TEACHING Platform for Human-Centric Autonomous Applications: Design and Overview

Valerio De Caro
Department of Computer Science
University of Pisa
Pisa, Italy

Christos Chronis
Department of Informatics and Telematics
Harokopio University of Athens
Athens, Greece

Massimo Coppola
Information Science and Technology Institute
National Research Council
Pisa, Italy

Vincenzo Lomonaco
Department of Computer Science
University of Pisa
Pisa, Italy

Claudio Gallicchio
Department of Computer Science
University of Pisa
Pisa, Italy

Konstantinos Tserpes
Department of Informatics and Telematics
Harokopio University of Athens
Athens, Greece

Davide Bacciu
Department of Computer Science
University of Pisa
Pisa, Italy

ABSTRACT

The TEACHING project enhances AI applications in pervasive environments via Humanistic Intelligence, fostering synergy between humans and Cyber-Physical Systems of Systems (CPSoS). Here, we present the TEACHING Platform, a microservice-based framework providing the technological advancements to represent humans and CPSoS as containerized software models that interact to mutually empower each other.

CCS CONCEPTS

• **Human-centered computing**; • **Computing methodologies** → **Artificial intelligence**; • **Software and its engineering** → **Software libraries and repositories**;

KEYWORDS

Human-centered computing, Artificial Intelligence, Software

ACM Reference Format:
Valerio De Caro, Christos Chronis, Massimo Coppola, Vincenzo Lomonaco, Claudio Gallicchio, Konstantinos Tserpes, and Davide Bacciu. 2024. TEACHING Platform for Human-Centric Autonomous Applications: Design and Overview. In *The 33rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '24), June 3–7, 2024, Pisa,Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3625549.3658813

1 INTRODUCTION

Humanistic Intelligence (HI) is a concept integral to the TEACHING project [1], focusing on the symbiotic relationship between humans and computational processes. Within this framework, the TEACHING project aims to develop a platform facilitating seamless interaction between humans and Cyber-Physical Systems of Systems (CPSoS), with the integration of AI for mutual empowerment (depicted in Figure 1). This paper discusses the design of the TEACHING CPSoS architecture, where human-centric design principles redefine the traditional human-computer interaction paradigm. Central to this approach is the concept of HI, wherein humans and computers are seen as intertwined entities, with computers augmenting human capabilities akin to a second brain. The paper addresses key design questions regarding incorporating human interaction into AI-powered systems and leveraging human intelligence to enhance AI. This human-centric design ethos is applied in TEACHING by ensuring continuous consideration of humans as central elements of the system. The TEACHING Platform is designed to enable HI by integrating human reactions into the system's feedback loop, thereby treating humans as first-class citizens. The platform's functional requirements include monitoring and quantifying human states, integrating them into decision-making processes, and providing programmability for dynamic application creation. By understanding these principles, this paper sets the stage for defining the TEACHING Platform and its specifications.

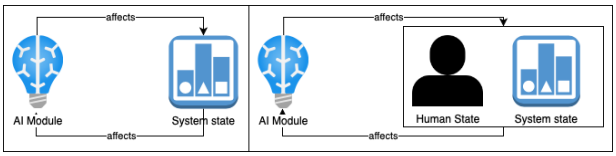

Figure 1: TEACHING enables the concept of Humanistic Intelligence by closing the loop between the human user(s) and the AI-enabled CPSoS. The mutual feedback about the counterpart status and behaviour allows the CPSes and the whole CPSoS to dynamically adapt to the user(s).

HPDC '24, June 3–7, 2024, Pisa, Italy

ACM ISBN 979-8-4007-0413-0/24/06
https://doi.org/10.1145/3625549.3658813

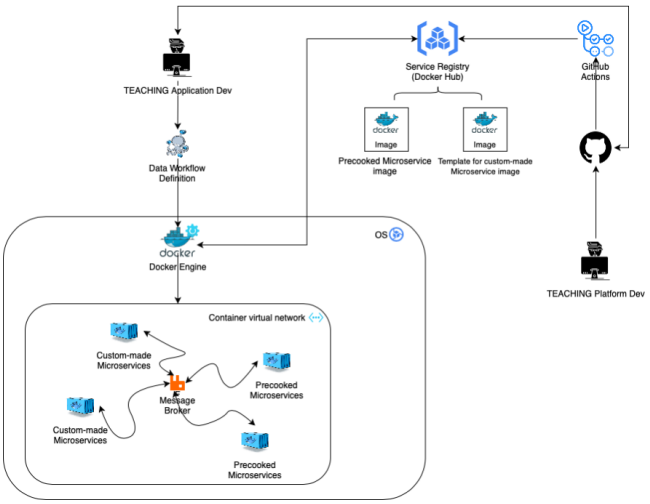

Figure 2: Architecture of the TEACHING Platform.

```yaml
influxdb_logger:
  image: teaching/teaching-data:latest
  container_name: influxdb_logger
  depends_on:
  - rabbitmq
  - influxdb
  environment:
    SERVICE_NAME: influxdb_logger
    SERVICE_TYPE: teaching.data.modules.influxdb.InfluxDBManager
  env_file: .env
```

```yaml
influxdb_logger:
  topics:
  - sensor.chest.value.*
  - prediction.stress.value.*
```

Figure 3: Example of definition of a service which logs sensor and stress data to InfluxDB. The upper figure defines the non-functional configuration of a service within the docker-compose file. The lower figure defines functional configuration of the module, i.e., the parameters of the Python class.

2 TEACHING PLATFORM ARCHITECTURE

The TEACHING Platform architecture follows a microservice-based architecture pattern. Using a message broker allows the dynamic definition of data workflows that may run in any platform that supports a Docker container. The workflows can be deployed across multiple platforms. The key idea is the application developers should be able to use an abstract language to describe the data workflows that they want to deploy, and the system should be able to implement that. In this context, a TEACHING Application is essentially modelled as a data workflow that is meant to close the feedback loop. Such a workflow may contain AI-based components which add further challenges to the task. As depicted in Figure 2, there are two types of end users for the TEACHING Platform: the TEACHING Platform developer and the application developer/provider. The components depicted on the top-right side of the figure are meant to facilitate the needs of the Platform Developer. The Developer manages the code through a GitHub project and an automated CI/CD pipeline builds the images to be deployed for the platform or each application. The Application developer then defines the application model as a data workflow and deploys it in one or more target systems. The Platform design focuses primarily on meeting the requirements of the application developer and encompasses four properties: monitoring, quantification, integration, and programmability. Monitoring and quantification are achieved by introducing data workflows that aggregate human and system monitoring data sources as well as ML models (quantification). Integration is possible through the inclusion of components at the ends of the workflow that can interact with the target AI-based system that is outside of TEACHING's control. Finally, programmability is achieved using architecture patterns that allow the creation of dynamic workflows over abstract declarations provided by the end users, with minimal intervention in their code. Towards that end, several features have been developed. In the following paragraphs, we provide an overview of how each respective feature was implemented, along with the baseline technology employed.

Microservice Architecture. The TEACHING Platform relies on Docker and Docker-compose to support dynamic workflows composed of small, reusable components. These components are managed in their containers, ensuring efficient lifecycle management and isolation for security and resilience. Docker-compose is extended to support the creation of application workflows, with RabbitMQ serving as the message broker for communication between components. This architecture enables the platform to implement abstract declarations provided by end-users, allowing for the creation of dynamic workflows with minimal intervention. The developer can easily define a TEACHING application by declaring built-in services in the docker-compose file and defining custom services by providing an appropriate business logic as in Figure 3.

CI/CD Pipeline. To streamline development and deployment processes, the TEACHING Platform employs a robust CI/CD pipeline based on GitHub Actions. This pipeline facilitates continuous integration and delivery, automating the building, testing, and deployment of platform modules. An easing factor for the CI/CD pipeline is the packaging of the Python library, where each package has its own, dedicated set of dependencies and a corresponding Docker image. This enables independent development and testing by multiple developers concurrently. Docker images are built and published to DockerHub, ensuring compatibility across different CPU architectures.

AI Models Support. The AI-Toolkit [4] within the TEACHING Platform serves as a comprehensive collection of AI modules, providing native support for various AI functionalities required in TEACHING applications. These functionalities include sequence learning, federated learning, stress prediction, reinforcement learning, anomaly detection, and cybersecurity. The toolkit offers flexibility for developers to customize and add their own learning modules, facilitating the development of AI-powered applications tailored to specific use cases and hardware platforms. Furthermore, ensuring cybersecurity in edge-to-cloud communication is paramount within the TEACHING Platform. The Model Transfer Service (MTS) secures federated communications between edge and cloud

by encrypting and decrypting transmitted model weights. MTS is deployed as a containerized microservice on both edge nodes and cloud servers, ensuring end-to-end encryption of model weights during transmission. Implemented with Java and the Spring Framework, MTS guarantees secure communication while maintaining efficiency and reliability. For further details regarding the AI functionalities of the TEACHING platform, we refer the reader to [4].

Stream Processing. Data Stream Processing (DSP) is a fundamental aspect of the TEACHING ecosystem, enabling high-speed processing of streaming data from sensors or distributed data producers. The WindFlow parallel library [5] is employed to define data-flow graphs of operators for intermediate computation, transforming inputs into outputs based on predefined business logic. WindFlow supports efficient execution on embedded devices, leveraging CPU processing capabilities and integrated GPUs to achieve high performance with low latency.

External Services Integration. The TEACHING Platform facilitates seamless integration of external services for data and knowledge aggregation across distributed platforms. This integration optimizes network traffic, privacy, and performance, leveraging resources available in the Cloud-Edge Continuum. By distributing specific tasks across the platform, TEACHING enables enhanced privacy, reduced service latency, and increased performance, catering to diverse application requirements.

2.1 Validation on an Autonomous Driving Use Case

One of the main use cases of the TEACHING project stands in the autonomous driving landscape. In short, the human interacts with the CPSoS (the car) by means of physiological data, the platform processes the physiological data by predicting a stress value and provides a new driving profile that aims to minimize the stress condition of the driver [2, 3].

Use case requirements. In this scenario (depicted in Figure 4), we assume a manufacturer wants to build an application embodying such a use case. The TEACHING platform can seamlessly mediate between the passengers and the AI system, operating on its own isolated environment, and integrating with the AI system's interfaces, performing dependable operations from a list of presets.

The manufacturer can use the TEACHING platform to: (1) monitor and quantify human preferences; (2) learn how to tweak the AI system's performance, creating a closed control loop. To achieve that, the manufacturer needs to implement a TEACHING application that retrieves data from sensors and then uses a deep learning model to quantify the stress that humans are experiencing. This quantified value is then passed on to another model, that considers it along with information about the current environment state to infer the option from the list of presets that will lead to the minimization of passenger stress.

As human passengers may have different tolerance levels, this last model must be adaptive and learn as it goes. As such, we refer to this model as the personalization model, and is trained in a federated fashion to transfer knowledge among multiple instances. To do so, the manufacturer sets up an external service with which the local TEACHING application in every vehicle is communicating over 4G

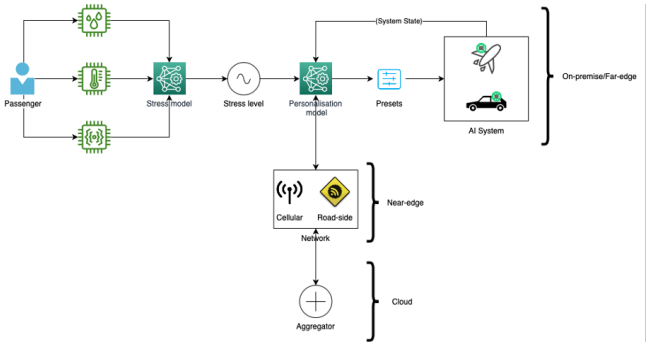

Figure 4: Workflow diagram of the automotive use case scenario.

or 5G, or through a road-side communication infrastructure, over short-range communication channels. These channels allow the model to be transferred during the training phase.

To ensure data protection as they are transmitted over a public link, they are encrypted and decrypted at each end of the communication channel. With this functionality, the TEACHING application is now extended to include both the local on-premises infrastructure, but also the external one, creating a continuum between the cloud and the near and far edge, that is, the roadside and on-premise resources respectively 4.

Simulated conditions. For the hardware board of the TEACHING Platform, we use the iMX8 Quad Max from NXP which embeds two ARM Cortex M4 real-time microcontrollers, four ARM Cortex A53 low power cores, and two Cortex A72 high performance cores. This is complemented by two GPUs making this architecture a viable choice for dealing with both the predictability requirements of dependable systems and the high-performance computing requirements of AI algorithms. We use this exact board and an x86 64 machine to set up two TEACHING application instances. We use CARLA to emulate the driving conditions and the vehicle, as well as an AI system that essentially navigates the vehicle toward its destination. To minimize the passengers' stress, the two instances are switching between three driving profiles that the CARLA AI navigation system is offering. The stress is measured with the help of an array of sensors from Shimmer, that measure the electrodermal activity of the passenger. To simulate the network conditions expected on the road, we use a network conditions emulator.

Development Process. The actions that the application provider (e.g., manufacturer's developers) needs to take are described in Figure 5. First, the developer starts with the selection of the needed microservices in the data workflow. These are the message broker, the sensor API, the stress, and personalization models, the DB, and a component called MTS that encrypts, and decrypts model weights and sends and receives them to the external model aggregation service. Each component will result in a service (as in Figure 3) within the final docker-compose, each being either pulled from a public image on DockerHub (according to the name and the desired architecture), or being built custom through the usage of the templates provided. The overall application consists of a docker-compose file for the client side, and one for the server side.

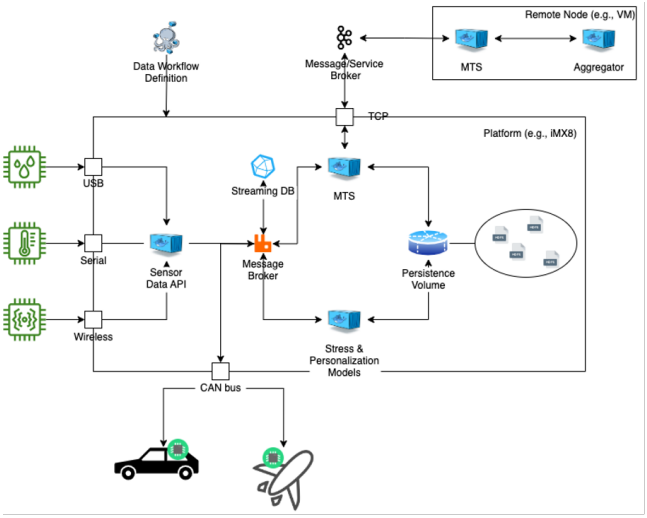

Figure 5: Deployment diagram for the use case.

Starting from the client side, to accommodate the requirement of dynamically creating data workflows, the TEACHING platform uses a message broker based on AMQP, which needs to be instantiated as a service within the docker-compose file. All the component templates include a RabbitMQ client allowing them to publish or subscribe on queues, which are defined via environment variables for flexibility of the workflow definition. Furthermore, each Python class implementing a service also internally exposes a listener interface, that the developer can use to have their code produce or consume data automatically. This definition of a service applies for the Sensor Data API, the Stress and Personalization Models and the MTS. The learning models are trained in a federated fashion, and communicate with the MTS to exchange weights with the server through Kafka. Additionally, to ensure the persistence of the data within the application, the client side includes an InfluxDB instance, which is fed data through an additional Python service which collects the data from a subset of topics and stores it via an InfluxDB client. Then, the workflow is ready to be deployed by simply running the docker-compose input file in the docker engine of each of the two machines.

The server side requires only the deployment of the external aggregation service for the federated learning algorithm, and the server side of the MTS. The deployment happens on a remote, physical machine and includes a version of the MTS and the aggregation service. Note that, like in the case of the local deployment, the developer is using a persistence volume and a fixed path to declare where the model weights will be laid on and picked up by the involved components. This too is defined at the docker-compose level allowing some flexibility in changing it. Once the deployment is over, the application is running, each component within a container.

3 CONCLUSIONS AND FUTURE WORK

The development of an application enabling the Humanistic Intelligence is challenging for reasons spanning from the need for deployment in pervasive environments (characterised by heterogeneous hardware) to the complexity of the interaction between humans and the CPSoS involved.

In this paper, we describe the TEACHING Platform, a framework developed in the context of the project of the same name. The platform addresses the problems described above by means of driving design principles that make applications flexible, allowing easy introduction of modules of any kind (off-the-shelf, custom, AI-driven etc.), robust development thanks to an effective CI/CD pipeline that, despite its agnosticism with respect to the underlying hardware, allows strong integration with them. We have described how a complex application in the field of autonomous driving can be developed simply by using the proposed platform.

In the future, we aim to provide a transpiling mechanism that allows an entire application to be defined using simple Python code. This will be further enhanced through a platform GUI, which will allow an application to be built from the simple graphical definition of the corresponding computational graph.

ACKNOWLEDGMENTS

This research was supported by the TEACHING project, funded by the EU Horizon 2020 research and innovation program under GA n. 871385.

REFERENCES

[1] Davide Bacciu, Siranush Akarmazyan, Eric Armengaud, Manlio Bacco, George Bravos, Calogero Calandra, Emanuele Carlini, Antonio Carta, Pietro Cassarà, Massimo Coppola, Charalampos Davalas, Patrizio Dazzi, Maria Carmela Degennaro, Daniele Di Sarli, Juergen Dobaj, Claudio Gallicchio, Sylvain Girbal, Alberto Gotta, Riccardo Groppo, Vincenzo Lomonaco, Georg Macher, Daniele Mazzei, Gabriele Mencagli, Dimitrios Michail, Alessio Micheli, Roberta Peroglio, Salvatore Petroni, Rosaria Potenza, Farank Pourdanesh, Christos Sardianos, Konstantinos Tserpes, Fulvio Tagliabó, Jakob Valtl, Iraklis Varlamis, and Omar Veledar. 2021. TEACHING - Trustworthy autonomous cyber-physical applications through human-centred intelligence. In *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*. 1–6. https://doi.org/10.1109/COINS51742.2021.9524099

[2] Valerio De Caro, Saira Bano, Achilles Machumilane, Alberto Gotta, Pietro Cassarà, Antonio Carta, Rudy Semola, Christos Sardianos, Christos Chronis, Iraklis Varlamis, Konstantinos Tserpes, Vincenzo Lomonaco, Claudio Gallicchio, and Davide Bacciu. 2022. AI-as-a-Service Toolkit for Human-Centered Intelligence in Autonomous Driving. In *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events (PerCom Workshops): Demos (PerCom Demos 2022)*.

[3] Valerio De Caro, Herbert Danzinger, Claudio Gallicchio, Clemens Könczöl, Vincenzo Lomonaco, Mina Marmpena, Sevasti Politi, Omar Veledar, and Davide Bacciu. 2023. Prediction of Driver's Stress Affection in Simulated Autonomous Driving Scenarios. In *2023 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW)*. 1–5. https://doi.org/10.1109/ICASSPW59220.2023.10193353

[4] Vincenzo Lomonaco, Valerio De Caro, Claudio Gallicchio, Antonio Carta, Christos Sardianos, Iraklis Varlamis, Konstantinos Tserpes, Massimo Coppola, Mina Marmpena, Sevasti Politi, Erwin Schoitsch, and Davide Bacciu. 2023. AI-Toolkit: A Microservices Architecture for Low-Code Decentralized Machine Intelligence. In *2023 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW)*. 1–5. https://doi.org/10.1109/ICASSPW59220.2023.10193222

[5] Gabriele Mencagli, Massimo Torquati, Andrea Cardaci, Alessandra Fais, Luca Rinaldi, and Marco Danelutto. 2021. WindFlow: High-Speed Continuous Stream Processing With Parallel Building Blocks. *IEEE Transactions on Parallel and Distributed Systems* 32, 11 (2021), 2748–2763. https://doi.org/10.1109/TPDS.2021.3073970