This project has received funding from the ECSEL Joint Undertaking under grant agreement No 101007326.
This Joint Undertaking receives support from the European Union's
Horizon 2020 research and innovation programme and the ECSEL member states.

Automotive Intelligence for/at Connected Shared Mobility

Deliverable	***Integrated AI platform demonstrator***		
Involved WPs	WP5	**Deliverable type**	Public
Project	***AI4CSM***	**Grant Agreement Number**	101007326
Deliverable File	D5.16	**Last Modified**	19.06.2024 (m38)
Due Date	04.2024 (m36)	**Actual Submission Date**	19.06.2024 (m38)
Status	Ready for Review	**Version**	1.0
Contact Person	Lorenz Klampfl	**Organisation**	AVL List GmbH
Phone	+43 664 88631658	**E-Mail**	Lorenz.klampfl@avl.com

Document history			
V	**Date**	**Author**	**Description**
0.01	12.04.2024	SINTEF	Template updated.
0.02	30.04.2024	AVL, TUGRAZ, AIT, AIDI	AIDI, TUG, AIT, AVL contributions integrated.
0.03	19.06.2024	TTTAUTO	TTTAUTO contributions integrated.
1.0	19.06.2024	AVL	Final version.

Disclaimer

The opinion stated in this document reflects the opinion of the authors and not the opinion of the European Commission. The Agency is not responsible for any use that may be made of the information contained (Art. 29.5 of the GA).

Table of contents

1 Executive/ Publishable summary

This document is intended to provide a short textual description complementary to the main contributions of this deliverable: demonstrator videos showcasing the preliminary results of the integration of the developed methods and components within SC7 and task T5.7 – "*Integration of AI and Virtualization Platform".* A comprehensive report describing the integration of the software platforms and methods developed will be provided in the scope of deliverable D5.17 in month M42.

In general, the aim of SC7 is to leverage AI for digital twins, learning, scene interpretation and manipulation, data analysis, and edge computing to advance the current boundaries and reduce the workload needed to develop, evaluate, and improve automated driving functionalities to ensure reliable and trustworthy systems.

The overall objective of task T5.7 focuses on the integration of the developed methods, tools, and processes based on the overall system level design established within WP2. The outcomes of this task will be facilitated in the final project phase within WP6, Validation and Testing. Within SC7, three main demonstrators will be utilized to showcase the successful development, integration and testing of advanced methods, tools, and processes for simulation and virtualization:

- SCD 7.1: Enriched virtual models based on standardized real-world data (lead: AVL)
- SCD 7.2: Virtualized time and latency critical AI processes on the in-car computing platform (lead: TTTech)
- SCD 7.3: AI based simulation and virtualization for multimodal mobility for virtual Smart Cities (lead: AIDI)

The following procedure was established to collect and compile activities performed in T5.7:

- Partners concentrated on their individual work package tasks and contributed to D5.16 by describing the work performed in T5.7.
- Each demonstrator lead collects and compiles the contributions from demonstrator partners.
- Collected inputs are evaluated and reworked based on discussions between the demonstrator partners in regular alignment meetings.

In the following chapters a short description complementary to the demonstrator videos is given.

2 Non publishable information

N.A. (This deliverable has dissemination level "Public").

3 Introduction & Scope

3.1 Purpose and target group

The AI4CSM project focuses on advancing sophisticated electronic components and systems (ECS) and architectures for upcoming mass-market ECAS (electric, connected, autonomous, shared) vehicles. This initiative drives digital transformation in the automotive industry by supporting mobility trends and accelerating the transition toward a sustainable ecosystem.

SC7 serves as a pivotal enabler for AI methods, playing a critical role in facilitating AI techniques, tools, and procedures. Its purpose is to ensure AI accountability, availability, collaboration, explainability, fairness, inclusivity, reliability, resilience, safety, security, trustworthiness, and transparency.

With this, SC7 leverages AI to expand the boundaries of digital twins, learning, scene interpretation and manipulation, data analysis, and edge computing. Its dual purpose involves streamlining programming efforts for automating and testing autonomous vehicles through visualization and simulation techniques. Additionally, SC7 utilizes genuine and digitally enhanced real-world data to enhance software-oriented training for automated driving functions.

To accomplish the goals, activities in SC7 include:

- Development and integration of interoperability as well as standardization between the different systems
- Development and integration of virtualization tools for AI processing at the edge
- Development and integration of AI supported real world data virtualization (digital twins)

To address the overall objective of SC7, the developed methods, tools, and processes are integrated into different environments to further on test and validate their functionality. T5.7 deals especially with the integration topic and provides the foundation for the upcoming validation and testing phase in WP6. As mentioned earlier, this document should serve as a complementary short description of the demonstrators showcasing the preliminary integration results. A detailed report will be provided within D5.17 in M42.

All demonstrator videos can be found on the project share: https://ai4csm-cloud.automotive.oth-aw.de/f/109852

3.2 Contributions of partners

Table 1 lists the partners participating in SC7 and briefly explains their activities and contributions in the various demonstrators and chapters.

TABLE 1: PARTNER CONTRIBUTIONS

Chapter	Partner	Contribution
1,2,3,4, 4.1, 4.1.1, 7	AVL	General description as well as description of AVL part, Data Collection & Virtualization, within SCD7.1.
4.1.2	AIT	Description of AIT part, AI-based controller learning, within SCD7.1.
4.1.3	TUGRAZ	Description of TUGRAZ part, controller validation, within SCD7.1.
5	TTTAUTO	Description of the prototypical demonstrator setup of SCD7.2.
6	AIDI	Description of the demonstrator SCD7.3.

3.3 Relation to other activities in the project

As a technology provider, SC7 is responsible for developing components, subcomponents, and software concepts and functions related to AI-based methods, simulation, and virtualization. The goal is to pave the way toward reliable systems and connected services. In addition to the demonstrators planned within SC7, the developed functionalities and methods will also be integrated

into the output enabler supply chain, SC1. To ensure a seamless development process and integration, close collaboration with the related supply chain was established from the early beginning of the project. Furthermore, SC7 engages in a bidirectional exchange with the value enabler supply chain, SC8, addressing Green Deal principles, standardization initiatives, certification, and ethical considerations to enhance the impact and exploitation of the developed solutions.

4 Prototype Integration Demonstrator SCD 7.1 – Enriched virtual models based on standardized real-world data (AVL, AIT, TUGRAZ)

4.1 Demonstrator prototype description

In Figure 1, the high-level structure and architecture of demonstrator, SCD7.1 is depicted. It comprises three main components: data collection and virtualization, AI-based controller learning, and controller validation. While parts of the demonstrator are developed independently and therefore can be utilized seperatly in different use cases, regular bi-weekly meetings enable also a complementary application within a complete toolchain or use case.

The first part of SCD7.1 focuses on real-world data and the implementation and integration of a data center, ensuring comprehensive signal collection for subsequent processing and conversion into a virtualized scenario based on standardized formats. In the second part, AIT utilizes virtual scenarios for AI-based controller learning. TUGRAZ contributes to controller validation, integrating monitoring systems to evaluate AI-based controller performance.

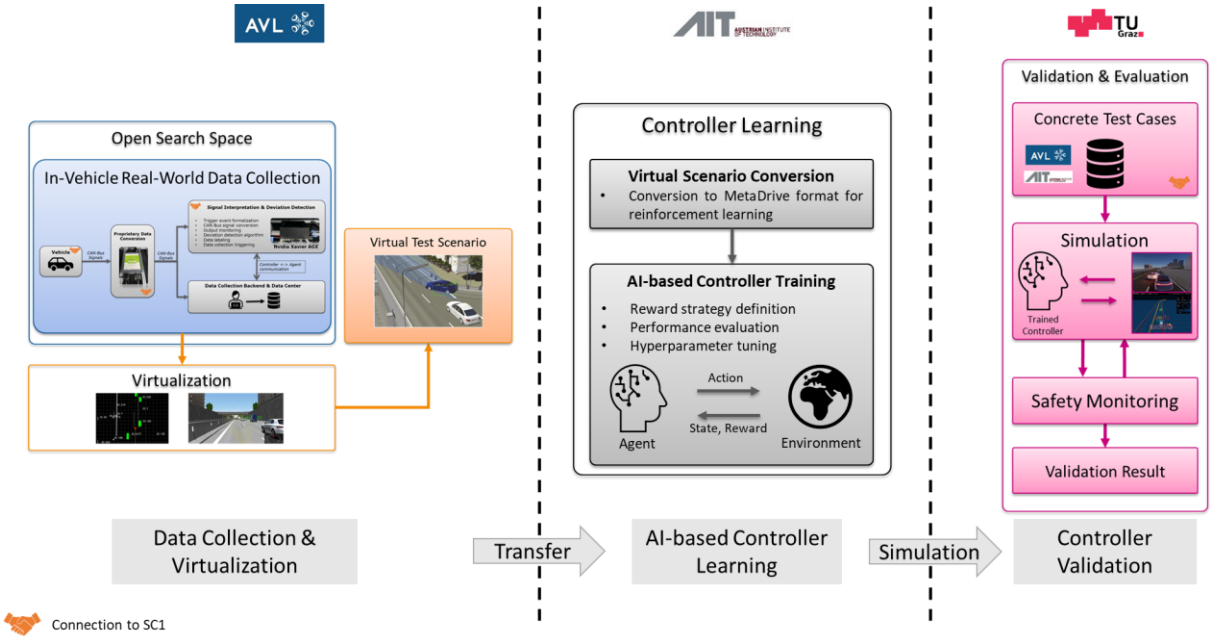

FIGURE 1 HIGH-LEVEL ARCHITECTURE OF DEMONSTRATOR SCD7.1

In the following, a short description of the work performed within T5.7 is given for each component. It should be noted that this should not serve as a detailed description, which will be included within D5.17, rather it should give some context to the demonstrator videos provided in the project cloud: https://ai4csm-cloud.automotive.oth-aw.de/f/109852

4.1.1 Data Collection & Virtualization (sub-demonstrator 1)

Data collection and virtualization constitute the initial phase of demonstrator SCD7.1. The overarching goal is to achieve efficiency in virtualizing real-world data, bridging the gap between data collection and enriched virtual models and scenarios. These virtual models and scenarios can be utilized for investigating potential ADAS/AD faults and for the development of future ADAS/AD systems.

4.1.1.1 High-level architecture of the demonstrator/sub-demonstrator prototype

Figure 2 illustrates the high-level architecture utilized for data collection and subsequent virtualization. Note that the data collection component is closely linked to the methods and topics addressed in SC1 and demonstrator SCD1.1. It stores only significant real-world data where deviations are detected but can collect all test drive data if necessary.

For virtualization, real-world data from the vehicle's CAN bus is converted into a readable format using a DBC file. This data is then translated into the Open Simulation Interface (OSI) format to ensure compatibility with various simulation tools. Pre-processing steps, such as sampling, coordinate alignment, and GPS improvement, are applied before converting the data into OpenSCENARIO and OpenDRIVE formats. These formats describe dynamic scenarios and road networks, respectively, and are integrated into the simulation environment. This process enables rigorous testing and validation of autonomous driving functions, ensuring robust performance in diverse scenarios.

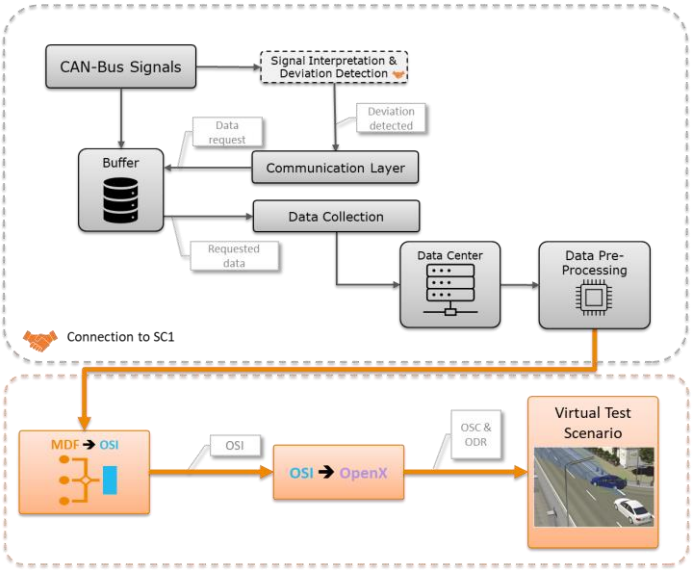

FIGURE 2 HIGH-LEVEL ARCHITECTURE OF SUB-DEMONSTRATOR DEALING WITH DATA COLLECTION AND VIRTUALIZATION

4.1.1.2 HW and SW components used in the demonstrator/sub-demonstrator prototype

Figure 3 illustrates the workflow of the software components developed for both offline and online virtualization on the dedicated hardware platform The conversion from real-world data to a virtualized scenario involves several steps. These steps begin with selecting the data snippet that captures the driving scenario to be virtualized, generating the road network, and including the detected objects along with their driving maneuvers performed during data capture. This figure demonstrates the results on a local machine; however, the next section provides an overview of how these methodologies were adapted and transferred to the dedicated hardware platform. A detailed description of the process will be provided in D5.17.

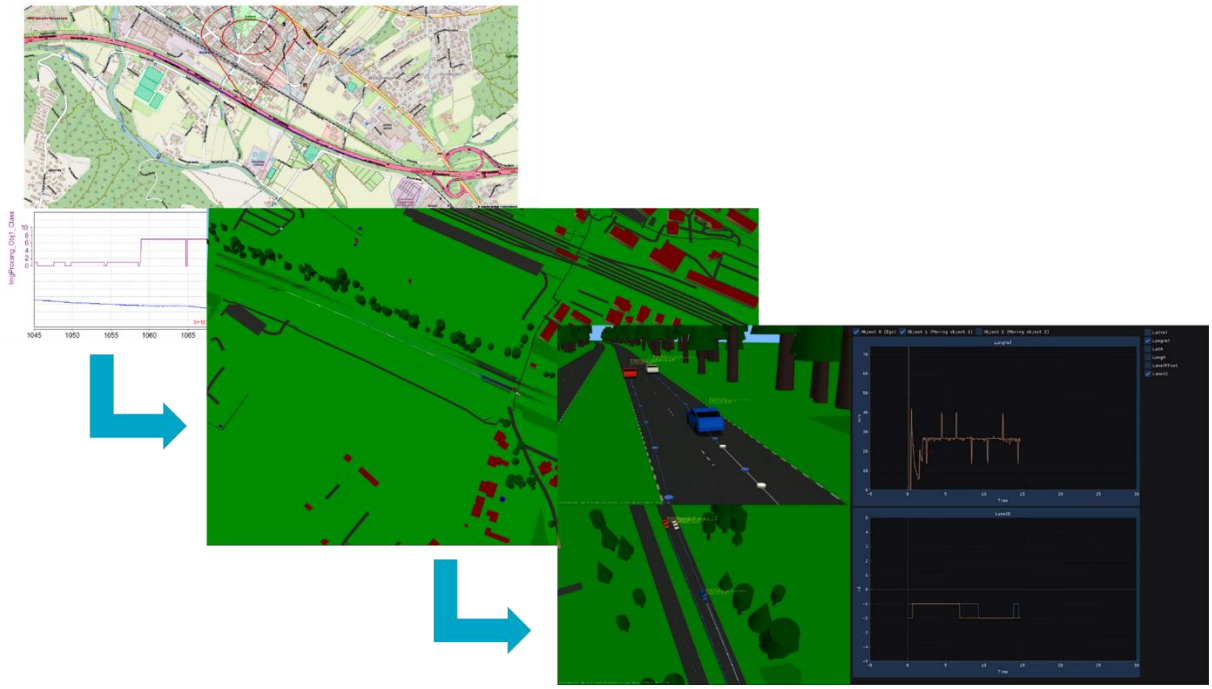

FIGURE 3 EXAMPLE RESULTS SHOWCASING THE OFFLINE VIRTUALIZATION BASED ON COLLECTED REAL-WORLD DATA.

Figure 4 shows the in-vehicle data collection infrastructure along with example real-world data that was collected. The main components used for real-world data collection and virtualization within the vehicle are:

- **Hardware:**
  - *FlexDevice-S:* Converts proprietary raw CAN signals into a readable format.
  - *Nvidia Xavier AGX:* Executes the developed algorithms, handles raw CAN signal interpretations, facilitates communication between the platform and the data collection backend, and runs the virtualization algorithms.
- **Software:**
  - *ESMINI:* Used to display the virtualized scenario, run on a conventional workstation or the Nvidia Xavier AGX.
  - *Developed Algorithms:* Implemented in Python, these algorithms are tailored for efficient virtualization of real-world data and adapted for deployment on the Nvidia platform.

A detailed description of the integration and deployment activities will be provided in the respective WP5 deliverable D5.17. This deliverable (D5.16) is intended to serve as an overview of the provided demonstrator videos.

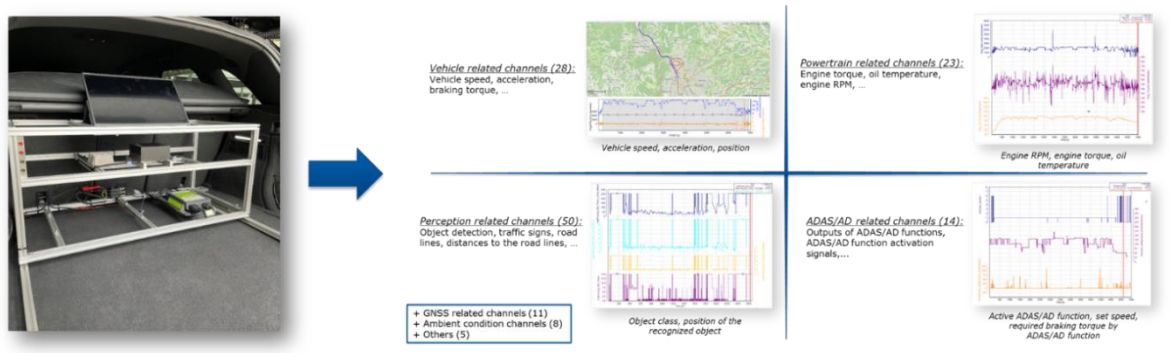

FIGURE 4 IN-VEHICLE INFRASTRUCTURE THAT IS USED FOR DATA COLLECTION AND ONLINE VIRTUALIZATION (LEFT) AND EXAMPLE DATA COLLECTED WITH THE INTEGRATED COMPONENTS.

4.1.1.3 Integration and Deployment onto the hardware platform

To deploy, integrate, and run our developed methodologies on the Nvidia board, it was necessary to optimize the algorithms for efficiency and build a dedicated Docker container including our algorithms for the arm64 architecture used by the hardware platform. Additionally, several tools had to be modified to work with the hardware platform's architecture, such as Protobuf compilers and Esmini libraries for running the generated scenarios. Figure 5 provides an overview of the workflow followed when a virtualization request is triggered. As shown, the process starts with the collected real-world data in MF4 format. The connection to the hardware platform is then established via an SSH tunnel, initiating the virtualization work. After successfully creating the virtualized scenario, the generated OpenSCENARIO and OpenDRIVE files can be executed in any simulator that supports these standards.

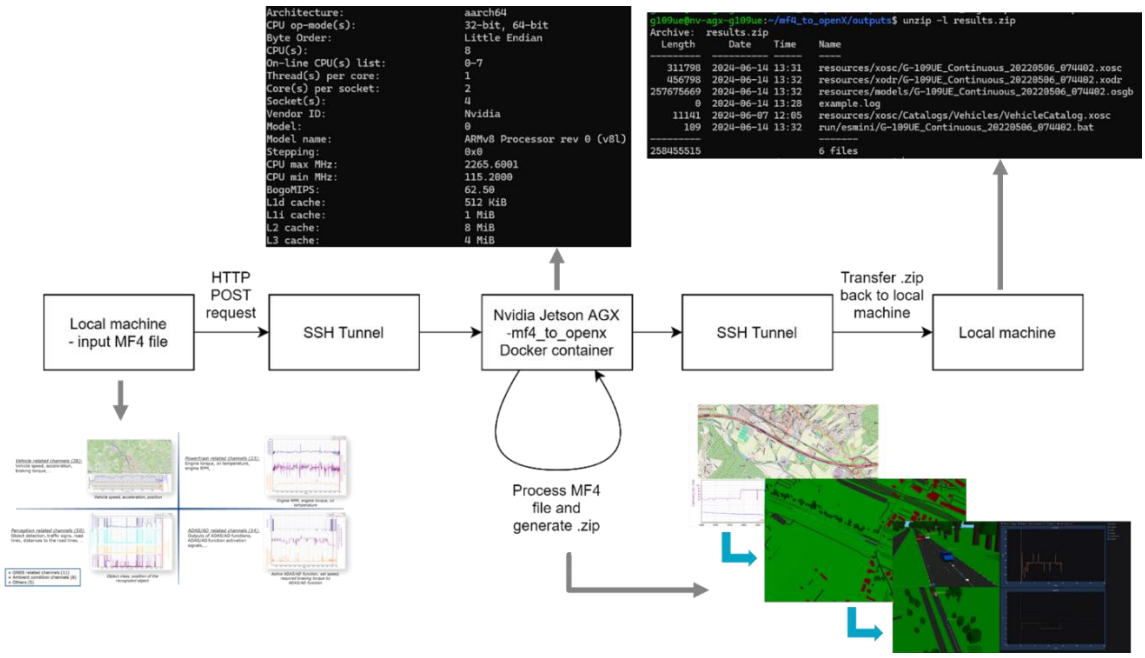

FIGURE 5 HIGH-LEVEL OVERVIEW SHOWING THE WORKFLOW OF VIRTUALIZATION WHEN DEPLOYED TO THE HARDWARE PLATFORM.

Figure 6 and Figure 7 show screenshots of the results obtained from the virtualization algorithms. Both figures illustrate the triggering of real-world data virtualization on the dedicated hardware. We implemented different methods for initiating this virtualization process. First, the transformation of real-world data into a virtual scenario—capturing the trajectories of the demonstrator vehicle and other traffic participants—can be triggered via a graphical user interface using Postman. Second, it can be directly initiated through the command line.

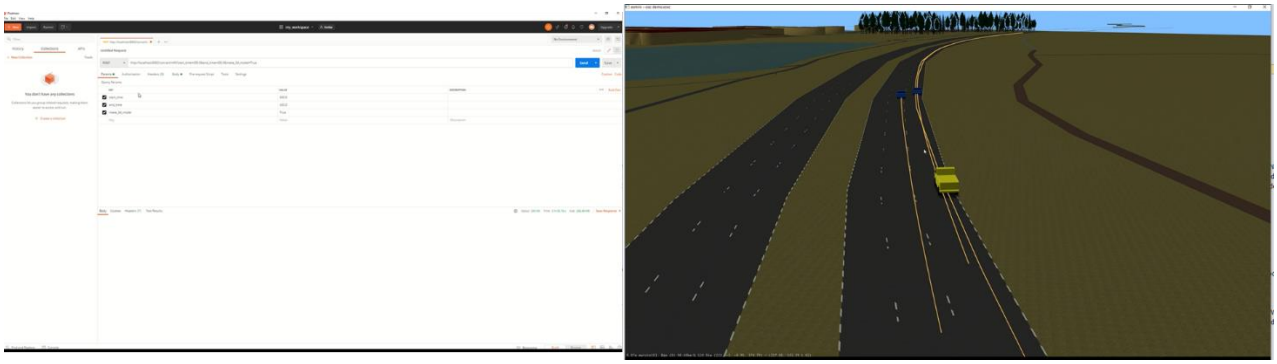

FIGURE 6 SCREENSHOT SHOWING THE RESULTS WHEN TRIGGERING THE VIRTUALIZATION ON THE HARDWARE PLATFORM VIA POSTMAN.

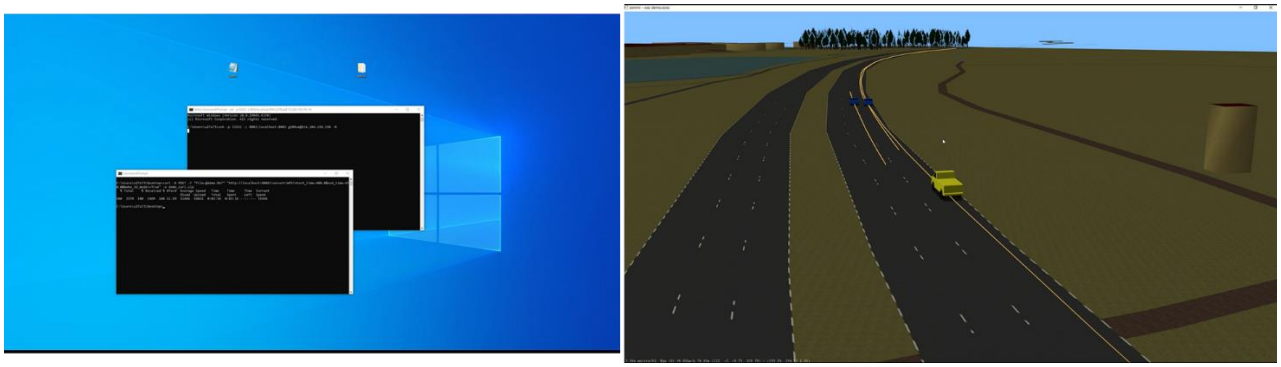

FIGURE 7 SCREENSHOT SHOWING THE RESULTS WHEN TRIGGERING THE VIRTUALIZATION ON THE HARDWARE PLATFORM VIA "CURL" (COMMAND LINE).

In addition, Figure 8 presents preliminary results of our efforts to make the virtualization process more user-friendly. The figure showcases the initial outcomes of a web application designed to virtualize real-world data.

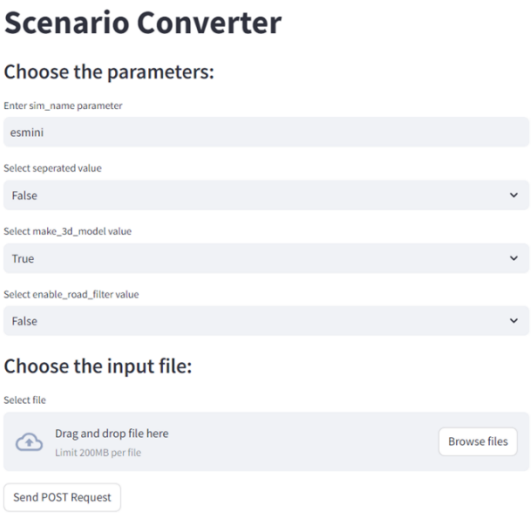

FIGURE 8 OUTLOOK ON THE BUILD WEB APPLICATION FOR VIRTUALIZATION.

4.1.1.4 Main achievements of the demonstrator/sub-demonstrator prototype integration.

In the scope of demonstrator SCD7.1, specifically focusing on data collection and virtualization part, we successfully integrated our developed algorithms into the respective environments. This integration involved coupling our methods with dedicated simulators by adhering to standards such as OpenSCENARIO and OpenDRIVE. Additionally, we successfully deployed, integrated, and built our software solutions onto a hardware platform integrated in a demonstrator vehicle, enabling us to virtualize scenarios during operation. A comprehensive description of the integration work will be provided in the connected deliverable D5.17, scheduled for month M42.

4.1.2 AI-based Controller Learning (sub-demonstrator 2)

The following paragraphs briefly describes the approach that we implemented as first step towards enabling transfer learning of autonomous driving agents. The description complements the attached video showcasing the approach in action. The video and this document are part of deliverable D5.16: Integrated AI platform demonstrator. The work presented in the video only showing the preliminary results, which was announced to the consortium, and is since AIT had a longer delay in hiring a replacement for a colleague who left the project. Final results will be reported in M42 in D5.17: Report in integration of AI platform.

4.1.2.1 Background

We assume that there exist two driving simulation environments in which the same autonomous driving agent should be able to operate safely. The SOURCE environment is used to train a driving agent. We assume that the agent is trained using Reinforcement Learning (RL); therefore, we assume the SOURCE environment enable fast simulation execution. The TARGET environment, instead, is used to test whether and to what extend the policy learnt by the agent in the SOURCE environment generalizes, i.e., test whether the same agent can safely drive in the TARGET environment. We assume that the TARGET environment shares the basic concepts with the SOURCE environment (e.g., road

geometry, basic physics); however, the level of details and accuracy is different. Specifically, the photo-realism and physics simulation in the TARGET are higher than the SOURCE environment.

In our experiments, we use the MetaDrive simulator as SOURCE environment and the BeamNG.tech simulator as TARGET environment. MetaDrive is designed with scalability in mind to address the main requirements of RL in autonomous driving; therefore, this simulator provides fast simulation execution that allows to train RL agents quicker than other existing driving simulators. BeamNG.tech is a photorealistic and physically accurate (soft-body physics) driving simulators. The simulator is designed for replicating advanced physics concepts relevant in the automotive domain, like bending/fractures, tear and wear of components, load distribution and inertia, and more. Consequently, BeamNG.tech simulations are more demanding than those implemented by MetaDrive.

4.1.2.2 Goal

MetaDrive trades efficiency off accuracy and photorealism; thus, it is not clear whether an RL driving agent trained using this simulator will safely work in more realistic environments. Conversely, training the RL agent directly in more realistic environments has prohibitive costs.

Our goal is to understand what the gaps between two driving simulation environments are, whether the agent's policy can cope with them, and how those gaps could be filled, thus enabling the agent to safely drive in both simulators.

As first step towards achieving this goal, we designed a pipeline for conducting **offline experiments** aimed at (1) quantifying the **observational gap** between the two simulations environments, i.e., the difference in the sensory inputs generated by the SOURCE and TARGET environments; and (2) assessing how this difference affects the agent's driving, i.e., how the control actions changes when the agent perception comes from the TARGET environment instead of the one used for training it.

We showcase the pipeline is the attached video. The following description completements and the video follows the same content organization for the sake of understanding.

4.1.2.3 Pipeline Overview

The pipeline consists of the following steps:

Step 0: Setup of the experiment. This preliminary step configures the environment. It specifies the (procedurally generated) road that will be used to collect observations, how many observations to collect, and which RL driving agent will be used to compute driving actions.

Step 1: Collect observations from the SOURCE environment. We execute the PPOExpertPolicy in MetaDrive and collect the observations at regular intervals. The pipeline stores the RGB images and the state data collected at each simulation step (frame) on the disk.

Step 2: Visualize placement and orientation of the observations. After executing the MetaDrive simulation, the pipeline collected the required observations. We visualize the placement (dots) and orientation (arrow) of the vehicle at the execution step the pipeline collected each observation in a plot. The plot shows the bird view of the roads, and the annotations/labels identify the order of the of observations.

Step 3: Collect images from the TARGET environment. The pipeline uses geometrical and semantic data about the generated road to recreate a similar scenario in BeamNG.tech. Noticeably, the recreated road has the same geometry and semantic (e.g., number of lanes, lane markings) of the original one;

however, because the two simulators use different materials and textures the resulting simulations look different. At this point, to collect images from BeamNG.tech the pipeline does not literally drive the vehicle following the same trajectory. Therefore, the pipeline uses the state observations collected in Step #1 to teleport the vehicle in the expected position before taking the picture from the vehicle's frontal camera. As before, the pipeline stores the collected images in a folder to enable manual inspection.

Step 4 and 5: Predict RL agents actions using images from SOURCE and TARGET environments. At this point, we have collected a set of state observations and two sets of images. The pipeline uses these data to compute the RL driving agent control actions, i.e., steering and throttle, when fed with state data and MetaDrive images and with the (same) state data but BeamNG.tech image. The pipeline enables to directly compare the control actions of the RL driving agent and report any difference between by the two simulators (i.e., the observational gap mentioned above).

Step 6: Metrics and Plotting. Finally, the pipeline computes the differences of input images and output control actions and plot them. Specifically, we investigate the following metrics to compute image difference: Pixel MAE, which is the mean absolute difference of the (colored) pixel in the two images, and SSIM, which is the standard Structural Similarity score computed over the gray scale version of the two images. For the output metrics, instead, we compute the absolute difference of steering and throttle.

4.1.2.4 Preliminary Observations and Achievements

The plot generated in Step #6 shows the distribution of differences for each pair of input (x-axis) and output (y-axis) metrics. Using this plot, we can observe that data are somewhat distributed around the diagonal, indicating that smaller differences in the input (bottom left corner) correspond to smaller discrepancies in the output, whereas larger differences in the inputs (top right corner) correspond to larger discrepancies in the output. This result follows our intuition that when the images from the two simulators are too different from each other, the driving agent might not produce the same control actions. However, we also observe that the difference in steering and control seems negligible (in the order of 1e-6), suggesting that (at least on this road) the RL driving agent should be able to drive in both simulators comparably.

4.1.3 Controller Validation (sub-demonstrator 3)

TUG's contribution includes the integration of a virtual autonomous vehicle simulator with our monitoring system. This integration will be achieved by connecting the two systems through an interface that transmits sensor data from the simulator to the monitoring system.

4.1.3.1 High-level architecture of the demonstrator/sub-demonstrator prototype

Our framework integrates an autonomous vehicle simulation environment with our monitoring system. This integration is achieved through the implementation of an interface that facilitates seamless communication between the simulator and the monitoring system. This interface enables the real-time transmission and evaluation of sensor data, thereby ensuring continuous and accurate monitoring of the simulated autonomous vehicle's performance.

4.1.3.2 HW and SW components used in the demonstrator/sub-demonstrator prototype

We use Esmini as the simulator of choice. An interface is implemented in python using the Esmini API to connect Esmini with the monitoring system. Simulation data is achieved in real time. Knowledge-

based rules are implemented in the monitoring system using ASP via Clingo, in order to evaluate and validate the simulation in real time. Scenarios run in Esmini are written in OPENScenario.

In addition, we introduce sensor fusion to our framework. The sensor fusion algorithm will make use of transmitting the same type of data from different sensors of the simulation vehicle. This data will be used as input for the Kalman filter algorithm. The result of Kalman filter as well as the data will be transmitted to the monitoring system. These results will undergo sensor fusion rule checks.

4.1.3.3 Main achievements of the demonstrator/sub-demonstrator prototype integration.

The achievements of our framework can be seen in the video we provided. In the video, we show a test case of an autonomous car driving in the highway. The default speed is determined by recognizing the type of the road (rural in this case). Based on the road type, the maximum speed is 100 km/h. When the car surpasses this speed, the monitoring system outputs UNSATISFIABLE. When the speed is under the limit, it outputs SATISFIABLE. Later, a speed sign of 80 km/h comes up. The monitoring system recognizes the sign and outputs UNSATISFIABLE when the car is speeding more than the sign. When the car goes under 80 km/h, output goes back to SATISFIABLE.

This test case showcases the implementation and functionality of knowledge-based rules in the monitoring system and real time evaluation and validation of the simulation.

5 Prototype integration Demonstrator SCD 7.2 - Virtualized time and latency critical AI processes on the in-car computing platform (TTTAUTO)

5.1 Demonstrator prototype description

Virtualization of hardware adds an abstraction layer and complexity to the overall system to orchestrate and manage the shared resource among different applications. On the other hand, such a capability provides an environment where the AI model is fully incapsulated with all the parameters that are needed. From the client's applications come inference requests and relevant data that should be used to run the current model on the connected AI/ML processor. The demonstrator is composed of four AI/ML workloads which are executed on an automotive ECU with hardware acceleration support. The developed middleware extension is intended to abstract the complexity and enable pre-defined task execution capabilities for the end-system.

The following paragraphs explain the architectural extensions as well as the software components and modules utilized to setup the demonstration and test environment.

5.1.1 High-level architecture of the demonstrator/sub-demonstrator prototype

Based on the subsystem design considerations in previous WPs the derived middleware architecture was extended as shown in Figure 9. Apart from adaptations in the general SW abstraction layer (shown in the middle in blue), two main layers have been visualized to reflect the additional capabilities of the SW environment. Firstly, the accelerator specific execution framework (in this particular case OpenCL/CUDA) in combination with the accelerator target on the automotive ECU. Secondly, the ML Library Abstraction layer (right middle section in dark-blue), which provides the services and methods to schedule, orchestrate AI/ML computational resources and manages the execution within the

desired requirement boundaries. The before mentioned tasks are represented as green boxes denoted as AI enabled applications on top of the figure.

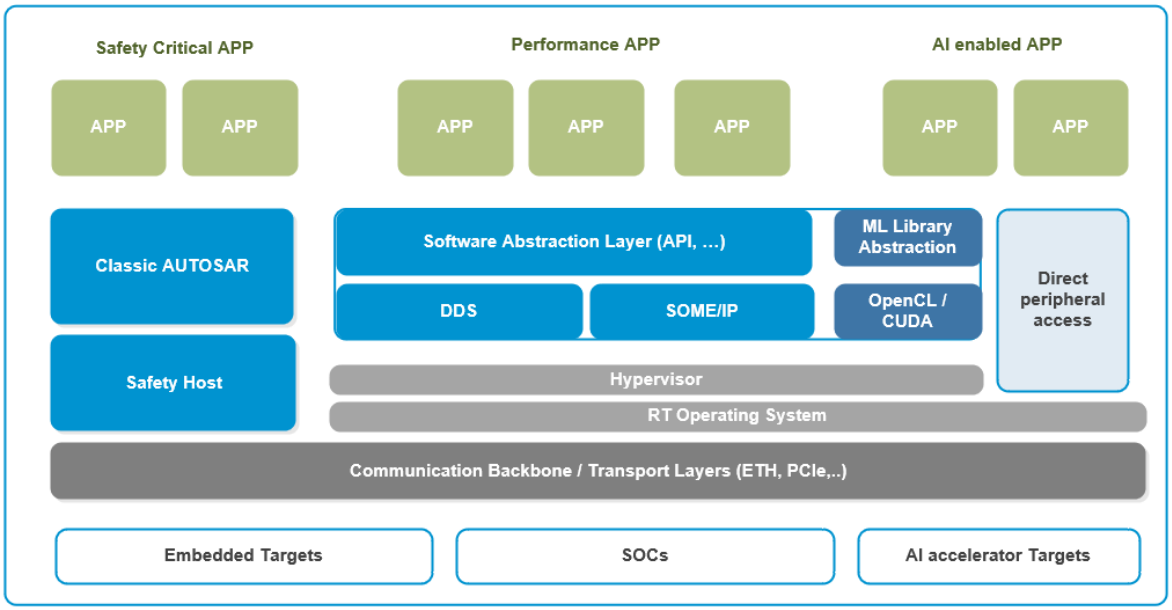

FIGURE 9 PROPOSED HIGH-LEVEL ARCHITECTURE OF EMBEDDING AI ENABLED APPLICATIONS INTO THE AUTOMOTIVE SOFTWARE ARCHITECTURE

5.1.2 HW and SW components used in the demonstrator/sub-demonstrator prototype

The target HW platform of the demonstrator is an automotive ECU with hardware acceleration. The platform has multiple cores on the embedded board which are safety and performance optimized. Additionally, a NVIDIA Jetson AGX Xavier (SoM) integrated on the ECU and provides the computational resources to execute DNN tasks on an embedded target.

The relevant software components and their involvement in the envisaged solution is depicted in Figure 10. The figure blends the application perspective with the relevant layer of the middleware framework. The example applications A and B request multiple AP/ML resources for completing various task over the lifetime of the application. Each task utilizes machine learning models and/or DLNN. The virtualization environment provides an abstraction that ensures secure execution environments, hardware consolidation and resources partitioning as well as the global scheduling requirements of the entire system (including safety and performance applications).

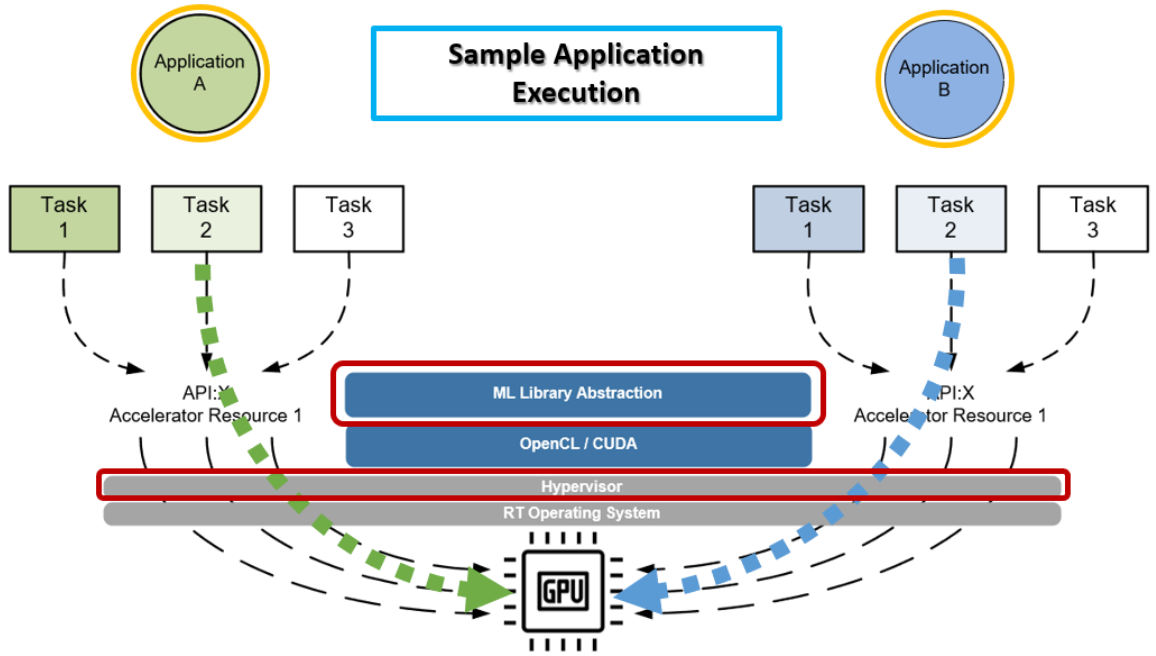

FIGURE 10 SIMPLIFIED VIEW OF SHARED AI/ML RESOURCE ON MULTI-ECU EMBEDDED TARGETS

The demonstrator- and test setup is intended to evaluate the required system properties. These activities are currently performed in WP6 activities. The properties can be categorized into:

- *Spatial independence:* avoiding wrong instructions and/or data transfer to the AI/ML resources.
- *Temporal independence*: avoid temporal behaviour failure in the execution of the applications and tasks from the perspective of the global scheduling context.
- *Safe inter-partition communication*: preventing communication interference among multiple hosts and processes on the same SoC.

5.1.3 AI-enabled systems used in the demonstrator/sub-demonstrator prototype

The demonstrator is setup to perform four individual AI/ML application or tasks, which are performed on the hardware accelerator target (NVIDIA Jetson Xavier). These applications have been selected for simplicity reason, since the integration efforts are limited, and the demonstration purpose is to showcase the SW middleware functionality. The applications are as follows:

1. A semantic segmentation workload has been selected and the application is able to distinguish between (i) pedestrian, (ii) vehicles, (iii) bicycles, (iv) roads. The segmentation works pixel wise and outputs the result again as image.
2. A simple crash estimation application based on simulated sensory inputs serves as second workload in the experiment.
3. An image classification task based on the GoogLeNet deep convolutional neural network has been selected to extend the ML workloads.
4. The fourth task replicates the image classification application.

The four applications are running on the target platform and depending on the overall system schedule they are sending service requests for AI accelerator resources to a dedicated scheduling algorithm denoted as kernel manager. This component handles data flow and scheduling of the software services from each for the four tasks in the testing setup. This is illustrated in Figure 10 with two applications in a simplified view, where the kernel manager can be located in the ML library abstraction layer.

For illustration purposes, the results of three applications are visualized together with a graphical representation of the task execution pipeline. Figure 11 summarizes the task execution as follows:

- Upper left section (1) in the figure visualizes the result of the semantic segmentation task with the four classes mentioned previously.
- The lower left (2) and right (3) depict the classification result of task three and four with the input image under investigation.
- The top right section (4) in the figure represents a graphical visualization of the execution order showing individual threads of the tasks in the execution pipeline. Each box in the grid depicts a processing timeframe of 100ms or less.

The sematic segmentation is sending requests every 500ms and the image classification tasks are sending service requests every 200ms to the Kernel Manager.

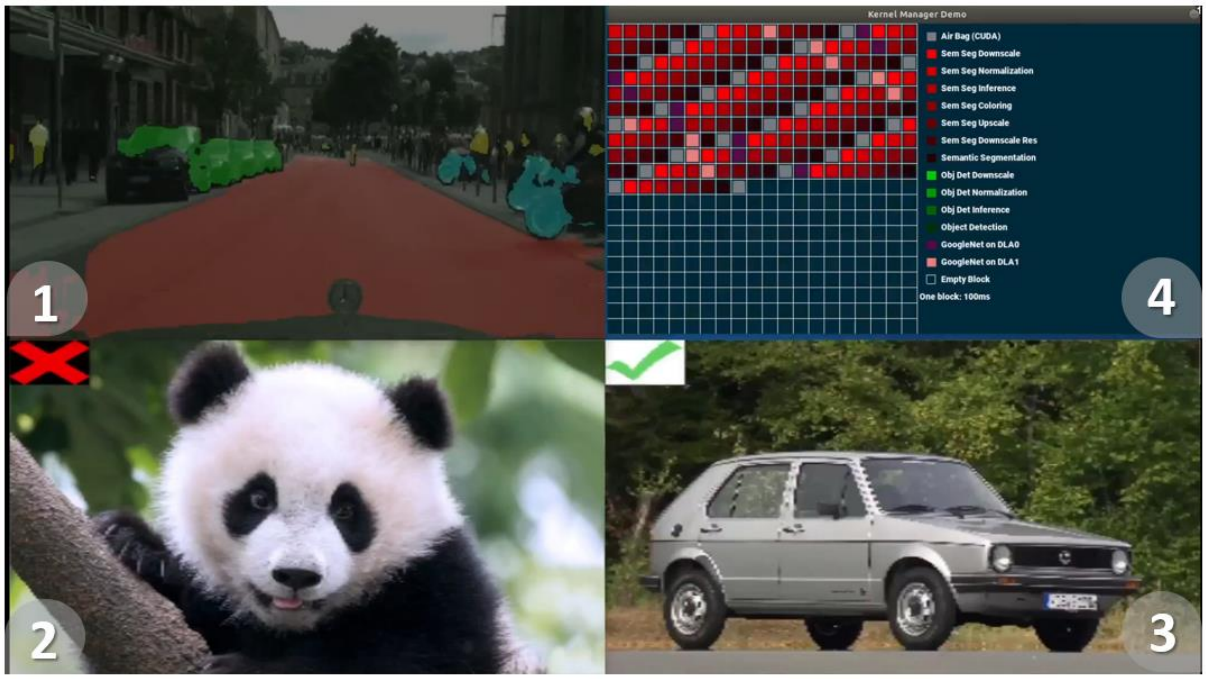

FIGURE 11 VISUALIZATION OF THE DEMONSTRATOR EXECUTING AI/ML TASKS

5.1.4 Main achievements of the demonstrator/sub-demonstrator prototype integration.

The demonstrator presents preliminary results from implementing the concepts and methods in WP4 (Task 4.7). It is planned to evaluate and analyze the defined KPIs based on a sample AI/ML workflow in a laboratory setting to be able to focus on the SW abstraction layer (Task 6.5). The setup currently demonstrates the virtualization of a dedicated AI processes on the in-car computing platform utilizing the onboard hardware accelerator.

The following milestones and achievements have been reached within the 3rd year of the project:

- Lab setup of the demonstrator for HIL deployment and testing.
- Example AI/ML workload available for testing the design and respective developments.
- Prototypical implementation ready for integration and verification in WP5, WP6.

Until the end of the project duration the demonstrator setup will be used for further testing and evaluation of the predefined KPIs.

6 Prototype Integration Demonstrator SCD 7.3 – AI based simulation and virtualization for multimodal mobility for virtual Smart Cities (AIDI)

6.1 Demonstrator prototype description

While driving, motor vehicles monitor, among other things, all systems that influence exhaust emissions, as well as various other control units whose information is recorded and stored via the on-board diagnostics (OBD). Errors are indicated to the driver by indicator lights, and these errors can later be queried and evaluated via the standardised OBD interface. The current standardised system is called OBD-2 or OBD II and access to vehicle diagnostics via OBD-2 is ensured by a 16-pin OBD-2 diagnostic socket (CARB socket or Diagnostic Link Connector (DLC)) in the vehicle. This diagnostic socket usually not only supplies data for the manufacturer-independent, emissions-relevant OBD-2 diagnostic protocol, but also for manufacturer-specific diagnostic protocols. The K-line or CAN bus is used as the physical interface. A Controller Area Network (CAN) bus is a communication protocol that is used in electronic systems and devices to communicate with each other.

6.1.1 Onboard Diagnostic Data Collection - Electrical Vehicle

6.1.1.1 High-level architecture of the demonstrator/sub-demonstrator prototype

A commercially available OBD2 diagnostic tool was also connected to a CAN splitter to record the usual OBD-2 information. This transmits data to an Android app via Bluetooth. To do this, an Android device must be connected to the OBD2 diagnostic tool before the journey begins. This Android device must be active during the entire journey so that the data can be recorded. An app based on the free EDIABAS library is used to read out the OBD2 data. The measured values recorded during the journey are then exported from the app as log files and transferred to the server. This process is currently carried out using a minitool. All data recorded by both the CAN logger and the OBD2 diagnostic device is collected on a server with a database. The data from the CAN logger is transferred here via WLAN directly from the research vehicle and requires no further processing or preparation of the data communication.

6.1.1.2 HW and SW components used in the demonstrator/sub-demonstrator prototype

A logging device called CANedge2 from CSS was used as the CAN logger for recording the vehicle data. In its basic configuration, the CAN logger only requires the main device, a power supply and DB9/OBD2 cables to the vehicle. The main device was permanently installed in the research vehicle; the power supply of the device was ensured by the power supply of the vehicle via DB9/OBD2 cable. In addition, the data logger was extended with a GPS-to-CAN module and a WiFi router.

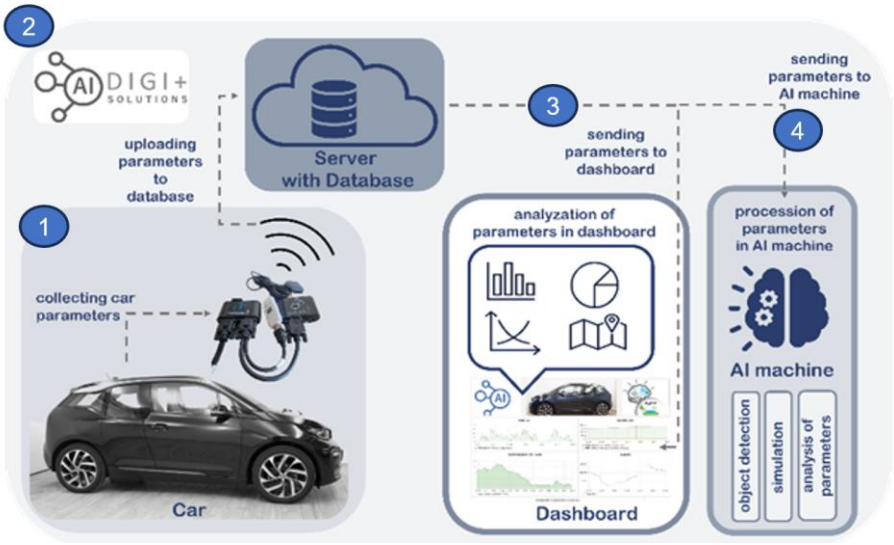

FIGURE 12 HIGH-LEVEL ARCHITECTURE OF DEMONSTRATOR SCD7.3

6.1.1.3 Main achievements of the demonstrator/sub-demonstrator prototype integration.

The Main Achievements are the application of a holistic approach based on Munich Agile concept and Model Based System Engineering in case of collection and processing of OBD2 Data for data analysis and value prediction. Furthermore, the entire process of data collection, data labelling and data validation have been created in a semi-automatic way. The entire data collection process from the research vehicle to the integration in a Dashboard have been created fully automated. Furthermore, bases on Model Based Systems engineering it was possible to create the entire process from Requirements, functions and the architecture itself based on SysML with Papyrus as a tools.

7 Conclusion and contribution to the overall picture

Within T5.7 and D5.16, contributing partners have provided dedicated demonstrator videos showcasing the results of the integration activities performed within WP5. The results demonstrate the successful application of the developed methods in various environments and introduce advanced solutions in the field of AI-based controller learning, validation methods, simulation, and virtualization. Further details will be presented in the upcoming deliverable D5.17 in M42.

List of figures

List of tables

- Last page of the document is intended to be blank! -