetting [63] adopts a different approach than EWC: it prevents large drifts in output activations by using the predictions of previous versions of the model as soft targets in a distillation loss [70].

Architectural strategies enhance model's plasticity by enabling dynamic expansion during training. Forgetting is mitigated by inhibiting learning on old components. The type of expansion ranges from adding single units [71] [72] to stack entire networks next to the existing architecture [73] [74]. Architectural strategies may also exploit pruning techniques to reuse parameters which are redundant for the current task [75] [76].

Replay strategies keep an internal memory in which to store previously encountered patterns. The content of the memory is added to the current training set to prevent forgetting. Patterns to be included in the memory can be chosen at random or by following specific criteria [77] [78]. If the number of tasks is large, replay strategies have high memory requirements.

In order to mitigate this disadvantage, generative replay is often employed [79] [80]. In this setting, a generative model like a Generative Adversarial Network [81] is trained to approximate the current task distribution. Then, it is used to generate inputs similar to previous patterns. Therefore, the replay memory size corresponds to the size of the generative model and no longer depends on the number of tasks.

Even if regularization, architectural and replay strategies are the most common ones, there are other approaches which look promising: from Bayesian learning [82] [83] to spiking neural networks [84] [85] and ESN [86]. Ultimately, the choice of a specific CL strategy strongly depends on the application it is intended for and on the constraints, it has to satisfy.

2.4. Metrics for Neural Networks' Dependability & Safety

2.4.1. Safety critical aspects in Neural Networks

In the last few years, the automotive market is facing a new challenge related to self-driving vehicles. The key elements that allow autonomous driving are software elements that shall analyze single images or a series of images (frames) making up a video to recognize objects, learn from people's personal habits to provide the driving style that best suits the involved person, decide the best thing to do in relation to the current situation and so on. To accomplish all of these tasks ML based models are used.

Software elements based on ML techniques might include NNs of several kinds. The important aspect to stress is that all these software elements use ML techniques to learn from a collection of data and to provide the best answer or output. This feature makes them different from the traditional software where a programmer decides how the software should behave.

The uniqueness of the characteristics of a ML model makes this kind of software different from the traditional software and this implies problems when this software is used in safety critical applications such as those related to autonomous driving.

A safety critical point of all ML models is about their non-transparency [87]; they contain knowledge in an encoded form, but this encoding is harder to interpret for humans.

A further safety issue affecting ML models is that they typically do not operate perfectly and exhibits some error rate [87]. Thus, "correctness" is not always ensured, but there is only a statistical guarantee about the reliability of answer of a ML model.

ML models as NNs are based on supervised learning and they are trained using a subset of possible inputs that could be encountered operationally. This is the cause of another safety issue because the training set is necessarily incomplete and there is no guarantee that it is even

representative of the space of possible inputs. In addition, learning may overfit a model by capturing details incidental to the training set rather than general to all inputs [87].

All these characteristics of the ML models make them not suitable to be designed and assessed according the Functional Safety standard for road vehicles ISO 26262.

The ISO 26262 recommends the use of a Hazard Analysis and Risk Assessment (HARA) method to identify hazardous events in the system and to specify safety goals that mitigate the hazards. Each safety goal will be characterized by an ASIL, a level of safety measure that shall be ensured. From each safety goal are derived the requirements that shall be developed to realize HW parts and SW parts with the required ASIL [87].

The development of the software part follows the well-known V model for engineering shown below in Figure 9.

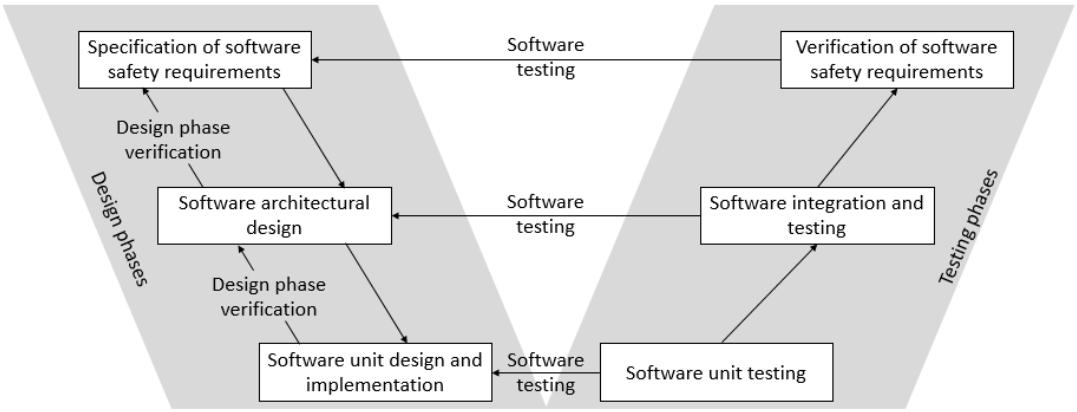

Figure 9 ISO 26262 part 6 - Product development at the software level

All these considerations led to the need to develop a methodology that would allow to evaluate the safety of an ML model in the family of NNs. These needs have led to the definition of the concept of dependability and its application to NNs in order to evaluate their efficiency under different points of view that involve also the safety.
In addition, the dependability allows to evaluate also security aspects; this shall be taken into account because some security issues can cause safety issues.

2.4.2. What is Dependability and its relationship with safety

In general, dependability can be defined as the credibility of a system, i.e., the degree of trust that can reasonably be placed in the answer of the system performing a particular task. The concept of dependability can be applied in general to each system or functionality but it has been analyzed and directed to be applied to NNs [88]. The fulfillment of dependability by a NNs ensures that a certain degree of safety is achieved. For this reason, the concept of dependability can be effectively applied to evaluate the safety of all kinds of NNs whose failure could have serious consequences on the people's health.

The attributes that a dependable system must have are: Robustness, Interpretability, Completeness and Correctness [88]. These attributes are also indicated with the acronym RICC, where each letter is the initial one of each attribute.

Here, is described the meaning of each attribute applied on a NN.

- Robustness: indicates how a NN is robust against various effects such as distortion or adversarial perturbation (which is closely related to security).
- Interpretability: indicates the understanding of what a NN has actually learned.
- Completeness: indicates if a NN has used training data possibly covering all important scenarios.
- Correctness: indicates how a NN is able to perform its task without errors.

Each of these attributes is closely related to the concept of safety.

Each safety related function shall be robust: this means that applying distortions or adversarial perturbations to the system, the function shall ensure that will not occur unforeseen behavior that can affect the safety of the involved persons [88]. This attribute is closely linked to the concept of stability of a NN concerning the initialization of an ESN.

Another attribute that each safety related function must have is the interpretability: the designer shall be aware of what the NN has learned and what the NN has not learned; in this way he can evaluate if the NN has the right knowledge of all those situations that if not correctly recognized could cause a physical harm to the involved people [88].

The third attribute that is related to the safety is the completeness; it is important to understand if the data used to train the NN model covers also those scenarios classified as potentially safety critical [88].

The last attribute regards the correctness; from a safety point of view this attribute ensure that the safety critical tasks are performed correctly [88].

Each NN dependability attribute can be estimated by determining a computable set of NN specific metrics. In Figure 10 we report all the metrics and their relationship with the dependability's attributes.

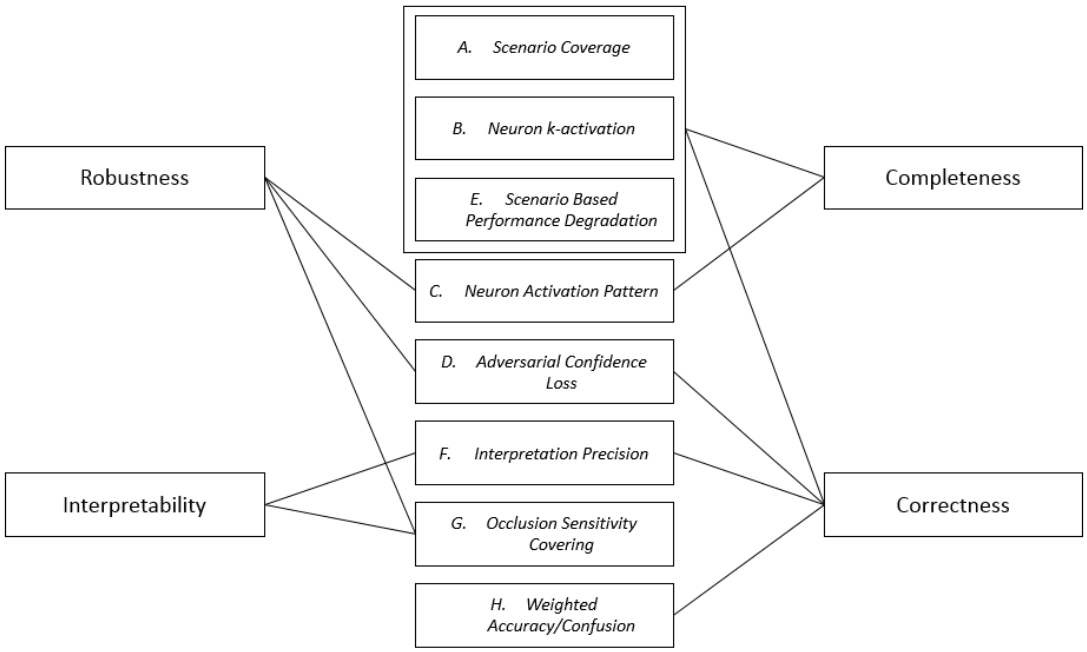

Figure 10 Relations between RICC attributes and the metrics.

2.4.3. Dependability evaluation metrics

The state-of-the-art concerning the calculation of metrics to quantify the dependability of ML models is still in its infancy. An approach which is enjoying good success is the one proposed in [88] where the metrics consider NNs models working on images. This approach can be suitable also in case we use time series data such as video streaming because the latter consist of series of images. Considering our context, we shall take into account that also other types of data coming from CPSoS (such as data streams coming from external vehicle sensors) are treated. However, some of the following metrics can be applied to different types of data because they give an output value by analysing the internal structure of the NN while this last is running; for example, they calculate the number of neurons activated with a given test set. On the other hand, the evaluation process of other metrics may need some adaptations. For the sake of simplicity in the following is shown the state-of-the-art in the same context considered in literature.

A. Scenario coverage metric M_{scene}

M_{scene}: metric to compute the scenarios quantity covered in the training step. For example given scenario there is a list $C = \langle C_1, \dots, C_n \rangle$ of operating conditions (e.g. weather condition, road condition, etc.), let $C_1 = \{sunny, cloudy, rainy\}$ represent the weather condition, $C_2 = \{stone, mud, tarmac\}$ represent the road surfacing and $C_3 = \{straight, curvy\}$ represent the incoming road orientation. Then $(sunny, stone, straight)$ and $(ruiny, tarmac, curvy)$ constitute two possible scenarios (see Figure 11). Since that to check the coverage of all possible scenarios is not feasible due to combinatorial explosion, the scenario coverage metric proposed is based on the concept of 2-projection.
To compute the metric shall be prepared a table recording all possible pairs of operating conditions (e.g.: road surfacing-weather, road orientation-weather and road surfacing-road orientation) and their possible values (e.g.: values of weather: sunny, cloudy, rainy). Then shall be filled all the cells of the table according the considered scenarios.

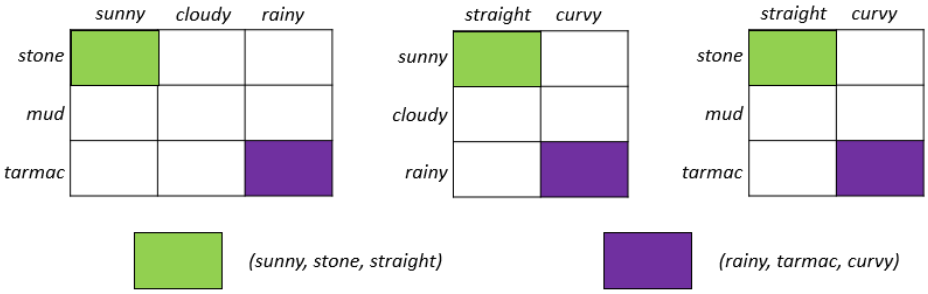

Figure 11 Computing scenario coverage metric via 2-projection table

The metric value will be the ratio between occupied cells and the total number of cells:

$$M_{scene} := \frac{\#\ of\ cells\ occupied\ by\ the\ data\ set}{\#\ of\ cells\ from\ 2-projection\ table}$$

In the provided example the value of the metric is:

$$M_{scene} := \frac{2+2+2}{9+6+6}$$

The scenario coverage metric affects completeness and correctness attributes of RICC.

B. Neuron k-activation metric $M_{neu-k-act}$

In a similar way to the previous metric, this metric observes the activation of neurons. One combination of neuron activation in the same layer can be compared to one scenario. In the same way to the case of the scenarios, all the combination of all neurons in the same layer generates a combinatorial explosion; for example, for a layer of 256 neurons there is a total of 2^{256} scenarios to be covered. Considering a specific layer to be analysed with *c* neurons and selecting an integer constant *k*, the metric can be calculated with the formula:

$$M_{neu-k-act} := \frac{\#\ of\ occupied\ cells\ due\ to\ the\ data\ set}{\binom{c}{k}\,(2^k)}$$

where the denominator represents the number of cells.

The neuron k-activation metric impacts completeness and correctness attributes.

C. Neuron activation pattern metric $M_{neu-pattern}$

While *k-activation* metric captures the completeness, the neuron activation pattern metric is used to understand the distribution of activation. For inputs within the same scenario, intuitively the activation pattern should be similar, implying that the number of activated neurons should be similar.

To calculate the metric the user shall consider an input set $\boldsymbol{X}$ consisting of images belonging to the same scenario and shall specify a layer of the NN (with *c* neurons) to be analyzed. Moreover, the user shall select from the input set **X**, a number γ of groups constituting a partition of the input set $\boldsymbol{X}$. For each input of each group $G_i(\boldsymbol{X})$, with $i \in \{1, \dots, \gamma\}$, the number of activated neurons in the specified layer shall be within the range $\left[\frac{c}{\gamma}(i-1), \frac{c}{\gamma}(i)\right]$.

Considered $G_j(\boldsymbol{X})$ the largest set among $G_1(\boldsymbol{X}), \dots, G_\gamma(\boldsymbol{X})$, the metric is evaluated by considering all inputs whose activation pattern, aggregated using the number of neurons being activated, significantly deviates from the majority. The neuron activation pattern metric can be calculated with the formula:

$$M_{neu-pattern} := \frac{\sum_{i\,:\,i \notin \{j-1, j, j+1\}} |G_i(\boldsymbol{X})|}{|\boldsymbol{X}|}$$

This metric reflects the robustness and completeness attribute

D. Adversarial confidence loss metric M_{adv}

This proposed metric adversarial confidence loss metric is useful in providing engineers an estimate of how a NN is robust. To provide a perturbed input, shall be chosen a parameter ϵ specifying the allowed perturbation and a set of perturbation techniques T_i. The function $T_i(x, \epsilon)$ gives in output a transformation of the perturbed input, while $\boldsymbol{y}(\boldsymbol{x})$ and $\boldsymbol{y}(T_i(\boldsymbol{x}, \epsilon)$ calculate the output of the NN respectively with the original input and with the perturbed and transformed input.

The equation to calculate the adversarial perturbation loss metric is:

$$M_{adv} := \frac{\sum_{in \in In} min_{i \in \{1,\dots,N\}} \boldsymbol{y}(T_i(\boldsymbol{x}, \epsilon)) - \boldsymbol{y}(\boldsymbol{x})}{|\boldsymbol{X}|}$$

The metric impacts robustness and correctness.

E. Scenario based performance degradation metric $M_{scen-perf-degr}$

Here can be used common performance metrics and detailed analysis considering each scenario can be performed. For missing input scenarios, the value of the metric can be discounted with a discount factor that can be taken from the computed scenario coverage metric. Example of commonly used performance metrics are validation accuracy and quantitative statistic measures such as MTBF (Mean Time Between Failures).

F. Interpretation precision metric $M_{interpret}$

This metric is aimed to evaluate if a NN for object detection makes its decision on the correct part of a single image or of a series of images (frames) making up a video.

When the NN is running, there is a probability *p* to correctly classify an object (obtaining a bounding box in the case of object detection). Then shall be calculated an occlusion sensitivity heatmap *H* where each pixel of the heatmap $h \in H$ maps to a position of the occlusion on the image.

Defined as P_{hot} the set of pixels that are classified as belonging to the object and $P_{occluding}$ the set of pixels constituting the object, the metric is computed as follows:

$$M_{interpret} := \frac{|P_{hot} \cap P_{occluding}|}{|P_{hot}|}$$

The interpretation precision metric affects the interpretability and correctness attributes.

G. Occlusion sensitivity covering metric M_{OccSen}

This metric tells us the ability of the network to recognize the object also if several parts of the images (frames) making up a video are occluded. The occlusion sensitivity covering metric is given by:

$$M_{OccSen} := \frac{|P_{hot} \cap P_{occluding}|}{|P_{occluding}|}$$

A high value of the metric implies that many positions of small occlusions can lead to a detection error. A low value of the metric implies that there is a greater chance of still detecting the object when it is partly occluded.

Occlusion sensitivity covering metric impacts robustness and interpretability attributes.

H. Weighted accuracy/confusion metric $M_{confusion}$

The goal of this metric is to assign different severity according the type of error that occurred. The severity of an error of a NN is not always the same. For example, confusing a pedestrian for a tree is more critical than in the opposite way. So, this metric considers different penalty terms as weights to capture different classification misses.

This is a general technique already used to evaluate ML algorithms but here the weights are determined.

Table 2 provides a summary over penalties to be applied in traffic scenarios.

Table 2 Qualitative severity of safety to be reflected as weights

A is classified to B	B (pedestrian)	B (vehicle)	B (background)
A (pedestrian)	n.a. (correct)	++	++++
A (vehicle)	+	n.a. (correct)	+++
A (background)	+	+	n.a. (correct)

The table assigns the highest penalty when a pedestrian (or bicycle) is incorrectly recognized as a background image (i.e., no object exists), as pedestrians are not protected, and it may easily lead to life threatening situations.

The metric gives indications about the correctness attribute.

2.5. Human State Monitoring (HSM) and perspectives

In the last decade, HSM is gaining interest from the scientific community and is mainly based on the analysis and fusion of multiple physiological signals. Physiological signals represent a rich source of information to address tasks such as Emotion Recognition (ER), Human Activity Recognition (HAR), Healthcare Applications (HA) such as Heart Diseases Classification (HDC) or Stress Detection (SD), Workload Detection (WLD), etc. Among the many published papers, several survey works attempt to provide a general picture of the field. These surveys focus on ER [89] [90], HAR [91], HA [92], and the application of DL techniques to physiological signal data [93].

The aspects of the human state that can be monitored fall into three main classes that will be discussed in more detail in the sections that follow: stress/discomfort, attention/distraction and fatigue detection.

2.5.1. Human in the loop

Aiming on advancing the orientation of current autonomous vehicles state-of-the-art, we introduce the aspect of human-machine collaboration, which refers to the development of more human-centered modules that put the human into the loop of the autonomous driving processes. In these terms, we emphasize on the symbiotic relation of drivers and self-driving vehicles, attempt to record all the parameters of this composite problem, and examine how HSM can be beneficial in this collaboration. The utter aim is to bring humans in this decision-making loop and to develop the ground for personalized solutions that will automatically align the machine behavior to the human preferences in any given circumstance.

For this purpose, in contrast to all current works in the field, the output of the human state monitoring process is not directed to the human operator in the form of alert, or to another human that is taking decisions. It is redirected to the system itself, in our case the autonomous vehicle, which will adjust its behavior using human state as an additional input, along with the vehicle and the environment status (road condition, obstacles etc.) as depicted in Figure 12. In these terms, we propose an enhanced perception system that is also affected by the physiometrics generated from the sensors that identify the driver state. The enhanced perception system in turns feeds the UGV control system with additional metrics like the stress level, the comfort or the attention of the driver/operator that are not currently taken into account by the UGV, providing the space for more personalized personalized behavioural control of the autonomous vehicle.

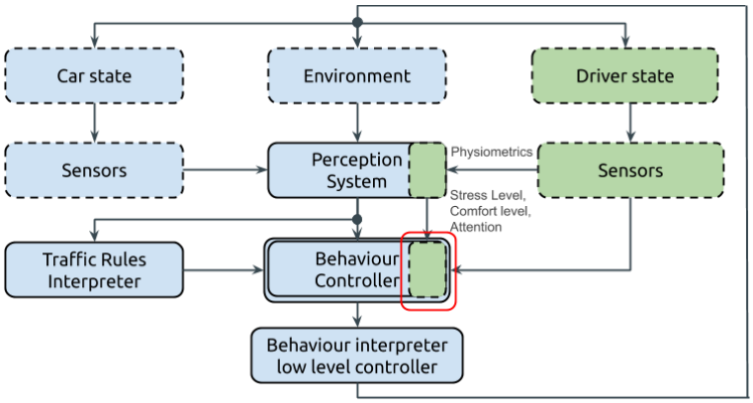

Figure 12: The proposed unmanned ground vehicle (UGV) control system that enhances extra input information (e.g., the driver's state, road conditions and weather context) into the vehicle control loop.

2.5.2. HSM tasks

With respect to the different aspects of the human state monitoring tasks, we focus on the case of autonomous cars emphasizing on the human-centric tasks, that could also be applied in the case of avionics under circumstances. In this direction, in the subsections that follow we divided the aspects being monitored into three status detection classes: **stress/discomfort**, **distraction/attention** and **fatigue**.

- **Detecting user stress/discomfort**

In a more general context, several research works employ physiological signals for identifying drivers'/operators' ***emotional state*** under varying conditions. In [94] authors combine several physiological signals for detecting drivers' mental workload in driving simulators, whereas in [95] the Heart Rate variability is the main factor for evaluating the mental load of professional drivers/operators. Another interesting problem that involves the state monitoring of users in driving scenarios and that has attracted the interest of the research community is the ***emotion recognition*** of vehicle drivers and passengers [96] [97].

Another group of related works focuses on the perceived ***comfort*** levels and the metrics that can be used for detecting them [98]. These comfort levels have been associated with human driving styles which are broadly categorised to comfortable (maximum perceived comfort), everyday driving, and dynamic (minimum comfort). The metrics are used for classifying every driving maneuver (deceleration, acceleration, lane change, follow at varying speed) to one of

the above styles. The input to these metrics mostly comes from vehicle-boarded sensors and is not directly refers to human state monitoring. The user personality is recorded with questionnaires and an association is performed between personality groups and driving styles [99]. However, such works are strongly related to the study of cyber-physical systems examined from the cyber perspective, which is complementary to the human one. Moving-base simulations are used for studying driving comfort in autonomous vehicles [100]. In such simulated scenarios, no input was asked from the driver, and comfort was mainly associated with vehicle-related factors, such as the gradient and the initial value of time to the minimum distance. However, the steering wheel and pedals were monitored, in order to record any driver input, which can be considered a sign of low comfort.

The perceived comfort level is subject to the human perception of the vehicle and its environment, as well as to the internal beliefs, preferences and expectations. Several studies on passenger comfort (or stress), fuse data from a large variety of sensors attached to the vehicle (e.g., accelerometers, gyroscopes etc). Then they attempt to detect the driving style of the driver and associate it to the driver/passengers' comfort or stress level.

The common baseline that most of the aforementioned tasks share is the input data that make use of. Mainly there are three main types of input data that may be available:

1. the state of the vehicle itself,
2. the driver's performance and
3. the driver's state

The state of the autonomous vehicle refers to the vehicle's environment during the drive and is based on the vehicle’s integrated equipment that tracks the outward context based on sensors and cameras either integrated to the vehicle or attached by the driver whilst the driver's performance is inferred from vehicle's data like speed, acceleration, deceleration rate, inter-vehicle distance and more. For identifying the state of the driver/operator most commonly physiological measures exploited are Heart Rate (HR), Electrocardiography (ECG), Electroencephalography (EEG), Electromyography (EMG), Electrodermal Activity (EDA), also known as galvanic skin response or skin conductance, Blood Volume Pulse (BVP), Skin Temperature (ST), Respiration (RSP), and Gaze Detection (GD), which are derived using either inwards monitoring cameras or physiological sensors, smart wearable devices, etc.

- **Detecting distraction**

The multi-modal sensing of the human state employs a wide variety of inputs, from visual, speech, and text to physiological signals. The early or late combination of signals to composite features and the temporal evolution of those features (time-series) and the training of the appropriate models is the main process used for the various human state detection tasks. Fusion techniques such as PCA, LSCE, etc, are frequently used for the joint analysis of eye-tracking, EEG, skin conductance, and other data and their association with user states of stress, drowsiness, or distraction in an attempt to bring human in the loop [101] [102].

In the case of assisted driving, it is more than obvious that attention is a key concept, referring to the driver as a direct machine operator or to the passengers. Regarding the former case, detecting driver's distraction [103] and drowsiness [104] is the most popular classification task related to driver state monitoring.

In the same driving context, the automatic identification of the driver's workload level [105] and the detection of stress level that it creates [106] [107] is just another human state monitoring task. The monitoring of the driver's physiological features and the correlation of their changes

with various kinds of stress that occur during the driving is the main challenge of the related works [108] [109].

In the case of autonomous driving vehicles, the tasks of detecting attention or fatigue are of limited applicability, since they are not so critical for driving safety, as in unassisted driving [110]. However, passengers' comfort is strongly correlated to the driver's driving style (i.e. acceleration and braking behaviour) [111] [112], which also applies even in the case of autonomous driving. The most prevalent task, in self-driving vehicles, becomes the evaluation of the driver's and passengers' stress levels in the presence of various stressors and in relation to the vehicle driving conditions. Authors in [106] use multiple bio-signals and an explicit stress signal for the driver in order to train a stress detector. Authors in [113] use a bio-electric physiological signal as input from the driver and Self Organizing Map techniques for distinguishing between low, medium, and high-stress driver states.

In the case of human-machine collaboration, the proper monitoring of user state is important for the successful execution of the collaborative activity (e.g., vehicle drivers, machine operators, construction workers) the main focus is on the detection of vigilance or loss of attention (distraction). When users are exposed to various risky and stressing situations, their level of attention is evaluated using wearable sensors (e.g. (EEG), (GSR) for measuring sweating levels or (ECG) for measuring heart rate) and cameras.

- **Detecting fatigue**

Identifying driver's lack of attention has been widely explored and is strongly related to fatigue but differs in the sense that driver's attention may be reduced by various means (such as mobile phones etc.) that are not related to fatigue.

Fatigue (physical or mental) is generally defined as the set of symptoms that affect human performance or disables a man from completing an activity [113]. Based on this concept, many algorithms and systems have been proposed and adopted in various brand's automobile driving assistants that track drivers state, identify drivers' fatigue, and alerts them accordingly (i.e., to take a short break from driving) [114] [115].

For example, in the case of truck drivers who perform repetitive activities for longer periods, brain and heart signals are used to detect mental fatigue or stress due to an increased mental workload. It is also essential to distinguish between wakefulness, drowsiness, loss of consciousness, and sleep status [104] as will be explained in the following.

2.5.3. HSM task formulation

All the above HSM tasks can be broadly presented in four main axes: emotion, attention, stress, comfort, as depicted in Figure 13.

All the aforementioned approaches have a common task formulation, which takes as input a collection of *k* time-series for user *(i)* $S^{(i)}(t) = \{s_1, s_2, \cdots, s_k\}$, at time *t*, where each time-series is of length *t* ($s_j =< s_j(1), \ldots, s_j(t) >$), and a (possibly empty) set of values that correspond to user-related features $U = \{u_1, u_2, \cdots, u_m\}$ and produces an output value $y^{(i)}(t)$ which corresponds to the perceived state level of the monitored human subject *(i)* at time *t* and is either a continuous or discrete value (i.e. level):

$$y^{(i)}(t) = f\left(S^{(i)}(t), U\right)$$

The learning task is usually formulated as an error minimization task, where the error in the output is defined on the difference between the output value $y^{(i)}(t)$ and the actual user state level $\bar{y}^{(i)}(t)$ at time *t*.

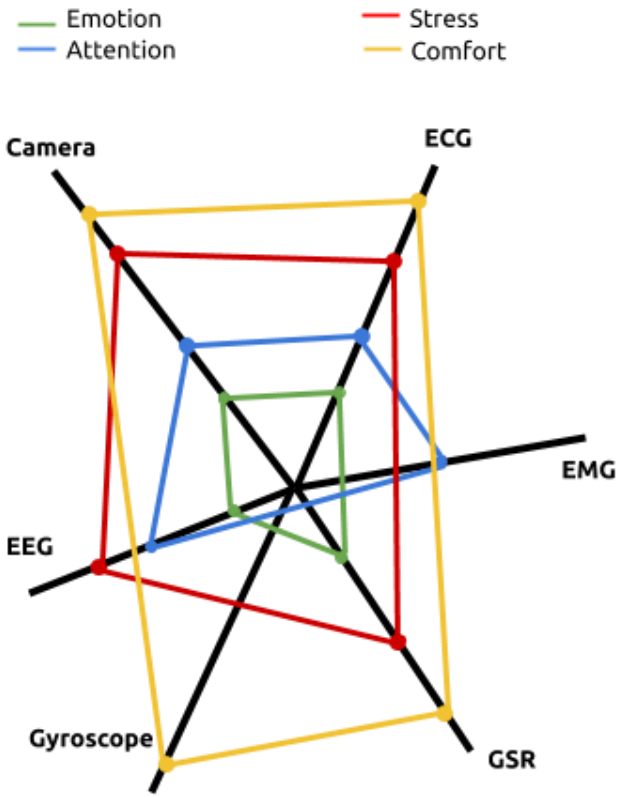

Figure 13: Summary of HSM tasks with respect to the inputs each task broadly uses.

In stress detection for example, $S^{(i)}(t)$ could contain the physiological signals of the subject *(i)*, while $U^{(i)}$ may represent user-specific features such as the average skin conductivity in the relaxed state, which can allow to use the same f to perform stress level recognition for a diversified set of subjects.

The inferred stress level can be represented as a continuous scalar value; in practice, however, it is often more practical to classify the stress level within a given number of macro-categories such as "low", "medium", and "high" [106], or "stressed" and "relaxed".

In discrete state prediction tasks, such as the detection of driver's emotions, the output vector $y \in \mathbb{R}^n$, where n is the number of core emotions (happiness, sadness, surprise etc.)

$$0 \leq state_x \leq 1 \ \ \forall\, x \ \in [1, n]$$

is processed to indicate different levels of intensity for each x of the n emotions, instead of continuous values between 0 and 1. This is extended in the task of emotion valence and arousal detection, where the dimensional emotional model associates the emotions of fear and sadness to low levels of arousal and negative valence, and happiness with high arousal and positive valence. In this case, the emotional state is a 2-dimensional vector $state \in \mathbb{R}^2$, where:

$$-1 \leq state_x \leq 1 \ \ \forall\, x \in [1, m].$$

2.6. Learning user's behaviour towards autonomous driving personalization

2.6.1. Human like driving background

The vision of a human-like driving of autonomous vehicles is not new and its main objective is to design a process of anthropomorphic adaptation to new situations [116]. The emotion model is based on sensor processing and interpretation modules that define human emotions in an abstract level (such as those defined in the Human State Monitoring subsection). In the early work of [116] the personalisation relies on the emotion model that intervenes between traffic situation and the driving controller parameters. In order to achieve personalisation, the emotion states are correlated with controller parameters, thus linking situations with individual human-like dynamic control behaviour.

One personalisation solution is to define individual controller sets and correlate them with emotion states. This allows the selection of autonomous driving modes (e.g., "time to target", "eco-friendly", "chauffeur", "sport", "race car" mode etc.) [117], by solving a cost optimisation problem for each mode. In this case the optimisation problem is to minimise the cost function that is approximated by a linear combination of several basic costs (e.g., steering angle cost, velocity change cost, etc.)

$$f = \sum_i w_i f_i$$

utilizing the weights $w_i \in [0, 1]$ which have to sum to 1 to avoid ambiguity. The optimisation is also subject to a set of driving (steering, speed or acceleration) constraints. The cost function minimizes over a certain time interval taking the dynamics of the car into account.

According to [116], when human is in the loop, the above optimization problem is a bilevel one, which assumes that humans minimise the cost function at the lower level (the linear combination of basic costs) and then at the upper-level problem, they aim in finding the weights w_i that minimize the squared distance between the observed and the predicted human actions.

Despite the few works that early stated the need for it, personalization in the automobile sector is still a relatively recent trend due to the fact that the underlying technology only recently reached a sufficient level of maturity and availability to support personalization and mainly focuses on personalizing the vehicle's advanced driver assistance system (ADAS) [118]. While the aim of ADAS systems is to improve driving comfort and safety, the driver's acceptance of ADAS interventions strongly depends on their skill, needs, and preferences. The goal in the personalization of such systems is to make their interventions more efficient and to improve the driving experience by adapting the systems to the individual preferences of the driver and in the field of autonomous driving personalization refers to be comfortable for different drivers, hence, the driving style of an autonomous vehicle should be adapted to the individual driver's preferences.

2.6.2. Personalising the autonomous car behaviour

The most important component in the task of personalizing a vehicle's ADAS is the driver model used to predict the user's behaviour to transform it in the relative ADAS actions. But most models are based on a normal average conscious user and their control parameters are predefined and cannot be altered based on the different vehicle drivers. In the context of autonomous driving personalization, the main task is to create the appropriate controller that fits the user based on the observations of human behaviour. This task mainly refers to creating

a model that is able to "learn from humans" and try to learn the way a real human would adjust the car's controls for various conditions.

Measuring driving performance and experience (discomfort, perceived safety, driving enjoyment) and correlating this with the effects of different driving styles is a key concept in various research works. The base of this task is the data collection through the use of sensor modules like the work of [119] where ANOVA was used to analyse these effects using a set of handset control and sensors, while the users' trust and acceptance was measured using a questionnaire approach for offline evaluation. In addition, a NN approach with multi-layer perceptrons (MLPs) can be used in order to recognise characteristics in physiological data with relation to discomfort affected by the current driving situation.

A task that is related with dynamic personalisation and adaption, is the **action recognition** of passengers and driver through computer vision. This is mostly based on human pose classification and a very limited set of driven behaviours. For example, [120], links activities to interior regions (instrument cluster region, gear region and steering wheel region). [121] detects three states: operating the shift gear, eating/smoking (combined together) and talking on the phone. The methods used for this task are either based on manually designed features or are end-to-end methods based on convolutional neural networks (CNNs). Drive&Act is the first large-scale dataset for driver activity recognition which covers both, autonomous and manual driving scenarios [122].

Under this prism, personalizing the behaviour of autonomous vehicles is initially bounded with the process of **learning** the user's actual behaviour, **adjusting** the system's response to this behaviour and **translating** this into commands that interact with the car's control units. Current state-of-the-art systems for autonomous driving are either implemented as a set of individual components that are trained to deal with a task each separately. They use a CNN to perform the driving pattern recognition and then adjust the system's response according to the perceived situation. They finally translate it into commands to the car control units. The main controller units for this purpose are either proportion integration differentiation (PIDs), which are low-level controllers, or model predictive control (MPC) components [123] [124], which is an advanced method of process control that is used to control a process while satisfying a set of constraints. The main advantage of MPC is the fact that it allows the current timeslot to be optimized, while keeping future timeslots in account. This is achieved by optimizing a finite time-horizon, but only implementing the current timeslot and then optimizing again, repeatedly. In addition, MPC has the ability to anticipate future events and can take control actions accordingly whilst PID controllers do not have this predictive ability.

On the other hand, deep learning has been used in systems that exploit end-to-end techniques to train one single model that imitates the behaviour of a human demonstrator [125]. The DAVE-2 System [125] combines input from cameras and the steering wheel in a CNN that predicts the correct steering angle that keeps the car on the road. However, such end-to-end solutions still do not introduce any personalisation aspects. In this direction, Reinforcement Learning is one of these state-of-the-art approaches that can be employed for continuous learning and adaptation of driving to the user state, thus providing an end-to-end solution for personalisation of autonomous driving. RL aims at training a model in order to maximize a reward function that is evaluated based on the feedback the model gets after each decision is performed in the agent's environment. Reinforcement learning, imitation learning and inverse reinforcement learning are common approaches for tailoring system behaviour and performance to better fit the user's profile.

Current state-of-the-art on this field explores different types of approaches for solving smaller problems (i.e., motion planner) that can be combined to compose a unified autonomous driving

controller. In such cases for example, reinforcement learning is used, in order to learn from human drivers based on the on-board sensing information and realize human-like longitudinal speed control through the learning from demonstration paradigm. Based on this knowledge, the target speed of the drivers is learned by the personalized learning system which translates this into low-level control commands by a proportion integration differentiation (PID) controller. Another variant of this approach is inverse reinforcement learning, which assumes that a human demonstrator follows an optimal policy with respect to an unknown reward function and once the reward function is recovered, reinforcement learning is also used to find the appropriate policy that imitates the expert.

For example, in the autonomous driving scenario, the goal is to maximizing the driving agent reward based on the car's state measures (i.e., speed, position on the road lane, distance from sidewalk, distance from other cars etc.). One of the latest achievements on this direction was proposed by [126] who designed a deep reinforcement learning algorithm and used the deep deterministic policy gradients algorithm for training an asynchronous advantage actor-critic (A3C) [127] RL agent and define the reward as the distance travelled by the vehicle without the interference of the human driver.

In these terms, RL algorithms are used for solving a Markov Decision Problem which consists of (**a**) a set of states S, (**b**) a set of actions$A$, (**c**) a transaction probability function $p: S \times A \rightarrow P(S)$, which assigns a probability distribution $p(\cdot|s,a)$ to every pair $(s,a) \in S \times A$, representing the probability of entering a state from state s using action a, (**d**) a reward function $R: S \times S \times A \rightarrow R$, that describes the reward $R(s_{t+1}, s_t, a_t)$, associated with entering state s_{t+1} from state s_t using action a_t and (**e**) a future discount factor representing how much we care about future rewards. Based on this approach, a set of sensors like RGBA cameras, LIDAR etc. is used in order to observe the set of states given to the algorithm as an input each observation step and the set of actions is composed of a 2-d space of pairs of steering angle and a speed setpoint in km/h. As pointed by the state-of-the-art, the reward function is defined as forward speed and an episode is terminated upon an infraction of traffic rules – thus the value of a given state $V(s_t)$ corresponds to the average distance travelled before an infraction.

In order to personalize the agent's decisions and make them adaptive to various conditions, including the human driver's state, it is necessary to develop a mechanism that dynamically adjusts the vehicle behaviour to the actual conditions. For this purpose, we decide to separate the vehicle controlling mechanism, which traditionally is a PID controller or an MPC controller as depicted in Figure 14, from the personalisation mechanism. As depicted in the figure, the MPC controller is controlling the vehicle behaviour based on the current and desired driving properties and a set of configuration parameters that affect how the vehicle reaches the desired properties. The result of this configured controlling is a less or more steep adjustment of the vehicle acceleration or steering, which can add to the physiological signals of the driver. For example, a sudden braking, continuous changes in the steering angle, or side accelerations may add to the driver stress, so the personalisation agent has to quickly adjust the MPC parameters in order to reduce stress. As shown in the Figure, a multi-series RNN model, considers the various input signals (time-series) from driver and car sensors and decides on the MPC parameters, which in turn affect the vehicle's behaviour and consequently the car and driver state. The point of interest in this approach is the quantification of the error in the prediction of the best MPC parameters at every moment, based on the resulting behaviour of the vehicle. In order to handle this issue, research and experiments are currently performed in a simulation environment, using an RL model with a reward function that is subject to the effect of MPC decisions on the driver state.

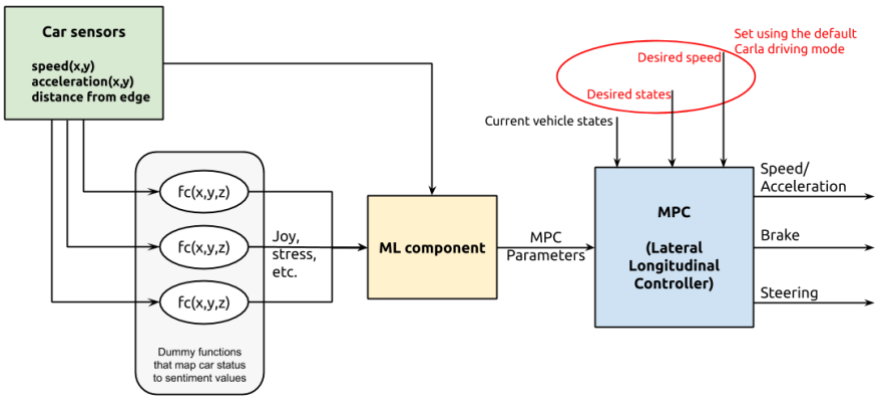

Figure 14: Pipeline for controlling MPC functionality based on the driving profile personalization

Another approach that can be exploited for adaptive personalisation in the autonomous driving task is Lifelong and Continual Learning (Section 2.3). A technique that has recently been used in chatbots to learn them to consequently adapt to new tasks [128] can be beneficial for personalised autonomous driving, where the learned driving preferences for a certain driver for a set of tasks are continuously challenged with new tasks, new driving conditions and situations. In the same time old knowledge must not be forgotten, thus the catastrophic forgetting aspect has also to be considered [129]. An indicative way of developing and validating successful human-machine symbiotic systems that promote user trust and protect user safety has lately been proposed by [130], who try to identify the key components of a validation, verification and certification process for Highly Automated Vehicles (HAVs). Also, in [131] [132] authors presented an approach for the verification and validation of robot assistants in the context of human–robot interactions, in terms of safety and trustworthiness, allowing different verification and validation techniques to corroborate one another.

2.7. Software Frameworks for ML

ML and DL are increasingly being used in industrial applications and, undoubtedly, one reason for their success is the availability of libraries and open-source tools that ease the design, development, and production of the learning models. A discussion on this kind of software frameworks from the perspective of the TEACHING computing platform is reported in D2.1 (Section 5.3). Among other alternatives (e.g., MXNet[4] and Caffe[5]), the two most popular choices are PyTorch[6] and TensorFlow[7]. As discussed in Section 5.3.2 of D1.1 (to which the reader is referred for further discussion), the TEACHING choice fell on TensorFlow, primarily for the availability of TensorFlow Lite[8], a set of tools that enable ML functionalities at the edge. TensorFlow Lite mainly consists of two components: an interpreter that can run optimized code for learning models on a huge variety of different hardware supports, including mobile devices and microcontrollers, and a converter, which converts TensorFlow models into a suitable form to be used by the interpreter. In addition, the choice of TensorFlow/Lite also naturally enables

[4] https://mxnet.apache.org

[5] https://caffe.berkeleyvision.org

[6] https://pytorch.org

[7] https://www.tensorflow.org

[8] https://www.tensorflow.org/lite

the use of high-level Keras[9] APIs, thereby including the sub-classing, functional and sequential APIs, which - for example - ease the process of composing the AIaaS learning-related modules.

TensorFlow and TensorFlow Lite are ideal frameworks for the implementation of the RNN approaches and techniques described in Section 2.1. In particular, TensorFlow's good support for sparse matrix operations is a useful feature for the implementation of efficient RC models. Moreover, the availability of the officially supported TensorFlow Federated framework makes it easy to implement ML models in the context of federated learning. Regarding deployment, the officially supported TensorFlow Lite is suited for the inference of ML models onboard low-power edge devices. In fact, TensorFlow Lite supports the conversion of ML models to more memory-efficient formats and also eases optimization techniques such as the quantization of the network weights to reduce the size of the model and to improve its inference speed. It is then possible to envisage configurations in which the RNN running on the edge devices are endowed with RC-based adaptation/training capabilities, which (as explained in Section 2.1) do not require the extra expense of gradients propagation.

Moreover, the computation of dependability measures (reported in Section 2.4) is amenable to implementations in both TensorFlow and TensorFlow Lite.

In relation to the human-feedback personalization techniques described in Section 2.6, this scenario illustrates a real-world application where new training data become available after the NN has already been trained. In these types of applications, CL networks and RL techniques have to be employed and TensorFlow (for training purposes) and TensorFlow Lite (for on-board prediction integration) are widely used platforms for building and deploying stable and high throughput ML-powered applications like these kinds of tasks. In this direction, TensorFlow and TensorFlow Lite will also be used for implementation along the different LMs described in Section 4.2.4 for the tasks of autonomous vehicle behaviour personalization.

2.8. Enabling TEACHING Use Cases

Both use cases considered for the project involve data in the form of time-series. In the automotive use case (D5.1, Section 3), the data will mainly come from biomedical sensors and vehicle sensors for supervised learning tasks (D5.1, Section 3.1). In the avionics use case (D5.1, Section 2), the AI will receive streams of data regarding the monitoring of hardware (and possibly software components) to perform anomaly detection and produce recommendations (D5.1, Section 2.7). Details on the TEACHING selected pool of sensors can be found in Section 5.2 of D1.1. The ML models discussed in the state-of-the-art Section 2.1 are thus ideal for processing these streams of time-series, in both cases.

Moreover, the use cases are perfect examples of applications where federation techniques can provide large advantages in terms of predictive performance by exploiting the knowledge available in the peers (i.e., other vehicles or aircrafts). The federated learning techniques from Section 2.2 will allow the AI systems to share knowledge while preserving privacy constraints.

Also, in both use cases, it can happen that the distribution of the data may drift over time. For example, the biological signals of a subject in a vehicle may exhibit periodic drifts (such as the skin conductivity in different seasons of the year) or permanent drifts (such as the stress level of a subject getting accustomed to a new road). This is why it is important to consider Continual Learning techniques from Section 2.3 to make sure to correctly adapt to these drifts.

[9] https://keras.io

In the automotive use case, the dependability concept shall be applied to reach an adequate safety level. Lack of safety could lead to catastrophic consequences for the human health. Data coming from external vehicle sensors shall be correctly interpreted from the appropriate LM to avoid errors that could potentially lead to a physical harm of the involved people. In this context the role of the metrics that calculate dependability becomes crucial because it permits to find LMs that do not reach a sufficient degree of dependability and so might not be considered safe.

3. Requirement Analysis

This section presents an analysis of the TEACHING AIaaS requirements, specifically detailing those originating as part of the technological view in WP4, or with an impact on it. Note that the TEACHING consortium developed a centralized repository that collects all the requirements of the project, reported in the Project requirements document release 1.0 in the form of a jointly edited, living document. To identify each requirement, we refer to the IDs assigned in the project's requirements document. For each requirement, together with a brief textual description, we report its priority, the impact on WPs, the reference use case domain (when relevant), and possible links to other project's requirements.

We start in Section 3.1 illustrating the architectural requirements. We then continue with a description of functional and non-functional requirements, respectively in Section 3.2 and Section 3.3. Safety and security requirements are described in Section 3.4. Finally, in Sections 3.5 and 3.6 we conclude with performance requirements and structural requirements.

3.1. Architectural Requirements

3.1.1. Data transfer for AIaaS federation

The edge system must be able to send and receive sets of ML model parameters to/from the cloud over the network. This communication channel can be subject to bandwidth and connectivity limitations.

ID: 72

Priority: high

Impact on WPs: WP2, WP4

3.1.2. Communication of the AIaaS modules with the vehicle

The AIaaS modules on the edge must be able to control the parameters exposed by vehicle. For example, they should be able to switch the configuration of the vehicle among different driving styles.

ID: 75

Priority: high

Impact on WPs: WP2, WP3, WP4

Domain: automotive

3.1.3. Communication of the vehicle with the AIaaS modules

The AIaaS modules on the edge must be able to receive as input the data from the sensors and the state of the vehicle.

ID: 76

Priority: high

Impact on WPs: WP2, WP3, WP4

Domain: automotive

Link to other requirements: 73, 74, 85

3.1.4. Communication of AIaaS suggested actions

The AIaaS system must be able to display suggested actions to the human through a suitable user interface.

ID: 77

Priority: medium

Domain: avionics

Impact on WPs: WP4, WP5

3.1.5. Intra-edge AIaaS communication

Within the same edge device, different modules of the AIaaS system need to exchange data and predictions. For example, the learning modules could be configured so that the output of model A is later used as input for model B.

ID: 78

Priority: high

Impact on WPs: WP2, WP4

3.1.6. Inter-edge AIaaS communication

Across different edge devices, different modules of the AIaaS system need to exchange data and predictions in a similar way to the intra-edge AIaaS communication. For example, the learning modules could be configured so that the output of model A on a given device is used as input for model B on a different device.

ID: 79

Priority: medium

Impact on WPs: WP2, WP4

3.1.7. Access to vehicle sensors' data

All the sensors on-board the vehicle used to acquire the biophysical parameter of the user, the automotive data and other environmental information must be readable through a standard M2M protocol from the communication entities of the infrastructure.

ID: 85

Priority: high

Domain: automotive

Impact on WPs: WP2, WP4, WP5

3.1.8. Data brokering within the AIaaS platform

A mechanism of data brokering should be designed to mediate the access of the functional modules of the AIaaS to the data generated by the wearable, automotive and environmental sensors, or avionics traces.

ID: 98

Priority: high

Impact on WPs: WP2, WP4

3.2. Functional Requirements

3.2.1. Detect changes in user state

Use heart rate (electrocardiogram), nerve and muscles signals (electromyogram), respiration, galvanic skin response, gaze direction and driver's and/or passengers' emotional state, which are derived using either inwards monitoring cameras or physiological sensors, smart wearable devices etc. to recognize changes in stress, emotional and drowsiness levels.

ID: 80

Priority: medium

Domain: automotive

Impact on WPs: WP4, WP5

3.2.2. Detect changes in car state or context

Use sensors and cameras either integrated to the car (accelerometers, gyroscopes, luminosity sensors, LIDAR, camera, IR beam and sonar) or attached by the driver to detect car's state and context (e.g., road and weather conditions, obstacles) to feed them in the human-centric personalized autonomous driving system.

ID: 81

Priority: medium

Domain: automotive

Impact on WPs: WP4, WP5

3.2.3. Perform ADAS adaptation for model fine tuning

Depending on the overall driving style of the passenger, his state, the car's context or the currently selected driving style of the passenger for the specific scenario, we change the model parameters in order to get a slightly different distribution at the variation of the model's parameters.

ID: 82

Priority: medium

Domain: automotive

Impact on WPs: WP4, WP5

3.3. Non-functional requirements

3.3.1. Annotated data for AIaaS (human state recognition)

It is needed to collect a dataset of physiological signals from human subjects in a vehicle or simulator in order to construct ML models capable of distinguishing between different physiological states such as relaxation, drowsiness, and stress in real-world conditions.

ID: 73

Priority: high

Domain: automotive

Impact on WPs: WP4, WP5

3.3.2. Annotated data for AIaaS (driving preference)

It is needed to collect a dataset of the different driving preferences associated to the different physiological states such as relaxation, drowsiness, and stress. The data is required in order to construct ML models capable of switching or suggesting different driving styles according to the physiological state of the human driver.

ID: 74

Priority: high

Domain: automotive

Impact on WPs: WP4, WP5

3.3.3. Compatibility between SW and HW components

The software implementation of AIaaS at the edge must run on the available hardware.

ID: 92

Priority: high

Impact on WPs: WP2, WP4,

3.3.4. Defining learning functionalities of the AIaaS

A list of functionalities and corresponding API should be defined for the modules of the AIaaS platforms such as RNN, Classifier, etc.

ID: 94

Priority: Critical

Impact on WPs: WP2, WP4

3.3.5. Defining the possible pattern for access the Edge storage

A pattern defines the characteristics of the data flow between the source and destination, such as: streaming/storing each value as it is available, sending batches of an assigned size, reading/writing data via queues. The patterns depend on AIaaS system implementation as well as from the application model we will adopt for the design of AIaaS.

ID: 95

Priority: Medium

Impact on WPs: WP2, WP4

3.3.6. Common interface for functional modules of AIaaS

Functional modules of the AIaaS should expose a common interface that allows orchestration operations such as initialization, setup, connect, etc..

ID: 96

Priority: High

Impact on WPs: WP2, WP4,

3.3.7. AIaaS application definition

An AIaaS application should be specified as a workflow graph of functional modules, including: its parameters, external communication channels, etc.

ID: 97

Priority: High

Impact on WPs: WP2, WP4

3.3.8. Common data format for the data brokering

The data brokering should have a defined format. Functional components of the AIaaS must either use the same format or implement an adapter.

ID: 99

Priority: High

Impact on WPs: WP2, WP4

3.3.9. Common meta-data format for the data brokering

The data brokering should have a defined format for the meta-data associated to each type of data sensor.

ID: 100

Priority: High

Impact on WPs: WP2, WP4

3.3.10. Annotated data for AIaaS (avionics traces)

Need to collect software/hardware traces from the aircraft sensors to train the AI modules

ID: 106

Priority: high

Domain: avionics

Impact on WPs: WP2, WP4, WP5

3.4. Safety and security requirements

3.4.1. Ensure safe mode transitions and awareness

Ensure that mode transitions are performed correctly and controlled by the vehicle operator affected if necessary. The vehicle operator affected has also to be aware of the current mode and their responsibility deriving from it. For example, actuating an automated mode is permitted only when inside the Operational Design Domain (ODD), and it will be deactivated prior to leaving the ODD or as a result of the vehicle operator taking control again.

ID: 32

Priority: high

Impact on WPs: WP1, WP2, WP3, WP4, WP5

3.4.2. React to insufficient nominal performance and other failures via degradation

Due to possibly unavailable nominal performance capabilities and other failures (e.g., based on hardware faults), CPS has to degrade within a well-defined amount of time.

ID: 33

Priority: high

Impact on WPs: WP1, WP2, WP3, WP4

3.4.3. Reduce system performance in the presence of failure for the fail-degraded mode

The reaction in case of failures during fail-degraded mode has to be defined.

ID: 34

Priority: high

Impact on WPs: WP1, WP2, WP3, WP4

3.4.4. Perform ODD functional adaption within reduced system constraints

CPS operation with ODD functional adaption is actuated as nominal capabilities with new limits. Multiple functional adaptions are possible. The limitations have to be defined such that the functional adaption can be stated as safe. Therefore, it may be necessary to avoid a permanent operation. A well-defined timeframe for an additional reaction is required.

ID: 35

Priority: medium

Impact on WPs: WP1, WP2, WP3, WP4, WP5

3.4.5. Attributes for dependability of Neural Networks

Many of AI systems are based on Neural Networks (NN). Attributes to evaluate the dependability of NN are: Robustness, Interpretability, Completeness and Correctness (RICC).

ID: 83

Priority: high

Impact on WPs: WP4

3.4.6. Metrics for attribute dependability

Definition of metrics to evaluate quantitatively the attributes RICC of dependability. Each metric reflects one or more attributes and their combination allows to have an evaluation of each attribute (Robustness, Interpretability, Completness and Correctness attributes).

ID: 84

Priority: high

Impact on WPs: WP4

3.4.7. Secure access from application to the adaptive system of the vehicle

The autonomous driving system should expose an interface that allows the application of the AIaaS to tune the parameter of the adaptive system of the vehicle in a secure way.

ID: 102

Priority: Critical

Domain: automotive

Impact on WPs: WP1, WP2, WP3, WP4, WP5

3.5. Performance requirements

3.5.1. Figures on data production rate and QoS requirements

Defining figures to quantify the amount of data information generated per unit of time by all the sensors, in the controlled environment. This is a fundamental requirement to properly design and to tune the communication and computing platform of the CPSoS.

ID: 88

Priority: high

Impact on WPs: WP2, WP4, WP5

3.5.2. Figures on data production rate and QoS requirements

Defining figures to quantify computing performances expected by AIaaS based applications, it is of paramount importance to properly design the computing platform and to tune the orchestration process.

ID: 89

Priority: high

Impact on WPs: WP2, WP4, WP5

3.6. Structural requirements

3.6.1. Hardware requirement of the functional modules of the AIaaS

AIaaS's functional modules should have associated their specific hardware requirements such as RAM, computational load, need of local storage, to assure their compatibility with the underlying hardware platform.

ID: 101

Priority: high

Impact on WPs: WP2, WP4

3.6.2. Non-volatile storage unit for the AIaaS platform

A non-volatile storage unit has to implemented to allow access to the AIaaS platform, for the storing of data and meta-data.

ID: 103

Priority: high

Impact on WPs: WP1, WP2, WP4, WP5

3.6.3. AIaaS subsytem to manage internal module violations

A non-volatile storage unit has to implemented to allow access to the AIaaS platform, for the storing of data and meta-data. The AIaaS platform requires a subsystem to collect and react to the violations of the reliability metrics raised by the functional modules of the platform

ID: 104

Priority: Medium

Impact on WPs: WP2, WP3, WP4

4. Design

This section describes the way the TEACHING platform supports AIaaS applications. In the first part (Section 4.1) we first introduce the overall approach to AI applications, which sees them as the result of composing commonly used, well specified basic AI building blocks. The same section then goes on to present the **software architecture** of the platform, describing the different modules and components it is comprised of.

The second part of this section (Section 4.2) focuses on the TEACHING **application model**, first by explaining more thoroughly the overall approach (Section 4.2.1), then by providing the definition and features of the **building blocks** that compose a TEACHING application (Section 4.2.2).

Then, Section 4.2.3 discusses the relationship between the AIaaS support, its hosted application and other parts of the TEACHING platform, focusing on communication mechanisms, local storage and data transfer.

Finally, subsection 4.2.4 lists the specific classes of **Learning Modules** that will contribute to the support of TEACHING use cases, also linking them to the research topics and learning techniques previously covered in the state-of-the-art (Section 2).

4.1. General Architecture

This section describes the overall architecture that supports AI application in the context of the AIaaS platform. The architecture is designed to be general enough to provide support for both the automotive and avionic use cases. The core idea is that such architecture should allow the application developer to compose a fully working application by means of the following set of abstract functionalities:

- Ready-made learning functional units, called *Learning Modules* (LM), that provide predefined learning functions. The LMs, akin to AI-enabling Lego blocks for the edge, are developed on top of an AI/learning framework suited to run on the edge devices (e.g., TensorFlow lite[10]);
- Ability of chaining and composition of LMs and automatic local routing of necessary data for training and prediction purposes;
- Access to a set of existing sensors that provide data stream as source for the LMs;
- Networking devices that allow external communication, including accessing Cloud based and other edge-based devices, e.g., RSUs for the automotive use case, or satellite for the avionic use case.

Based on this first description, in the following Section 4.1.1 we present the overall software architecture of the AIaaS platform, define the roles of its composing modules and their relationship with each other.

[10] https://www.tensorflow.org/lite

4.1.1. Architecture Components Description

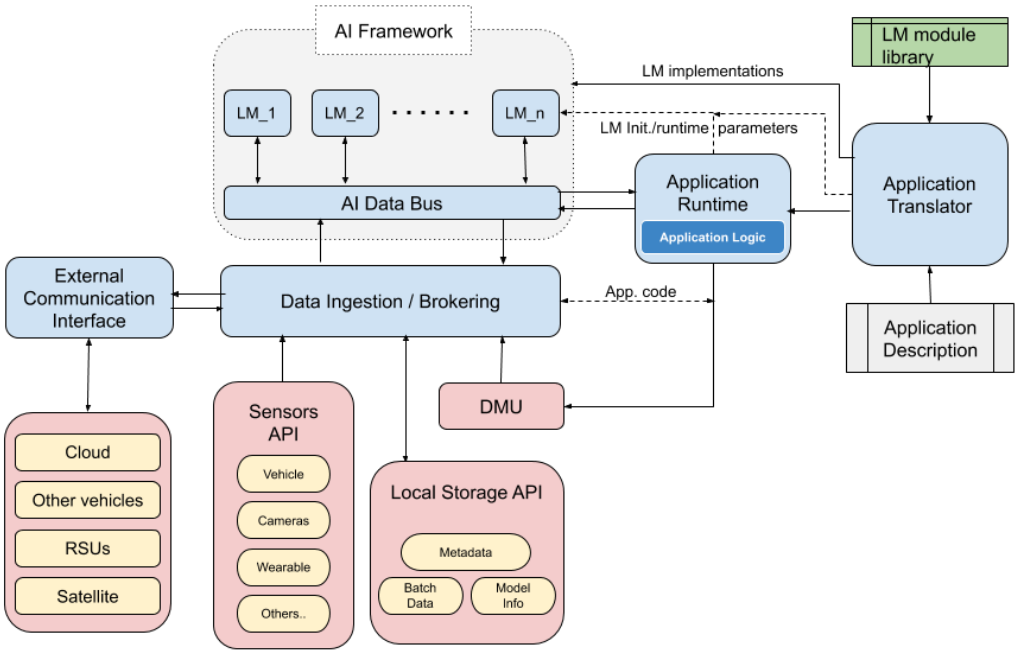

Figure 15 Software Architecture of the AIaaS Platform on the Edge

The functional components of the AIaaS system and their relationships are depicted in Figure 15. The core components of the systems are the ones devoted to the execution of AI applications, and the overall architecture is designed taking into account the requirements listed in Section 3. The application is composed by a set of LMs that operate together toward a specific end; for example, in the context of the TEACHING use cases they generate a feedback for the controller of a vehicle or for the pilot of an aircraft.

The application deployment inside the AIaaS platform, according to a model that is more fully described in Section 4.2, revolves around four elements: the Application description, the Application Translator, the Application Runtime and the Application Logic. The **Application Description** is a document containing the definition of the application in a well-defined format, and it is described more in detail in Section 4.2.2.

The **Application Translator** software component plays two roles: first, it generates and configures the ready-to-be-executed instances of the LMs that are part of the application, and, second, it provides the application runtime component with the Application Logic elements as derived from the Application Description.

The **Application Runtime** component manages the main execution workflow of an application. Its core functions include: the instantiation of the underlying **AI framework** that hosts the learning modules; instantiation and execution of the LMs of the application; configuration of the AI data bus to correctly route the data stream to the relevant LMs. The application runtime also manages the Application Logic, triggering and providing data to its function.

Finally, the **Application Logic** is a per-application software module. It embodies those parts of the application which are in charge of handling all special cases and actions that depart from the main workflow of the Application Runtime (deployment, data collection, and data analysis). For instance, custom snippets of code from the Application Description to be triggered by some

specific LMs behaviour, checking conditions and enacting specific application reactions, are all collected in this software module.

A **LM** is a function module that implements a single machine-learning (ML) task. The LM scope might include both the training of an ML model and the execution of such a model against incoming data, plus other functions (e.g., data processing and mining) that will be useful or necessary for the development of the reference applications. LMs are implemented via the AI framework, but they may be defined to address more complex tasks than those already provided by the AI framework. The structure of LMs and their interplay with the application model are described in Section 4.2. The LM functions considered within the project are reported in Section 4.2.4.

The collection of the LM implementations that are available to the Application Translator for deploying applications is stored within the local **LM library**. This component acts as a local repository providing for each LM at least one implementation suitable to execute within the AI framework, together with metadata needed to guide and perform the LM deployment.

Data that feeds the LMs is produced by either sensors (typically time-series of observations for certain physical measures) or outputs produced by other software components, potentially paired with supervision or controls data.

The sensors are accessed through a common, dedicated **Sensors API**, but not directly, in order to provide a place for adapter components that insulate the application support from the specifics of different kind of sensors. If foreseen, intra vehicle communication (e.g., interfacing to mobile on-board devices to exploit their sensing capabilities) may also be placed within the Sensors API component.

The data is then collected by the **Data Brokering** component, which takes the role of main distributor of data inside the AIaaS platform. As a routing and interfacing component, the Data Brokering will provide an interface to select the data of interest from the various components, such as publish/subscribe interfaces and filtering capabilities alike those of modern stream processing system (e.g., Apache Kafka[11]). It will also provide the option to implement to some degree custom management of communication / interactions, as it may be required either by the application or by the handling of extravehicular links/protocols (possibly addressing link reliability and latency characteristics, or synchronization constraints).

The **AI Data Bus** is in charge of dispatching the data to the correct LM component. It will generally exploit the DB component as well as the APIs of other components in order to let the data flow. The specification of what data goes in which LM and when is provided by the Application Logic.

The **Local Storage API** provides the interfaces to a non-volatile, local storage that allows applications to store different types of data, including configurations, temporary data, cached data and results. This assures that whenever the AIaaS platform needs to be restarted, all the work can continue from the last checkpoint rather than restart from scratch.

Apart from the sensors, the system communicates with two other kinds of external entities. The feedback from the application to the CPS system goes through the **DMU** (Decision Making Units, e.g., controller interfaces), that interfaces to system in-vehicle actuators or the pilot. As the whole CPS has critical dependability constraints, the DMU component will perform

[11] https://kafka.apache.org/

additional checks and impose constraints on any action that the application may want to perform. The **External communication interface** component is used to mediate the communication with other entities outside the vehicle, and in the case of the automotive use cases other entities such as RSUs, and other vehicles. Practically, this component can be internally divided into multiple components each dealing with the different means of communication necessary.

4.2. Application model

In this section we describe the overall approach to modelling and describing AIaaS applications. We provide an in-depth description of the specific design choices made and under evaluation for the application model, the resulting research and technological issues, as well as the interplay and the technological constraints that affect the AIaaS architecture, WP4 activities and other related project activities and components.

As a first step, we explain and motivate the overall approach to modelling AI applications.

4.2.1. Approach to AIaaS Applications

The application model in TEACHING is a composition of abstract basic AI units (called Learning Modules) which provide predefined useful AI functions to the App developer.

The **Application Description** document that we described in Section 4.1.1 contains all the relevant information about the application. The overall AIaaS support is capable of transforming an application described there, expressed as a composition of "building blocks" plus suitable metadata, into an actual executable artefact plus the necessary workflow of steps and checks required to manage it.

Application management tasks include both the deployment of the executable(s) and any runtime action needed either by the framework (e.g. react to system events like power down, application termination or hardware failures) or by the application itself (e.g. react to application status changes, to reliability metrics changes, perform actions on the CPS system based on application results).

The "building blocks" of applications, the LMs, are AI modules whose semantics and interfaces are agreed upon and defined by TEACHING. Section 4.2.4 of this deliverable summarizes the current list of LMs identified so far. The rationale backing the definition of a set of LM is to develop a high-level description of generic AI apps that is fully operational, the following key points standing:

- Typical AI applications are easily composed with commonly used functional blocks, which are separately verified and debugged, and whose behaviour can still be configured by providing them (hyper)parameters;
- The implementation details of each LM over possibly different instances of the execution framework can be safely hidden to the application developer;
- The LM abstraction allows optimizing the execution exploiting the available hardware;
- The LM abstraction allows most management activities to be performed by the AIaaS framework in a homogeneous way, while conveying specific events to custom application code whenever needed;
- The list of available LMs can be extended and made fully general, but the approach allows a quick bootstrap within the project by focusing on those learning tasks that the TEACHING use cases require.

The application model definition can be seen as an Application Specific Language (ASL) for edge AI, providing enhanced high-level features to easily express a certain class of problems, without sacrificing expressive power, for a specific class of applications.

The reference class of applications comprises complex AI and learning applications for CPS, where related previous work is plenty both from the theoretical standpoint as well as from the technological one. As a consequence, the definition of the application model is not a new ASL designed in a vacuum, and developed from scratch, it is instead a specialization and extension of an existing formalism for AI applications.

The TEACHING approach to defining the application model focuses on easing the addition of metadata about the application structure and its execution features, in order to allow easier automated deployment and management of applications. The specification of the application structure and the snippets of management code are designed to simplify the application source code, removing most of the burden from low-level management code.

4.2.2. Elements of the application description

The Application Description artefact must contain all information needed by the AIaaS platform in order to select resources, deploy the app, manage its execution (implementing reactions to both ordinary events and sporadic ones), perform an orderly app termination and clean-up if needed.

We already discussed the overall approach of exploiting **Learning Modules** (LMs) in composing applications. LMs are ready-made learning functional units that provide predefined learning functions of a small set of initialization parameters and one or more input/output data streams (hosting actual data, weight-sets, or models). The LMs, akin to AI-enabling Lego blocks for the edge, are developed on top of an AI/learning framework suited to the edge devices (e.g. TensorFlow lite).

Any Application Description references and instantiates one or more of such LMs, providing them (hyper) parameters, possibly initial internal state, and specifying the type of data that the LM needs to compute. Additionally, the source of each data stream(s) and the destination of the result stream(s), are specified as discussed later on in this section about the Application Graph.

- It is a requirement that at least some of the modules are abstract enough to be configured in two ways: to apply a preconfigured knowledge model, or to learn/refine the same kind of model. The instantiation of the LM within the application has to specify (at the latest at deployment time) the way that module is used.
- Specific LMs may have more than one implementation, differing in the execution constraints and bound in terms of HW and SW resources (e.g. specific devices like GPUs, available SW libraries or features of the supporting framework, amount of memory, performance or of power efficiency). The Application may specify constraints and features to allow the Application Translator to pick up the most appropriate LM implementation from the Learning Module Library

The **Application Graph** is the graph collecting the LMs that compose an application. Irrespectively of the actual syntax used to describe the LM data connections (e.g. 1-to-1 channels, pub/sub interfaces), we see the directed (multi)graph of the application as an abstract data structure whose nodes are LMs and whose edges are data exchanges of any type. The Application Graph contains information about

- Chaining/composition of LMs, the description of the data they exchange, including information needed for automatic local routing. Note that different types of data

exchange semantics may be adopted, with (buffered) stream communication being the obvious one and asynchronous exception/like message exchanges being another important use case.
- Existing sensors as a source of data streams to be collected
- Destinations of data streams like specific DMUs or Application Logic methods
- Local storage as destination, source or cache of data that needs to be communicated
- Information allowing the data ingestion/brokering support to implement the communication tasks that provide access to external networks. As a significant special case, networking devices that allow inter- and intra-vehicle communication can be designed as the source and destination of data (providing the application with a flexible access to Cloud-based and other edge-based devices).

The **Application Logic** is a part of the Application Description that collects code stubs and methods designed by the app developer to performing specific tasks supporting the overall learning process.

While most of the application code is actually produced from the Application Translator exploiting the information from the LMs and the Application Graph, and managed in a safe and standard way by the Application Runtime, for the sake of expressiveness we cannot rule out that applications need to perform specific data or event-triggered tasks that are not or not efficiently implemented by LMs.

As a rule of thumb, in well-defined programming frameworks the amount of low-level code that can introduce unforeseen dependencies should be as little as possible. Fully strict models are however detrimental to programming flexibility, especially during the process of developing the runtime support for a new or improved programming paradigm, when experimenting with different solutions is even more of value. Hence, custom code stubs will be allowed to deal with special use cases, e.g. any custom processing needed by sensor-produced data or by commands to be addressed to DMU actuators. Last but not least, reacting to asynchronous events may be simpler and more efficient to do within ordinary sequential code than with an LM running inside the AI framework. This is true especially if the action to be performed implies changing the state of the AI framework, by removing, replacing, adding LMs to the application, or by tuning their hyperparameters, which are all relevant actions for the use cases in the TEACHING project.

4.2.3. Relationship with other TEACHING subsystems

This section provides more detail about two issues related to the data flow within the AIaaS platform, namely:

- the type of communication required by distinct, interacting parts of the application
- the use of local storage that we foresee in TEACHING applications.

Concerning the first topic, the main interaction between LMs is natively performed within the AI framework. We choose to provide a communication support (the Data Ingestion/Brokering Support) outside the AI framework as we identified via the requirement analysis several potential data flows with different characteristics within the use cases. The DBS is likely to implement a pub/sub model to allow subsets of the same data to be routed to different LMs and platform modules, possibly separately filtered as each one of them requires. The routing and filtering data streams is a common need in these settings, and it is more efficient to implement it as feature of the mediator component. The approach is flexible and also efficient for volatile data with the expected bandwidth. However, in addition to regular communications that are well modelled by queue systems, we also identified the need to implement out-of-band,

extemporal, exception-like information exchanges which will need to reach at least the LMs, the Application Logic and the control API of the AI framework.

Concerning the exploitation of **Local Storage API**, the need is clear from the requirements for a stable data storage beyond the volatile one provided by the data brokering component. A stable storage will deal with both sensor data streams and model data (e.g., LM model weights and parameters, which in federated/continuous learning can also be seen as a stream). The Local Storage API needs to provide several features:

- Access to non-volatile storage, as it needs to survive a reboot
- Support data and associated metadata (e.g., how old is the data in storage, what LM used/generated the data items)
- As the size of stored data will be constrained, a small solid-state device is fit for the two following purposes
    - Perform a backup/dump/reload of LM models, in order to allow a continuous learning process to survive system restart (model data size is quite small)
    - Stack up input data in a local cache in order to perform batch computation (data size is comparable to the system board RAM memory size for this use case)

Information stored or sent to external devices (including the Cloud) can generically be either a learned model, raw sensor data or anything in between. The fact mandates a pause and a second look at the potential privacy and system abuse issues.

As we worked to analyze the constraints on moving data from local storage to non-local, external storage, we realized that the architecture design phase must stress a distinction between **privacy-oriented design** and **security**. Privacy-oriented design is most obviously within the scope of the project. Security beyond the state of the art, on the other hand, that is designing a system which cannot be abused by application developers, is not in the project scope.

From an architectural standpoint we may be tempted to avoid those functions that transfer data buffers to remote storage, or to adopt a sophisticated authorization mechanism for those transfers. However, we must observe that:

- the distinction is not quite trivial to implement, and it makes much more complex the component APIs.
- Project goals require us to be able to send a model (e.g., the output of some LM) to a different system. Any restriction on sending data is immaterial if we can define an LM that sidesteps the constraint by encoding the raw data inside its model.
- Our AIaaS platform works in a very secure and industrially verified environment, where all components checked right away since their design, and the risks of allowing an intrusion are much higher than data theft.

Security and privacy in the presence of *malicious developers* are thus protected by application inspection and by LM code design: the whole process of developing edge AI apps already is, and needs to be involved in preserving those aspects.

4.2.4. Classes of Learning Modules

Here we present the rationale, purpose and main features of specific classes of Learning Modules that either are relevant for the research to be performed, or whose need has been identified by the partners in order to support TEACHING use cases. We identify the main classes of LMs the project will design and implement, as well as their relationship with the research aspects previously described in Section 2 about the state-of-the-art analysis.

For each class of LM we describe here, we foresee one or more concrete implementations to be developed by the project. Besides providing a specified machine learning function, the different implementations of a given LM may vary in the set of features provided (e.g., having some parameters/hyperparameters fixed or tuneable) as well as in their non-functional features (e.g., being optimized for lower energy consumption, for larger memory size, for GPU usage etc.).

One of the principal functionalities provided by LMs is that of performing predictions from **time-series data** (as implied by requirements such as 3.2.1, 3.2.2, 3.3.1, 3.3.2, and 3.3.10). In this context, the most promising approach for implementing such LMs appears to be that of employing RNNs (see Section 2.1). In particular, given the efficiency requirements, special focus will be given to RC implementations, such as the ESNs introduced in Section 2.1.3. In the case of ESNs, the implementations will make use of the standard NN architecture, but also of the advanced reservoir topologies that have been described in Section 2.1.4. These include the ring topology (which makes it possible to share a reservoir by only transmitting one floating-point value instead of a quadratic number of weights), the DeepESN topology (which is particularly suited to capturing multiple time-scale dynamics in the data), and the GatedESN topology (which is useful when the data contains long-term dependencies).

The LMs will also feature **Federated Learning** and **Continual Learning** functionalities as described in Sections 2.2 and 2.3. In this regard, LMs will provide functionalities for federated training strategies (such as the averaging or incremental strategies described in Section 2.2) by exposing appropriate methods to the other LMs. Similarly, LMs related to Continual Learning will expose the strategies described in Section 2.3.1.

In addition to the functionalities that are strictly related to learning, LMs with application supporting functionalities will also be implemented. For instance, a task of automatic hyperparameter tuning and model selection of other LMs may be performed by dedicated, specifically designed LMs as well as dedicated code modules within the Application Logic.

Another set of functionalities that are provided by LMs is the **classification** of the preferred driving mode for the user and a **regression** module of the preferred parameters that define the driving behaviour for the user at a moment, using Continual Learning functionalities (Section 2.6) for the autonomous-driving scenario. With regards to the scenario of avionics, variations of these modules will also be applicable mainly focused on detecting anomalies based on continuously learning from the streams of data coming from the aerial vehicles' sensors and software monitoring the hardware behaviour, providing relative recommendations. In addition, driving mode choice is also a key functionality towards autonomous car behaviour personalization, as described in Section 2.6.2, that the LMs will support. Specifically, LMs will support the choice of the appropriate driving mode and parameterization of the related controller parameters abiding by all the safety rules, while taking also into account the monitored human state along with the input parameters (i.e., car status, road/environment conditions, user feedback etc.).

A further set of functionalities provided by LMs are those to **estimate the dependability** of NNs used in safety critical tasks, such as RNNs for object detection from video streams. In particular, the LMs shall implement functions to calculate each of the dependability **metrics**

introduced in section 2.4.3 to provide an evaluation of the dependability attributes introduced in section 2.4.2. Accordingly, the LMs will implement functionalities related to the attributes of dependability: Robustness, Interpretability, Completeness and Correctness.

The other sub-functionalities to be implemented are at a lower layer of abstraction and they are needed to compute each of the previous functionalities. Such sub-functionalities are those related to the computation of the metrics: M_{scene}, $M_{neu-k-act}$, $M_{neu-pattern}$, M_{adv}, $M_{scen-perf-degr}$, $M_{interpret}$, M_{OccSen}, $M_{confusion}$. As required in section 2.4.2, to estimate the value of each dependability metric, these LMs shall use the parameters of the considered NNs and shall run these models providing them the appropriate inputs required from each metric.

As mentioned above, this metrics are born in a task of object detection/recognition from images and videos. However, we can easily reuse or adapt M_{scene}, $M_{neu-pattern}$, M_{adv}, $M_{scen-perf-degr}$ and $M_{confusion}$ to work with different types of data streams (such as human physiological data streams).

5. Preliminary Results

As a part of the activities of WP4, we have already started some aspects of the investigation, analysis, and development regarding relevant methodological concepts for the work that is expected during Y2. The major outcomes of this preliminary work (spanning all active tasks T4.1, T4.2, and T4.3) are reported in this section.

Specifically, we have started investigating Federated Learning approaches to Reservoir Computing design of Recurrent Neural Networks, reported in Section 5.1. Interestingly, by exploiting the incremental approach for readout training we find that it is possible to accurately train ESN neural networks in a decentralized fashion in tasks related to stress and affect detection, as well as for human activity recognition. Moreover, despite the difficulties involved by the COVID-19 pandemic, we are setting up a pilot lab for human monitoring on UNIPI premises. Such a local experimental setup includes a variety of wearable sensors (e.g., ECG, GSR, EMG), EEG cap, eye trackers, cameras, high-quality and omnidirectional microphones, a large display, and accelerator cards (FPGA and GPU). In this context, in Section 5.2 we illustrate some basic functionalities of a stress estimation demo that is currently under development. The system, called ANSIA (A Neural System that Infers Affects), gathers real-time sensor readings from wearable Shimmersensing devices and is equipped with both stress prediction functionalities (based on Deep Reservoir Computing, see Section 2.1.4) and visualization capabilities. Finally, we have started working on the aspects of modifying the driving functionality of the vehicle controllers based on the various inputs (driver state sensors, car perception sensors, etc.). In Section 5.3 we describe the work that is being performed and illustrate the preliminary results on two fundamental tasks related to driving model personalization based on the estimated human level of stress, and adjustment of non-linear driving control systems.

Further ongoing works regard experimental benchmarking assessment of deep randomized neural models in human monitoring and stress estimation, and Continual Learning for Reservoir Computing models adaptation. Given the still highly preliminary nature of these last works, the results along these lines of research will be illustrated in the next reports on WP4 activities.

5.1. Federated Reservoir Computing

The objective in Federated Learning is to learn a model *m* by using a subset of clients *u* from a set of all clients *p*. In our case, the model *m* is an ESN. Each client has a local dataset *D*, which is a time-series data, and an ESN model *m*. The learning process of each ESN is not iterative like in Stochastic Gradient Descent, but rather, optimal weights are computed by a closed formula.

The weights of the input layer and the reservoir layer are randomly initialized and are the same for all clients. Each local dataset *D* is a matrix of the shape *(rows, features)*. In the first step, the input matrix is reshaped into *(time steps, sequence length, features)* in order to be fed into the ESN model. Data is time-series, and sequence length is fixed for all time steps.

Random weights of the input and reservoir layers are fixed and are not trained during learning. The next step is calculation of the *final state* matrix, which has the shape *(time steps, units)*. ESN calculates the *final state* matrix by using input and reservoir layer weights. In the final step, weights of the output layer are multiplied by the state matrix to compute our predictions **Y**. Therefore, we have the following equation:

$$\mathbf{Y} = \mathbf{UH}$$

In the equation above, **H** is the state matrix, and **U** is the trainable weights of the output layer (readout). **Y** has the shape *(time steps, classes)* and **U** is in the shape of *(units, classes)*.

We are investigating two main approaches for learning the model in the federated scenario: federated averaging and incremental federated learning. These approaches have been described in Section 2.2.

For benchmark, we have simulated a Federated Learning scenario for human state and activity recognition, employing the WESAD [133] and Heterogeneity Activity Recognition [134] datasets. Both datasets are publicly available.

The WESAD dataset contains data for performing wearable stress and affect detection. It includes data about blood volume pulse, electrocardiogram, electrodermal activity, electromyogram, respiration, body temperature, and three-axis acceleration. The associated time-series are labelled to distinguish between four different affective states, namely *neutral*, *stress*, *amusement* and *meditation*. It includes recordings from 15 different subjects.

The Heterogeneity Activity Recognition (HAR) dataset contains time-series from sensors in smartphones and smartwatches with the goal of human activity recognition. The sensors include accelerometers and gyroscopes, while the labelled activities are *biking*, *sitting*, *standing*, *walking*, *stair up*, and *stair down*. It includes recordings from 9 different subjects.

The subjects available in the datasets have been used to train individual predictors, one per subject. All predictors are implemented as ESNs, all with the same shared reservoir topology. After the individual predictors have been trained (thus simulating the training that happens in the devices on the edge), the parameters of the readout are shared with a centralized entity (represented by the cloud in the federation) where they are aggregated.

Table 3 Training and test accuracy on the WESAD dataset with respect to different numbers of clients, using the "averaging" aggregation strategy.

Users	Local Accuracy (%)	Local Accuracy Std	Train Accuracy (%)	Train Accuracy Std	Test Accuracy (%)	Test Accuracy Std
25%	97.83	0.02	81.55	0.09	63.42	0.10
50%	98.01	0.03	77.63	0.09	71.75	0.12
75%	98.02	0.024	76.77	0.11	74.69	0.09
100%	98.02	0.028	76.57	0.12	75.81	0.10

Table 4 Training and test accuracy on the WESAD dataset with respect to different numbers of clients, using the "incremental learning" aggregation strategy.

Users	Local Accuracy (%)	Local Accuracy Std	Train Accuracy (%)	Train Accuracy Std	Test Accuracy (%)	Test Accuracy Std
25%	98.36	0.01	93.14	0.04	65.96	0.08
50%	98.60	0.01	87.51	0.06	74.89	0.09
75%	98.59	0.02	84.45	0.05	76.56	0.07
100%	98.51	0.02	83.78	0.06	77.92	0.07

Table 5 Training and test accuracy on the HAR dataset with respect to different numbers of clients, using the "averaging" aggregation strategy.

Users	Local Accuracy (%)	Local Accuracy Std	Train Accuracy (%)	Train Accuracy Std	Test Accuracy (%)	Test Accuracy Std
25%	93.39	0.00	93.39	0.00	59.71	0.07
50%	93.74	0.02	78.02	0.08	67.96	0.08
75%	93.77	0.02	71.37	0.09	70.70	0.06
100%	93.69	0.03	71.38	0.11	73.74	0.05

Table 6 Training and test accuracy on the HAR dataset with respect to different numbers of clients, using the "incremental learning" aggregation strategy.

Users	Local Accuracy(%)	Local Accuracy Std	Train Accuracy (%)	Train Accuracy Std	Test Accuracy (%)	Test Accuracy Std
25%	92.73	0.00	92.73	0.00	55.96	0.11
50%	93.60	0.02	85.66	0.05	70.28	0.05
75%	93.84	0.03	81.62	0.07	75.69	0.04
100%	93.64	0.03	79.59	0.10	80.19	0.03

As expected, all experiments show an increase in predictive accuracy with respect to increasing numbers of subjects (i.e., edge nodes in the federation). Both aggregation approaches (weight averaging and incremental learning) lead to good predictive accuracy, as shown in Table 3 and Table 4 for WESAD, and in Table 5 and Table 6 for HAR. The "local accuracy" columns indicate the performance of the model when trained only on the locally available data.

Future work will include a thorough investigation of the differences, with respect to predictive accuracy and efficiency, between aggregation approaches. Moreover, investigations will focus on the case in which the reservoir topologies are not shared between the edge nodes. The experimental results will be strengthened by the use of additional real-world datasets for benchmarking the proposed techniques.

5.2. Reservoir Computing for Stress Estimation: Demo

We are working to the development of the ANSIA (A Neural System that Infers Affects) software, an application that provides an interface to the Consensys Bundle Development Kit, an all-in-one solution that allows the testing of sensory sensing capabilities offered by the platform Shimmer3[12], as illustrated in Figure 16.

[12] http://shimmersensing.com

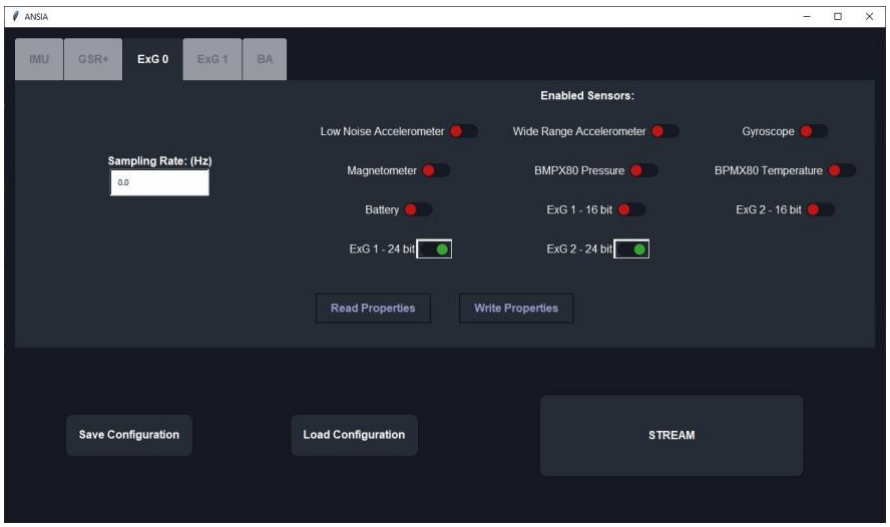

Figure 16 Sample screenshot from the ANSIA software.

The software provides two modes of use: WESAD and experimental.

The ***WESAD mode*** offers the user the demo of an emotional state predictor. The artificial intelligence on which this is based will be composed of a Deep ESN trained through the dataset WESAD (WEarable Stress and Affect Detection), a multi-modal dataset created with the intent to provide a public resource that is a reference for the application of Machine Learning technologies (and not) to the topic of human emotionality classification (as already described in the previous Section 5.1).

During the first access in this mode the user will be guided to the installation of the sensor setup by a short tutorial aimed at suggesting the correct positioning of the different measuring sensor devices. For the correct execution of the NN predictor it is essential that the user follows the guidelines defined by the software. There are three devices needed to use it: one Shimmer3 GSR+ placed on the wrist and two Shimmer3 ExG placed on the chest. Through the first one the electrodermal activity (EDA) and the blood volume pulse (BVP) will be measured, from which the heart rate is obtained. With the second, an electromyography (EMG) and a measurement of respiration (RESP) are performed:

- Using finger electrodes, the EDA signal is measured on the fingers of the non-dominant hand;
- Using an optical pulse sensor, the signal related to BVP is measured through the index finger (upper part) or from the earlobe (solution that ensures greater stability)
- The breathing signal is recorded through the use of electrodes, placed in the middle axillary lines (one on each side).
- The EMG measurement is instead set on the basis of the WESAD experiment: the signal is recorded from the upper muscles of the trapezius.

The predictions emitted by the Deep ESN model will be displayed in real-time on the screen. The software will also proceed to save the proposed predictions in a .csv file, usable for future analysis.

The ***experimental mode***, on the other hand, wants to be a free interface to the various features offered by the Shimmer platform. The user will be free to connect the devices he prefers, enabling and disabling at will the sensors from which to receive data. Compared to the

previously described scenario, this time is the user himself that decides and controls the configuration of each device.

5.3. Vehicle control and autonomous driving profile personalization

Being able to modify the driving functionality of the vehicle controllers based on the various inputs (driver state sensors, car perception sensors etc.) is a key factor towards autonomous driving profile personalization. In more details, we have identified two main individual tasks:

- Choose a different driving model based on the driver's stress level
- Use model predictive control (MPC) for smooth adjustment of the nonlinear driving control system

For research purposes we use an open-source simulator for autonomous driving (CARLA) for experimenting and evaluating our approaches. Currently, Carla's behaviour controller:

- Implements three types of navigation agents
- Utilizes three types of behaviour (let's call them driving modes) that define the way the autonomous vehicle responds to driving decisions. These modes are referred to as: **Cautious**, **Normal** and **Aggressive**
- And uses PID controllers to control the longitudinal, lateral and speed control

Based on the type of behaviour/driving mode chosen for the execution of a driving scenario, the simulator passes different parameter values to the PID controllers that control the autonomous vehicle when the autopilot is turned on.

Given the default Carla behaviour controller, we have modified the behaviour agent used by the autopilot so that we record a keystroke that simulates user's expression of stress, which in the context of the WP4 tasks driver's stress will finally be provided by a stress level recognition learning module. Based on the level of distress expressed by the driver, we dynamically change the agent's driving mode. At this point, we are also working on training a Reinforcement Learning agent using Carla's camera sensors and longitudinal, lateral proximity sensors to test whether this agent can be used to control the car's behaviour based on the driver's stress levels.

Finally, in the context of WP4 task T4.3, we are exploring Gekko and do-mpc libraries to build an MPC controller that will be added to the Carla autopilot functionality. This MPC integration will allow of more refined control of the autonomous vehicle and provide the ability of modifying the vehicle's control parameters on the fly based on various input parameters like the user stress.

Based our initial tests, this MPC can provide fine grained autonomous vehicle control and at this point we are in the process of integrating this MPC controller into Carla's source code to examined realistic driving scenarios on Carla's integrated world maps. In parallel with this task, we are also designing a machine learning module that will learn the desired MPC coefficients based on a set of sensor parameters as well as user emotions (i.e., stress). This will allow real-time fine-tuning of the MPC for different drivers, changing driver profiles or adjusting the driving behaviour under different driving conditions or as driver and/or driver's emotions change etc.

Under the prism of validation and verification with regards to the perceived human safety and trustworthiness raised in Section 2.6.2 we plan on following a similar approach for evaluating the effect of our model with respect to the different scenarios and types of feedback provided to the user. We will also simulate various driving scenarios in the Carla simulator and the resulting impact on users' trust [132] and comfort will be taken into account. All of our current development on the Carla simulator for evaluating our LMs towards autonomous driving

personalization is planned with respect to following the same set of rules and traffic situations as well as using the same set of evaluation metrics as they have already been set in the Carla Autonomous Driving Challenge.

6. Conclusions

WP4 aims at designing the necessary methodologies for creating the TEACHING AIaaS software infrastructure. The currently active tasks focus on the development of the core ML-related components (T4.1), on the monitoring of the human state in CPSoS from sensor data (T4.2), and on human-centric personalization (T4.3).

In this document we provided a state-of-the-art analysis of the involved ML methodology, highlighting the identified roles of Recurrent and Reservoir Computing NNs, Federated and Continual Learning, metrics for NN dependability and safety, human state monitoring, and human-centric personalization in autonomous vehicles. We also gave a preliminary analysis of the TEACHING AIaaS requirements and the system's design. Finally, we provided a brief insight into the preliminary results achieved in the active tasks, in the fields of Federated Reservoir Computing, stress estimation from physiological data, and autonomous driving personalization.

Following the envisaged project's lifecycle, the outcomes of the work conducted in WP4 and described in this document, along with that of the other technical WPs, will drive the efforts in the core technology building during Y2 (Phase 2). From the WP4 viewpoint, the effort will concentrate on implementing the main concepts of dependable, distributed, federated, and lightweight RNNs for CPSoS, human monitoring, and personalization in autonomous vehicles. These activities will be complemented by the work on privacy-aware AI models from Task T4.3 (starting at M13) and will result in the integrated mockup of the AIaaS system in D4.2, hence contributing to the project's milestone MS2 on the first integrated setup of the TEACHING platform at M20.

7. Bibliography

[1] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[2] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature,* vol. 521, no. 7553, pp. 436-444, 2015.

[3] C. Gallicchio and S. Scardapane, "Deep Randomized Neural Networks," in *Recent Trends in Learning From Data*, Springer, Cham, 2020, pp. 43-68.

[4] S. Scardapane and D. Wang, "Randomness in neural networks: an overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 7, no. 2, p. e1200, 2017.

[5] L. Zhang and P. N. Suganthan, "A survey of randomized algorithms for training neural networks," *Information Sciences,* vol. 364, pp. 146-155, 2016.

[6] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review,* vol. 3, no. 3, pp. 127-149, 2009.

[7] A. Rahimi and B. Recht, "Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning," in *Advances in neural information processing systems*, 2008.

[8] C. Gallicchio, "Deep Randomized Neural Networks, tutorial of AAAI-21," [Online]. Available: https://aaai.org/Conferences/AAAI-21/aaai21tutorials/.

[9] P. Dominey, M. Arbib and J. Joseph, "A model of corticostriatal plasticity for learning oculomotor associations and sequences," *Journal of cognitive neuroscience,* vol. 5, no. 3, pp. 311-336, 1995.

[10] S. C. Kremer and J. F. Kolen, Field Guide to Dynamical Recurrent Networks, Wiley-IEEE Press, 2001.

[11] Y. Bengio, P. Y. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks,* vol. 5, no. 2, pp. 157-166, 1994.

[12] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Networks,* pp. 100-123, 2019.

[13] H. Jaeger, *The "echo state" approach to analysing and training recurrent neural networks-with an Erratum note,* 2001.

[14] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science,* pp. 78-80, 2004.

[15] H. Jaeger, M. Lukosevicius, D. Popovici and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Networks,* pp. 335-352, 2007.

[16] T. Strauss, W. Wustlich and R. Labahn, "Design strategies for weight matrices of echo state networks," *Neural computation,* vol. 24, no. 12, 2012.

[17] O. L. White, D. D. Lee and H. Sompolinsky, "Short-term memory in orthogonal neural networks," *Physical review letters,* vol. 92, no. 14, 2004.

[18] J. Boedecker, O. Obst, N. M. Mayer and M. Asada, "Studies on reservoir initialization and dynamics shaping in echo state networks.," in *ESANN*, 2009.

[19] M. A. Hajnal and A. Lőrincz, "Critical echo state networks," in *International Conference on Artificial Neural Networks*, 2006.

[20] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE transactions on neural networks,* vol. 22, no. 1, 2010.

[21] H. Jaeger, "Short term memory in echo state networks.," in *GMD-German National Research Institute for Computer Science* , 2002.

[22] I. Farkaš, R. Bosák and P. Gergeľ, "Computational analysis of memory capacity in echo state networks," *Neural Networks,* vol. 83, 2016.

[23] C. Gallicchio and A. Micheli, "Deep Reservoir Computing: A Critical Analysis," in *ESANN,* 2016.

[24] C. Gallicchio, A. Micheli and L. Pedrelli, "Deep reservoir computing: A critical experimental analysis," *Neurocomputing*, pp. 87-99, 2017.

[25] C. Gallicchio, A. Micheli and L. Pedrelli, "Design of deep echo state networks," *Neural Networks,* 2018.

[26] C. Gallicchio and A. Micheli, "Echo state property of deep reservoir computing networks," *Cognitive Computation,* vol. 9, no. 3, pp. 337-350, 2018.

[27] D. Di Sarli, C. Gallicchio and A. Micheli, "Gated Echo State Networks: a preliminary study.," in *INISTA*, 2020.

[28] K. Cho, B. v. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *EMNLP,* 2014.

[29] M. Lukoševičius, H. Jaeger and B. Schrauwen, "Reservoir computing trends," *KI-Künstliche Intelligenz,* 2012.

[30] B. Schrauwen, D. Verstraeten and J. Van Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th european symposium on artificial neural networks*, 2007.

[31] F. Palumbo, C. Gallicchio, R. Pucci and A. Micheli, "Human activity recognition using multisensor data fusion based on reservoir computing," *Journal of Ambient Intelligence and Smart Environments,* 2016.

[32] E. Crisostomi, C. Gallicchio, A. Micheli, M. Raugi and M. Tucci, "Prediction of the Italian electricity price for smart grid applications}," *Neurocomputing,* 2015.

[33] D. Bacciu, P. Barsocchi, S. Chessa, C. Gallicchio and A. Micheli, "An experimental characterization of reservoir computing in ambient assisted living applications," *Neural Comput. Appl.,* pp. 1451-1464, 2014.

[34] C. Gallicchio and A. Micheli, "Experimental Analysis of Deep Echo State Networks for Ambient Assisted Living.," in *AI* AAL@ AI* IA,* 2017.

[35] C. Gallicchio, A. Micheli and L. Pedrelli, "Deep Echo State Networks for Diagnosis of Parkinson's Disease," in *ESANN,* 2018.

[36] C. Gallicchio, A. Micheli and L. Pedrelli, "Comparison between DeepESNs and gated RNNs on multivariate time-series prediction," in *ESANN,* 2018.

[37] M. Alizamir, S. Kim, O. Kisi and M. Zounemat-Kermani, "Deep echo state network: a novel machine learning approach to model dew point temperature using meteorological variables," *Hydrological Sciences Journal,* 2020.

[38] T. Kim and B. R. King, "Time series prediction using deep echo state networks," *Neural Computing and Applications,* 2020.

[39] Q. Li, Z. Wu, R. Ling, L. Feng and K. Liu, "Multi-reservoir echo state computing for solar irradiance prediction: A fast yet efficient deep learning approach," *Applied Soft Computing,* 2020.

[40] H. Hu, L. Wang and S.-X. Lv, "Forecasting energy consumption and wind power generation using deep echo state network," *Renewable Energy,* 2020.

[41] Z. Song, K. Wu and J. Shao, "Destination prediction using deep echo state network," *Neurocomputing,* 2020.

[42] S. Dettori, I. Matino, V. Colla and R. Speets, "Deep Echo State Networks in Industrial Applications," in *IFIP International Conference on Artificial Intelligence Applications and Innovations,* 2020.

[43] V. Colla, I. Matino, S. Dettori, S. Cateni and R. Matino, "Reservoir computing approaches applied to energy management in industry," in *International Conference on Engineering Applications of Neural Networks,* 2019.

[44] D. Bacciu, S. Chessa, C. Gallicchio, A. Micheli, L. Pedrelli, E. Ferro, L. Fortunati, D. La Rosa, F. Palumbo, F. Vozzi and O. Parodi, "A learning system for automatic Berg Balance Scale score estimation," *Engineering Applications of Artificial Intelligence,* vol. 66, pp. 60-74, 2017.

[45] M. Dragone, G. Amato, D. Bacciu, S. Chessa, S. Coleman, M. Di Rocco, C. Gallicchio, C. Gennaro, H. Lozano, L. Maguire and M. McGinnity, "A cognitive robotic ecology approach to self-configuring and evolving AAL systems," *Engineering Applications of Artificial Intelligence,* vol. 45, pp. 269-280, 2015.

[46] G. Amato, D. Bacciu, M. Broxvall, S. Chessa, S. Coleman, M. Di Rocco, M. Dragone, C. Gallicchio, C. Gennaro, H. Lozano and T. McGinnity, "Robotic ubiquitous cognitive ecology for smart homes," *Journal of Intelligent & Robotic Systems,* vol. 80, no. 1, pp. 57-81, 2015.

[47] D. Bacciu, C. Gallicchio, A. Micheli, S. Chessa and P. Barsocchi, "Predicting user movements in heterogeneous indoor environments by reservoir computing," in *Proc. of*

the IJCAI Workshop on Space, Time and Ambient Intelligence (STAMI), Barcelona, Spain, 2011.

[48] C. Gallicchio, A. Micheli, P. Barsocchi and S. Chessa, "User movements forecasting by reservoir computing using signal streams produced by mote-class sensors.," in *In International Conference on Mobile Lightweight Wireless Systems*, 2011.

[49] D. Kleyko, E. P. Frady and E. Osipov, "Integer Echo State Networks: Hyperdimensional Reservoir Computing," *CoRR*, vol. abs/1706.00280, 2017.

[50] K. Sozinov, V. Vlassov and S. Girdzijauskas, "Human Activity Recognition Using Federated Learning," in *IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications*, 2018.

[51] Ditzler, Gregory, Manuel Roveri, Cesare Alippi, and Robi Polikar, "Learning in nonstationary environments: A survey," *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12-25, 2015.

[52] G. I. Parisi, J. Tani, C. Weber, and S. Wermter, "Lifelong Learning of Spatiotemporal Representations With Dual-Memory Recurrent Self- Organization," *Frontiers in Neurorobotics*, vol. 12, 2018.

[53] G. M. van de Ven and A. S. Tolias, "Three scenarios for continual learning,," in *Continual Learning Workshop NeurIPS*, 2018.

[54] Maltoni, Davide, and Vincenzo Lomonaco, "Continuous learning in single-incremental-task scenarios," *Neural Networks*, vol. 116, pp. 56-73, 2019.

[55] Díaz-Rodríguez, Natalia, Vincenzo Lomonaco, David Filliat, and Davide Maltoni, "Don't forget, there is more than forgetting: new metrics for Continual Learning," arXiv preprint arXiv:1810.13166, 2018.

[56] Lopez-Paz, David, and Marc'Aurelio Ranzato, "Gradient episodic memory for continual learning," in *Advances in neural information processing systems*, 2017.

[57] G. Parisi, R. Kemker, J. Part, C. Kanan and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54-71, 2019.

[58] B. Pfülb and A. Gepperth, "A comprehensive, application-oriented study of catastrophic forgetting in DNNs," in *ICLR*, 2019.

[59] R. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128-135, 1999.

[60] G. A. Carpenter and S. Grossberg, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network," *Computer*, vol. 21, no. 3, pp. 77-88, 1988.

[61] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123-146, 1995.

[62] L. Abbott and S. Nelson, "Synaptic plasticity: taming the beast," *Nature neuroscience*, vol. 3, no. 11, pp. 1178-1183, 2000.

[63] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence,* vol. 40, no. 12, pp. 2935-2947, 2017.

[64] X. Liu, M. Masana, L. Herranz, J. Van de Weijer, A. Lopez and A. Bagdanov, "Rotate your networks: Better weight consolidation and less catastrophic forgetting," in *24th International Conference on Pattern Recognition (ICPR)*, 2018.

[65] J. Pomponi, S. Scardapane, V. Lomonaco and A. Uncini, "Efficient continual learning in neural networks with embedding regularization," *Neurocomputing,* 2020.

[66] H. Ahn, S. L. D. Cha and T. Moon, "Uncertainty-based continual learning with adaptive regularization," in *Advances in Neural Information Processing Systems*, 2019.

[67] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska and D. Hassabis, "Overcoming catastrophic forgetting in neural networks," in *Proceedings of the national academy of sciences*, 2017.

[68] F. Zenke, B. Poole and S. Ganguli, "Continual Learning Through Synaptic Intelligence," in *International Conference on Machine Learning*, 2017.

[69] A. Chaudhry, P. Dokania, T. Ajanthan and P. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[70] G. Hinton, O. Vinyals and J. Dean, "Distilling the knowledge in a neural network," in *Deep Learning and Representation Learning Workshop. In Neural Information Processing Systems*, 2015.

[71] T. Draelos, N. Miner, C. Lamb, J. Cox, C. Vineyard, K. Carlson, W. Severa, C. James and J. Aimone, "Neurogenesis deep learning: Extending deep networks to accommodate new classes," in *International Joint Conference on Neural Networks (IJCNN)*, 2017.

[72] J. Yoon, E. Yang, J. Lee and S. Hwang, "Lifelong learning with dynamically expandable networks," in *International Conference on Learning Representations*, 2018.

[73] A. Rusu, N. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu and R. Hadsell, "Progressive neural networks," in *arXiv preprint arXiv:1606.04671,* 2016.

[74] A. Cossu, A. Carta and D. Bacciu, "Continual Learning with Gated Incremental Memories for sequential data processing," in *International Joint Conference on Artificial Neural Networks*, 2020.

[75] C. Hung, C. Tu, C. Wu, C. Chen, Y. Chan and C. Chen, "Compacting, picking and growing for unforgetting continual learning," in *Advances in Neural Information Processing Systems*, 2019.

[76] S. Srivastava, M. Berman, M. Blaschko and D. Tuia, "Adaptive compression-based lifelong learning," in *30th British machine vision conference - BMVC*, 2019.

[77] R. Aljundi, E. Belilovsky, T. Tuytelaars, L. Charlin, M. Caccia, M. Lin and L. Page-Caccia, "Online continual learning with maximal interfered retrieval," in *Advances in Neural Information Processing Systems*, 2019.

[78] D. Isele and A. Cosgun, "Selective Experience Replay for Lifelong Learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[79] H. Shin, J. K. Lee, J. Kim and J. Kim, "Continual Learning with Deep Generative Replay,," in *Advances in Neural Information Processing Systems*, 2017.

[80] Z. Wang, C. Subakan, E. Tzinis, P. Smaragdis and L. Charlin, "Continual Learning of New Sound Classes Using Generative Replay," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2019.

[81] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014.

[82] H. Li, P. Barnaghi, S. Enshaeifar and F. Ganz, "Continual learning using bayesian neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[83] S. Ebrahimi, M. Elhoseiny, T. Darrell and M. Rohrbach, "Uncertainty-guided continual learning with bayesian neural networks," in *International Conference on Learning Representations*, 2020.

[84] J. Allred and K. Roy, "Controlled Forgetting: Targeted Stimulation and Dopaminergic Plasticity Modulation for Unsupervised Lifelong Learning in Spiking Neural Networks," *Frontiers in neuroscience*, vol. 14, 2020.

[85] A. Ororbia, "Spiking Neural Predictive Coding for Continual Learning from Data Streams," *arXiv preprint arXiv:1908.08655*, vol. 2019.

[86] T. Kobayashi and T. Sugino, "Continual Learning Exploiting Structure of Fractal Reservoir Computing," in *In International Conference on Artificial Neural Networks*, 2019.

[87] R. Salay, R. Queiroz and K. Czarnecki, *An Analysis of ISO 26262: Using Machine Learning Safely in Automotive Software*, arXiv, 2017.

[88] C.-H. Cheng, G. N¨uhrenberg, C.-H. Huang, H. Ruess and H. Yasuoka, *Towards Dependability Metrics for Neural Networks*, arXiv, 2018.

[89] S. Jerritta, M. Murugappan, R. Nagarajan and K. Wan, "Physiological signals based human emotion recognition: a review," in *2011 IEEE 7th International Colloquium on Signal Processing and its Applications*, 2011.

[90] P. J. Bota, C. Wang, A. L. N. Fred and H. Plácido Da Silva, "A Review, Current Challenges, and Future Possibilities on Emotion Recognition Using Machine Learning and Physiological Signals," *IEEE Access*, vol. 7, p. 140990–141020, 2019.

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

[92] O. Faust, Y. Hagiwara, T. J. Hong, O. S. Lih and U. R. Acharya, "Deep Learning for Healthcare Applications Based on Physiological Signals: A Review," *Computer Methods and Programs in Biomedicine*, vol. 161, p. 1–13, 7 2018.

[93] B. Rim, N.-J. Sung, S. Min and M. Hong, "Deep Learning in Physiological Signal Data: A Survey," *Sensors,* vol. 20, p. 969, 2020.

[94] K. A. Brookhuis and D. De Waard, "Monitoring drivers' mental workload in driving simulators using physiological measures," *Accident Analysis & Prevention,* vol. 42, p. 898–903, 2010.

[95] S. Begum, M. U. Ahmed, P. Funk and R. Filla, "Mental state monitoring system for the professional drivers based on Heart Rate Variability analysis and Case-Based Reasoning," in *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2012.

[96] Z. Kowalczuk, M. Czubenko and T. Merta, "Emotion monitoring system for drivers," *IFAC-PapersOnLine,* vol. 52, p. 200–205, 2019.

[97] Z. Kowalczuk, M. Czubenko and T. Merta, "Interpretation and modeling of emotions in the management of autonomous robots using a control paradigm based on a scheduling variable," *Engineering Applications of Artificial Intelligence,* vol. 91, p. 103562, 2020.

[98] H. Bellem, T. Schönenberg, J. F. Krems and M. Schrauf, "Objective metrics of comfort: developing a driving style for highly automated vehicles," *Transportation research part F: traffic psychology and behaviour,* vol. 41, p. 45–54, 2016.

[99] H. Bellem, B. Thiel, M. Schrauf and J. F. Krems, "Comfort in automated driving: An analysis of preferences for different automated driving styles and their dependence on personality traits," *Transportation research part F: traffic psychology and behaviour,* vol. 55, p. 90–100, 2018.

[100] H. Bellem, M. Klüver, M. Schrauf, H.-P. Schöner, H. Hecht and J. F. Krems, "Can we study autonomous driving comfort in moving-base driving simulators? A validation study," *Human factors,* vol. 59, p. 442–456, 2017.

[101] T. Chakraborti, S. Sreedharan, A. Kulkarni and S. Kambhampati, "Alternative modes of interaction in proximal human-in-the-loop operation of robots," *arXiv preprint arXiv:1703.08930,* 2017.

[102] J. J. Meyer, W.-L. Hu, Z. Wang, D. E. Adams, T. Reid and A. Chaturvedi, "Application of SHM Pattern Recognition to Assess Decision Making of Humans in the Loop," *Structural Health Monitoring 2015,* 2015.

[103] A. Al-Rahayfeh and M. Faezipour, "Eye tracking and head movement detection: A state-of-art survey," *IEEE journal of translational engineering in health and medicine,* vol. 1, p. 2100212–2100212, 2013.

[104] J. D. Fuletra and D. Bosamiya, "A survey on driver's drowsiness detection techniques," *International Journal on Recent and Innovation Trends in Computing and Communication,* vol. 1, p. 816–819, 2013.

[105] E. T. Solovey, M. Zec, E. A. Garcia Perez, B. Reimer and B. Mehler, "Classifying driver workload using physiological and driving performance data: two field studies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014.

[106] J. A. Healey and R. W. Picard, "Detecting stress during real-world driving tasks using physiological sensors," *IEEE Transactions on intelligent transportation systems,* vol. 6, p. 156–166, 2005.

[107] L.-l. Chen, Y. Zhao, P.-f. Ye, J. Zhang and J.-z. Zou, "Detecting driving stress in physiological signals based on multimodal feature analysis and kernel classifiers," *Expert Systems with Applications,* vol. 85, p. 279–291, 2017.

[108] J. Healey and R. Picard, "SmartCar: detecting driver stress," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000,* 2000.

[109] N. Sharma and T. Gedeon, "Objective measures, sensors and computational techniques for stress recognition and classification: A survey," *Computer methods and programs in biomedicine,* vol. 108, p. 1287–1301, 2012.

[110] S. Kaplan, M. A. Guvensan, A. G. Yavuz and Y. Karalurt, "Driver behavior analysis for safe driving: A survey," *IEEE Transactions on Intelligent Transportation Systems,* vol. 16, p. 3017–3032, 2015.

[111] A. E. Af Wåhlberg, "Short-term effects of training in economical driving: Passenger comfort and driver acceleration behavior," *International Journal of Industrial Ergonomics,* vol. 36, p. 151–163, 2006.

[112] Y. He, X. Yan, C. Wu, D. Chu and L. Peng, "Effects of Driver's Unsafe Acceleration Behaviors on Passengers' Comfort for Coach Buses," in *ICTIS 2013: Improving Multimodal Transportation Systems-Information, Safety, and Integration,* 2013, p. 1649–1655.

[113] H. Singh, J. S. Bhatia and J. Kaur, "Eye tracking based driver fatigue monitoring and warning system," in *India International Conference on Power Electronics 2010 (IICPE2010),* 2011.

[114] O. Ursulescu, B. Ilie and G. Simion, "Driver Drowsiness Detection Based on Eye Analysis," in *2018 International Symposium on Electronics and Telecommunications (ISETC),* 2018.

[115] S. Arun, M. Murugappan and K. Sundaraj, "Hypovigilance warning system: A review on driver alerting techniques," in *2011 IEEE Control and System Graduate Research Colloquium,* 2011.

[116] S. Kraus, S. Albrecht, M. Sobotka, B. Heißing and M. Ulbrich, "Optimisation-based identification of situation determined cost functions for the implementation of a human-like driving style in an autonomous car," in *International Symposium on Advanced Vehicle Control,* 2010.

[117] T. J. Pallett, J. A. Doering, A. D. Tsakiris and K. R. Post, "Selectable autonomous driving modes". Washington, DC: U.S. Patent and Trademark Office Patent 9,365,218, 2016.

[118] F. Walldén, S. Löfving and B. Dai, "Supporting personalisation of ADAS through driver characteristics-a design science study," 2019.

[119] M. Beggiato, F. Hartwich, P. Roßner, A. Dettmann, S. Enhuber, T. Pech, ... and J. Krems, "KomfoPilot—Comfortable Automated Driving," *Smart Automotive Mobility,* pp. 71-154, 2020.

[120] E. Ohn-Bar, S. Martin, A. Tawari and M. M. Trivedi, "Head, eye, and hand patterns for driver activity recognition," in *2014 22nd International Conference on Pattern Recognition,* 2014.

[121] C. Yan, F. Coenen and B. Zhang, "Driving posture recognition by convolutional neural networks," *IET Computer Vision,* pp. 10(2), 103-114, 2016.

[122] M. Martin, A. Roitberg, M. Haurilet, M. Horne, S. Reiß, M. Voit and R. Stiefelhagen, "Drive&act: A multi-modal dataset for fine-grained driver behavior recognition in autonomous vehicles," in *IEEE international conference on computer vision,* 2019.

[123] H. Guo, D. Cao, H. Chen, Z. Sun and Y. Hu, "Model predictive path following control for autonomous cars considering a measurable disturbance: Implementation, testing, and verification," *Mechanical Systems and Signal Processing,* pp. 118, 41-60, 2019.

[124] S. Dixit, U. Montanaro, M. Dianati, D. Oxtoby, T. Mizutani, A. Mouzakitis and S. Fallah, "Trajectory planning for autonomous high-speed overtaking in structured environments using robust MPC," *IEEE Transactions on Intelligent Transportation Systems,* pp. 21(6), 2310-2323, 2019.

[125] M. Flad, P. Karg, A. Roitberg, M. Martin, M. Mazewitsch, C. Lange, ... and B. Karakaya, "Personalisation and Control Transition Between Automation and Driver in Highly Automated Cars," *In Smart Automotive Mobility,* pp. 1-70, 2020.

[126] A. Kendall, J. Hawke, D. Janz, P. R. D. A. J. Mazur, ... and A. Shah, "Learning to Drive in a Day," in *International Conference on Robotics and Automation (ICRA),* 2019.

[127] V. Mnih, A. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, ... and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning,* 2016.

[128] B. Liu and S. Mazumder, "Lifelong and continual learning dialogue systems: learning during conversation," in *Proceedings of AAAI-2021, 2021*.

[129] B. Pfülb, A. Gepperth, S. Abdullah and A. Kilian, "Catastrophic forgetting: still a problem for DNNs," in *International Conference on Artificial Neural Networks*, 2018.

[130] P. Koopman, R. Hierons, S. Khastgir, J. Clark, M. Fisher, R. Alexander, ... and A. Blake, "Certification of Highly Automated Vehicles for Use on UK Roads: Creating An Industry-Wide Framework for Safety," 2019.

[131] M. Webster, D. Western, D. Araiza-Illan, C. Dixon, K. Eder, M. Fisher and A. G. Pipe, "A corroborative approach to verification and validation of human–robot teams," *International Journal of Robotics Research,* pp. 39(1), 73-99, 2020.

[132] R. Flook, A. Shrinah, L. Wijnen, K. Eder, C. Melhuish and S. Lemaignan, "On the impact of different types of errors on trust in human-robot interaction: Are laboratory-based HRI experiments trustworthy?," *Interaction Studies,* pp. 20(3), 455-486, 2019.

[133] P. Schmidt, A. Reiss, R. Durichen, C. Marberger and K. V. Laerhoven, "Introducing WESAD, a Multimodal Dataset for Wearable Stress and Affect Detection," in

Proceedings of the 2018 on International Conference on Multimodal Interaction, ICMI 2018, Boulder, CO, USA, October 16-20, 2018, 2018.

[134] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne and M. M. Jensen, "Smart Devices are Different: Assessing and Mitigating Mobile Sensing Heterogeneities for Activity Recognition," in *13th ACM Conference on Embedded Networked Sensor Systems*, 2015.

[135] H. Jaeger and M. Lukosevicius, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127-149, 2009.

[136] R. R. Singh, S. Conjeti and R. Banerjee, "Biosignal based on-road stress monitoring for automotive drivers," in *2012 National Conference on Communications (NCC)*, 2012.