Hardware Abstraction Layer for a European Software Defined Vehicle Approach

D3.2 – Modelling and simulation activities for the RISC-V integration

30.03.2026

Co-funded by the European Union
Chips JU

Deliverable	D3.2 – Modelling and simulation activities for the RISC-V integration
Work Package(s)	WP3
Task(s)	Task 3.2
Dissemination Level	Public
Due Date	31.03.2026
Actual Submission Date	30.03.2026
WP Leader	Dirk Slama
Task Leader	Daniele Jahier Pagliari
Deliverable Leader	Daniele Jahier Pagliari
Contact Person	Daniele Jahier Pagliari
E-mail	daniele.jahier@polito.it

Document History		
Revision	Date	Description
V0.1	10.12.2025	Initial version
V0.2	28.01.2026	First draft with partners contributions
V0.3	31.01.2026	Ship out for internal revision
V0.4	07.03.2026	Internal review finalized, version for PSB review
V0.5	18.03.2026	PSB review finalized, version for PGA review
V1.0	30.03.2026	PGA cleared, final version for submission.

Authors	
Name	Partner Short Name
Daniele Jahier Pagliari	POLITO
Giuseppe Tagliavini	UNIBO
Vittorio Zaccaria	POLIMI
Nenad Petrovic	TUM
Paolo Burgio	UNIMORE
Matthias Stammler	KIT
Bruno Bodin	CEA
Mario Driussi	VIF

Co-funded by the European Union

Chips JU

Acknowledgement

The project is co-funded by the Chips Joint Undertaking (Chips JU) and National Authorities under grant agreement n° 101139789.

Co-funded by the European Union

Disclaimer

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or National Authorities. Neither the European Union nor the granting authorities can be held responsible for them.

Table of contents

List of figures

List of tables

Co-funded by the European Union

Abbreviations

A-DECA	Automated Design Space Exploration for Computing Architectures
AI	Artificial Intelligence
API	Application Programming Interface
BRAM	Block Random Access Memory
CAN	Controller Area Network
CLIC	Core-Local Interrupt Controller
CPS	Cyber-Physical System
DCLS	Dual-Core Lockstep
DSE	Design Space Exploration
EMF	Eclipse Modelling Framework
FPGA	Field-Programmable Gate Array
GenAI	Generative Artificial Intelligence
HMR	Hybrid Modular Redundancy
HW	Hardware
iDMA	integrated Direct Memory Access
LLC	Last-Level Cache
LLM	Large Language Model
LTTng	Linux Trace Toolkit: next generation
MBPTA	Measurement-Based Probabilistic Timing Analysis
MCS	Mixed-Criticality System
MCU	Microcontroller Unit
OCL	Object Constraint Language
PPA	Power, Performance, Area
PCR	Platform Control Register
PMCA	Programmable Multi-Core Accelerator
PULP	Parallel Ultra Low Power
RAG	Retrieval-Augmented Generation
ROS 2	Robot Operating System 2
RTL	Register-Transfer Level
RTOS	Real-Time Operating System
SDV	Software-Defined Vehicle
SoC	System-on-Chip
TCLS	Triple-Core Lockstep
VLM	Vision–Language Model
VP	Virtual Platform
VSS	Vehicle Signal Specification
WCET	Worst-Case Execution Time
XMI	XML Metadata Interchange

Co-funded by the European Union

1. Summary

This deliverable reports on the modelling and simulation activities carried out in Task 3.2 of the HAL4SDV project. The objective of Task 3.2 is to enable the integration of open-source RISC-V hardware platforms into the HAL4SDV ecosystem and virtual lab/playground, so that software and system-level design decisions can be explored, validated, and iterated early, before the final target hardware is available. In the SDV domain, this early feedback is essential: software stacks are large and rapidly evolving, computing architectures are increasingly heterogeneous, and system-level requirements (e.g., timing predictability, safety, and security) can only be fully assessed when hardware and software are considered together.

Task 3.2 is tightly connected to Task 3.4, which encompasses modelling and simulation activities for generic HAL4SDV tools, that are not specific to RISC-V integration. Together, the two tasks cover a layered tool landscape ranging from low-level virtual platforms to higher-level architectural analysis, optimization, and AI-assisted validation pipelines. Figure 1 provides a consolidated view of the modelling and simulation activities of Tasks 3.2 and 3.4 across abstraction layers and functional domains.

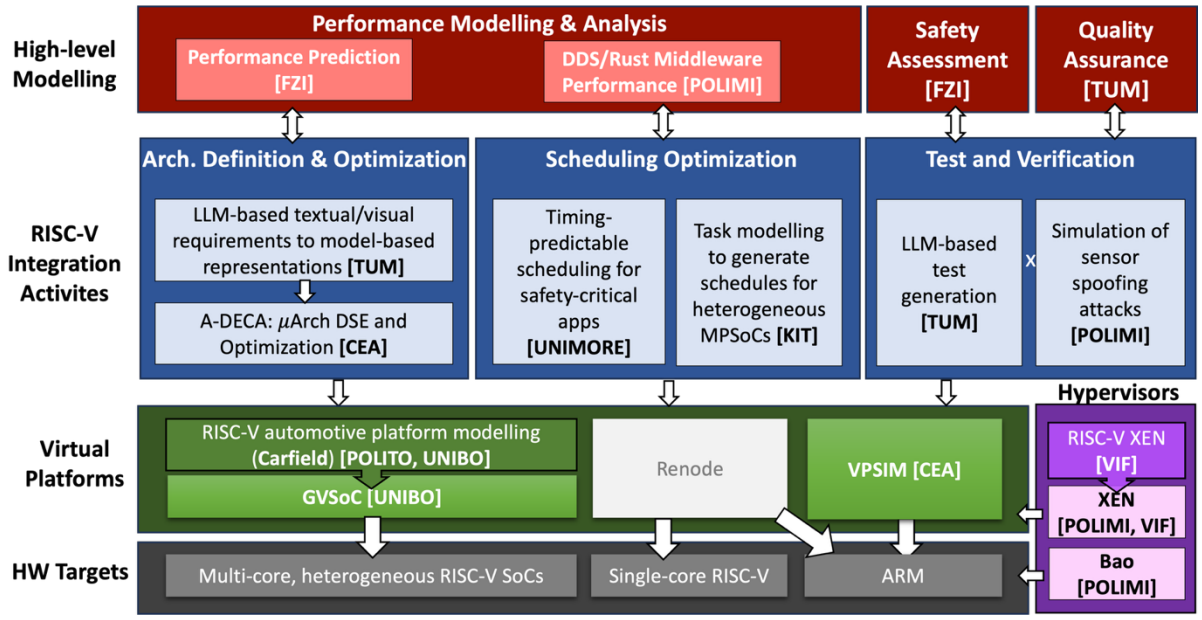

Figure 1: Overview of the HAL4SDV modelling and simulation activities (T3.2 and T3.4)

Some tools (e.g., Renode) are greyed out because they are not actively developed inside the project, but higher-level tools use them.

Figure 2 zooms in on the subset of activities that fall within Task 3.2 and are therefore documented in this deliverable.

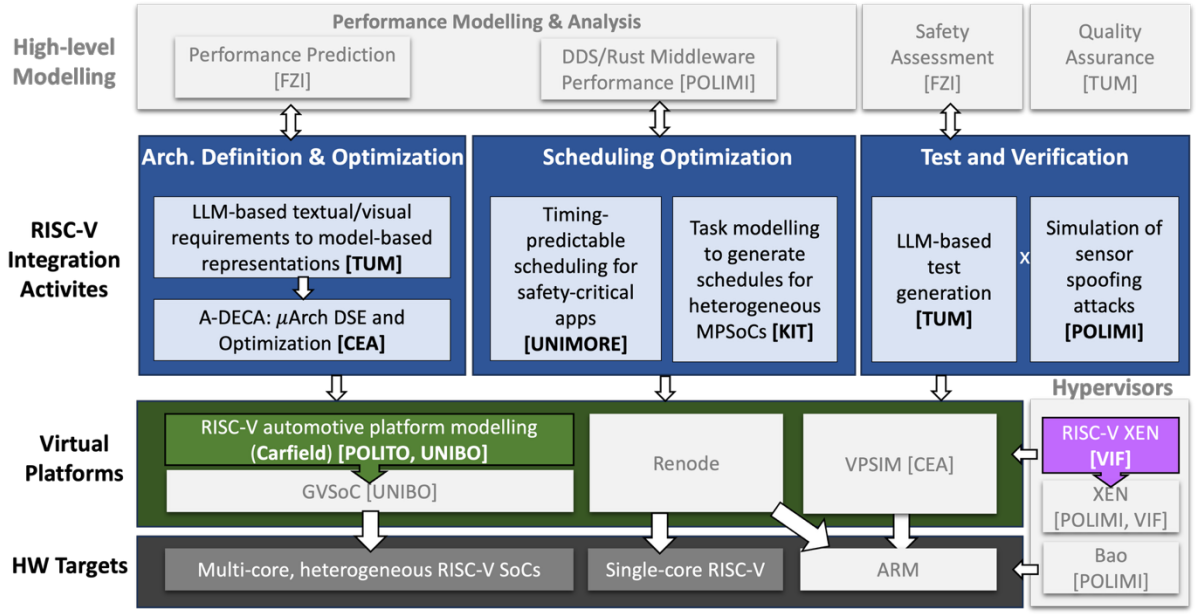

Figure 2: Modelling and simulation activities related to the RISC-V integration (T3.2)

The contributions within Task 3.2 are organized around three complementary goals that together facilitate and promote RISC-V adoption for SDV platforms.

First, Task 3.2 activities include the development of new **virtual prototyping targets for RISC-V automotive platforms** and **hypervisors support for this architecture**. These activities provide practical "hardware targets" that can be reused by multiple partners and connected to the broader HAL4SDV playground, and enhanced virtualization support for RISC-V, fundamental in the automotive domain.

Second, Task 3.2 extends **architecture definition, optimization, and design space exploration** capabilities towards realistic heterogeneous RISC-V platforms. In particular, the partners extended architectural optimization tools that previously only supported ARM-based systems, or simple single-core RISC-V processors, to be compatible with realistic heterogeneous automotive-grade RISC-V SoCs (such as those virtualized in the first category of contributions, see above). Those SoCs must include multiple compute domains and accelerators while meeting tight constraints on performance, power, area, and real-time predictability. The work reported in this deliverable therefore includes extensions of architecture exploration and optimization tooling, as well as model-based and AI-assisted methods that help bridge natural-language and diagram-based requirements to architecture-level representations.

Third, Task 3.2 develops dedicated **scheduling, test, and verification techniques for heterogeneous and mixed-criticality RISC-V platforms**. The contributions in this category include work on task modelling and schedule generation for heterogeneous platforms, and automation of verification workflows using GenAI, spanning requirements-to-test generation and interface consistency checking. In addition, **security-focused simulation work** is included to study the impact of sensor spoofing and data-oriented attacks on timing predictability and availability in mixed-criticality systems.

Overall, the activities described in this deliverable provide a significant step forward in the maturity of modelling and simulation infrastructure for RISC-V, from executable virtual

Co-funded by the European Union

prototyping targets to architecture exploration and optimization, to scheduling, verification, and security-oriented evaluation. Therefore, the outcomes of Task 3.2 represent an important progress towards the HAL4SDV's objective of de-risking the integration of RISC-V hardware into SDV platforms.

In the rest of this deliverable, each Task 3.2 component or development activity is described using a consistent template: contributing partners, category, description and achievements, current release status (including availability and licensing/access information), and interactions with other tools across WP3 and the broader project.

2. Modelling and simulation activities for the RISC-V integration

2.1 RISC-V automotive platform modelling in GVSoC (Carfield)

2.1.1 Contributing Partners

POLITO, UNIBO

2.1.2 Category

Virtual Platform

2.1.3 Description and Achievements

GVSoC is the Virtual Platform developed for the family of Parallel Ultra Low Power (PULP) RISC-V Systems on Chip (SoCs). This VP has been actively maintained for several years by the PULP team (part of which coincides with the UNIBO team involved in HAL4SDV).

Part of the activities that POLITO and UNIBO collaboratively carried out for the HAL4SDV project consists of *extending GVSoC to support an automotive-specific RISC-V SoC from the PULP family, called Carfield*.

This work fits into a broader collaboration between UNIBO and POLITO (over multiple projects, including the previous CHIPS-JU initiatives on RISC-V TRISTAN and ISOLDE), in which POLITO provides software support for the PULP hardware developed by UNIBO, especially for what concerns the realization of AI applications on-board these platforms. This activity includes the development of AI compilers, optimized SW backend kernels, and, importantly, *high-level VPs/simulators that permit the functionally and timing accurate validation of complete applications* prior to tape out, which is impractical to do at RTL due to the excessive simulation time.

Specifically, Carfield is an open-source heterogeneous platform targeting the automotive domain. It provides several features that make it a Mixed-Criticality System (MCS), organized into multiple domains, as shown in Figure 3. In particular, the platform is composed of the following domains:

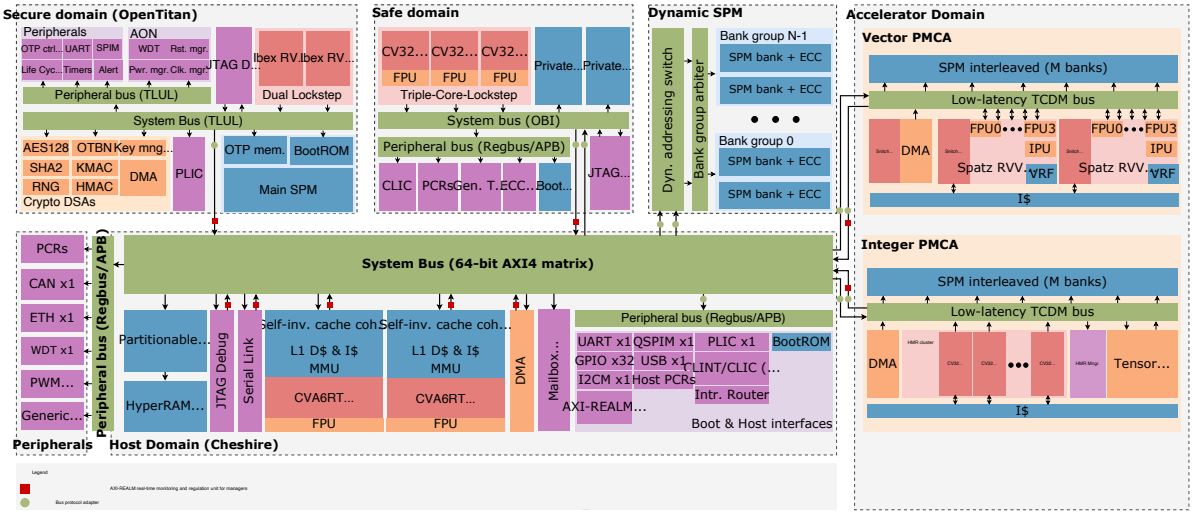

Figure 3: Schematic overview of the Carfield platform.

- *Host domain* (Cheshire): a Linux-capable RV64 system based on dual-core CVA6 processors, featuring a self-invalidation cache coherency mechanism.
- *Safe domain*: a Triple-Core-Lockstep (TCLS) RV32 microcontroller system based on CV32E40P, with fast interrupt handling enabled by the RISC-V CLIC.
- *Secure domain*: a Dual-Core-Lockstep (DCLS) RV32 Hardware Root of Trust (HW RoT) system that ensures secure boot for the entire platform, acts as a secure monitor, and provides cryptographic acceleration through dedicated crypto accelerators.
- *Accelerator domain*: composed of two programmable multi-core accelerators (PMCAs), namely a 12-core integer cluster (*PULP cluster*) with Hybrid Modular Redundancy (HMR) capabilities targeting compute-intensive integer workloads such as AI, and a vector cluster (*Spatz cluster*) with floating-point vector processing capabilities for accelerating control-oriented workloads.

In Task T3.2 of the project, POLITO, with the support of UNIBO, worked on integrating a model of the Carfield platform into the GVSoC simulator. An initial analysis was performed to identify the minimum functionality required for a first working version, that is: a simple but functional *host-plus-cluster system*. Based on this analysis, two domains were selected as initial targets: the host domain (*Cheshire*) and one accelerator domain, namely the integer cluster (*PULP cluster*).

This required the initial modelling of the Cheshire platform, which is also designed to be reusable as a host domain for other chips. The modelling effort from POLITO involved interconnecting a basic CVA6-based RV64 processor with standard components already supported by GVSoC, such as memories and peripherals, as well as with custom components adapted specifically for this project, such as the iDMA.

The standalone Cheshire domain was validated by POLITO using the unmodified software toolchain originally employed for FPGA prototyping and RTL simulation.

Subsequently, POLITO extended the Cheshire platform to more closely resemble the host domain configuration used in Carfield, by adding additional components such as platform control registers (PCRs), the last-level cache (LLC), and other system-level features.

Among these extensions, particular relevance was given to automotive-oriented support for the CAN controller, interfaced with the standard Linux CAN stack (SocketCAN), which is especially useful for the targeted application domain.

This enables the connection of Carfield-based nodes (simulated on GVSoC) to the *Mix&Match platform* described in D3.1 and consequently to the HAL4SDV playground.

Finally, the PULP cluster was integrated into the Carfield platform and interfaced with the customized Cheshire host domain. Although a PULP cluster model was already available in GVSoC, several modifications were required to ensure compatibility with the Carfield specifications. These included the introduction of specific interconnects between the host and accelerator domains and alignment of the memory map exposed to software.

Like for the Cheshire model, complete host-domain-plus-PULP-cluster system was also validated by POLITO using offload examples from the original RTL and FPGA toolchain. These tests confirmed that Carfield workloads involving offloading computations to the integer cluster can now be successfully executed on the GVSoC simulator.

2.1.4 Current Release

The current version of the Carfield model has integrated into the main GVSoC repository through a pull request and is now publicly available at https://github.com/gvsoc/gvsoc.

Development and integration activities are tracked through dedicated issue threads within the same repository. In particular:

- Cheshire platform development: https://github.com/gvsoc/gvsoc/issues/155;
- Carfield platform development: https://github.com/gvsoc/gvsoc/issues/223.

All components developed within this project are released as open source and follow the licensing scheme adopted by the GVSoC repository.

2.1.5 Interactions with other tools

First, this activity is strictly related to those carried out by UNIBO and POLITO in T3.4 (described in D3.4). Moreover, the integration of Carfield support at the VP level is functional to enable *all higher-level RISC-V integration activities* (blue background in **Fehler! Verweisquelle konnte nicht gefunden werden.**) that could leverage Carfield as a hardware target. This includes automatic architectural optimization, scheduling optimization, AI-based test generation etc.

In particular, POLITO is currently working with CEA to enable architectural Design Space Exploration (DSE) using their A-DECA tool, targeting a configurable GVSoC plus Carfield template (e.g. memory dimensions, number of accelerator cores, etc), targeting the optimization of on-vehicle AI applications, compiled for Carfield using POLITO's open-source neural network deployment tool MATCH (https://github.com/eml-eda/match). While this activity falls beyond the scope of the present deliverable, it represents an important development that is enabled by Carfield's integration into GVSoC.

2.2 RISC-V Xen based virtualization platform for automotive

2.2.1 Contributing Partners

VIF

2.2.2 Category

Hypervisors

2.2.3 Description and Achievements

In this task, we are establishing a shared development, (performance) testing, simulation, build, deployment, and monitoring environment that can be reused across heterogeneous hardware targets, x86_64, Arm, and, once available, RISC-V (Xen Hypervisor Linux Guest HW drivers not ready at this time), so that workflows and observability remain consistent rather than platform-specific. A key objective is to run performance analyses in different operational contexts (lab, cloud, vehicle) to explore how far virtual validation can be pushed and which performance indicators remain comparable across environments. In addition, the environment is designed to demonstrate software portability with a "build once, run anywhere" approach: the same built artifacts are executed unchanged in the lab, cloud, and vehicle without recompilation, enabling a clean separation between portability constraints and runtime performance effects.

We established a reproducible virtualization baseline on Intel (NUC) and Arm64 (Raspberry Pi 5, 16 GB) using Yocto-based builds. On Arm, we implemented and exercised both deployment models that are relevant for our target embedded and mixed-criticality scenarios: Dom0less (for a lightweight, static partitioning style) and a classic Linux-based Dom0 setup. In parallel, we kept a Linux-as-Dom0 configuration available on both Arm and Intel, enabling comparable host/guest workflows across heterogeneous hardware.

On top of these platforms, we executed representative guest workloads and used a combined tracing, profiling, and monitoring toolchain to characterize system behaviour end-to-end. Specifically, we relied on guest-level tracing (e.g., LTTng) together with Xen-centric observability utilities (xentrace for hypervisor trace buffers, xenoprof for profiling, and xentop for runtime monitoring). This combination is important because it allows us to correlate guest-visible effects (scheduler latency, I/O patterns, middleware activity) with hypervisor-level events and resource accounting. The resulting baseline is intended to serve as the reference point for evaluating both performance and portability of the overall software stack across CPU architectures.

Planned extension to RISC-V platform

The same tracing/profiling/monitoring concept is planned to be ported and rebuilt for RISC-V to evaluate the performance and portability of the complete virtualized software stacks under Xen. Our evaluation target is explicitly not limited to "hello world" guests: we aim to assess

full guest stacks consisting of an operating system (Linux), a middleware layer (ROS 2 / Autoware), and corresponding applications. This type of workload is inherently resource-intensive and requires a stable hypervisor feature set to ensure that measurements reflect the behaviour of the stack under test rather than artifacts from platform immaturity.

At the present time, we have not started the RISC-V Target activity for concrete reason:

- The Xen RISC-V port is still in an “early enablement / under active development” phase, and key capabilities are not yet at the maturity level we require. Full driver support for Linux guests.
- The Xen 4.21 announcement highlights “early RISC-V enablement” rather than a fully supported production port.

For these reasons, we deliberately postponed the RISC-V implementation of the tracing/profiling/monitoring concept. The activity is planned if Xen RISC-V Linux guest drivers are available and provide the required features with sufficient stability for running and measuring full guest stacks (Linux + ROS 2/Autoware + applications) without confounding effects from platform incompleteness.

2.2.4 Current Release

Virtualized Intel and Arm Targets with Xen Hypervisor and Linux DOM0 and DomU, Zephyr as ControlDomain, Linux as DriverDomain and DomU prepared for Performance Analysis in different environments (lab, cloud)

2.2.5 Interaction with other tools

This activity uses tools and concepts worked out in Task 3.3 and Task 3.4

2.3 A-DECA: Microarchitectural Design Space Exploration and Optimization

2.3.1 Contributing Partners

CEA

2.3.2 Category

Architecture Definition and Optimization

2.3.3 Description and Achievements

The increasing complexity of modern computing architectures, particularly in the context of automotive Software-Defined Vehicles (SDVs), raises two major challenges for architecture tuning. First, there is a need for fast and accurate automatic exploration of large microarchitectural design spaces given HW/SW needs. Second, architectures must satisfy multiple, often conflicting design objectives, such as performance, power consumption, silicon area, and real-time constraints. Traditional methodologies focusing on one or two

metrics are no longer sufficient to address the complexity and safety requirements of automotive systems.

A-DECA (Automated Design Space Exploration for Computing Architectures) is a modular framework developed by CEA, designed to address these challenges by automating the exploration and optimization of microarchitectural parameters. It combines multiple simulators, analytical models, and exploration strategies to efficiently evaluate large design spaces while maintaining reasonable execution times.

Within the HAL4SDV project, A-DECA has been extended and adapted to support embedded automotive SoCs, with a particular focus on the Carfield RISC-V–based architecture. The methodology enables the automatic optimization of computing platforms by jointly improving several criteria relevant to SDVs, including performance, power, and area (PPA). Key innovations provided by A-DECA include:

- Automation: Fully automated design space exploration, enabling rapid evaluation of thousands of architectural configurations across different workloads without manual intervention.
- Modularity: A flexible architecture that allows easy integration of different simulators, power models, and exploration engines, supporting heterogeneous computing environments.
- Multi-objective optimization: Native support for Pareto-based optimization, enabling balanced exploration of conflicting PPA metrics.

The inputs to A-DECA include:

- A parametric template of the target architecture, describing RISC-V cores, memory hierarchy, interconnects, and optional accelerators.
- A set of tunable microarchitectural parameters.
- Optimization objectives, such as maximizing performance or minimizing power and area.
- Design constraints, including memory size, interconnect limitations, and embedded system requirements.

The outputs:

- Set of non-dominated solution represented as a Pareto Front

The exploration engine is based on the A- DECA approach presented in [Zaourar et al., 2023] and previously applied to system-level high-performance computing architectures [Fu et al., 2024]. In A-DECA, several optimization strategies (such as Bayesian optimization, genetic algorithms, and machine-learning-based methods) are used to explore candidate architectures automatically.

For each configuration, simulators and analytical tools are invoked to evaluate relevant PPA metrics. The framework outputs a set of Pareto-optimal configurations, allowing architects to analyse trade-offs between competing objectives and select solutions best suited to automotive SDV requirements.

In HAL4SDV, A-DECA has been specifically updated to address embedded RISC-V automotive platforms. In collaboration with Politecnico di Torino, the framework is applied to the Carfield use case, using GVSoC as the main simulation backend for performance evaluation. Power and area estimations are derived from complementary analytical models and technology-aware estimation flows, enabling early-stage architectural decisions consistent with automotive constraints.

2.3.4 Current Release

The current version of A-DECA used in HAL4SDV includes:

- Support for embedded SoC design space exploration.
- Integration with GVSoC for RISC-V–based performance simulation.
- Multi-objective optimization engines tailored to PPA trade-off analysis.

2.3.5 Interactions with other tools

Within HAL4SDV, A-DECA interacts with several tools and activities across WP3 and other work packages:

- Simulation tools: in addition to VPSim, CEA teams realized the integration with GVSoC enables detailed performance evaluation of Carfield RISC-V configurations.
- Power and area estimation tools: Results from analytical or technology-aware models are used as inputs to the multi-objective optimization process.
- Use-case workloads: Automotive-oriented applications provided by project partners are used to guide exploration toward SDV-relevant configurations.

These interactions position A-DECA as a central architectural optimization component supporting early design decisions for RISC-V–based automotive SDV platforms.

2.4 LLM-based textual/visual requirements to model-based representations

2.4.1 Contributing Partners

TUM

2.4.2 Category

Architecture Definition and Optimization

2.4.3 Description and Achievements

2.4.3.1 LLM-empowered SDV system extendability

Figure 4 illustrates the proposed workflow for convenient automotive system updates that leverages LLM-driven hardware abstraction and automated compliance checking.

Co-funded by the European Union

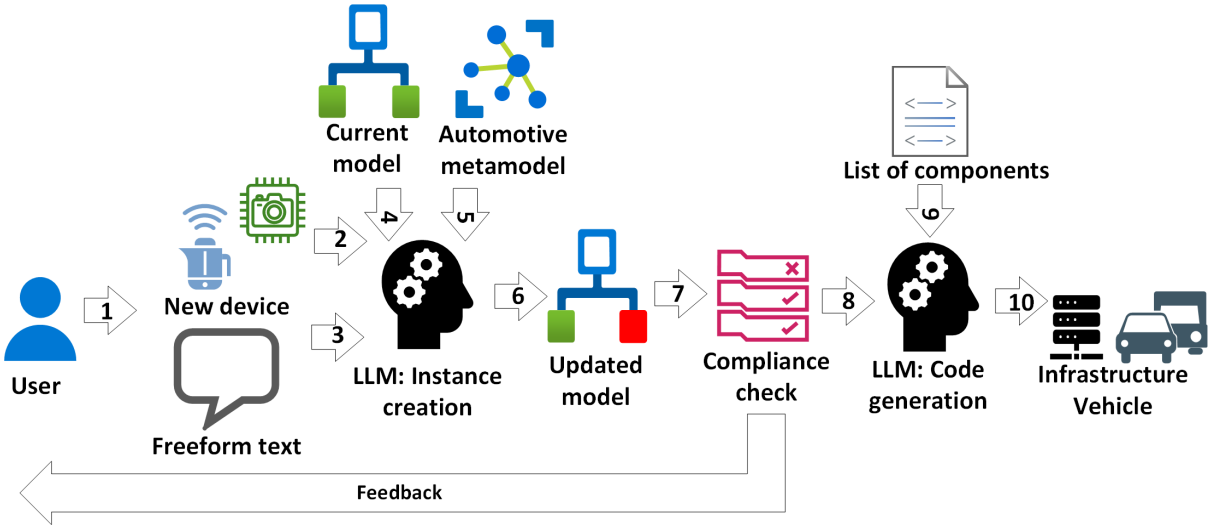

Figure 4: LLM-enabled updates workflow: User input; 2) new device description; 3) usage scenario for the new device; 4) XMI model instance; 5) Ecore metamodel; 6) updated XMI model instance; 7) predefined set of OCL rules; 8) compliance check pass

In this case study, we assume that a user intends to add a new device to an existing vehicular system, such as a sensor, an actuator, or an application-specific hardware accelerator. The specification of the new device is assumed to be available, either entered manually by the user or automatically retrieved from a device descriptor. In addition, the user provides free-form textual input describing the intended purpose and usage of the new device.

The workflow takes into account the current system state, which is represented as a model instance, as well as the underlying system metamodel. Based on the user-provided text or diagrams, as well as the device specification, an updated system model is constructed. This step builds on our previous work on LLM-based instance model creation [1]. The result is a model-based representation of the updated ("to-be") system. Metamodel-related aspects are implemented using the Eclipse Modelling Framework (EMF) and its Ecore framework, while concrete model instances are represented in XMI format.

In parallel, a predefined set of design space constraints is applied to the updated system model. These constraints are expressed as Object Constraint Language (OCL) rules derived from an assumed reference architecture. The purpose of these rules is to determine whether the updated system configuration is capable of fulfilling the intended user scenario. If all constraints are satisfied for the updated instance model representing the "to-be" system, the workflow proceeds to the next step.

Subsequently, LLM-based code generation is performed. In addition to the updated system model, this step relies on a catalogue of predefined components that are assumed to be available, including:

- a list of device-specific drivers for the target host architecture,
- publish/subscribe message topics for the ROS 2 middleware,
- a list of available Docker containers.

The commands generated as the final outcome of this process are then executed on the target infrastructure.

A summary of the evaluated scenarios is provided in Table 1.

We first assume a target vehicle configuration consisting of two cameras (2 × 8.3 MP), five radars, and ten ultrasonic sensors, with an optional driver monitoring camera (1 × 8.3 MP). This configuration serves as the basis for generating the corresponding OCL rules. In this experiment, the goal is to implement supervised parking functionality using this sensor setup inspired by reference architecture from [2].

Next, we consider a situation in which the current system configuration is missing one camera. If a new camera with a 2.1 MP resolution is added to the system, the OCL rule validation fails, and the workflow stops before code generation. In contrast, if the added camera has a resolution equal to or higher than 8.3 MP, the OCL constraints are satisfied, and the workflow proceeds to command generation. This includes installing the corresponding driver, subscribing to the required ROS 2 topics, and launching the Docker container responsible for parking assistance functionality.

Table 1: Updates and Hardware Abstraction Examples

Target Configuration	Current Configuration	New Device	Expected Outcome
2 × 8.3 MP cameras, 5 radars, 10 ultrasonic sensors, optional 1 × 8.3 MP driver monitoring camera	1 × 8.3 MP camera, 5 radars, 10 ultrasonic sensors	Camera model c0 – 2.1 MP	Camera resolution too low
2 × 8.3 MP cameras, 5 radars, 10 ultrasonic sensors, optional 1 × 8.3 MP driver monitoring camera	1 × 8.3 MP camera, 5 radars, 10 ultrasonic sensors	Camera model c1 – 8.3 MP	Install c1, run Docker container for parking assistance, publish ROS 2 topic *camera2parking*

Summary of the experiment results is given in Table 2.

Table 2: Experiment summary and evaluation results

Prompts / Steps	Inputs	Execution Time [s]	Correctness

Step 1: OCL Rule Generation • Generate OCL rules based on reference requirements. Step 2: Model Update • Update the current system model instance to add a new device that satisfies the new requirements, taking the metamodel into account. Step 3: Model Validation Step 4: Code Generation • Generate driver installation commands for the new device. • Generate Docker container run commands based on templates. • Subscribe to ROS 2 topics based on the provided list.	• Current system model • Metamodel • New device specification • New requirements • Reference requirements • Docker command template • List of ROS 2 topics	38.55	9/10 (Code generation) 10/10 (Decision)

More details about this work can be found in the paper presented at GACLM conference [1].

2.4.3.2 LLM-driven interface compatibility checker

This contribution leverages Large Language Models (LLMs) to perform automated interface compatibility checking when integrating a new software component into an existing system. As shown in Figure 5, the approach takes as input the specifications and source code of the existing system, the specifications and source code of the new component, and a set of new or modified requirements. These inputs are analysed jointly to assess whether the new component can be safely and effectively integrated.

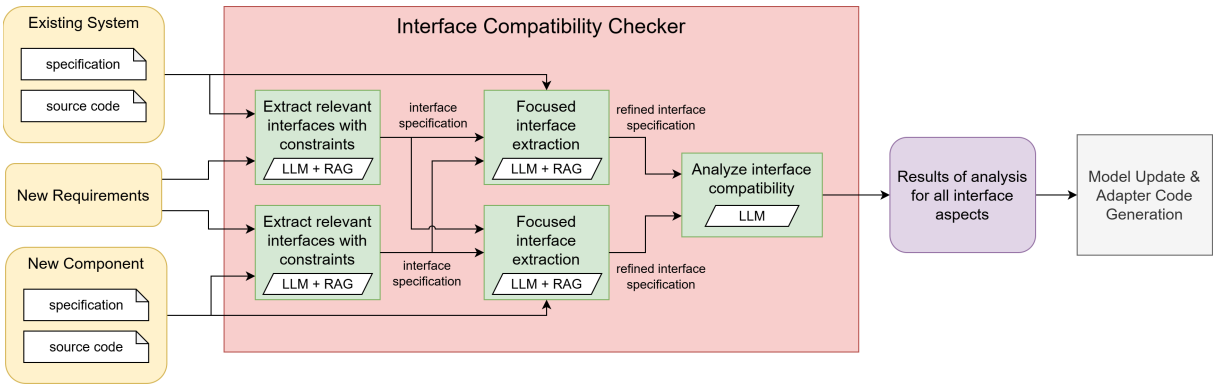

Figure 5: LLM-based interface compatibility checker

Co-funded by the European Union

The specifications of both the existing system and the new component may consist of heterogeneous artifacts, including natural-language documentation, architectural or behavioural models, and implementation-level source code. Each artifact type provides complementary insights into the system interfaces. For example, documentation may reveal implicit assumptions, third-party dependencies, or usage constraints that are not explicitly encoded in the source code, while code artifacts provide precise information about data types, method signatures, and control flow. The new requirements capture changes to existing functionality or the introduction of new capabilities that the integrated system is expected to support.

As depicted in the figure, the Interface Compatibility Checker first applies an LLM combined with Retrieval-Augmented Generation (RAG) to extract the relevant interfaces and associated constraints from the existing system artifacts and the new component artifacts. This step produces initial interface specifications that focus only on those interfaces affected by the new requirements or involved in the planned integration. By filtering out irrelevant information early, the approach reduces analysis complexity and improves the quality of subsequent reasoning.

In the next stage, a focused interface extraction step is performed, again using an LLM with RAG support. Here, the previously extracted interface specifications are refined and normalized to obtain a consistent and comparable representation of the interfaces on both sides. This refinement process resolves ambiguities, aligns terminology, and makes implicit assumptions explicit, resulting in refined interface specifications that are suitable for detailed compatibility analysis.

The refined interface specifications are then analysed by the Interface Compatibility Checker using an LLM-based reasoning process. This analysis examines multiple dimensions of interface compatibility, including syntactic compatibility (e.g., method signatures and data types), semantic compatibility of exchanged data and operations, alignment of pre-conditions and post-conditions, preservation of interface invariants, and consistency of communication mechanisms such as invocation styles, protocols, or data formats. Potential mismatches, such as incompatible assumptions about error handling or execution order, are explicitly identified.

The outcome of this process is a comprehensive set of analysis results covering all relevant interface aspects, as illustrated on the right-hand side of Figure 5. These results provide a structured assessment of the degree and nature of incompatibilities between the existing system and the new component. Based on this assessment, developers can determine whether integration is feasible without changes, whether adapter or wrapper code is required, or whether updates to the system model or source code are necessary. In this way, the approach supports informed decision-making during system evolution and reduces the risk and cost associated with component integration.

As proof-of-concept, we applied the proposed approach to Autoware-inspired case study, considering the fact that vehicle manufacturers rely on diverse communication standards and proprietary protocols, integrating Autoware with a specific vehicle often requires the development of custom interface adapters. A representative example is the PACMod3 ROS driver, which enables communication with vehicles equipped with the PACMod drive-by-wire system via CAN-based protocols. This driver standardizes actuation commands across supported platforms and exposes them through ROS 2 message types such as

pacmod3_msgs/SteeringCmd. To make these commands usable within Autoware, an additional adapter—commonly referred to as pacmod_interface—is required. This adapter translates PACMod-specific messages into Autoware-defined message types such as autoware_control_msgs/Control.

This adapter-based integration process is prone to multiple forms of interface incompatibility. During the development of pacmod_interface, several such incompatibilities are likely to have been encountered and resolved. These incompatibilities can be broadly categorized as follows.

Data type mismatches occur when corresponding interface fields use different data representations. For example, the steering command field SteeringCmd.rotation_rate in PACMod3 is defined as a 64-bit floating-point value, whereas the corresponding field Lateral.steering_tire_rotation_rate in Autoware uses a 32-bit floating-point type. Although these types are nominally compatible, unhandled conversions may result in precision loss or runtime errors.

Semantic mismatches arise when interfaces use the same data types but assign different meanings to the values. In the steering example, the two systems may adopt different conventions for steering direction, such as clockwise versus counterclockwise rotation. Without explicit semantic alignment, such differences can lead to incorrect vehicle behaviour despite syntactically valid communication.

Communication mechanism mismatches relate to differences in how and when data is exchanged. Even when both systems rely on publish/subscribe communication, incompatibilities may occur due to differing assumptions about message frequency, timing, or synchronization. For instance, a steering command published at 50 Hz may not satisfy a subscriber that expects updates at 100 Hz, potentially degrading control performance.

To systematically identify and address these incompatibilities, an LLM-based Software Interface Compatibility Checker can be applied, as illustrated in the accompanying figure. In this workflow, the specifications and source code of the existing system (Autoware), the new component (PACMod3 driver), and the new or modified requirements are provided as inputs. Using a combination of Large Language Models and Retrieval-Augmented Generation (RAG), the checker first extracts the relevant interfaces and associated constraints from both systems. These interfaces are then refined into focused interface specifications that explicitly capture data types, semantic assumptions, and communication properties.

Based on the refined specifications, the Interface Compatibility Checker performs a comprehensive analysis across all relevant interface aspects. The outcome is a structured assessment of potential incompatibilities, along with recommendations for resolving them. These recommendations may include explicit data type conversions, semantic transformations, rate adaptation mechanisms, or the automatic generation of adapter code to bridge the identified gaps.

To evaluate this approach in a controlled manner, a set of simplified ROS 2-based evaluation scenarios was constructed. These scenarios abstract away from real vehicle hardware and instead focus on minimal message exchange configurations that isolate specific incompatibility types. Each scenario consists of a data-producing node (either a publisher or

a service server) and a data-consuming node (a subscriber or client). The scenarios and their expected outcomes are summarized in Table 3.

Table 3: LLM-based interface compatibility scenarios

Scenario	Setup	Expected Outcome
Data type mismatch	Rotation rate published as a 32-bit floating-point value; subscriber expects a 64-bit floating-point value.	Precision loss identified; explicit type conversion suggested.
Semantic mismatch	Publisher and subscriber both use 32-bit floating-point rotation rates; documentation specifies different steering direction conventions.	Semantic mismatch detected; transformation between conventions recommended.
Communication mechanism mismatch	Rotation rate command published at 50 Hz; subscriber expects updates at 100 Hz.	Rate mismatch identified; solutions such as interpolation or resampling proposed.

Summary of the experimental results can be seen in Table 4 We give overview of crucial steps, the main instructional content embedded in the prompts, as well as performance in terms of execution time and percentage of correctness, given as average based on 10 executions. We used OpenAI's GPT-4o LLM in all of the presented experiments.

Table 4: Experiments summary and evaluation

Prompts / Steps	Inputs	Execution Time [s]	Correctness
Step 1: Extraction • Extract all parts of the system's interfaces relevant to the new requirement. • Extract all parts of the component's interfaces relevant to the new requirement. **Step 2: Refinement** • Refine the system's interface specification to ensure it covers all aspects required for integrating the new component. • Refine the component's interface specification to ensure seamless integration into the existing system. **Step 3: Compatibility Analysis** • Analyse all aspects of both interfaces to assess their compatibility.	• Existing system specification • Existing system source code • New component specification • New component source code • New requirement	**Mismatch cases:** Data type: 89.08 Semantic: 90.05 Communication mechanism: 101.64	**Mismatch cases:** Data type: 9/10 Semantic: 3/10 Communication mechanism: 9/10

The contributions of this topic were presented as part of paper [3] presented at GACLM conference in Valencia, Spain.

2.4.3.3 Software Architect Assistant

To support the broader task of software architecture evolution, we propose a Software Architect Assistant. This assistant processes the same core inputs as the Interface Compatibility Checker, namely the specifications of the existing system, new requirements, and the specifications of a new software component. In addition, it explicitly incorporates input from the software architect. Such input may include design preferences or expert knowledge that is not captured in formal specifications, for example planned transitions to new architectural patterns (e.g., the use of private databases for microservices) or insights into future functionalities defined in the product roadmap.

Based on these inputs, the Architect Assistant generates a set of analysed architecture modification options. For each option, the assistant highlights the associated advantages and disadvantages, enabling the software architect to make an informed decision and select the most appropriate solution. As part of this process, the assistant implicitly evaluates the interface compatibility between the new software component and the existing system. It also derives and compares multiple alternatives, including options that integrate the new component as well as options that fulfil the new requirements without introducing an additional component.

Once a preferred architecture modification option has been selected, a more detailed implementation plan can be derived, for example in the form of feature definitions or user stories, which then serve as a basis for implementation. While these subsequent steps—such as option selection, detailed planning, and implementation—can also be supported or automated using LLMs, they are outside the scope of the present work.

The overall workflow of the Software Architect Assistant is illustrated in Figure 6.

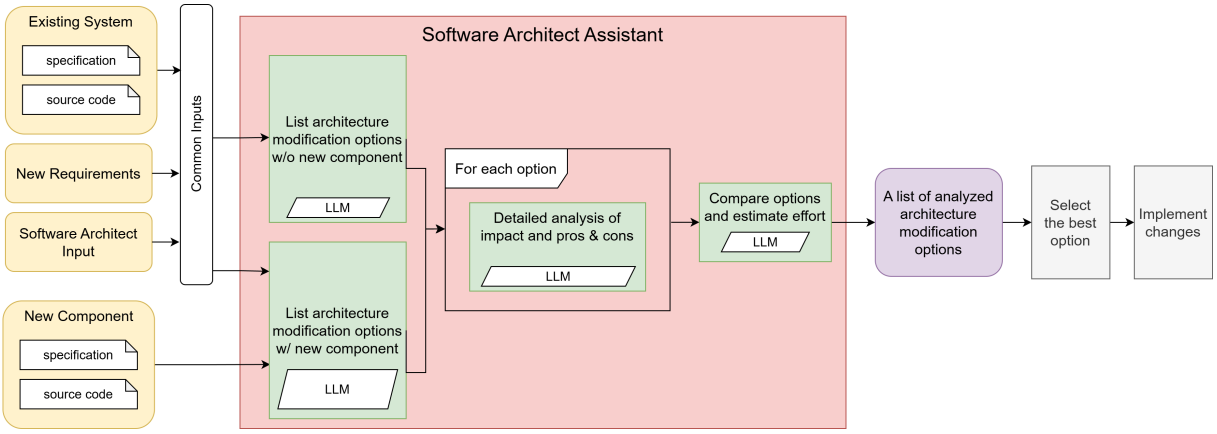

Figure 6: Workflow of the Software Architect Assistant.

The Architect Assistant approach was evaluated using a simple face detection system for a webcam, implemented with ROS 2. The system setup consists of three nodes: a ROS 2 driver for USB cameras (usb_cam), an image pre-processing node (image_filter), and a face detection node (face_detector). The image_filter node subscribes to raw image data published by usb_cam, crops the images to the required size, and republishes the processed images. The face_detector node subscribes to these cropped images, performs face detection using the OpenCV face detector based on the Haar Cascade classifier, and publishes images with detected faces for visualization in a client application such as rviz2.

In the experiment, the existing system used image_filter_small to crop images to a resolution of 640 × 480. A new requirement introduced support for a higher resolution of 1280 × 720, and a modified image filter node (image_filter_large) was provided as a new software component to perform cropping at this resolution. The objective of the evaluation was to determine whether the Architect Assistant could correctly propose different options for modifying the existing image_filter_small implementation, as well as suggest replacing image_filter_small with image_filter_large after verifying interface compatibility.

The specifications of all components in the existing system, as well as of the newly introduced component, were generated in Markdown format and manually reviewed before being used as input for the Architect Assistant. In addition, the source code of both the existing system and the new component was provided as a single string, consisting of the folder structure of each component followed by the concatenated contents of all source files. The input from the software architect included general design guidelines for the assistant, such as adherence to the Single Responsibility Principle and the Open/Closed Principle.

Evaluation summary for GPT-4o based on 10 executions is given as Table 5.

Table 5: Architecture assistant evaluation

Prompts / Steps	Inputs	Execution Time [s]	Correctness
Step 1: Architectural Options Identification • Describe options for modifying the existing system architecture to fulfil the new requirement without introducing a new software component. • Describe options for modifying the architecture by introducing a new software component to fulfil the requirement. **Step 2: Detailed Analysis of Modifications** • For each proposed modification, perform an analysis including the required changes and the advantages and disadvantages of each approach. **Step 3: Effort and Impact Estimation** • Compare the proposed modifications and provide relative estimates of implementation effort and impact for each option.	• Existing system specification • Existing system source code • New component specification • New component source code • New requirement • Software architect input	116.41	10/10

2.4.4 Current Release

We intend to open-source all the related software assets. Currently, HAL4SDV partners can ask for access to the source code at the following address:

https://gitlab.lrz.de/hal4sdv/sensor-integration

2.4.5 Interactions with other tools

n.a.

2.5 Carfield Integration and timing-predictable scheduling for safety-critical apps

2.5.1 Contributing Partners

UNIMORE

2.5.2 Category

Co-funded by the European Union

Scheduling Optimization

2.5.3 Description and Achievements

UNIMORE supported UNIBO for the integration of Carfield platform on the ZCU102, that is the reference platform for the demonstrator. The ZCU102 offers a smaller programmable area compared to the boards that are supported by Carfield. UNIMORE's activities focused on defining the memory requirements to enable the Carfield implementation targeting the ZCU102.

In parallel, testing of the Carfield flow with the objectives of:

- Building the hardware targeting the ZCU102.
- Verifying the RISC-V compilation flow and execution on the hardware.

The next step is to implement the interconnection between the two subsystems via BRAM, to ensure predictable communication across components.

2.5.4 Current Release

The current release consists of a prototype integration of the Carfield platform targeting the AMD ZCU102 platform. At this stage the hardware for ZCU102 is successfully set up and validated and the RISC-V compilation flow verified.

The activities related to Carfield integration are still ongoing including the implementation of predictable memory banks shared between Host and RISC-V using BRAMs.

The current version is available as internal development version for research. No current plans to release it opensource.

2.5.5 Interactions with other tools

Within Task T3.3, UNIMORE proposed a set of software components developed using the ROS2 framework. These software components, referred to as tasksets, provide localization capabilities within a known environment.

The hardware platform developed in this activity is intended to support the execution of such software stacks. In particular, the Carfield-based hardware targeting the ZCU102 platform is designed to host and execute the RISC-V binaries generated from the ROS2 tasksets, enabling the evaluation of hardware/software co-design aspects and predictable execution.

2.6 Task modelling to generate schedules for heterogeneous MPSoCs

2.6.1 Contributing Partners

KIT

2.6.2 Category

Scheduling Optimization

2.6.3 Description and Achievements

The KIT is developing a scheduler in conjunction with a scheduling model inside HAL4SDV. While the scheduler itself is described in other deliverables, this chapter introduces the task model. The model was published in [5] and a case study is made available in [6].

The scheduler used in this work is an extension of HM2GP [5]. The HM2GP framework already provides a rich modelling basis, supporting heterogeneous execution units, multiple shared resources, multi-version tasks, gang tasks, and explicit precedence constraints between tasks. Building upon this foundation, we further extend the model to explicitly support multiple execution modes, allowing the scheduler to adapt task behaviour and implementation choices depending on the selected mode of operation. An overview of the resulting scheduling model is provided in Table 6.

The scheduling model is defined by three fundamental sets: a task set $\mathcal{T}$, a set of execution units $\mathcal{U}$, and a set of shared resources $\mathcal{R}$. All tasks in the system are assumed to be periodic. A subset of tasks is assigned a period D together with a permissible slack γ, which defines the allowable deviation from strict periodic execution. If a task is not assigned a deadline, the scheduling tool does not automatically enforce its execution. Due to the periodic nature of tasks, the period can be interpreted as an implicit deadline when no explicit deadline is provided.

To capture different operational behaviours, execution modes are introduced through the set $\mathcal{M}_n$. Each execution mode $\mathcal{M}_n$ defines the n-th mode by assigning a set of available task versions to each task $\tau_i \in \mathcal{T}$. In other words, for a given mode, each task may be executed using one of several alternative versions, enabling flexible trade-offs between resource usage, execution time, and scheduling feasibility.

Each version v of a task $\tau_i \in \mathcal{T}$ is described using five defining elements. First, a worst-case execution time (WCET) estimate C_i^v characterizes the maximum execution time of the task version. Second, a set of resource access phases A_i^v specifies when and how the task accesses shared resources, with no restriction on the temporal placement of these phases within the task execution. Each access phase is further constrained by resource-specific bounds B_i^v, which limit the fraction of available resource capacity that may be consumed. At any given time step, a phase may use between zero and the full available resource capacity, with each access phase $A_i^v(r)$ being assigned a value in the interval [0, 1] to indicate this fraction.

Third, a set of precedence constraints P_i^v defines the allowable temporal relationships between the start times of jobs belonging to different tasks. These constraints ensure that required execution orders and timing relationships are respected. Fourth, each task version specifies a set of valid execution units $U_i^v \subseteq \mathcal{U}$, restricting where the task may execute depending on the selected version.

Based on this information, the scheduler constructs a concrete schedule by assigning three values to each job j of each task τ_i. The start time $\theta_j(\tau_i)$ determines when the job begins execution, the version selection $\lambda_j(\tau_i)$ identifies which task version is executed, and the execution unit assignment $\phi_j(\tau_i)$ specifies the hardware unit on which the job runs.

Schedule generation is subject to several global constraints to ensure correctness and feasibility. First, the total usage of each shared resource is bounded such that, at every time

step, the sum of all resource accesses does not exceed one. This is enforced by ensuring that for every resource $r \in \mathcal{R}$ and each time step $t \in [0, t_p)$, the accumulated values of B_i^v across all access phases A_i^v of all jobs scheduled at that time remain within this bound.

Second, exclusive use of execution units is guaranteed by preventing overlapping execution intervals of jobs assigned to the same unit. As a result, no two jobs may execute concurrently on the same execution unit. Third, all jobs that are required to be scheduled—either because their corresponding task has an assigned period $D(\tau_i) \neq \emptyset$ or because they are constrained by precedence relationships—must satisfy the precedence constraints $P_i^{\lambda_j(\tau_i)}$ of the selected task version. Finally, the scheduler enforces that each job is assigned only to an execution unit that is valid for the chosen version, ensuring consistency between version selection and hardware capabilities.

Table 6: Model Components

Model Component	Description
$\mathcal{T} = \{\tau_i \mid i \in [0, N_{\mathcal{T}})\}$	Task set
$\mathcal{U} = \{u_i \mid i \in [0, N_{\mathcal{U}})\}$	Unit set
$\mathcal{R} = \{r_i \mid i \in [0, N_{\mathcal{R}})\}$	Resource set
$D : \mathcal{T} \to \mathbb{Q}^+ \cup \emptyset$	Task period
$\gamma : \mathcal{T} \to [0, 1]$	Allowed slack
$\mathcal{M}_n : \mathcal{T} \to \wp(\mathbb{N})$	Description of the n-th execution mode
$C_i^v \in \mathbb{Q}^+$	WCET estimation of task τ_i, version v
$A_i^v : \mathcal{R} \to \wp(\mathbb{Q}^+)$	Resource access phases of task τ_i, version v
$B_i^v : \mathcal{R} \to [0, 1]^{\{}$	$A_i^v(r)$
$P_i^v : \mathcal{T} \to \wp(\mathbb{Q}^+)$	Precedence constraints of task τ_i, version v
$U_i^v \subseteq \mathcal{U}$	Valid execution units for task τ_i, version v
t_p	Hyper period
$\theta_j : \mathcal{T} \to [0, t_p)$	Start time of the j-th job
$\lambda_j : \mathcal{T} \to [0, N_v)$	Selected version for the j-th job
$\phi_j : \mathcal{T} \to [0, N_{\mathcal{U}})$	Selected execution unit for the j-th job

2.6.4 Current Release

The scheduling tool (and with that the model) was published in [5] [6]

2.6.5 Interactions with other tools

Using the represented model and tool, KIT was able to generate schedules for a UNIMORE RoboRacer task stack and was able to deploy synthetic task sets onto existing works from FZI, as shown in [6].

2.7 LLM-based test generation

2.7.1 Contributing Partners

TUM, Mercedes-Benz AG and Ferdinand-Steinbeis-Institut der Steinbeis-Stiftung

2.7.2 Category

Test and Verification

2.7.3 Description and Achievements

Regarding the testing and simulation, TUM collaborated with other partners: Mercedes-Benz AG and Ferdinand-Steinbeis-Institut der Steinbeis-Stiftung on Req2Road (Requirements-to-Road) workflow, as depicted in Fig. 7. It is a GenAI-based process that transforms specification artifacts into test scenarios for Software-Defined Vehicles (SDVs).

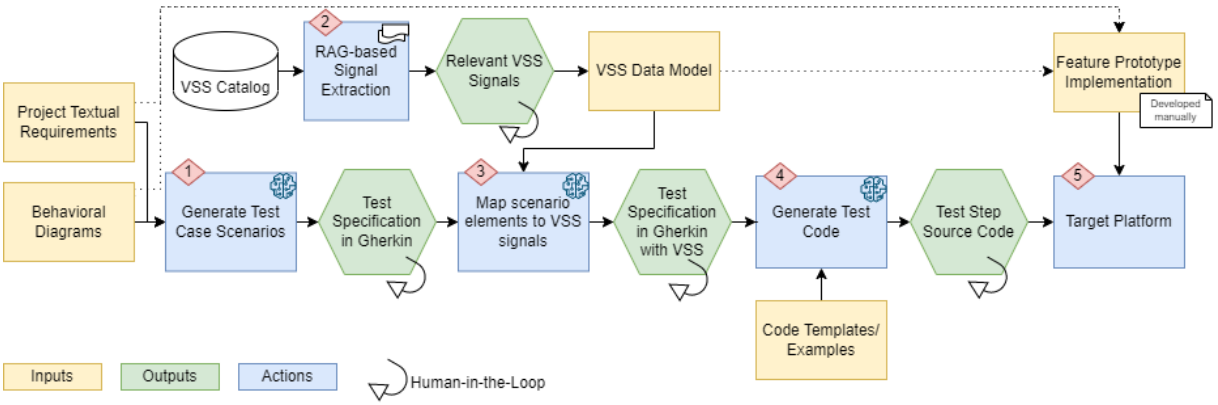

Figure 7: GenAI-Based Test Pipeline Req2Road workflow: requirements to executable test artifacts

The pipeline consists of four main phases:

- Initial Gherkin scenario generation,
- Vehicle Signal Specification (VSS) signal mapping,
- Refined Gherkin scenario generation,
- Test code generation

The process starts from project-specific textual requirements and behavioural diagrams. These inputs, together with the VSS catalogue, form the basis for test generation. A Retrieval Augmented Generation (RAG)-based signal extraction module identifies relevant VSS signals as an intermediate result and constructs a corresponding VSS data model, which guides the subsequent pipeline stages.

Two AI-assisted phases are dedicated to Gherkin scenario generation. In the initial phase, preliminary Gherkin test scenarios are derived from textual and behavioural specifications, serving as a starting point for Human-in-the-Loop review and correction. In the refined phase, the corrected scenarios are mapped to VSS signals using a combination of LLMs and VLMs. This enables both semantic and visual grounding of scenario elements. The outcome is a VSS-enriched Gherkin test specification that ensures consistency between test logic and vehicle signal semantics.

Co-funded by the European Union

In the final phase, executable test code is generated using predefined code templates and deployed to the target platform. A manually developed feature prototype acts as the system under test. In the workflow illustration, colour coding distinguishes inputs (yellow), outputs (green), and processing steps (blue), highlighting the collaborative interaction between human expertise and AI-based automation. This pipeline bridges requirements interpretation, model-based signal mapping, and automated test realization.

When it comes to test generation, the first step of the pipeline is the generation of test cases in Gherkin syntax based on natural-language requirements and the escalation flowchart. To generate Gherkin scenarios from the natural-language requirements and provided flowchart, we used GPT-5 in combination with a Vision–Language Model (VLM). Among the evaluated VLMs, Gemini 2.5 Pro produced the strongest overall results. However, for locally deployable solutions, Qwen 2.5 VL 72B performed best and was therefore selected for integration into the Req2Road pipeline. The use of locally deployable GenAI models is particularly important for handling sensitive, automotive-specific artifacts—such as requirements and user stories—that should not be shared externally.

The primary prompting techniques applied for test case generation were Chain-of-Thought prompting and Few-Shot Learning. The prompt template used for this step instructed the model to act as an automotive software testing assistant and to generate structured test cases in Cucumber Gherkin syntax using the Given/When/Then format. Each generated test case was required to correspond to a specific transition in the escalation flowchart, explicitly reflect timing conditions, and remain consistent with the functional requirements. In cases of ambiguity or inconsistency, the model was instructed to request clarification.

All generated test cases were manually reviewed as part of a human-in-the-loop validation process to identify semantic and syntactic issues. Overall, 32 out of 36 requirements (89%) were transformed into executable Gherkin scenarios without modification. The remaining four scenarios required targeted adjustments, primarily due to ambiguously specified edge cases or requirements containing OR conditions that did not explicitly enumerate all alternatives.

To avoid overwhelming the model and to simplify validation, the initial Gherkin test case generation was intentionally separated from a subsequent refinement step. In this refinement phase, the corresponding Vehicle Signal Specification (VSS) signal paths were injected into the scenarios. For VSS mapping, GPT-4o-mini was selected as the LLM backend, as it provided the best balance between correct matches and false positives on the curated candidate signal pool.

In the test script generation step, GPT-4.1 was used to translate the VSS-enriched Gherkin scenarios into executable Python-based test artifacts. The input to this step consisted of three elements:

- A Gherkin scenario capturing the intended test logic,
- An example of the KUKSA client API in Python, and
- A minimal example using the Behave testing framework.

The KUKSA client API example demonstrated the use of the synchronous simplified API from the kuksa-python-sdk and served as a template for generating code to read and modify VSS values. The KUKSA client was deliberately chosen over the Velocitas framework (commonly used in the digital.auto playground), as direct manipulation of sensor values was essential for simulating the required test conditions. In addition, the synchronous API was sufficient for this use case and avoided the complexity associated with asynchronous interactions.

The Behave framework example illustrated how to structure the test environment and step definitions, including the organization of files such as environment.py and steps/.py*, as well as the integration of the VSS client fixture into the Behave lifecycle hooks.

These three inputs were combined into a structured prompt template. The system message instructed the model to act as an automotive software test engineer, while the human message supplied the Gherkin scenario, the KUKSA client API example, and the Behave framework example. This setup enabled the model to map high-level behavioural specifications to executable code while adhering to established usage patterns of both the KUKSA API and the Behave framework. The outcomes of TUM's work were targeting execution within digital.auto SDV simulation platform.

Initial works on this topic can be found in paper presented at DTF Symposium [4], while the paper containing extended workflow with more detailed evaluation was submitted for CAiSE 2026.

2.7.4 Current Release

We intend to open-source all the related software assets. Currently, HAL4SDV partners can ask for access to the source code at the following address: https://gitlab.lrz.de/hal4sdv/vss-test-generation

2.7.5 Interactions with other tools

Takes the outputs of QA module from D3.4, denoted "2.4- Model-based Representations Tools and QA Plugin" as input in order to generate test code (digitalauto or target platform) based on Gherkin scenarios.

2.8 Simulation of sensor spoofing attacks

2.8.1 Contributing Partners

POLIMI

2.8.2 Category

Test and Verification

2.8.3 Description and Achievements

As computing becomes ubiquitous, Cyber-Physical Systems (CPSs) form the basis of modern infrastructure, from vehicles to smart grids. CPS software is heterogeneous, with tasks having different safety assurance levels. To reduce energy, size, and cost, *consolidation* of software

Co-funded by the European Union

on shared platforms has become a priority. Mixed-Criticality Systems (MCSs) enable this by enforcing temporal and spatial isolation between co-running tasks.

Timing analysis is crucial for estimating Worst-Case Execution Time (WCET), essential for timing correctness. In this task, we focused on probabilistic techniques like Measurement-Based Probabilistic Timing Analysis (MBPTA) which allows less pessimistic results and higher resource utilization than traditional approaches. However, complex mechanisms in modern architectures and operating systems—such as preemption or speculative execution—introduce execution time variability (*jitter*). Another factor that might impact availability when using MBPTA is **poor representativity** during the design phase.

In this case, the system responds by dropping less critical—but potentially essential—tasks to preserve safety-critical functionality, entering a *degraded* state. This availability drop makes representativity a vector for Denial-of-Service (DoS) attacks which require some sort of system compromise. Given the prevalence of sensors in modern CPSs and memory corruption vulnerabilities, we focus on two emerging threats: Data-Oriented Attacks (DOAs) and Sensor Spoofing Attacks (SSAs). These allow adversaries to exercise previously unseen execution paths, pushing the system into unexplored operating regions.

In this task we addressed two questions:

- RQ_1: To what extent can data-oriented and sensor spoofing attacks cause timing anomalies by exploiting poor input representativity in probabilistically time-analysed mixed-criticality systems?
- RQ_2: Given heterogeneity of task temporal behaviour in mixed-criticality systems, can robustness of state-of-the-art timing anomaly detection approaches—based on execution time iid-ness—be improved by detecting deviations in tasks' execution time distributions?

To answer this questions, we have built a **simulated test-prototype** based on a 32-bit RISC-V MCU with out-of-order speculative execution, 8KB set-associative cache (32B line size), and 4MB main memory. The platform was emulated in Renode running Zephyr RTOS. Benchmarks were rewritten as Zephyr applications with a *sensor* task simulating readings and a *control* task running benchmark logic. Output traces were fed into the TBM µ-architectural simulator for realistic execution times.

Preliminary results confirmed our hypotheses. Timing jitter from execution paths under poor input representativity can effectively enable DoS attacks in mixed-criticality systems. Data-oriented and sensor spoofing attacks showed significant potential for timing anomalies, even on simple algorithms, with a maximum overrun of 229.68%.

We have also found that a probabilistic analysis of WCET distribution showed 61% reduction in F_1 score standard deviation compared to current state-of-the-art, demonstrating reduced sensitivity to execution time variability. This robustness is desirable in mixed-criticality settings with diverse timing behaviours. However, detection latency increased by one order of magnitude.

2.8.4 Current Release

We don't plan to open source the testbench, however HAL4SDV partners can ask for access to the source code at the following private repo: https://github.com/polimi-aos-lab/timecrush-poc

2.8.5 Interactions with other tools

n.a.

3. Conclusion

This deliverable reports the modelling and simulation activities carried out in Task 3.2 of the HAL4SDV project, with a specific focus on enabling the integration of open-source RISC-V hardware platforms into an SDV development ecosystem and virtual lab/playground. The reported outcomes address key challenges of RISC-V adoption in automotive contexts by providing reusable virtual prototyping targets, architecture-level exploration and optimisation capabilities, and methods that support predictable integration and systematic validation on heterogeneous and mixed-criticality platforms.

The activities covered span multiple abstraction levels. At the virtual prototyping layer, the Carfield RISC-V automotive platform is integrated into the GVSoC simulator to provide a shared, executable "hardware target" for cross-partner experimentation and early software/platform evaluation. In parallel, a reproducible Xen-based virtualisation baseline and observability toolchain are established on Intel and Arm, designed to be portable towards RISC-V as hypervisor and guest-driver maturity increases. At higher levels, the deliverable includes extensions for architecture definition and design space exploration (e.g., A-DECA), together with AI-assisted and model-based methods that translate requirements into explicit constraints and support interface compatibility checking during system evolution. Complementing these, dedicated scheduling, testing, and verification contributions, such as heterogeneous task modelling and schedule generation (HM2GP), GenAI-based requirements-to-test workflows (Req2Road), and security-oriented simulation of sensor spoofing and data-oriented attacks, illustrate how functional, timing, and robustness aspects can be assessed early and iteratively.

Overall, the results presented form a coherent and cohesive set of RISC-V–focused building blocks that complement the cross-platform tooling of Task 3.4 and contribute to WP3's objective of integrated HAL4SDV toolchains and reusable SDV development workflows.

Co-funded by the European Union

4. References

[1] F. Pan, N. Petrovic, V. Zolfaghari, L. Wen and A. Knoll, "LLM-enabled Instance Model Generation," 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Grand Rapids, MI, USA, 2025, pp. 586-595, doi: 10.1109/MODELS-C68889.2025.00082.

[2] Autonomous Vehicle Computing Consortium, TR-001: Conceptual Architecture for Automated and Assisted Driving Systems [Online]. Available: https://avcc.org/tr001/ Accessed: Jan. 11, 2026.

[3] N. Petrovic, L. Mazur and A. Knoll, "LLM-Based Approach for Enhancing Maintainability of Automotive Architectures," 2025 2nd International Generative AI and Computational Language Modelling Conference (GACLM), Valencia, Spain, 2025, pp. 279-284, doi: 10.1109/GACLM67198.2025.11231880.

[4] D. Zyberaj, L. Mazur, N. Petrović, P. Verma, P. Hirmer, D. Slama, X. Cheng, and A. Knoll, “GenAI-based test case generation and execution in SDV platform,” arXiv preprint arXiv:2509.05112, Sep. 2025.

[5] STAMMLER, Matthias; LESNIAK, Fabian; KUMAR, Niraj; EASWARAN, Arvind and BECKER, Jürgen: “HM2GP: A Multi-Version Task Scheduler with Extended Precedence Constraints on COTS SoCs”. In: 2025 IEEE 38th International System-on-Chip Conference (SOCC). Dubai, United Arab Emirates: IEEE, Sept. 2025, pp. 1–6. DOI: 10.1109/SOCC66126.2025.11235395.

[6] STAMMLER, Matthias; SCHEIDT, Henrik; HARBAUM, Tanja; BECKER, Jürgen; DUDZIK, Konstantin; PAZMINO, Victor; GAVIOLI, Federico; BURGIO, Paolo; EASWARAN, Arvind and ECKEL, Andreas: “Multi-Partner Project: Scheduling-Deployment Workflow for Autonomous RoboRacer Driving Stacks in the HAL4SDV Project”. In: 2026 Design, Automation & Test in Europe Conference & Exhibition (DATE). Verona, Italy: IEEE, Apr. 2026, pp. 1–6. DOI: TBD