Safe and Explainable
Critical Embedded Systems based on AI

D5.2 Case study porting and integration

Version 1.0

Documentation Information

Contract Number	101069595
Project Website	www.safexplain.eu
Contratual Deadline	31.05.2025
Dissemination Level	PU
Nature	R
Authors	Frank Geujen(NAV), Gabriele Giordana (AIKO), Joanes Plazaola (IKR)
Contributors	Enrico Mezzetti (BSC), William Guarienti (EXI), Lucas Tosi (NAV), Rob Lavreysen (NAV)
Reviewed by	Thanh Bui (RISE)
Keywords	Safe, Explainable, AI, Space, Automotive, Railway

This project has received funding from the European Union's Horizon Europe programme under grant agreement number 101069595.

Change Log

Version	Description Change
V0.1	First draft
V0.2	Reviewed version
V1.0	Final version

Table of Contents

List of Figures

Executive Summary

This report presents the outcomes of integrating AI safety components into a Minimum Viable Product (MVP) demonstrator and most importantly three domain-specific demonstrators—**space**, **automotive**, and **railway**—as part of the SAFEXPLAIN project, funded by the EU Horizon Europe programme. Each demonstrator applies Safety Pattern 2, where AI contributes to decision-making but does not solely determine system behaviour, ensuring increased traceability and safety.

In the **space** use case, pose estimation for satellite docking is achieved through diverse AI models and redundancy, integrating real-time diagnostics and anomaly detection. The **automotive** demonstrator implements DS3.1-compliant logic using YOLOS-Tiny and VAE-based supervision to detect pedestrians and ensure braking safety in simulated driving scenarios on embedded hardware. The **railway** case study combines stereo vision with depth estimation to identify obstacles on tracks, leveraging multiple redundancy strategies and visual simulation in Unreal Engine.

Across all domains, the systems integrate layered diagnostics, real-time supervisor monitoring, fallback mechanisms, and embedded deployment on platforms like Jetson Orin AGX. These results demonstrate the portability, robustness, and explainability of the SAFEXPLAIN safety framework in safety-critical, AI-enabled applications.

1 Introduction

This deliverable (D5.2) presents the status of the SAFEXPLAIN case studies, focusing on the integration and porting of system components developed across the project. It demonstrates how the Safety Pattern 2 (SP2) architecture—defined in earlier work—has been applied to an MVP demonstrator and three demonstrators in representative domains: space, automotive, and railway.

The objective of this phase is to bring together AI components, diagnostic mechanisms, supervision functions, and control logic into cohesive, operational demonstrators. Emphasis is placed on achieving modular integration, ensuring platform compatibility (including embedded execution), and preparing each case study for validation.

D5.2 builds on preparatory work conducted in WP2, WP3, WP4, and early WP5 activities. The reported results mark a significant step toward the final evaluation and verification stages, confirming that integration is on track and the system's behaviour is reproducible and portable across domains.

2 MVP Demonstrator

The MVP demonstrator aims to provide a simplified yet comprehensive representation of a reference AI-based system, highlighting SAFEXPLAIN technologies and tools, together with the platform's key resources. Initially conceived as Proof of Concept (PoC) to guide the implementation of platform features, it matured into a more robust and consolidated example of application of the SAFEXPLAIN approach and is now considered an open demonstrator that can be exploited to share and promote the project results outside the consortium.

The demonstrator models an application scenario that meets Safety Pattern 2 (SP2[1]) specifications, where the AI component is not exclusively responsible for the system behaviour but can contribute to the decision process and ultimately to the operation. This reference setup includes instantiations of the required software packages, communication interfaces, instrumentation, and a standardized parameter configuration, ensuring compliance with the selected safety pattern.

As a close collaboration among WP2, WP3, WP4 and WP5, further details of the implementation and demonstrator feature are available in the deliverables D4.2[2] and D3.3[3].

2.1 Architecture

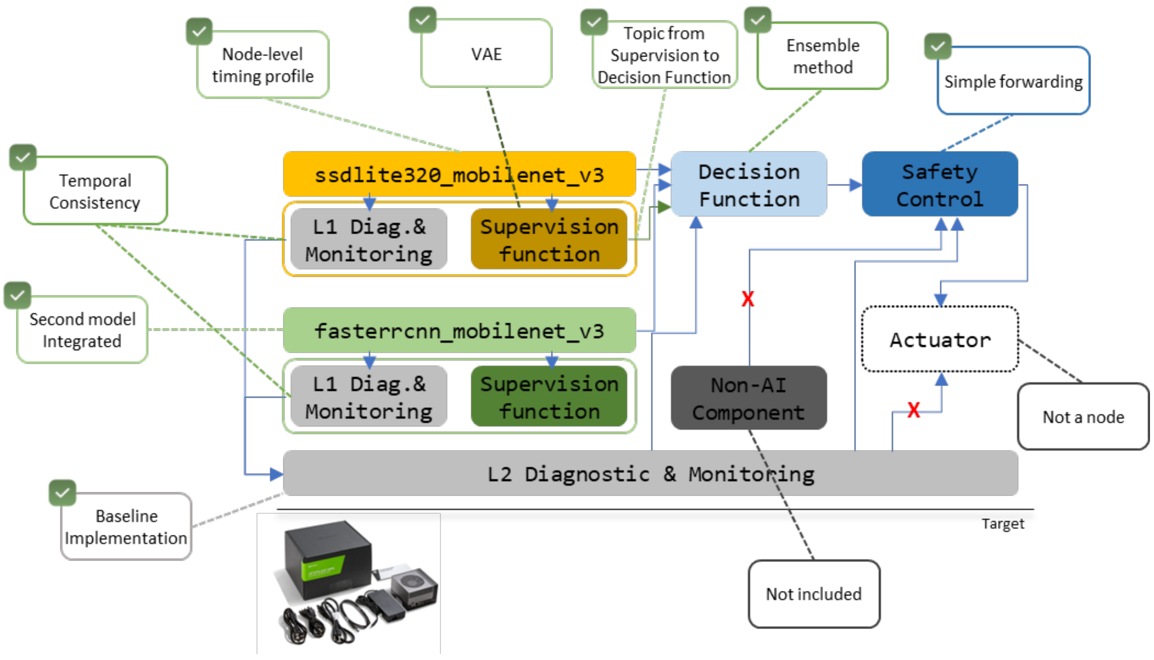


Figure 1. Demonstrator architecture

Figure 1 shows the architecture of the demonstrator system, integrating the AI models and all components required by the safety pattern to ensure the system safety. A lightweight implementation was preferred to showcase the functioning without introducing excessive complexity. The details are reported in the following subsections. Accordingly, the Demo is focusing on the AI-subsystem alone and does not consider a traditional non-AI sub-system, which is instead the main responsible for the system operation according to the SP2 semantics. Non-AI subsystems safety can be guaranteed and assessed within the traditional Functional Safety approach and thus not in focus of this demonstrator.

The input stream of the system comprises timeseries sequences of images generated from the dataset pipeline of AIKO[4]. These datasets simulate a spacecraft agent navigating on trajectories and taking images with a monocular grayscale camera. In the field of view, a target satellite is present and represents the aim of the object detection system deployed in the demonstrator.

2.1.1 AI/ML constituent

The AI (DL) models employed for the foundational constituents of the system were chosen among open-source object detection models, to allow full accessibility of the whole system. They employ Deep Learning techniques (Convolutional Neural Networks – CNN – and Multilayer Perceptron MLP in particular) to identify in an image some specific targets that they were trained to recognise. In the vast range of choices, lightweight and easily portable models were preferred, and were selected from the TorchVision library to facilitate compatibility with the SAFEXPLAIN middleware.

Two different models are trained for object detection and run in parallel and independent nodes:

- **SSDLite**: ssdlite320_mobilenet_v3_large, a fast model relying on a MobileNet backbone, with reduced computational needs which is optimized for embedded environments, and thus particularly suitable in the project context (see documentation at [5])
- **Faster CNN**: fasterrcnn_mobilenet_v3_large_320_fpn, also using a MobileNet backbone. This model is more accurate than the SSDLite but slower, since it requires more resources at inference time (see [6]).

2.1.2 Diverse redundancy

The redundancy of the AI/ML constituent is ensured by the presence of two DL models, with different architectures and separate training, that provide the same functionality to the system (see previous subsection).

They are implemented in two different ROS2 nodes, which are segregated by the SAFEXPLAIN middleware, and their outputs go through a process of comparison and reconciliation carried out in the Decision function.

2.1.3 L1DM mechanisms

The L1 Diagnostics and Monitoring component is required to monitor inputs and AI constituents to ensure that runtime errors and model insufficiencies do not harm the system's safety goals. In this case, the component implements two mechanisms:

- **Input temporal consistency**: this technique implements verification on two consecutive input images, ensuring that their difference is below a certain threshold estimated from the training dataset of the system; this allows to detect lost frames, sensor lagging and faults and to avoid that the system is fed with inconsistent, and thus erroneous, input.
- **Verification of the AI/ML constituent**: the health status reported by the two nodes implementing the AI/ML constituent is also received by the component and constant monitoring is performed.

All results of the monitoring process are fed to the Decision function, which integrates them with the other outputs from supervisory monitors and AI models to provide a consistent and informed system output.

2.1.4 L2DM mechanisms

L2 Diagnostic and Monitoring mechanisms aim to detect runtime errors on the platform hardware and software components. The health of all the other components is constantly shared and collected by the L2DM component, which provides the information to the Decision function node for verification of the overall execution.

2.1.5 Supervision function

Supervision function component consists of several supervisory monitors which have also been described in deliverable D3.3 from the theoretical perspective. In this section, technical details of that realization in the MVP demo are provided.

2.1.5.1 Anomaly detectors

The 'detect_anomaly' function takes an input and a pre-trained Variational Autoencoder (VAE) descriptor model, preprocesses the image, runs it through the model to get a reconstruction, and then calculates a Mean Squared Error (MSE) between the original and reconstructed data. This MSE error serves as an anomaly score – higher scores indicate more likely anomalies.

3 separate VAE models have been trained to handle domain uncertainty related anomalies of input images, neuron activation patterns (extracted from a selected layer in SSDLite model head.classification_head.module_list[0][0][2]), and a cropped detected object. They are named input, model activation and output anomaly detectors respectively.

2.1.5.2 Surrogate model

Surrogate model is implemented as a trained Random Forest (RF) regression model to predict object bounding boxes based on traditional image features. It extracts features related to key points, Gaussian Mixture Model (GMM) parameters, and Local Binary Patterns (LBP) from the same input satellite images. The trained surrogate RF model provides interpretable logics that can be tested and certified during the AI-FSM phases.

2.1.5.3 Uncertainty-aware model

This uncertainty-aware model builds upon the main model (SSDLite) architecture by incorporating dropout layers into its feature maps. This addition enables the estimation of epistemic uncertainty in the model's predictions. Leveraging the MobileNet backbone of the original SSDLite, the model extracts feature and then applies dropout to each feature map before passing them to the SSD prediction heads. To quantify uncertainty, the model runs multiple stochastic forward passes (with varying dropout masks) on the same input images. The resulting distribution of detected object bounding boxes is then used to compute a 95% Confidence Interval (CI95), represented by inner and outer bounding boxes that define the range of likely object locations.

2.1.6 Decision Function

The decision function is built based on an ensemble model to aggregates the outputs of multiple prediction instances, combining detected object bounding boxes and associated anomaly scores. The function operates by comparing predictions against a pre-computed baseline stored in a JSON file. This baseline represents the expected boundaries for in-distribution data and serves as a reference for anomaly detection. A metric calculation based on this baseline is performed during initialization.

The function receives prediction outputs (bounding boxes, labels, confidence scores) from diverse redundant main models alongside with the anomaly scores provided by the three anomaly detectors. It uses these inputs to determine if a given prediction is anomalous based on its deviation from the baseline. If anomaly is found, the function will also provide associated explanations.

2.1.7 Non-AI subsystem

In the demonstrator architecture, a non-AI subsystem was not perceived to be necessary, since the models provided in the AI/ML constituent already implemented the functionalities required by the operational scenario. Since the safety pattern architecture is a reference adaptable to the specific needs of the case, the component was excluded from the final integration.

2.1.8 Safety Control

To avoid unnecessary complexities, the safety control did not implement explicit features. The Decision function computes an aggregate bounding box for the target, while integrating outcomes from diagnostics and monitoring components, and provides a dependable output. The safety control node, in this case, received the result and forwards it to become the final output of the system.

2.2 Timing profiling

The Demo exercises the transparent support for timing profiling of single nodes, offered by the Middleware layer. The profiling in the Demo is addressing the node implementing the Single Shot Detector (SSDLite) model and is enabled by the middleware configuration script defined for the Demo.

The Demo also demonstrates the application of the Timing Analysis methods described in deliverable D4.2[2]. The profiling information gathered on the node is automatically fed to the RestK script to compute the node's Probabilistic Worst-Case Execution time (WCET) distribution. Finally, the distribution is plotted on a dedicated UI window.

2.3 Demo development and testing

The Demo have been developed on top of the reference Middleware skeleton for Safety Pattern 2. The middleware skeleton already provided a software architectural design where all nodes are providing an empty semantics. The Demo development, therefore, mainly consisted in replacing dummy semantics in the architecture with concrete semantically relevant elements. The integration of these elements followed an incremental approach where the integration of each element was followed by a quick regression testing campaign. Integration started from the core inference model and proceeded by adding the redundant model, and providing the implementation for the supervision function, borrowed from DLLib[3]. Successively, the Demo was extended with L1 and L2 diagnostic and monitoring functionalities, partially building on existing ROS2 support.

Integration testing and validation have been conducted exploiting the functionalities developed in the scope of WP4, for visualization and inspection of the system logs and internal state. More details on logging and visualization support can be found in [2].

3 Space case study

The space case study takes place in a scenario with high relevance for the space industry, a spacecraft agent navigating towards another satellite target and attempting a docking manoeuvre, represented in Figure 2. This is a typical situation of an In-Orbit Servicing mission, where commercial and scientific satellites or space stations have to interact with other assets in a precise, synchronized way.

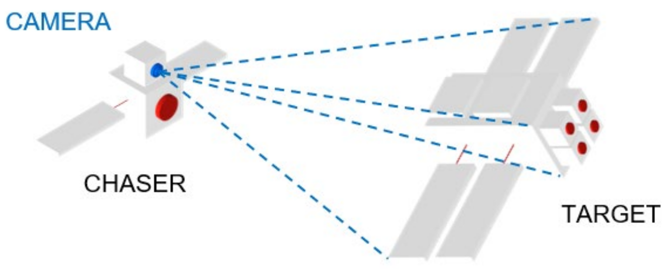


Figure 2. Space scenario for the case study

The agent can exhibit a varying degree of autonomy in operating the trajectory and approaching manoeuvres. Simple sensors such as grayscale optical cameras can provide information on the target and the environment, and that is where the adoption of AI and its safety are paramount for enabling unprecedented capabilities.

The case study envisions a component of the Guidance, Navigation and Control (GNC) system, the pose estimation, shown in Figure 3. It is an essential block of the pipeline, which is required to provide an accurate pose (translation and attitude) of the target, allowing the other components to compute a trajectory and the relative actuator commands to follow it. For further information on the case study scenario, see the project's previous deliverable D5.1[4].

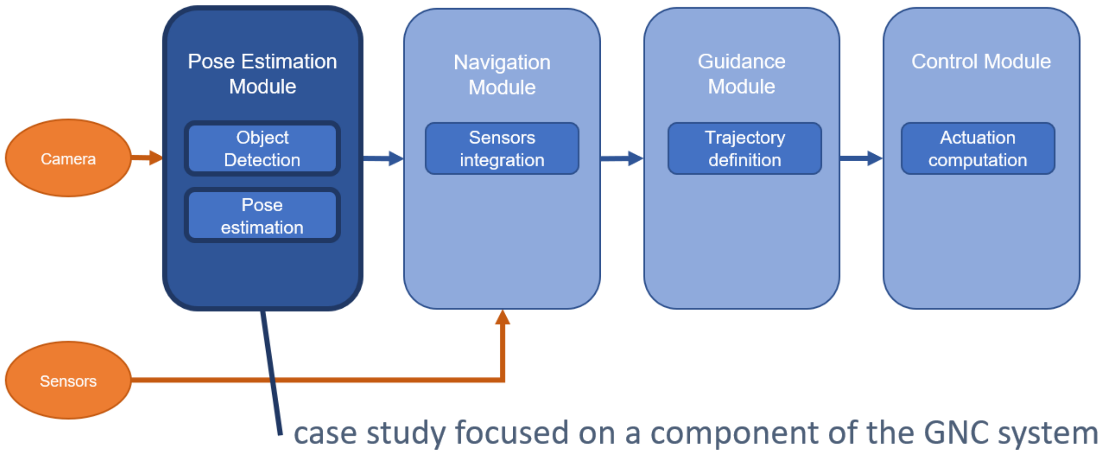


Figure 3. GNC architecture and the pose estimation component

3.1 Architecture

The pose estimation, being the core of the system that was built as the case study, was encapsulated in an architecture comprising different components to meet Safety Pattern 2[1], where AI components have safety-critical responsibilities together with traditional-software ones.

In fact, in the pose estimation, the AI models represent a core feature for the final output, and thus AI has a relevant impact on the critical operations of the system. On the other side, non-AI elements in the component itself are present and provide pose estimation results; furthermore, in the context of the GNC system which would include the pose estimation module, non-AI functionalities would be implemented and would share the safety-critical role.

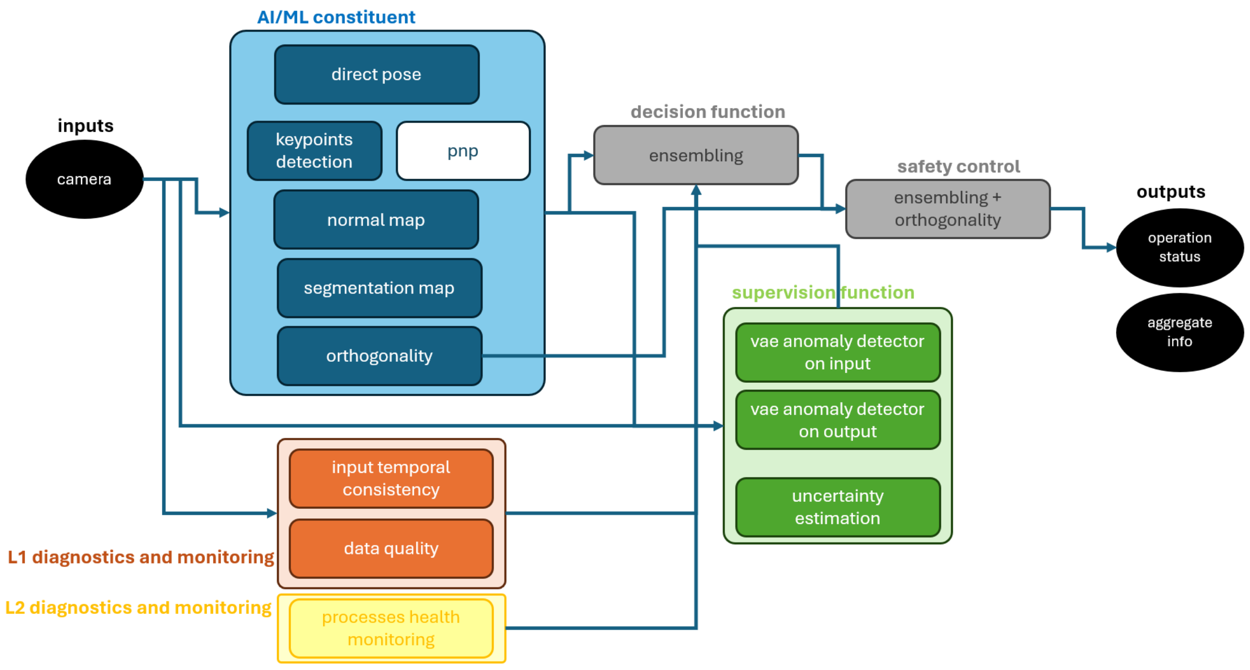


Figure 4. System architecture

The final system setup is illustrated in Figure 4.

The main components are identified by colour and explained in the following subsections, and the main communication patterns are represented by arrows (communication internal to the components is excluded for better visualization, but details are reported in the corresponding subsection).

3.1.1 AI/ML constituent

The case study was prepared during the first phase of WP5, stubbing and preparation of algorithms and datasets, as reported with details in deliverable D5.1[4]. Most of the features have not changed, as shown in Figure 5.

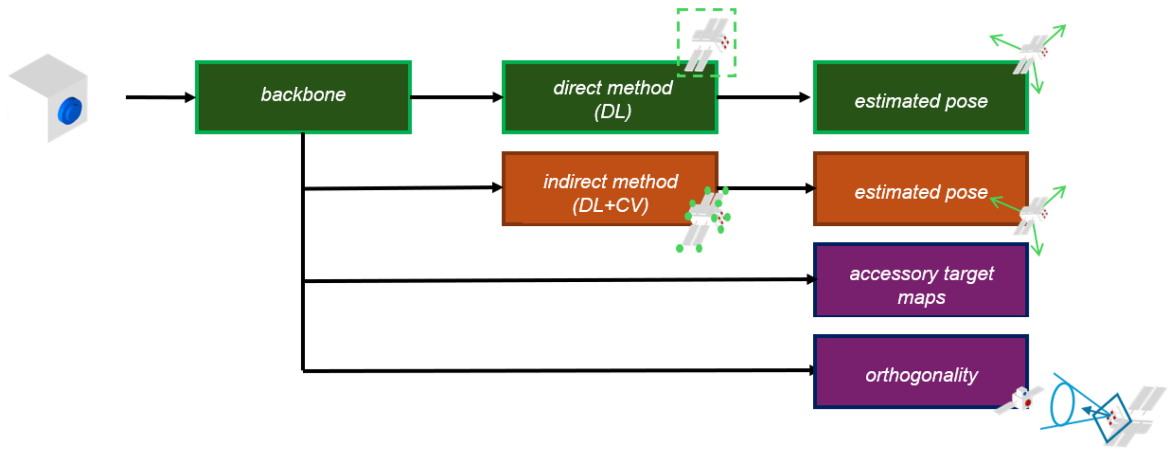


Figure 5. AI/ML constituent modules

The network is a single-input, multiple-output (SIMO) framework, consuming an image and providing pose estimations and secondary outputs. It is based on a unified backbone, relying on an EfficientNet[7]. This model processes the input and distributes its learned features to the different heads, optimising the use of computational resources. The models leaning on the backbone are:

- **Direct method for pose estimation** computes the 6-dimensional pose (3 for position with respect to the agent, and 3 for rotation or attitude) directly from the input, without intermediate explicit steps or outputs. The bounding box is provided together with the pose.
- **Indirect method for pose estimation** computes the same pose estimation with a different technique, adopting two steps:
    - Detection of the key points of the target (the vertices of the asset, spikes of antennas and so on) computing a heatmap of the points and then applying non-maximum suppression (NMS) algorithms.
    - Computation of the pose using a Perspective-n-Point (PnP) algorithm, a geometrical approach taking the key points as inputs.
- **Segmentation map**: provides a binary map identifying the silhouette of the target, a secondary output useful for the identification of the object.
- **Normal map**: provides an RGB mask encoding the orientation of the target surfaces with respect to the target, another secondary output for further insights into the model’s understanding of the environment.

- **Orthogonality to the docking area**: a small, critical model that estimates the deviation from the normal to the docking site on the target, where the agent must lock into it. This allows to verify how much the trajectory of the agent is precise and does not endanger the manoeuvre.

Inputs: input image from the camera sensor node.

Outputs: two pose estimations, bounding box, normal and segmentation map to the Supervision function node for monitoring, and to the Decision function node for ensembling and forwarding; orthogonality to the docking site to the Safety control node for the operation verification and final decision on the system output.

3.1.2 Diverse redundancy

The redundancy mechanism is a first essential tool for ensuring a safe execution in a critical system. In this case, it is implemented in different ways:

The main, critical functionality of the system is the estimation of the pose. For this reason, the computation is implemented in two ways: two different models, with different architectures and different approaches are used (Figure 6); one is a direct, full Deep Learning technique, the other employs different models, both DL-based and non-AI, and it is more interpretable and human-readable.

These two models are implemented in two different ROS2 nodes, which are run in segregation in the SAFEXPLAIN middleware. The two pose estimations are then ensembled and a compared, balanced output is computed by the Decision function. If the estimations though are inconsistent, a malfunction warning is triggered.

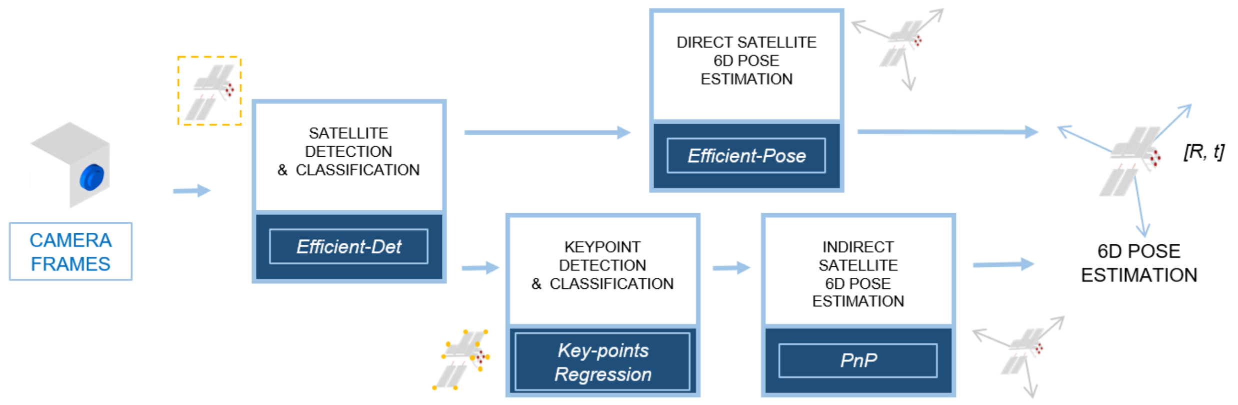

CAMERA FRAMES
SATELLITE DETECTION & CLASSIFICATION
Efficient-Det
DIRECT SATELLITE 6D POSE ESTIMATION
Efficient-Pose
KEYPOINT DETECTION & CLASSIFICATION
Key-points Regression
INDIRECT SATELLITE 6D POSE ESTIMATION
PnP
[R, t]
6D POSE ESTIMATION

Figure 6. Model redundancy for pose estimation

Secondly, other outputs are redundant: the bounding box is computed directly, but it is also inferred from the set of key points, so that two different computations are again provided and compared. The bounding box can then again be compared to the normal and segmentation map to verify their consistency.

3.1.3 L1DM mechanisms

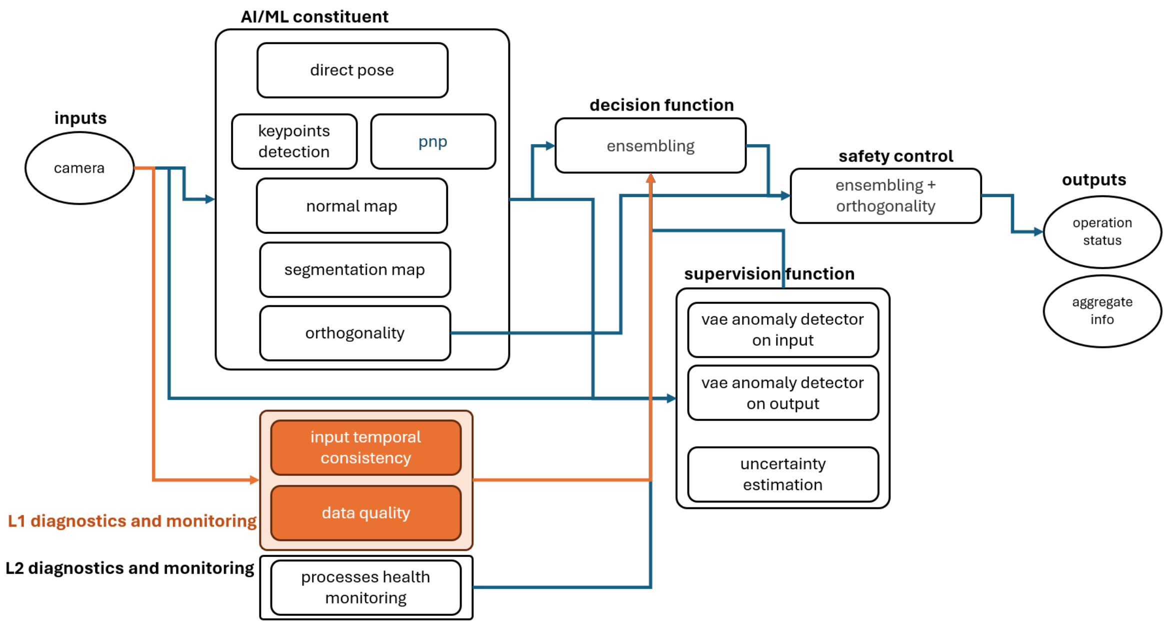


Figure 7. L1DM component

The L1 Diagnostics and Monitoring component (Figure 7) was implemented to protect the system against runtime errors, input anomalies and model insufficiencies. This monitoring activity is performed in parallel to the execution on multiple elements, by the L1DM component implementing the following techniques:

- **Input temporal consistency**: this technique implements verification on two consecutive input images, ensuring that their difference is below a certain threshold estimated from the training dataset of the system; this allows to detect lost frames, sensor lagging and faults and to avoid that the system is fed with inconsistent input.
- **Data quality**: this technique verifies the quality of the image by analysing the edges present in the picture and detecting potential anomalies. Both techniques are implemented using the OpenCV library, a well-known open set of tools for computer vision.
- **Monitoring of the AI/ML constituent health status**: the AI/ML constituent nodes report their health status, to which this component is subscribed to track its correct functioning.

All results of the monitoring process are fed to the Decision function, which integrates them with the other outputs from supervision and AI models to provide a consistent and informed system output.

<u>Inputs</u>: input camera image, AI/ML constituent outputs.

<u>Outputs</u>: nominal or anomalous result for temporal consistency, for data quality, and AI models health; forwarded to the Decision function and consequently to the Safety control nodes.

3.1.4 L2DM mechanisms

The L2 Diagnostics and Monitoring component (Figure 8) is tasked to track the processes in the system and report about their correct functioning, on a lower level than L1DM, less focused on the functional aspect and more on the software components' execution.

The SAFEXPLAIN middleware periodically receives reports from all the components in the system, keeps track of the components that must be launched and be alive at any time, and promptly provides any runtime error or fault to the Decision function, and consequently the Safety control.

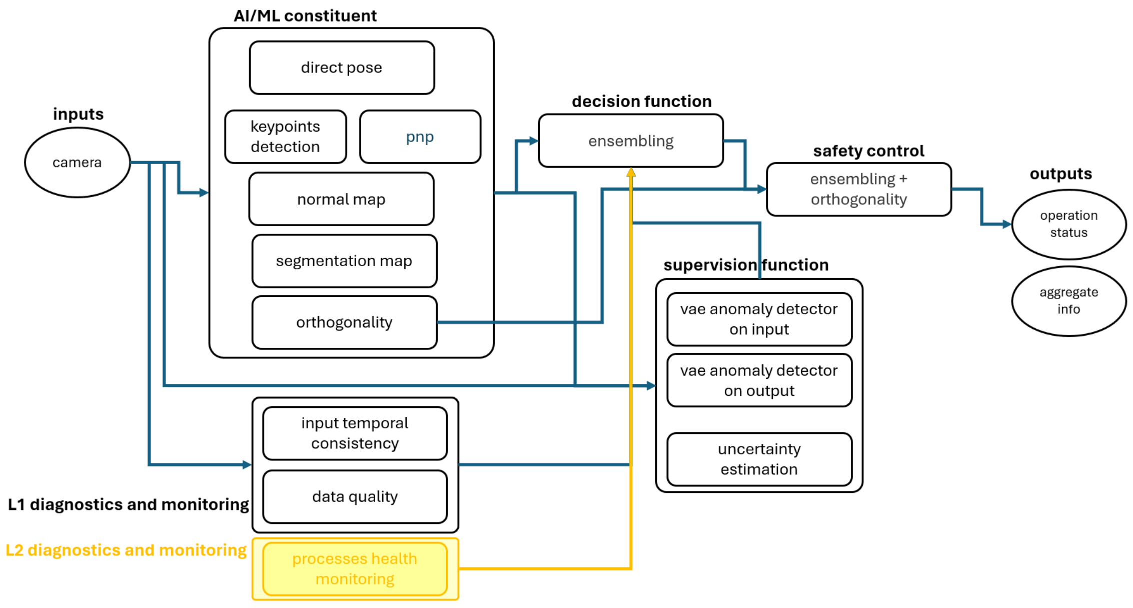


Figure 8. L2DM component

Inputs: health status from all the other components.

Outputs: status of the system sent to the Decision function node and consequently to the Safety control node.

3.1.5 Supervision function

The Supervision function (Figure 9) has the essential role of monitoring the AI/ML constituent on the functional side and protecting the system from any malfunction, insufficiency and anomaly that may arise: the input may be corrupted or the data distribution may differ strongly from what the model was trained on, the model itself may suffer from inference anomalies and provide untrustworthy outputs, and more.

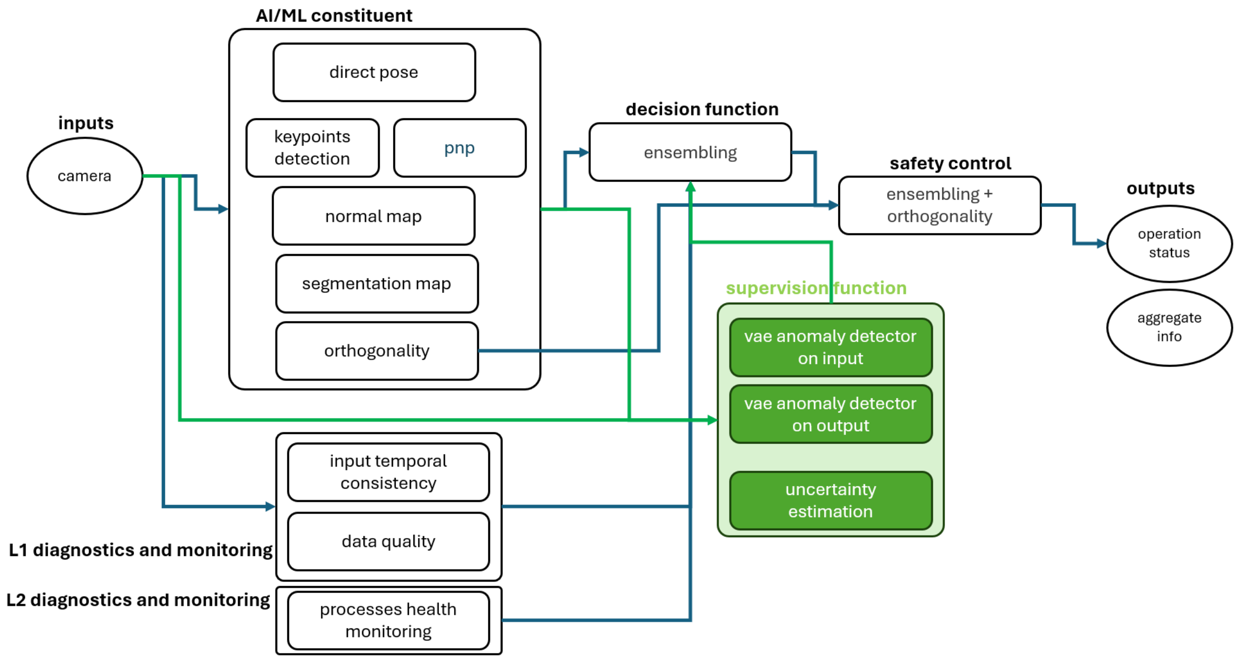


Figure 9. Supervision function component

For this reason, the component implements:

- **Anomaly detection on the input**: a DL model based on a Variational AutoEncoder (VAE) was trained on the same training dataset of the AI/ML constituent; it learned to encode the images in an embedding and then reconstruct them, internalising the distribution of features in the images. At inference time, it tries to reconstruct the image it is given, and the reconstructed image is compared to the original one: the degree of deviation is taken as a score for anomalous input; the more the image is not in the expected distribution, the more the VAE will have difficulty recognising and reconstructing the patterns.
- **Anomaly detection on the output**: the same technique is applied to the output of the AI/ML constituent, to monitor potential anomalies in the model inference: images cropped on the bounding box are used as input, so that another VAE model learns to expect pictures where the target exactly fits the frame. The same mechanism is applied to check the bounding box that is produced by the system.
- Another technique was made available by the work of WP3 (see[3] for further details): the **model uncertainty estimation**. This technique allows to estimate the degree of confidence with which the AI/ML constituent is giving its output; this can provide valuable information about its trustworthiness. The integration of the uncertainty estimation is currently under evaluation since the performance overhead on the system execution must be verified across multiple test cases; the components already instantiate many models and processes, and a fair trade-off between safety, redundancy and computational performance must be reached. If it will be proved to be sustainable, it will be added to the system as a further level of supervision, otherwise the prior modules will be enough to guarantee the functionality.

Inputs: input image from sensor camera node, AI/ML constituent outputs.

Outputs: anomaly scores for input and output, forwarded to the Decision function node.

3.1.6 Decision Function

While the previous components are employed to compute information about the task at hand, the environment and the system well-functioning, the Decision function plays the first essential decisional role: it receives multiple inputs from such components and it aggregates them, producing a coherent and enriched output.

The core of the component (Figure 10) is implemented as an ensemble method, which takes care of comparing the pose estimations from the two methods and verifying that they are consistent with each other. Secondly, the bounding box is compared with the set of key points, to ensure consistency on this aspect, and with the maps provided by the secondary models of the AI/ML constituent. Finally, anomaly scores, health reports and potential warnings are integrated to decide whether all components have been executing without issues and if the previously computed outputs can be trusted and forwarded. Integrated AI outputs and monitoring reports are then delivered to the Safety control node.

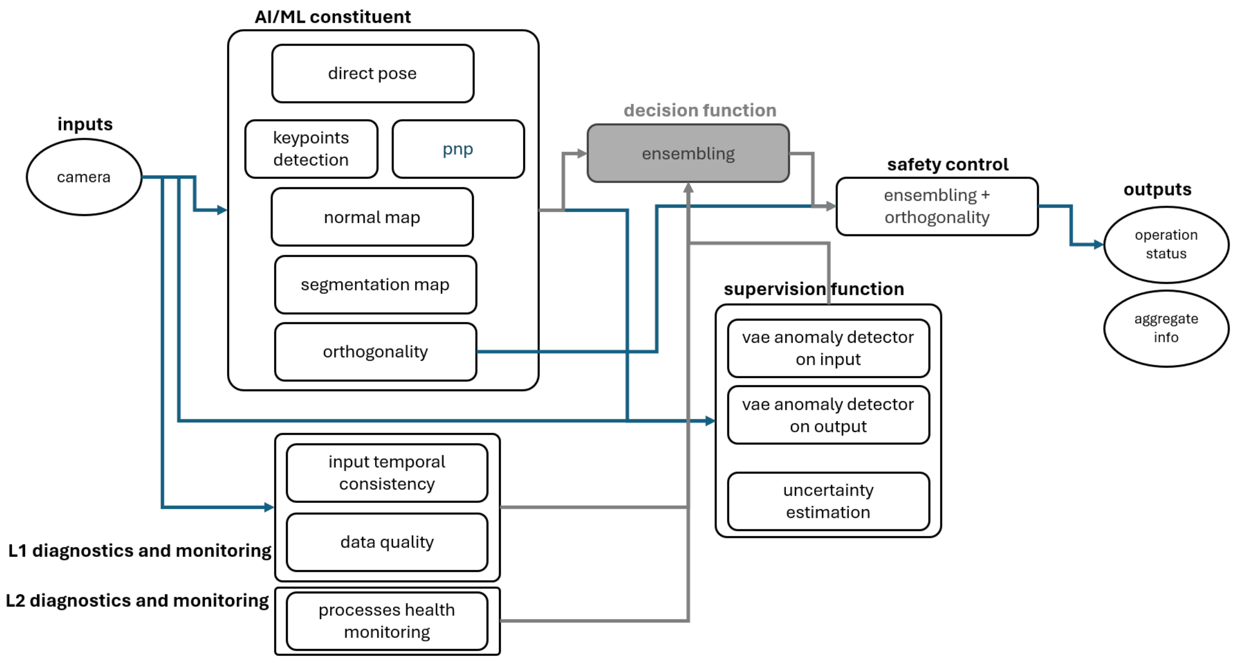


Figure 10. Decision function component

Inputs: AI/ML constituent outputs, L1DM report and health status, L2DM processes health report, Supervision function anomaly report.

Outputs: integrated pose estimation and health and anomalies report, sent to the Safety control node.

3.1.7 Non-AI subsystem

In this system, there is a non-AI subsystem which is part of the pipeline of the core feature of pose estimation: the Perspective-n-Point algorithm.

Functionally, the module is seamlessly integrated into the AI/ML constituent, providing the second step of the indirect method for pose estimation (see subsection 3.1.1). From the perspective of the technology, though, it employs an algorithm which does not involve Deep Learning but just geometrical computations to provide the pose given a set of key points whose position is known in the target, so it falls under this category.

Inputs: set of key points from the key point detection module, internal to the AI/ML constituent.

Outputs: pose estimation, joined to the other outputs from the AI/ML constituent and sent to the same components.

3.1.8 Safety Control

The Safety control (Figure 11) is the last component in the pipeline of the system before the outputs are delivered (to the agent, the user, or the outer systems interacting with it). It hosts the verifications of the outputs of the Decision function and the final considerations necessary to decide if the current operational status is nominal, and the outputs can be delivered as trustworthy, or anomalies and malfunctions hindered the execution and a warning or transition to safe mode must be issued instead.

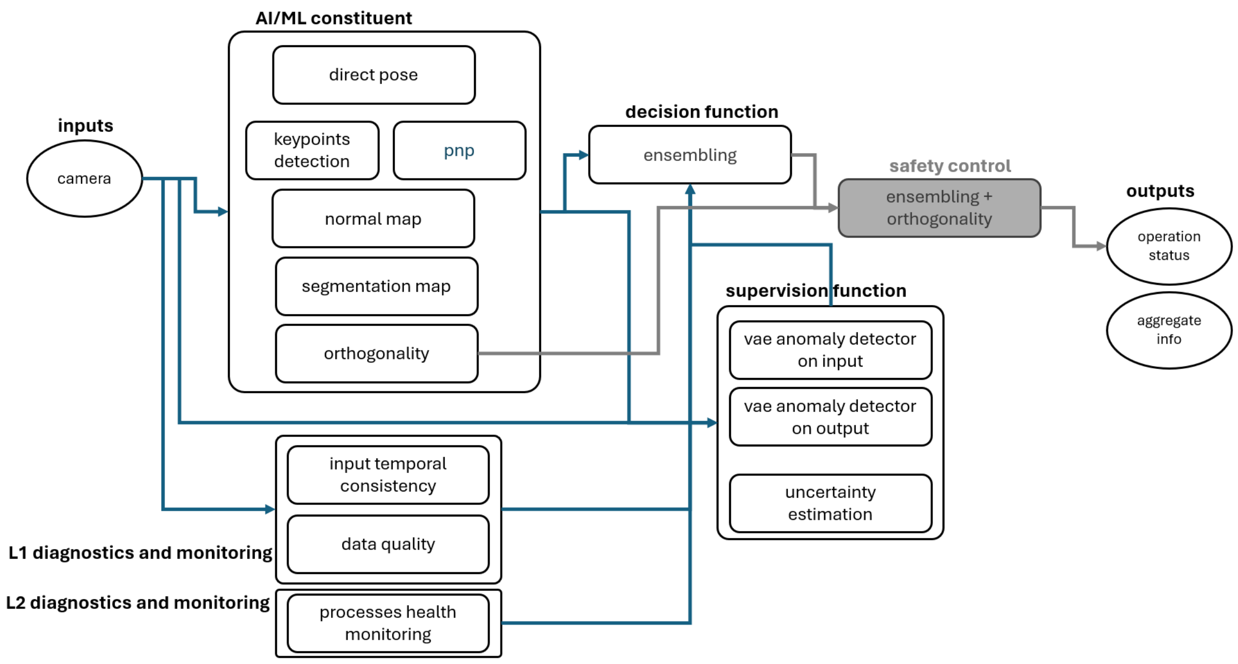


Figure 11. Safety control component

The component bases the decision on analysing the conclusions of the Decision function, including the consistency of the pose estimation results, the health of the processes and the retrieved anomaly scores. In addition, the orthogonality verification is carried out: the corresponding model from the AI/ML constituent delivers its estimation, and the Safety control ensures that the agent is approaching the target with a minimum level of orthogonality to the docking site. If this check is not passed, a relative warning is sent out; otherwise, according to the Decision function's opinion, the pose estimation can be validated and transmitted.

Inputs: Decision function output (pose estimations, L1DM, L2DM and supervision results), orthogonality estimation from AI/ML constituent.

Outputs: operational status of the system (nominal or anomalous), pose estimation and additional information from the decision and monitoring.

3.2 Testing

Along with the integration of the system, incremental testing was conducted to assess the system functionalities. The single components were tested as standalone items, during and after integration, to assess their capability of providing the specified feature (subscription to topics and input receiving, monitoring, supervision or inference depending on the case). As the integration progressed, testing of the system was carried out to verify the interaction of the components with each other.

The tests major need was the generation of datasets of images and relative ground truths; this was already provided in the first phase of WP5, as the data generation pipeline for the space case study provided the capability of generating both random and trajectory-based datasets. The data was then split into training and test sets for later use.

The assessment of the tests was carried out by verifying the correctness and accuracy of the outputs, like the pose estimation error rates, and the expected operational status; if certain input images contained anomalies, the relative components were ensured to output the corresponding warnings. Also, secondary outputs like segmentation and normal maps, bounding boxes and such were taken into consideration during tests inspection, to ensure the consistency of the full pipeline.

The testing activity was carried out locally, especially during development and integration, and after this first assessment it continued the platform: the system instantiation on the Orin was incrementally updated and its correct execution was verified in the embedded environment.

This testing activity was only the first step towards a full verification of the system: the main verification process will be held in the last phase of WP5, evaluation and assessment of the case studies: in the following months, real scenarios simulated from the tests catalogue provided by WP2 will be fed to the system and the results collected. Further improvements of the system and its components will be applied whenever possible, making sure that they have a low impact on the readiness of the case study.

3.3 Challenges and solutions

Integrating such a complex system, in a dedicated environment, compatible with an embedded device and using outcomes from different work packages was a challenge, and it brought about some issues that point to the most delicate parts of the work:

- During the integration of the components, the main design and implementation worry was to keep inputs, intermediate and final outputs consistent with each other, ensuring that information from different images didn’t mix up. For this, it was needed a careful of where needed to **enforce topic synchronisation** and on which arguments of the topics to make the comparison. This led sometimes to changes in the topic message’s structure, adding ids or further information.
- A sometimes-tricky experience was **debugging the system from a certain level of complexity** onward, due to the high number of layers onto which the AI solution relies (the components themselves, the SAFEXPLAIN middleware, ROS2, the operating system, etc.); for overcoming this, we set up set of logging commands and we carefully monitored the components.
- A complex activity was **porting the system to the platform**, ensuring that all dependencies were available and compatible with the embedded environment. We chose to run the application in a containerised environment to have full control over it, and we had to update some dependencies mid-way and rebuild some upper layers. This effort was nonetheless expected from experience on previous porting tasks.
- A delicate matter, that is still under supervision, is **maintaining an adequate performance** of the core function of providing the pose estimation. The AI developers build their standalone model, but integrating it with other components, algorithms and

communication patterns introduces an overhead which must be monitored and tweaked to preserve the functionality. Experiments and testing, as said in the previous section, are still going on and will be the focus of the last period of the project, when we'll be ready for any adjustments and fixes that will prove necessary for the case study improvement.

4 Automotive case study

4.1 Architecture

Figure 12 illustrates the **end-to-end system architecture** implemented in this case study. It captures the **modular ROS 2 pipeline** comprising **simulation input (CARLA)**, **real-time perception (YOLOS-Tiny)** for pedestrian and car detection, **YOLOPV2** for **lane detection**, **decision logic**, **safety supervision** via a **Variational Autoencoder (VAE)**, and final **control output** through the **Controller Node**.

The architecture follows a **layered structure**, where **perception**, **decision**, **supervision**, and **control** are separated into **ROS 2 components** that interact through **structured messaging**. **SAFEXPLAIN middleware** enables **lifecycle coordination**, **health diagnostics**, and **fallback management** throughout the system.

All components are designed to support **traceable decision-making** and **functional safety compliance**. Interactions between nodes are handled via **ROS 2 topics**, including **perception outputs**, **anomaly flags**, **system warnings**, **braking commands**, and **diagnostic reports**.

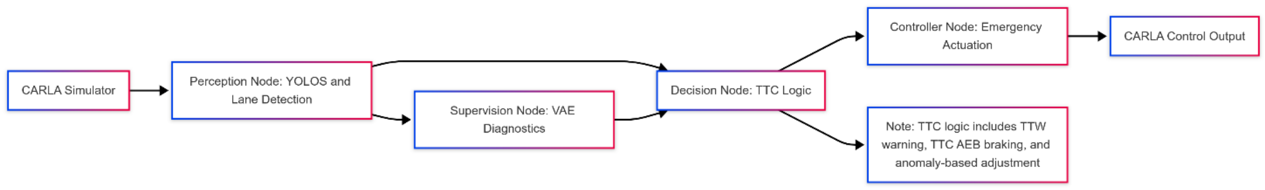

CARLA Simulator
Perception Node: YOLOS and Lane Detection
Supervision Node: VAE Diagnostics
Decision Node: TTC Logic
Controller Node: Emergency Actuation
CARLA Control Output
Note: TTC logic includes TTW warning, TTC AEB braking, and anomaly-based adjustment

Figure 12: End-to-end system architecture deployed on NVIDIA Orin AGX.

The system includes full **DS3.1** logic handling, which relies on key timing thresholds to assess collision risk and determine appropriate responses.

DS-3.1 is a driving safety scenario defined within the SAFEXPLAIN project that addresses autonomous vehicle behaviour in urban environments where pedestrians may suddenly enter the vehicle's path, aligning with safety and regulatory frameworks such as UN Regulation No. 152; it evaluates the system's ability to detect pedestrians early, issue timely warnings, perform emergency braking when necessary, and ensure the vehicle operates within its functional safety limits.

These include **Time-to-Collision (TTC)**, **Time-to-Warning (TTW)**, and **Time-to-Collision Autonomous-Emergency-Braking (TTC AEB),** which are explained below:

- **TTC:** The estimated time before a potential collision occurs between the ego vehicle and an object ahead, assuming constant speeds and trajectories.
- **TTW:** The threshold at which the system should issue a visual or auditory warning to alert the driver or autonomous controller of a potential risk.
- **TTC AEB:** A critical TTC threshold at which the system must initiate emergency braking to avoid or mitigate a collision.

Based on these thresholds, the system distinguishes the following intervention steps:

- **Step 1:** TTC > TTW → Monitoring only
- **Step 2:** TTC == TTW → Warning (visual + audible)
- **Step 3:** TTC ≤ TTC AEB → Emergency braking (≥5 m/s²)

The Decision Node computes TTC and adjusts thresholds when the Supervision Node flags anomalies. The Controller executes the final actuation.

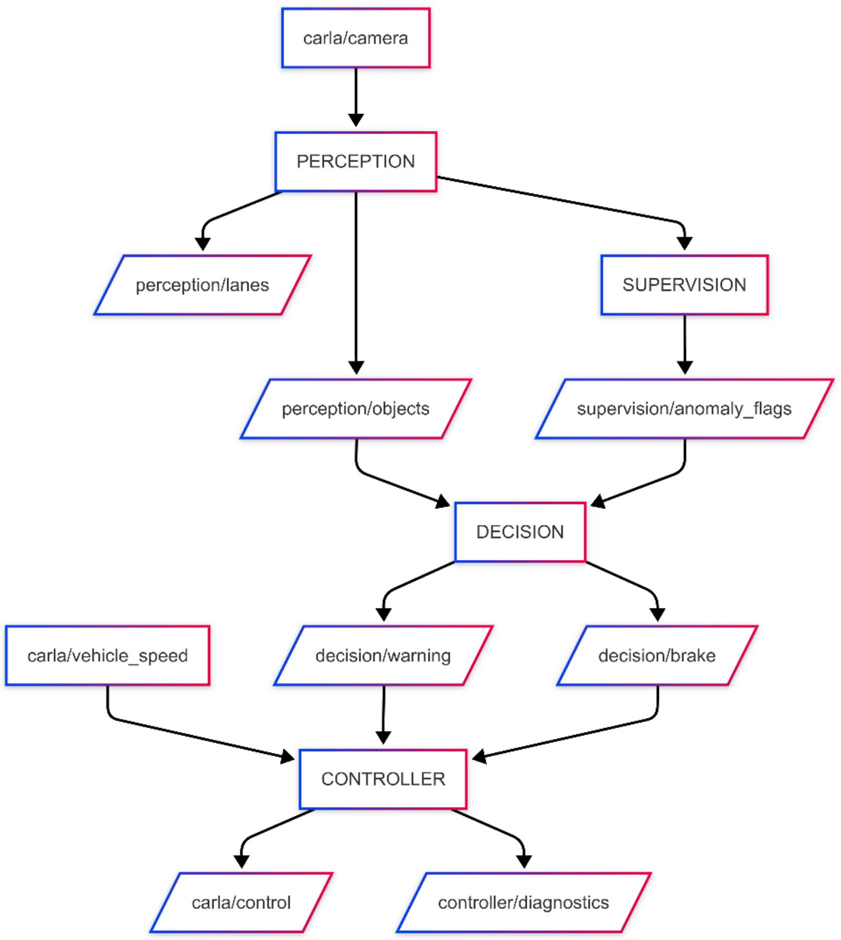


Figure 13: Topic Graph Showing ROS-Based Communication Between Nodes in the Automotive Demonstrator

The topic-level architecture of the automotive demonstrator includes key diagnostic and actuation-related topics that support structured safety logic, traceability, and fallback behaviour. The communication flow covers message paths such as /supervision/anomaly_flags (to signal degraded input conditions), /decision/warning and /decision/brake (to trigger driver alerts or emergency braking), and /controller/diagnostics (to monitor execution latency or delays).

These elements enable the system to respond appropriately across DS3.1 Step 1–3 conditions—ranging from passive monitoring to active braking—while supporting traceable validation of decision logic, anomaly propagation, and control actuation. The topic graph enables a modular and transparent safety pipeline, aligned with **Safety of the Intended Functionality (SOTIF)**[8] and UN Regulation No. 152[9] requirements.

4.1.1 AI/ML constituent

The AI/ML constituent of the system encompasses all components responsible for data-driven perception and unsupervised anomaly detection. It forms the core of the perception–decision loop by enabling real-time understanding of the driving environment and detecting deviations from expected input patterns.

This constituent includes the following key modules:

- YOLOS-Tiny[10] Pedestrian Detector: Deployed within the Perception Node, this model identifies pedestrians and cars in the forward-facing camera feed. The hustvl/yolos-tiny model is based on the YOLO (You Only Look Once) object detection framework. However, unlike traditional YOLO versions, it uses a Vision Transformer (ViT) architecture instead of convolutional layers. This makes it more like transformer-based models like DEtection TRansformer (DETR). The "tiny" variant refers to a lightweight version suitable for faster inference. So, it’s a YOLO-inspired model reimagined with transformer technology. A confidence threshold of 0.5 is used to ensure detection reliability while minimizing false positives. We will validate the coming months if this value requires any tweaking/retraining.
- YOLOPv2[11] Lane Detection Module: This module uses a pre-trained, open-source neural network to extract lane markings. YOLOPv2 stands for "You Only Look Once Perception version 2", a unified vision model designed for autonomous driving perception tasks. The output complements object detections by providing structural context in the scene.

Anomaly detection is handled independently by the Supervision Node, which encapsulates the VAE logic. For full details, refer to the Supervision Function section.

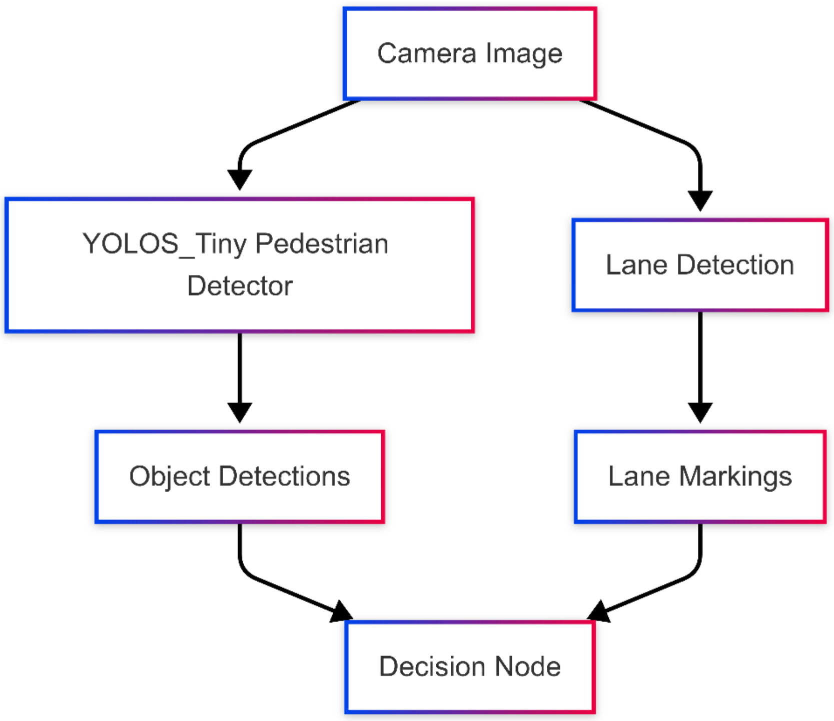


Figure 14: Parallel processing of camera input by pedestrian detector and lane detection module.

4.1.2 Diverse redundancy

To support functional safety in AI-based perception systems, the architecture applies the principle of **diverse redundancy**, as defined in D2.2. This principle involves using **independent implementations of the same functional capability**, such as object detection, to reduce the risk of **correlated failures** due to shared architecture or training biases.

In the current system, **diverse redundancy is achieved** within the object detection component using:

- **YOLOS-Tiny** – a **transformer-based** object detector used for recognizing pedestrians, vehicles, and other road actors.
- **YOLOP-V2** – a **CNN-based** multi-task model that includes object detection, lane detection, and drivable area segmentation.

Although both models operate on the same camera input, their **internal architectures are fundamentally different**, enabling fault detection through architectural diversity. When overlapping outputs (e.g., object bounding boxes) are cross validated, this setup supports early detection of perception inconsistencies and enhances safety.

It is important to note that while **YOLOP-V2's lane and drivable area predictions** provide additional environmental understanding, these functions are **complementary**, not redundant with YOLOS-Tiny. The **diverse redundancy** specifically applies to their **shared object detection role**.

In addition to model-level redundancy, the system includes structural and temporal safeguards:

- Independent diagnostics in various nodes monitor lens obstruction, message delays, and sensor health.
- Timestamp verification and signal freshness checks across modules prevent stale data propagation.

This multi-layered redundancy strategy supports both **L0 diversity** and downstream diagnostics, contributing to the robustness of the overall safety architecture.

4.1.3 L1DM mechanisms

Level 1 Diagnostic Monitoring (L1DM) refers to diagnostics implemented directly within individual functional nodes. These mechanisms are designed to detect internal inconsistencies, sensor quality issues, or unexpected deviations at the component level—before they escalate into system-wide failures. L1DM ensures that each module can self-assess and report its operational integrity in real time.

In the context of this system, the following L1DM mechanisms are implemented:

- **Camera Quality Monitoring (Perception Node):** The system evaluates the incoming camera feed for degradation indicators such as blur, over/underexposure, and brightness shifts. These checks are lightweight and run in parallel with the primary perception logic.
- **Obstruction and Calibration Checks (Perception Node):** The node assesses whether the visual input is partially occluded (e.g. by dirt or objects on the lens) and monitors for camera calibration drift. If detected, the node flags warnings to downstream modules.

- **Anomaly Flagging via VAE (Supervision Node):** The VAE computes anomaly scores frame-by-frame and publishes as /supervision/anomaly_flags, which serve as L1-level diagnostics.
- **Bounding Box Plausibility Validation (Decision Node):** The Decision Node evaluates bounding box size, location, and consistency across frames to filter out implausible object detections (road actors) that may indicate faulty perception output.
- **Signal Delay Detection (Controller Node):** The Controller node continuously compares the time since the last speed-feedback update and the time since the most recent object detection to their respective thresholds (speed_update_threshold and camera_detection_timeout). If either threshold is exceeded, it logs a warning and publishes on /controller/velocity_calculation_warning or /controller/camera_performance_warning, ensuring any critical message delay is flagged and real-time control remains reliable. Each of these mechanisms produces structured warnings or confidence indicators that can be used for:
- Local fallback decisions (e.g. discard corrupted frames)
- Triggering L2 system-level diagnostics
- Supporting traceability and audit of failure conditions

These L1DM capabilities contribute directly to DS3.1 compliance and functional safety assurance by making every node accountable for its input integrity and internal decision quality.

In addition to localized diagnostics, the SAFEXPLAIN middleware plays a coordinating role, aggregating health reports from all nodes and enabling structured recovery, lifecycle control, and traceable logging. This is illustrated in Figure 15.

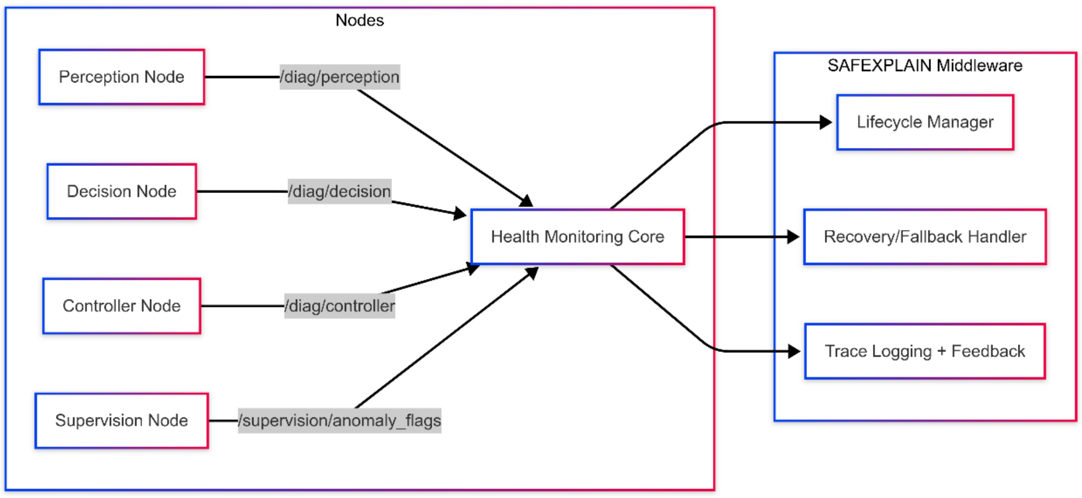


Figure 15: Role of SAFEXPLAIN middleware in aggregating diagnostics, managing recovery, coordinating lifecycle transitions, and maintaining cross-node traceability.

4.1.4 L2DM mechanisms

Level 2 Diagnostic Monitoring (L2DM) refers to system-wide diagnostic mechanisms that aggregate, interpret, and report the health status of the overall pipeline. Unlike L1DM, which operates locally within nodes, L2DM spans across components and relies on structured interfaces and middleware to ensure runtime observability and traceability.

The core function of L2DM is to collect diagnostic outputs from individual modules—such as perception, decision, control, and supervision—and consolidate them into a coherent system status. This information is used for health reporting, fault propagation analysis, and lifecycle management.

In this architecture, L2DM is implemented through the SAFEXPLAIN middleware, which provides a ROS 2–compliant interface for diagnostics. Nodes emit status messages, error flags, and anomaly indicators using structured topics, which are then aggregated and published as part of the system's runtime health state.

Key L2DM functions include:

- **Lifecycle coordination:** Ensuring that nodes transition through initialization, operational, and error states in a traceable and managed way.
- **Cross-node consistency checks:** Verifying that inputs, outputs, and diagnostic states across nodes remain synchronized and up to date.
- **Health state publishing:** Emitting aggregated health reports for external monitoring tools or safety managers.
- **Diagnostic escalation:** Triggering fallback modes, alerts, or safe-state transitions when certain thresholds are breached across multiple components.

The Supervision Node and BaseApplication interfaces play a central role in aligning node behaviour with SAFEXPLAIN's structured diagnostic scheme. By enabling system-level awareness, L2DM mechanisms contribute to DS3.1 compliance, enhance fault traceability, and support safety certification goals.

4.1.5 Supervision function

By decoupling monitoring from control and perception, the Supervision Function introduces a robust and explainable safety layer that reinforces system resilience and diagnostic transparency. The Supervision Node communicates directly with the Decision Node, adjusting reaction thresholds in real-time and publishing frame-specific diagnostics.

At its core, the **Supervision Node** leverages a **VAE** to evaluate incoming camera frames for distributional anomalies. By reconstructing each frame and comparing it to its original, the VAE identifies **out-of-distribution inputs**, **visual degradation**, or **unexpected scene elements** that could compromise perception reliability. The result of this process is communicated via structured diagnostic flags (e.g., /supervision/anomaly_flags). The structure of the Supervision Node—including its use of VAE-generated anomaly scores, and integrated health monitoring—is depicted in Figure 16.

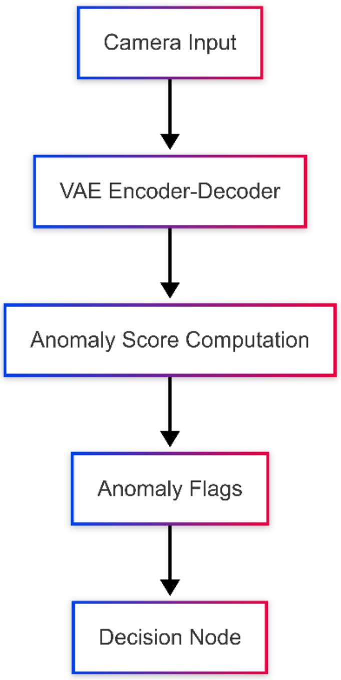


Figure 16: Supervision node flow with VAE anomaly detection and health diagnostics (orthogonal to primary object detection)

Beyond the VAE, the **Supervision Function** performs a range of L1DM and L2DM activities:

- **L1DM:** Detects localized anomalies such as blur, brightness imbalance, or image obstruction within the frame.
- **L2DM:** Contributes to system-level **health reporting** and integrates with the **SAFEXPLAIN middleware** for lifecycle coordination and escalation handling.
  Figure 17 illustrates how L0 functional logic, L1 local node diagnostics, and L2 supervision and middleware coordination are integrated to provide traceable DS3.1-compliant execution and safety monitoring behaviour across the full automotive pipeline.

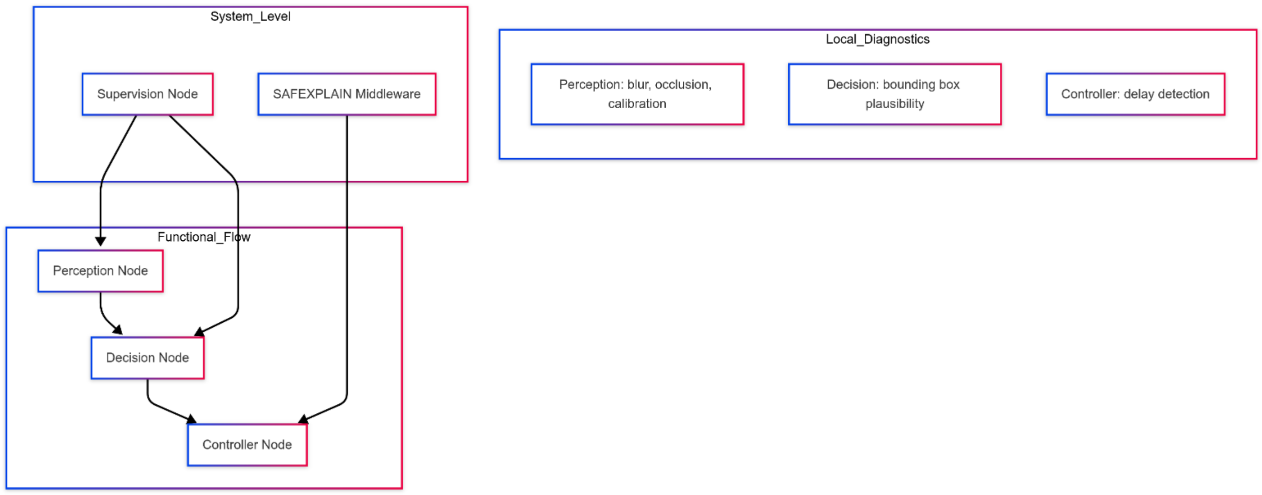


Figure 17: Layered diagnostic architecture showing functional flow (L0), local node-level diagnostics (L1), and system-level health aggregation (L2).

Additionally, the **Supervision Node** plays a supervisory role over other nodes by:

- Monitoring message **timing**, **freshness**, and **arrival intervals**
- Verifying the **consistency** of diagnostic signals across modules
- Flagging **fault propagation patterns** that suggest systemic risk

The **Supervision Function** enhances the system's ability to detect **latent failures**, including those that may not be immediately reflected in object detection results. It provides a complementary, independent validation path and supports **functional safety assurance** under both **DS3.1** and **SOTIF**-relevant criteria.

By decoupling monitoring from control and perception, the **Supervision Function** introduces a robust and explainable safety layer that reinforces system resilience and diagnostic transparency.

Although the VAE operates on the same camera input as the perception models, it serves a **different functional role:** detecting anomalous or out-of-distribution inputs instead of classifying objects. Therefore, it does not qualify as a diverse redundancy mechanism but rather as an **orthogonal diagnostic** strategy that enhances system observability and integrity.

4.1.6 Decision Function

The Decision Function serves as the core reasoning layer within the perception–control pipeline. It synthesizes information from perception, vehicle state, and diagnostics to determine whether warnings or braking actions should be issued. The goal of this function is to ensure timely, explainable, and context-aware responses to potential hazards in the driving environment.

The Decision Node processes three key inputs:

- **/perception/objects** – structured object detections from the Perception Node, including bounding boxes, classes, and confidence scores.
- **/vehicle/state** – real-time vehicle telemetry, such as speed.
- **/supervision/anomaly_flags** – safety and quality indicators from the Supervision Node, flagging degraded sensor input or suspicious conditions.

The internal evaluation logic of the node—based on TTC, object plausibility, and diagnostic conditions—is visualized in Figure 18, showing how specific inputs trigger safety-relevant outputs.

To support traceable and configurable DS3.1 behaviour, the Decision Function uses two explicitly declared thresholds:

- TTW: 3.0 seconds – a warning is triggered when the computed TTC falls below this value but remains above the braking threshold.
- TTC AEB: 1.0 second – a brake request is issued when the TTC drops below this value and no driver override is detected.

These thresholds **are exposed as ROS 2** parameters (ttw_threshold, ttc_aeb_threshold) and may be tuned based on **vehicle dynamics** or **testing requirements**. Additionally, when anomaly flags are received from the Supervision Node, the Decision Function adjusts these thresholds downward to ensure earlier intervention under degraded conditions.

Runtime enforcement ensures that TTW is **always** greater than TTC AEB, preserving logical consistency. These values influence the warning and brake output topics as well as the system state transitions published by the node.

Detailed Logic Explanation:

For each detected object, the Decision Function computes **TTC** using the formula: **TTC = distance / relative speed,** where relative speed defaults to ego speed if object speed is unavailable.

If **TTC ≤ TTW**, a visual warning is triggered.

If **TTC ≤ TTC AEB**, and no override conditions are present, **a brake request** is issued.

When **anomaly flags** from the Supervision Node are active, the system **applies conservative adjustments** to the thresholds, ensuring earlier warnings and braking.

Each object is processed **independently**, and the **most critical object** (smallest TTC) determines the system response.

The node includes **fallback logic** for delayed or missing vehicle state data (e.g., speed), allowing it to estimate values locally and maintain decision continuity.

These behaviours ensure alignment with **DS3.1 compliance** and support **SOTIF safety assurance** by mitigating risks under degraded conditions.

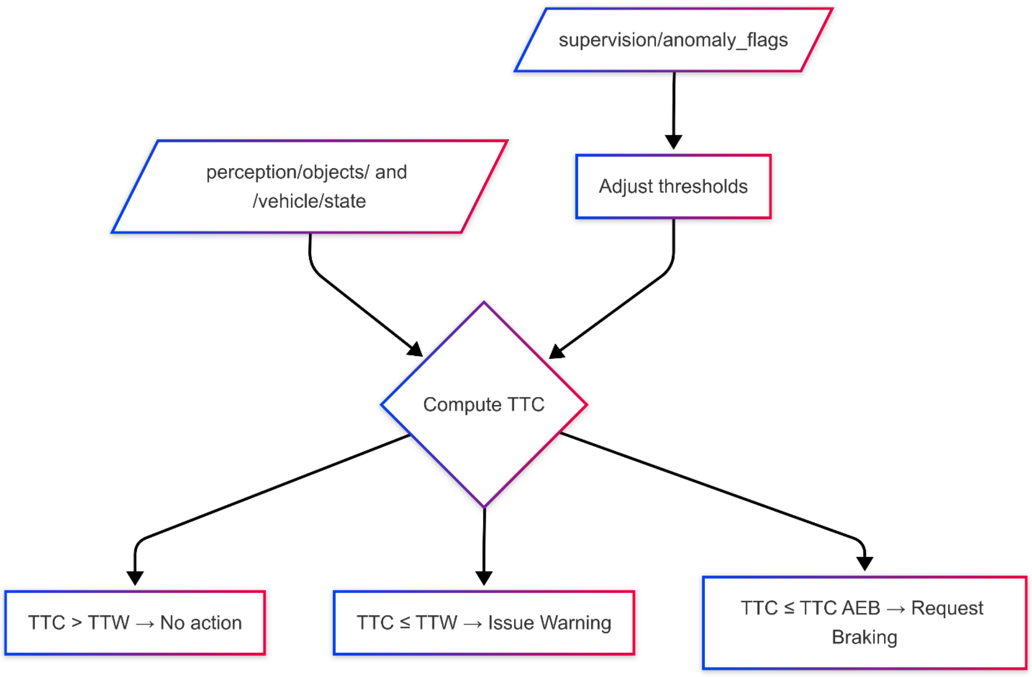

supervision/anomaly_flags
perception/objects/ and /vehicle/state
Adjust thresholds
Compute TTC
TTC > TTW → No action
TTC ≤ TTW → Issue Warning
TTC ≤ TTC AEB → Request Braking

Figure 18: Decision function flow showing TTC-based logic, anomaly integration, and outcome paths for warnings, braking, or continued monitoring.

The Decision Function evaluates each detected object based on a **computed TTC** value derived from the ego vehicle's **speed** and object **distance**. When TTC exceeds the warning threshold (TTW), no action is taken. When TTC reaches or falls **below** TTW, a visual and audible warning is issued. If TTC reaches the TTC AEB threshold and **no override** is detected, the system requests **braking**. Anomaly flags received from the Supervision Node cause the Decision Node to adjust these thresholds dynamically, enabling earlier intervention under degraded input conditions. This logic ensures alignment with DS3.1 Step 1 (monitoring), Step 2 (warning), and Step 3 (braking), while also supporting SOTIF coverage for unpredictable edge cases or visual anomalies.

The **Decision Function** bridges perception and actuation with safety-verified logic, ensuring that complex AI-driven observations are converted into reliable, certifiable actions.

4.1.7 Non-AI subsystem

The **Non-AI Subsystem**—also referred to as the **Safety Control Layer**—is responsible for enforcing deterministic safety behaviour in response to decisions made by upstream components. Unlike perception or decision-making modules that rely on machine learning, this subsystem operates using **rule-based logic** designed for **predictability**, **traceability**, and **fail-safe actuation**.

The subsystem receives the following inputs:

- **/decision/warning** – alerts that signal elevated risk conditions.
- **/decision/brake** – commands that request immediate braking action.
- **/carla/vehicle_speed** – real-time speed feedback from the simulator or vehicle.

Based on these inputs, the Non-AI Subsystem determines the appropriate control response and outputs:

- **/carla/control** – low-level vehicle control commands, including throttle, brake overrides.
- **Safety warnings** – optional messages that may be propagated to other components or external interfaces.

This layer acts as the final **control authority** within the system and plays a critical role in **reducing response latency** and **ensuring system stability** during high-risk scenarios. It can initiate emergency braking independently of upstream confidence levels, and it maintains basic driving logic even in the absence of AI inputs (e.g., during fallback or degraded modes).

Additionally, the Non-AI Subsystem contributes to:

- **Safety layer isolation** – providing a separation between AI-driven logic and actuation.
- **Runtime validation** – monitoring the timing and integrity of control messages.
- **DS3.1 alignment** – supporting traceable actuation responses and system-level safety guarantees.

By complementing the AI stack with a deterministic safety core, the **Non-AI Subsystem** reinforces the system's **functional safety**, improves its **resilience to uncertainty**, and ensures a reliable bridge to **real-world actuation**.

4.1.8 Safety Control

The **Safety Control** layer serves as the final line of defence in the system architecture, bridging the gap between AI-driven decision-making and physical actuation. Its purpose is to ensure that, regardless of the upstream logic or perception quality, the system can execute safe and reliable control actions under all operating conditions.

This layer is implemented through the **Non-AI Subsystem**, which operates independently of learning-based modules. It follows rule-based control logic that is deterministic, transparent, and fully auditable—key qualities for **functional safety certification** and **DS3.1 compliance**.

Key functions of the **Safety Control** layer include:

- **Command validation:** Ensures that control commands received from the Decision Node (such as /decision/brake) are well-formed, timely, and consistent with current vehicle state.
- **Emergency actuation:** Issues low-level control outputs (e.g., /carla/control) to enforce braking, throttle reduction in response to high-risk conditions.

- **Fallback handling:** Maintains minimal safe behaviour even when AI modules are unavailable, slow to respond, or degraded due to poor sensor inputs.
- **Signal monitoring:** Continuously checks inputs such as /vehicle/state and /supervision/anomaly_flags for anomalies, timeouts, or inconsistencies.

The Safety Control system is designed to act **conservatively** under **uncertainty**. For instance, in the presence of late or conflicting messages, it can block unsafe commands, apply default braking behaviour, or revert to pre-defined safe states.

By separating control execution from AI perception and reasoning, the architecture ensures **fail-operational behaviour**, enhances s**ystem robustness**, and **reduces** the likelihood of **cascading faults.** This decoupling is fundamental to achieving SOTIF objectives.

4.2 Integration & Testing

To ensure the system meets its safety, reliability, and functional performance goals, a comprehensive testing strategy should be applied across all components of the pipeline. The limited testing campaign so far focused on validating both nominal functionality and failure response behaviour under a variety of simulated and controlled scenarios. This will be implemented in the period from M32-M36.

The system was yet (limited) tested within the CARLA simulation environment, where test cases should be systematically executed with precise control over sensor inputs, environmental conditions, and timing. These tests are designed (but not yet implemented) to replicate realistic driving events and to introduce fault scenarios relevant to DS3.1 and SOTIF compliance.

Key aspects of the testing approach include:

- **Functional validation:** Confirming that each node—Perception, Decision, Supervision, and Controller—operates correctly under normal input conditions and generates the expected outputs.
- **Degradation testing:** Introducing visual noise, sensor dropout, timestamp drift, and other perturbations to evaluate the system's robustness and ability to fallback safely.
- **Anomaly detection validation:** Measuring the effectiveness of the VAE-based supervision in identifying out-of-distribution inputs, scene anomalies, and visual degradation.
- **Timing and synchronization:** Assessing how well the system maintains temporal consistency across nodes and handles delayed or missing messages.
- **Diagnostic behaviour:** Verifying that L1DM and L2DM mechanisms activate appropriately and publish structured health indicators in real time.

The testing framework also produced **logs**, **anomaly flags**, and **health reports** that should be reviewed post-execution to confirm system correctness and identify areas for tuning or refinement.

The results of this process should be documented in a series of test cases, each mapped to specific safety goals and supported by implementation evidence. Together, these tests should provide confidence in the system's ability to perform reliably under operational and degraded conditions.

See Figure 16, Figure 17, Figure 18 for visual representation of component logic and supervision diagnostics.

4.2.1 Deployment and Embedded Integration (Jetson Orin AGX)

To evaluate the portability and real-time embedded execution of the DS3.1-compliant automotive demonstrator, the entire Automotive Use Case pipeline was deployed on the NVIDIA Jetson Orin AGX platform. This deployment aimed to confirm that all SAFEXPLAIN components—including the Perception Node, Decision Node, Supervision Node, Controller, and middleware—operate reliably on an embedded ARM64-based system.

CARLA[12], the driving simulator, was executed on a separate desktop PC. Sensor data was streamed via TCP/IP to the Orin platform, where the runtime components executed the full safety pipeline in real time. This setup preserved ROS 2 topic communication and SAFEXPLAIN-based lifecycle coordination across the hardware boundary.

4.2.1.1 Visual Setup and Power Profiling

Error! Reference source not found. shows the complete demonstrator configuration: the embedded Orin AGX connected via network to a CARLA-running PC and visualized through the Foxglove Dashboard.

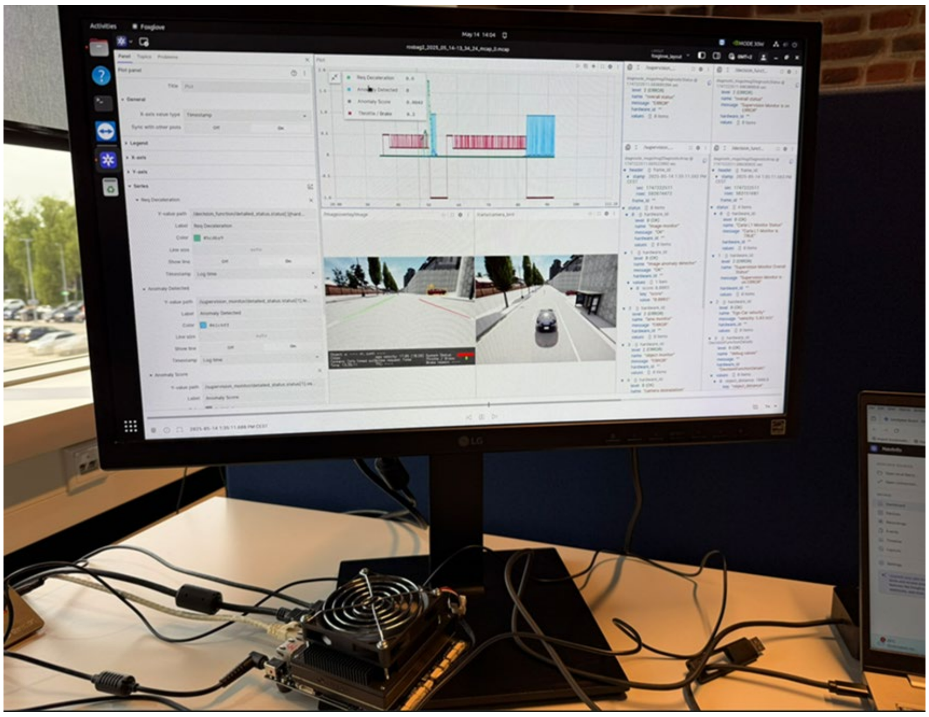


Figure 19: Jetson Orin AGX running automotive use case and Foxglove Dashboard

Error! Reference source not found. illustrate the use of NVIDIA Jetson Power GUI to monitor power and resource usage during the execution of DS3.1 scenarios on Orin.

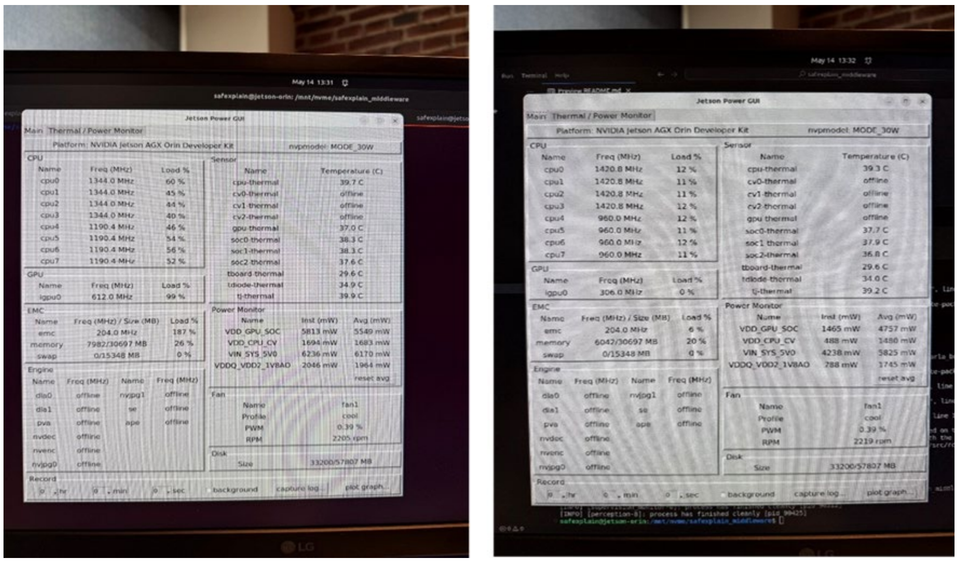


Figure 20: Visual representation of the Power GUI tools running on Jetson Orin

4.2.1.2 Integration Considerations

To ensure compatibility with JetPack 6 and ARM64 constraints, several software-level adjustments were necessary:

- **TorchVision**: No official ARM64 CUDA wheels were available for the required version (0.18.0+), so it was built manually from source.
- **PyTorch compatibility**: CUDA-enabled builds were tested iteratively to match the Jetson ecosystem.
- **NumPy ABI**: Compatibility issues were resolved by reverting to NumPy 1.24.3.
- **Image decoding**: CARLA occasionally emitted unsupported formats; fallback decoding logic was introduced to enforce compatibility (e.g., bgr8 format).
- **YOLOS-Tiny model initialization**: Resource spikes on Orin during model load were mitigated through deferred loading and RAM caching strategies.
- **ROS 2 lifecycle alignment**: All nodes were refactored to properly support kConfigure and Active states using the BaseApplication interface, ensuring smooth transitions and middleware compatibility.

4.2.1.3 Current Status and Future Work

- While the ported system is operational and stable, full validation of runtime behaviour and safety logic on the embedded platform remains planned for the period M32-M36. This includes:
- Scenario replay using MCAP files recorded from real test cases.
- Confirming correct trigger behaviour for DS3.1 Steps 1–3.
- Measuring system latency and response timing under load.
- Verifying structured health reporting and fallback behaviour in degraded modes.
- This embedded deployment confirms the demonstrator’s portability, its modular architecture’s readiness for edge applications, and alignment with SAFEXPLAIN goals of traceable safety assurance across heterogeneous platforms.

4.2.2 Validation Tooling: Overlay Interface and Foxglove Dashboard

To support **functional safety validation** and **demonstration** of DS3.1 compliance, two complementary interfaces have been developed:

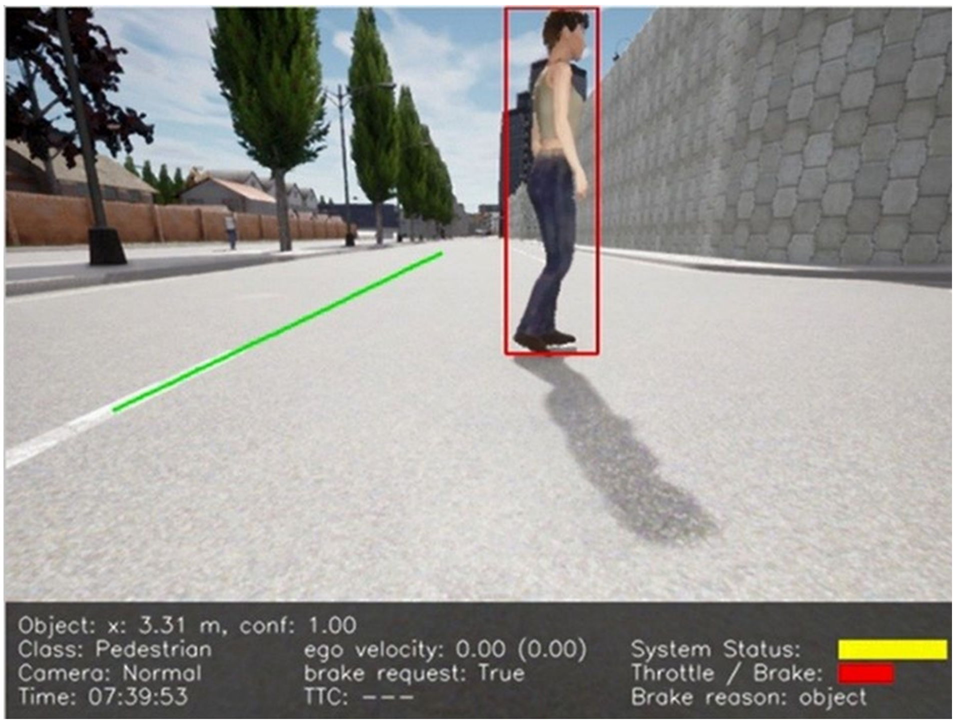
4.2.2.1 Visual Overlay Dashboard Interface

The **Overlay Interface** provides **real-time visualization** of critical perception and control data overlaid on camera inputs (**Error! Reference source not found.**). It serves both:

- **Validation roles** (e.g., seeing system state during test playback).
- **Driver-oriented insight** into braking, warning, and object detections.

Key Features:

- Visualizes:
    - Bounding boxes for detected objects with class, distance, confidence.
    - TTC bar, brake status, system state (green/yellow/red indicator).
    - Ego vehicle speed, brake request flag, anomaly state.
    - Visual and Audio warning
- Designed for scenario playback from simulation.
    - Used for quick identification of:
        - Correct object detection triggering.
        - Warning and braking events according to DS3.1 steps.



```
Object: x: 3.31 m, conf: 1.00
Class: Pedestrian          ego velocity: 0.00 (0.00)     System Status:
Camera: Normal             brake request: True           Throttle / Brake:
Time: 07:39:53             TTC: ---                      Brake reason: object
```

Figure 21: Visual Overlay Dashboard per frame/image

**** Disclaimer: this image is not (yet) reflecting the UI improvements related to visual and audio pre warning and the AEB specific related braking actions, we might be able to replace these images last minute before formal release ****

Pedestrian detected at 3.31m, brake request True, system state Yellow indicating active intervention.

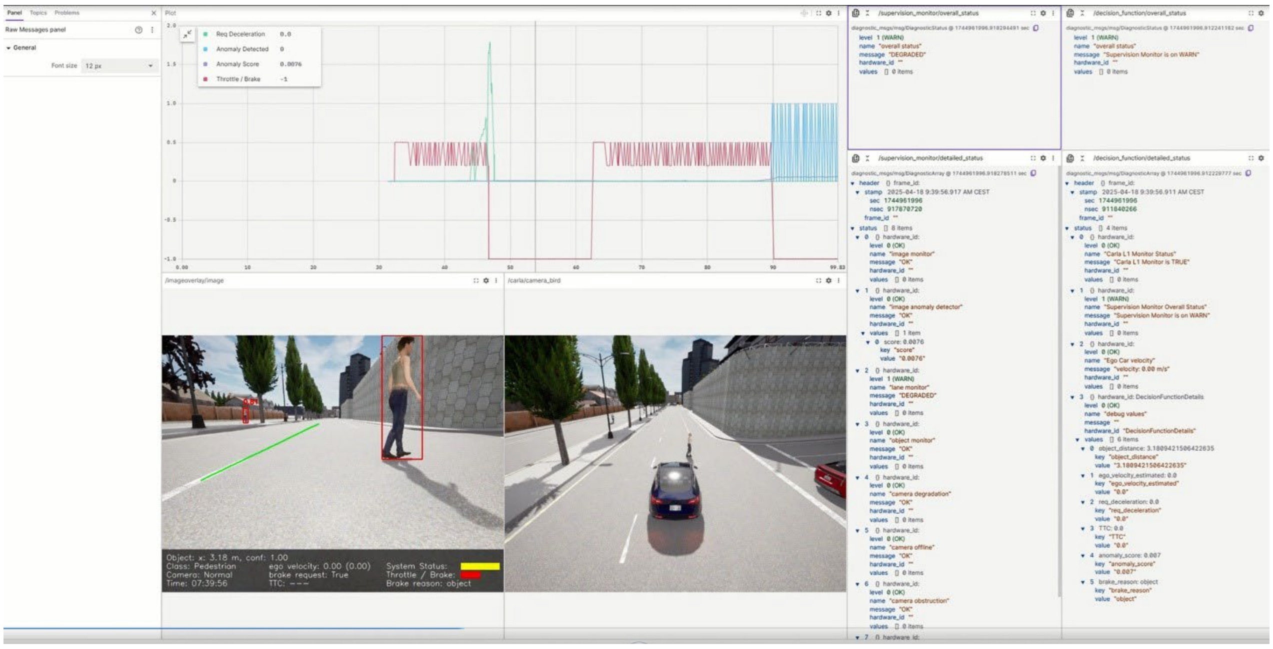
4.2.2.2 Foxglove Studio Dashboard for Deep Diagnostics

The **Foxglove Dashboard** is used for **in-depth post-analysis** of recorded test scenarios (Figure 2, Figure 3). It provides:

- **Detailed introspection** into all **ROS topics** and **diagnostics.**
- Replay of **MCAP-converted Rosbags** covering full DS3.1 scenarios.

Key Features:

- Multi-panel visualization**:**
  - **Plots:** deceleration requests, anomaly scores, throttle/brake activity.
  - **Messages:** detailed status from **supervision** and **decision** nodes**.**
  - **Camera views:** front and bird's-eye perspective synchronized with control actions.
- Enables**:**
  - Frame-by-frame **verification of DS3.1 test steps.**
  - **Inspection of diagnostics** like **anomaly flags, VAE scores,** and **supervision status.**
- Useful for debugging**,** functional safety audits, and scenario demonstrations.



```
Object: x: 3.18 m, conf: 1.00
Class: Pedestrian        ego velocity: 0.00 (0.00)    System Status:
Camera: Normal           brake request: True          Throttle / Brake:
Time: 07:39:56           TTC: ---                     Brake reason: object
```

```
/supervision_monitor/overall_status
diagnostic_msgs/msg/DiagnosticStatus @ 1744961996.918294491 sec
  level 1 (WARN)
  name "overall status"
  message "DEGRADED"
  hardware_id ""
  values [] 0 items
```

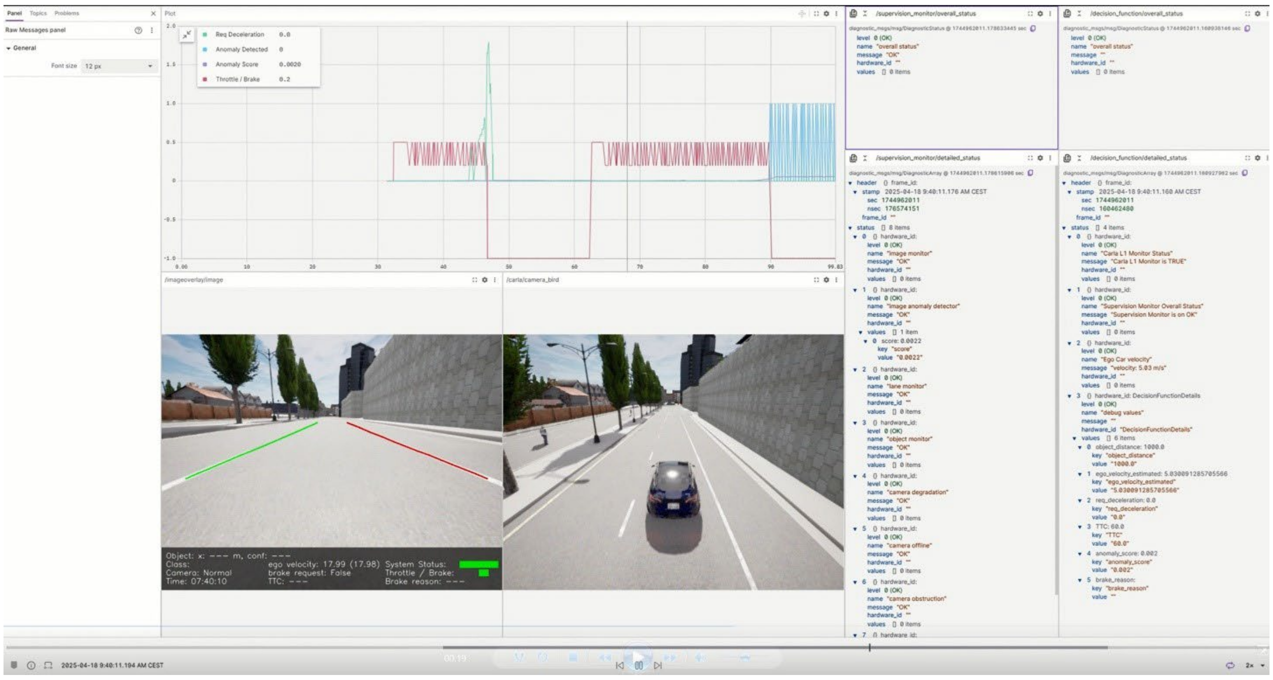
```
/decision_function/overall_status
diagnostic_msgs/msg/DiagnosticStatus @ 1744961996.912241182 sec
  level 1 (WARN)
  name "overall status"
  message "Supervision Monitor is on WARN"
  hardware_id ""
  values [] 0 items
```

```
/supervision_monitor/detailed_status
diagnostic_msgs/msg/DiagnosticArray @ 1744961996.918278511 sec
▾ header {} frame_id:
  ▾ stamp 2025-04-18 9:39:56.917 AM CEST
      sec 1744961996
      nsec 917870720
    frame_id ""
▾ status [] 8 items
  ▾ 0 {} hardware_id:
      level 0 (OK)
      name "image monitor"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 1 {} hardware_id:
      level 0 (OK)
      name "image anomaly detector"
      message "OK"
      hardware_id ""
    ▾ values [] 1 item
      ▾ 0 score: 0.0076
          key "score"
          value "0.0076"
  ▾ 2 {} hardware_id:
      level 1 (WARN)
      name "lane monitor"
      message "DEGRADED"
      hardware_id ""
      values [] 0 items
  ▾ 3 {} hardware_id:
      level 0 (OK)
      name "object monitor"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 4 {} hardware_id:
      level 0 (OK)
      name "camera degradation"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 5 {} hardware_id:
      level 0 (OK)
      name "camera offline"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 6 {} hardware_id:
      level 0 (OK)
      name "camera obstruction"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 7 {} hardware_id:
```

```
/decision_function/detailed_status
diagnostic_msgs/msg/DiagnosticArray @ 1744961996.912229777 sec
▾ header {} frame_id:
  ▾ stamp 2025-04-18 9:39:56.911 AM CEST
      sec 1744961996
      nsec 911840266
    frame_id ""
▾ status [] 4 items
  ▾ 0 {} hardware_id:
      level 0 (OK)
      name "Carla L1 Monitor Status"
      message "Carla L1 Monitor is TRUE"
      hardware_id ""
      values [] 0 items
  ▾ 1 {} hardware_id:
      level 1 (WARN)
      name "Supervision Monitor Overall Status"
      message "Supervision Monitor is on WARN"
      hardware_id ""
      values [] 0 items
  ▾ 2 {} hardware_id:
      level 0 (OK)
      name "Ego Car velocity"
      message "velocity: 0.00 m/s"
      hardware_id ""
      values [] 0 items
  ▾ 3 {} hardware_id: DecisionFunctionDetails
      level 0 (OK)
      name "debug values"
      message ""
      hardware_id "DecisionFunctionDetails"
    ▾ values [] 6 items
      ▾ 0 object_distance: 3.1809421506422635
          key "object_distance"
          value "3.1809421506422635"
      ▾ 1 ego_velocity_estimated: 0.0
          key "ego_velocity_estimated"
          value "0.0"
      ▾ 2 req_deceleration: 0.0
          key "req_deceleration"
          value "0.0"
      ▾ 3 TTC: 0.0
          key "TTC"
          value "0.0"
      ▾ 4 anomaly_score: 0.007
          key "anomaly_score"
          value "0.007"
      ▾ 5 brake_reason: object
          key "brake_reason"
          value "object"
```

Figure 22: Full braking scenario on Foxglove

***** *Disclaimer: this overlay used in this Foxglove dashboard visual; is not (yet) reflecting the UI improvements related to visual and audio pre warning and the AEB specific related braking actions, we might be able to replace these images last minute before formal release******



```
Object: x: --- m, conf: ---
Class:                    ego velocity: 17.99 (17.98)  System Status:
Camera: Normal            brake request: False         Throttle / Brake:
Time: 07:40:10            TTC: ---                     Brake reason: ---
```

```
/supervision_monitor/overall_status
diagnostic_msgs/msg/DiagnosticStatus @ 1744962011.178633445 sec
  level 0 (OK)
  name "overall status"
  message "OK"
  hardware_id ""
  values [] 0 items
```

```
/supervision_monitor/detailed_status
diagnostic_msgs/msg/DiagnosticArray @ 1744962011.178615906 sec
▾ header {} frame_id:
  ▾ stamp 2025-04-18 9:40:11.176 AM CEST
      sec 1744962011
      nsec 176574151
    frame_id ""
▾ status [] 8 items
  ▾ 0 {} hardware_id:
      level 0 (OK)
      name "image monitor"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 1 {} hardware_id:
      level 0 (OK)
      name "image anomaly detector"
      message "OK"
      hardware_id ""
    ▾ values [] 1 item
      ▾ 0 score: 0.0022
          key "score"
          value "0.0022"
  ▾ 2 {} hardware_id:
      level 0 (OK)
      name "lane monitor"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 3 {} hardware_id:
      level 0 (OK)
      name "object monitor"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 4 {} hardware_id:
      level 0 (OK)
      name "camera degradation"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 5 {} hardware_id:
      level 0 (OK)
      name "camera offline"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 6 {} hardware_id:
      level 0 (OK)
      name "camera obstruction"
      message "OK"
      hardware_id ""
      values [] 0 items
  ▾ 7 {} hardware_id:
```

```
/decision_function/overall_status
diagnostic_msgs/msg/DiagnosticStatus @ 1744962011.160938146 sec
  level 0 (OK)
  name "overall status"
  message ""
  hardware_id ""
  values [] 0 items
```

```
/decision_function/detailed_status
diagnostic_msgs/msg/DiagnosticArray @ 1744962011.160927962 sec
▾ header {} frame_id:
  ▾ stamp 2025-04-18 9:40:11.160 AM CEST
      sec 1744962011
      nsec 160462480
    frame_id ""
▾ status [] 4 items
  ▾ 0 {} hardware_id:
      level 0 (OK)
      name "Carla L1 Monitor Status"
      message "Carla L1 Monitor is TRUE"
      hardware_id ""
      values [] 0 items
  ▾ 1 {} hardware_id:
      level 0 (OK)
      name "Supervision Monitor Overall Status"
      message "Supervision Monitor is on OK"
      hardware_id ""
      values [] 0 items
  ▾ 2 {} hardware_id:
      level 0 (OK)
      name "Ego Car velocity"
      message "velocity: 5.03 m/s"
      hardware_id ""
      values [] 0 items
  ▾ 3 {} hardware_id: DecisionFunctionDetails
      level 0 (OK)
      name "debug values"
      message ""
      hardware_id "DecisionFunctionDetails"
    ▾ values [] 6 items
      ▾ 0 object_distance: 1000.0
          key "object_distance"
          value "1000.0"
      ▾ 1 ego_velocity_estimated: 5.030091285705566
          key "ego_velocity_estimated"
          value "5.030091285705566"
      ▾ 2 req_deceleration: 0.0
          key "req_deceleration"
          value "0.0"
      ▾ 3 TTC: 60.0
          key "TTC"
          value "60.0"
      ▾ 4 anomaly_score: 0.002
          key "anomaly_score"
          value "0.002"
      ▾ 5 brake_reason:
          key "brake_reason"
          value ""
```

Figure 23: Unobstructed driving scenario on Foxglove

Visualization of an unobstructed driving sequence, showing **no anomaly detection**, no **brake commands**, and **no deceleration profile**. It shows acceleration and green system health and no TTC.

4.2.2.3 Comparison of Interfaces

Overlay Interface	**Foxglove Dashboard**
Focused on real-time visual clarity	Focused on deep diagnostic insight
Tailored for driver-perspective demonstration	Tailored for developer & auditor-level validation
Shows critical driving info	Shows full system internals, raw ROS data
Live overlay of system state and detections	Replay & analyse pre-recorded, MCAP-converted logs

Integration with DS3.1 Validation:
Both tools are actively used in **scenario testing**, enabling:

- Validation of TCDS_3.1 Steps 1–3.
- Confirmation of warning/brake trigger points.
- Visual evidence supporting test case traceability.

4.3 Challenges and solutions

Throughout the integration and validation of the automotive AI pipeline, several engineering challenges were encountered—ranging from hardware limitations and middleware behaviour to model deployment and runtime diagnostics. Addressing these challenges required targeted solutions that balanced performance, safety, and system compatibility.

Below is an overview of the key challenges and the solutions applied:

- **CARLA compatibility with embedded hardware:** The NVIDIA Jetson Orin AGX could not run the CARLA simulator natively due to its PC- Architecture requirements.
  ➤ *Solution*: CARLA was executed on an external development machine, with sensor data streamed via TCP/IP to the embedded system, preserving real-time input flow.
- **Bridge node startup dependencies**: The bridge node responsible for interfacing with CARLA would block other ROS 2 components if the simulator wasn't available at launch. ➤ *Solution*: A **conditional launcher** and **retry logic** were implemented to delay ROS node initialization until a successful connection was established.
- **Health reporting inconsistencies**: Initial implementations lacked alignment with **SAFEXPLAIN's diagnostic expectations** and did not follow a uniform reporting scheme.
  ➤ *Solution*: All nodes were refactored to inherit from a shared **BaseApplication interface**, enabling **standardized health reporting** and lifecycle handling.
- **Timing drift and message synchronization**: The use of CARLA-generated timestamps led to misalignment between simulation time and local execution.
  ➤ *Solution*: Timestamps were replaced with **locally generated clocks** (e.g., time.time()), improving synchronization and **real-time consistency**.
- **YOLO model initialization spikes**: On embedded hardware, the initial loading of **YOLOS Tiny** created resource spikes, occasionally causing instability.
  ➤ *Solution*: **Deferred model loading**, **RAM caching**, and **threshold tuning** were applied to reduce overhead and stabilize system performance.
- **Unsupported image encoding**: CARLA occasionally produced image formats incompatible with OpenCV or ROS decoding standards.
  ➤ *Solution*: **Fallback decoding logic** was introduced to enforce safe formats like **bgr8**.
- **Log file overgrowth**: Unfiltered ROS logging led to rapid log file expansion on the limited embedded filesystem.
  ➤ *Solution*: **Log verbosity control** and **automatic log rotation** were implemented.
- **Diagnostic parameter variability**: Simulation and real-world environments required different thresholds for diagnostics.
  ➤ *Solution*: All **diagnostic thresholds** were exposed as **ROS 2 parameters**, enabling runtime configuration per scenario.
- **Difficulties to find the right CUDA-enabled version of Pytorch**
  ➤ *Solution*: **research in communities which version should be used on the Orin with Jetpack 6 and try and error till fixed.**
- **Compatibility of NumPy 1.x ABI**
  ➤ *Solution*: **reinstalled NumPy 1.24.3**
- **Installing the right version of TorchVision** (with the right ARM64 CUDA wheels)
  ➤ *Solution*: Building and installing a matching version of TorchVision: since no official ARM64 CUDA wheels were available for TorchVision 0.18.0+ on the public index, we built it from source. Resulting in the perception node now successfully completes its kConfigure transition, and moves to the Active state, allowing the LifecycleManager to proceed without aborting.

Each of these challenges and their respective solutions contributed to system **stability**, **safety**, and **portability**, enabling a robust execution pipeline across embedded and simulation platforms.

The functional and diagnostic flow described above should be evaluated through simulation-based testing and integrated validation scenarios. 4 visualizes the key signal pathways and verification hooks used to assess system behaviour against DS3.1 safety criteria.

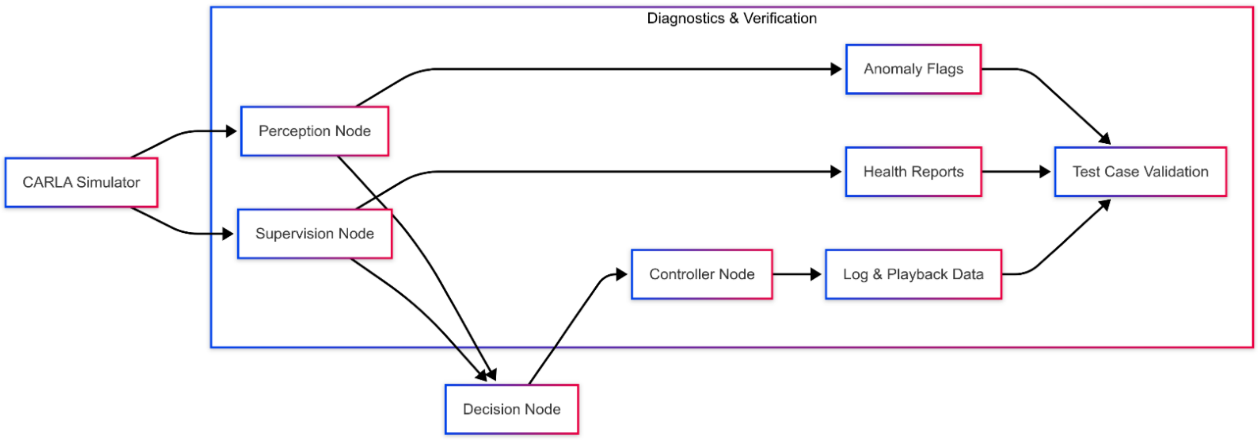


Figure 24: Testing and validation flow from CARLA input through all node's layers. Diagnostic outputs feed into DS3.1 test case validation.

To ensure comprehensive validation of the system's safety logic and alignment with **DS3.1** requirements, each test case was explicitly mapped to the node(s) responsible for its implementation. The diagrams below visualize this mapping, grouping test cases by functional role within the perception–decision–control stack.

Figure 25 shows test cases related to the **Perception Node**, including image quality, calibration, and detection plausibility.

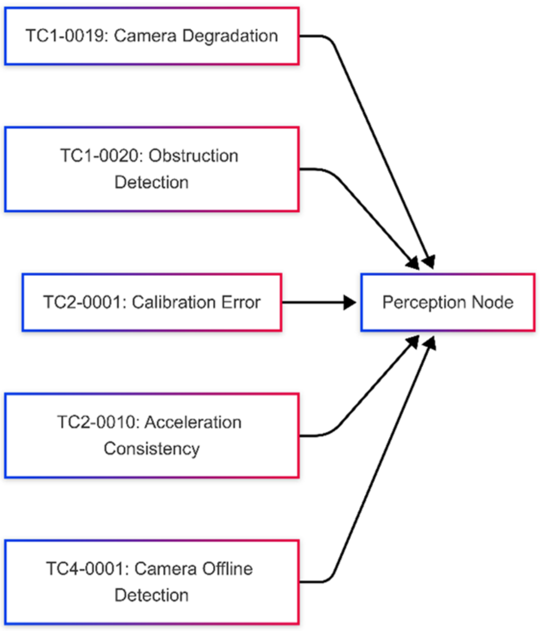

TC1-0019: Camera Degradation
TC1-0020: Obstruction Detection
TC2-0001: Calibration Error
Perception Node
TC2-0010: Acceleration Consistency
TC4-0001: Camera Offline Detection

Figure 25: Perception Node Test Coverage

Figure 26 illustrates test coverage of the **Decision Node**, with a focus on time-to-collision (TTC) evaluation, object classification, and signal delays.

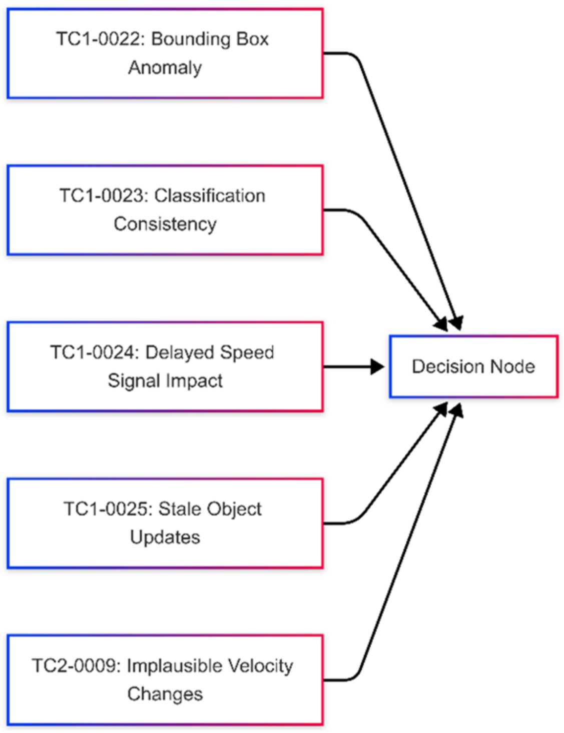

TC1-0022: Bounding Box Anomaly
TC1-0023: Classification Consistency
TC1-0024: Delayed Speed Signal Impact
TC1-0025: Stale Object Updates
TC2-0009: Implausible Velocity Changes
Decision Node

Figure 26: Perception Node Test Coverage

Figure 27 outlines test cases implemented in the **Controller** and **Supervision Nodes**, capturing low-level actuation safety and high-level anomaly monitoring.

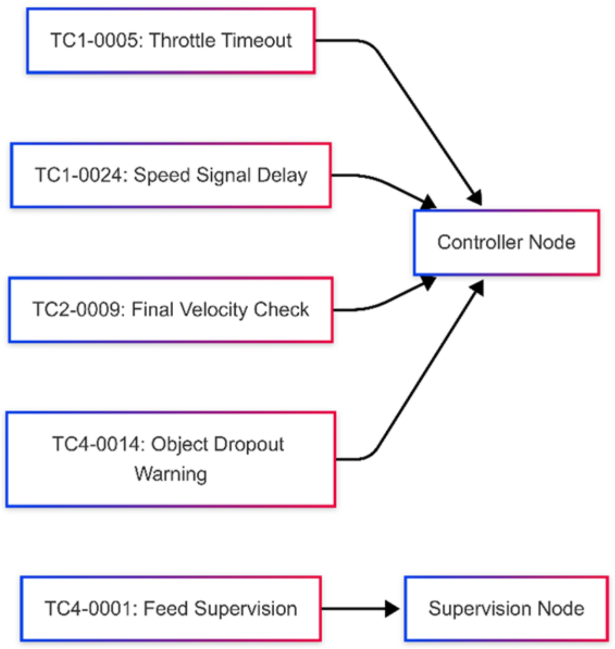

TC1-0005: Throttle Timeout
TC1-0024: Speed Signal Delay
TC2-0009: Final Velocity Check
TC4-0014: Object Dropout Warning
Controller Node
TC4-0001: Feed Supervision
Supervision Node

Figure 27: Controller & Supervision Node Test Coverage

These visuals support traceability between test artifacts and architecture components, reinforcing both functional validation and system-level safety assurance.

The following core safety mechanisms should be validated through simulation and diagnostic evaluation. These mechanisms span across perception, decision-making, control, and supervisory layers:

Validated safety mechanisms:

- Signal delay detection and override logic
- Anomaly detection via VAE
- TTC-based decision gating
- Node health reporting and lifecycle coordination
- Fallback behaviour under degraded inputs

Validated safety mechanisms in detail:

The following core safety mechanisms were validated through simulation, test case execution, and diagnostic monitoring. Each contributes to DS3.1 compliance and enhances system robustness across perception, decision-making, control, and supervision layers:

- **Signal delay detection and override logic**: Implemented in the **Controller Node** to detect delayed or missing vehicle state signals (e.g. speed).
  Test Cases: TC1-0005, TC1-0024, TC2-0009
- **Anomaly detection via VAE**: Performed by the **Supervision Node**, which detects out-of-distribution inputs and visual degradation.
  Test Cases: TC4-0001, TC2-0010
- **TTC-based decision gating:** Handled by the **Decision Node**, evaluating time-to-collision (TTC) to issue warnings or braking commands.
  Test Cases: TC1-0022, TC1-0023, TC1-0025
- **Node health reporting and lifecycle coordination**: Managed by the **SAFEXPLAIN middleware**, which collects node health statuses and coordinates lifecycle transitions. **Covers**: All L2DM reporting paths, aligned with TC4-0014 and supervisory monitoring behaviour

- **Fallback behaviour under degraded inputs**: Observed across all nodes when sensor input quality degrades or control signals become unreliable.
  Test Cases: TC1-0019, TC1-0020, TC2-0001

5 Railway case study

The railway use case focuses on obstacle detection in railway environments through stereo vision and AI-based perception. The objective is to detect potentially hazardous elements on the tracks, such as people or vehicles leveraging depth-aware perception and robust AI pipelines deployable on embedded platforms. The system follows the SP2 safety pattern, integrating AI models, diagnostic monitoring (L1/L2), and decision/safety control layers for safe and traceable operation.

5.1 Architecture

The end-to-end architecture (Figure 28) consists of stereo image input, a rail segmentation module, an obstacle detection module, a distance estimation module, a monitoring and decision layer, and a final safety control stage that issues warnings when required. A non-AI module performs deterministic depth estimation to provide an additional source of trustworthy information.

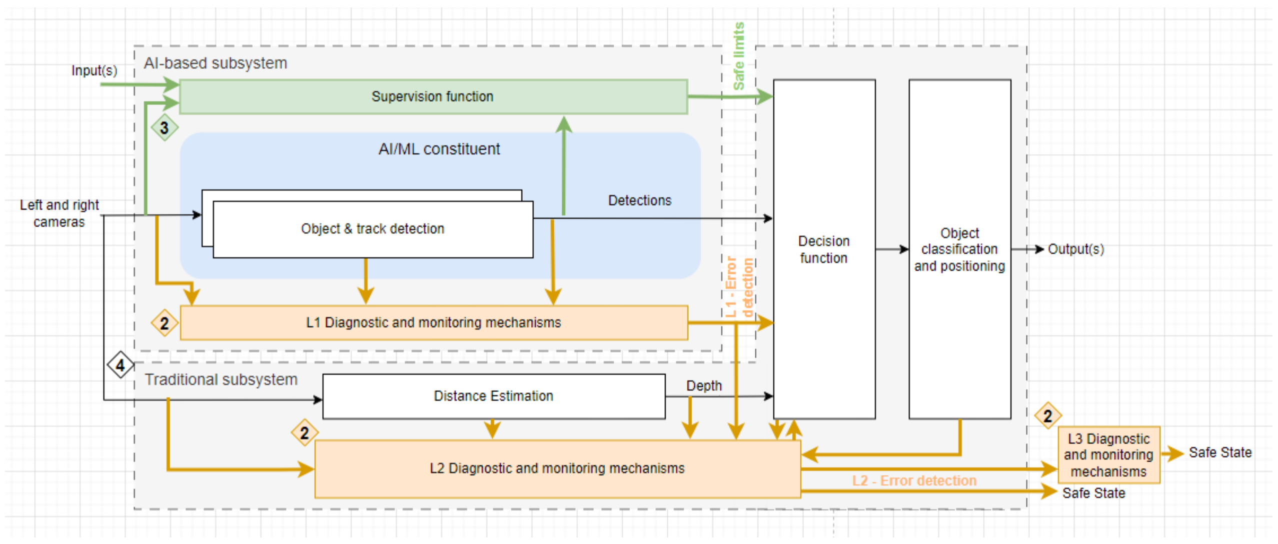


Figure 28: Railway end-to-end system architecture

5.1.1 AI/ML constituent

Compared to D5.1, the AI constituent has evolved from a unified master model (YOLOv8-based object and rail detection) into two distinct and specialised models:

- Rail segmentation model (YOLOv8-seg): performs semantic segmentation to identify the rail tracks.
- Obstacle detection model (YOLOv8): performs object detection for people, vehicles, and other relevant classes.

Justification for the Change

- **Class imbalance**: Every railway image contains rails, creating a class imbalance that made it difficult to train a unified model. Splitting the tasks enables better handling of this issue.
- **Dataset availability**: Object detection datasets are far more common and diverse than semantic segmentation datasets specific to railway scenarios.
- **Modularity and evaluation**: Smaller, task-specific models are easier to evaluate, deploy, and optimise for embedded performance.

Object detection

The *object_detection* node receives two images from a stereo camera (left and right) and is responsible for detecting objects in both images based on its trained model. Multiple instances of this node can be deployed, with each instance representing a different trained model. In this configuration, detections are performed in parallel and independently.

ROS Interfaces:

- Subscriber:
  - /ikerlan_ml_constituent/ikerlan/input → Type: ikerlan_interfaces.msg.StereoImage
- Publisher:
  - /ikerlan_ml_constituent/"node_name"/output → Type: ikerlan_interfaces.msg.StereoDetectionArrayParameters:

  - Model → Specifies the trained model that will be used to perform the detection

5.1.2 Diverse redundancy

To create the redundancy schemes, +model has been trained to recognize the following classes: person, bike, car, motorbike, bus, and truck.

In the case of the duplicated redundancy, the baseline model has simply been used twice in parallel. In the case of diverse redundancy, there are 4 different scenarios.

- **Diverse concept**: In the diverse concept, in addition to the baseline model, there is also a separate model which runs in parallel and has a different concept for detecting people. Instead of detecting the entire person, it detects body parts, which allows us to correct some of the false negatives from the baseline model.
- **Diverse execution platform**: In the diverse execution platform, in addition to the baseline model ran on the GPU, there is a second baseline model ran on the CPU. Interestingly, running the same model on CPU and GPU yields slightly different results, which is why it's considered a diverse redundancy scenario. This is due to the differences in how CPUs and GPUs handle inference. CPUs are deterministic, meaning that running the same inference multiple times will always yield the same results. In contrast, GPUs operate in parallel, so the order of operations may vary slightly between runs, leading to minor variations in the output.
- **Diverse framework**: In the case of the diverse framework, in addition to the original model, there is a second baseline model which has been exported to TensorRT. Like the CPU diverse execution platform scenario, the baseline model and the TensorRT baseline model yield different results. This is because when the model is exported from Pytorch to ONNX and ONNX to TensorRT it has endured some changes in the weights and also even in the architecture. Additionally, the runtimes are different for PyTorch and TensorRT and it contributes to the output difference. A second aim of employing diverse frameworks is to protect the system against common-cause errors. By comparing the outputs of both models, we can detect whether a common-cause error has occurred and determine if any of the frameworks contain a bug or fault. In other words, if PyTorch has a bug, it would only affect one of the two models, which would not be critical.
- **Lockstep**: In the Lockstep scenario, two baseline models are run in parallel, but they are delayed in different ways. The first model generates detections and stores them for 5 frames. The second model waits for five frames then generates detections. The idea is to protect the system against random hardware faults. They both yield the same exact result if there is no fault.

5.1.3 L1DM mechanisms

The L1 monitor node is designed to ensure the health and consistency of input images and detections from an ML model within a ROS environment. It focuses on three aspects: input data quality, input temporal consistency, and output trajectory prediction. The node continuously monitors the health of the input images and ML detections and reports any issues through a health status message.

- **Input data quality**: Canny edge is used to identify edges in the input images. Anomalous edges are checked, which can indicate potential issues with the image quality. If anomalous edges are detected, the node updates the health status to reflect a data quality issue.
- **Input Temporal Consistency**: The optical flow between consecutive frames is calculated to measure the temporal consistency of the input images. Specifically, it compares the calculated optical flow with a predefined threshold to determine if the images are

consistent over time. If the optical flow exceeds the threshold, the node updates the health status to indicate a temporal consistency issue.

- **Output Trajectory Prediction:** The node uses Kalman Filters to track the detections produced by the ML model. It checks for objects that are not being tracked and have exceeded a warning age. If untracked objects above the warning age are detected, the node updates the health status to reflect a trajectory prediction issue.

5.1.4 L2DM mechanisms

For the integration of this mechanism, a small patch was developed. When it was integrated, the final implementation of the *ReportHealth()* function was not yet available. Therefore, from each node that needs to publish any kind of health message information, a publisher has been implemented to handle it. Once the final version is available, this small patch will be removed and the *ReportHealth()* implementation will be used directly.

5.1.5 Supervision function

The Supervision Function plays a critical role in enhancing the reliability and trustworthiness of the AI/ML constituent. It acts as an independent monitoring layer that evaluates whether the model inputs and outputs fall within the expected distribution.

To fulfil this role, the current supervision function includes visual explainability integration via EigenGrad-CAM heatmaps applied to the obstacle detection model, enabling per-frame interpretability of predictions.

These visual cues allow the system to highlight which parts of the image influenced a detection, increasing transparency and enabling post hoc analysis of anomalous behaviour.

Note on performance:

After integrating the EigenGrad-CAM explainability module, significant latency was observed during inference on Jetson Orin. The computation time for generating saliency maps is currently too high for real-time deployment, especially when combined with the primary models. A more detailed performance profiling is being planned, and lightweight alternatives are under consideration for future iterations.

5.1.6 Decision Function

The *decision function* node is responsible for verifying that the detections produced by two different sources (or models) are consistent and for grouping the corresponding detections. In addition, the node monitors the health of the overall system and issues alerts if any irregularities are detected. It can operate in two modes—duplicate redundancy and diverse redundancy—which determine the criteria for matching the detections.

Node procedure:

To ensure that the detections from both sources are processed in tandem, the node uses a time synchronizer. This synchronizer collects messages that are closely timestamped, ensuring that comparisons are made on corresponding data samples.

Depending on the *redundancy type* parameter:

- Duplicate Mode: The node calls *process_detections_duplicate*(), expecting near-identical detections from both sources.
- Diverse Mode: The node would call *process_detections_diverse*() implying a different matching approach when detections come from different models.

Duplicate mode:

The function performs pairwise matching by iterating over detections in the first array and finding the best corresponding detection in the second array. It calculates an overlap score using the Intersection over Union (IoU) based on bounding box positions and sizes. A match is considered valid if the overlap score meets or exceeds the 0.9 threshold, ensuring each detection in the second array is used only once. Finally, the function returns matched detections for both arrays along with a nested array of overlap scores.

If the two input vectors do not have the same number of detections or if any pairing has a score lower than the threshold, the error is reported through the health message.

Diverse mode:

process_detections_diverse()

- **Validation:** The function first verifies that detection arrays from both sources (for left and right images) are not empty. If any array is missing, the function returns immediately.
- **Applying Non-Maximum Suppression (NMS):** It calls a helper function non_maximum_suppression() to process the detections from both sources. This helper function internally sorts the detections based on their confidence scores. It Iterates over the sorted detections and calculates the overlap score using Intersection over Union (IoU) between each pair. Finally, it removes detections that overlap beyond a specified threshold, and it therefore retains only the highest confidence detection for each group.
- **Output Construction:** After applying NMS, the function builds a unified StereoDetectionArray message that includes the filtered left and right detections. Additionally, it publishes a Float32 health/score message (e.g., using the minimum score among the retained detections).
- **Resetting State:** The node clears its stored detection arrays to prepare for the next cycle.

ROS Interfaces:

- Subscriber:
  - /decision_function/output → Type: ikerlan_interfaces.msg. StereoDetectionArray
  - /ikerlan_distance_estimation/output → Type: sensor_msgs.msg.Image
- Publisher:
  - /ikerlan_ml_constituent/distance_warning → Type: std_msgs.msg.Bool
  - /ikerlan_ml_constituent/distance_danger → Type: std_msgs.msg.Bool
  - /ikerlan_ml_constituent/health → Type: smw_interfaces.Msg.Health
- Parameters:
  - Redundancy_type → duplicate / diverse

5.1.7 Non-AI subsystem

The non-AI component in the railway use case is represented by the stereo depth estimation node, which does not rely on AI but uses traditional computer vision techniques (e.g. disparity map estimation and stereo rectification). This node provides an interpretable and deterministic depth estimation, complementing AI-based detection with distance estimation for obstacle warnings.

5.1.7.1 Distance estimation

The *distance estimation* node is a ROS node that receives two images from a stereo camera (left and right) as input and computes a corresponding depth image. The node operates as follows:

Node procedure:

Upon receiving a ROS message containing the stereo image (left and right), the node executes the following steps:

1. **Camera Position and Baseline Calculation:**
    - The node extracts the rotation and translation data from the cameras' extrinsic matrices, computes their positions, and determines the baseline as the Euclidean distance between them.
2. **Stereo Rectification:**
    - The node determines the image size from the left image, computes the relative rotation and translation between cameras, and converts intrinsic matrices to double precision with zero distortion. It then calculates rectification transforms, projection matrices, and the disparity-to-depth mapping matrix.
3. **Rectification Mapping:**
    - The node generates rectification maps and applies them, producing rectified images aligned for disparity computation.
4. **Disparity Calculation:**
    - The node configures a StereoSGBM matcher with parameters such as minimum disparity, number of disparities, block size, penalty values, and additional settings. The rectified images are then used to compute the disparity map, which is converted to float format and scaled accordingly.

After the disparity (depth) image is calculated, it is normalized to a range between 0 and 255 and published using the corresponding ROS message. If any part of the process fails or if the resulting image is completely black, the error is reported through the health message.

ROS Interfaces:

- Subscribers:
    - */ikerlan_ml_constituent/ikerlan/input* → Type: ikerlan_interfaces.msg.StereoImage
- Publishers:
    - */ikerlan_distance_estimation/output* → Type: sensor_msgs.msg.Image
    - */ikerlan_ml_constituent/health* → Type: smw_interfaces.Msg.Health

5.1.8 Safety Control

The Safety Control node consolidates decisions made by the AI models and diagnostic layers to generate appropriate warnings. Two boolean topics are issued:

- /distance_warning — when obstacles are detected within a configurable range.
- /distance_danger — for critical proximity situations requiring immediate attention.

The node uses outputs from the Decision Function, which ensures redundancy agreement and health validation, and optionally depth estimation results. It acts as the final gatekeeper in the pipeline, issuing either a nominal or degraded operational status.

5.1.9 Object classification and positioning

The *object_classification_and_positioning* node integrates stereo detection data with depth information to accurately identify obstacles that lie on the railway track. By filtering detections, verifying their location relative to the rail, checking for overlaps between left and right detections, and calculating distances via the depth image, the node determines whether an object presents a warning or danger. It then publishes Boolean messages, ensuring that the train's safety system is informed about potential obstructions in its path.

Node procedure:

Upon receiving a ROS message containing the stereo detections and the depth image, the node executes the following steps:

- **Input Validation:** The function begins by checking if the stereo detection message (and its left/right arrays) as well as the depth image message are valid. If any of these are missing, the error is reported through the health message.
- **Separate Detections by class:** This function filters the detections by class, separating them into two: on one side, the rail area region, and on the other, the rest of the detections.
- **Verifying Detections on the Railway:** With the filtered detections, the next step is to identify if any detection matches the rail area. This rail area is dynamically detected using a segmentation model, which processes the scene to distinguish the railway from the surrounding environment, allowing for a more accurate comparison of obstacles within the detected rail region.
- **Intersection Analysis:** The node compares detections from the left and right sides. This step identifies overlapping bounding boxes between the two sets, which helps confirm the presence of obstacles.
- **Distance Calculation and State Determination:** Based on the relative number of detections from the left versus the intersections from the right., For each detection:
    - The centre of the bounding box is determined.
    - The depth image is used to compute an average disparity.
    - A state machine updates the current state (normal, warning, or danger) based on preset distance thresholds. Once the initial warning distance threshold is crossed, this threshold is increased, making it harder to return to a safe state. Similarly, when transitioning from danger to warning, the limit is raised to prevent rapid state fluctuations and ensure a more stable assessment of risk. Accordingly, Boolean messages are published indicating a warning or a danger condition.

5.1.10 Traffic light

The *traffic_light* node has been developed to provide a simple visualization of the obstacle status on the track. It implements a traffic light system where, if there is no object on the track, green is displayed. If an obstacle is detected at a considerable distance (i.e., the *distance_warning* topic is true), the light turns yellow. If the *distance_danger* topic is true, indicating that an obstacle is close to the train on the track, the traffic light will display red.

This node is for visualizing the results, meaning it is purely a debugging node. If it fails, nothing happens. That is why no health message is sent.

ROS Interfaces:

- Subscribers:
    - */ikerlan_distance_estimation/distande_warning* → Type: std_msgs.msg.Bool
    - */ikerlan_distance_estimation/distande_danger* → Type: std_msgs.msg.Bool
- Publishers:
    - */ikerlan_ml_constituent /traffic_light_markers* → Type: visualization_msgs.msg.MarkerArray

5.1.11 Train operation control

The train operation control node has a very basic function, which consists of communication between the system and the driver. The system continuously suggests whether to reduce speed or stop the train based on the distance to obstacles on the track. If there are no obstacles, it does not provide any suggestions.

The system cannot stop the train but can reduce its speed when the distance_warning is active. The driver also has a button to disable the system's ability to reduce speed, leaving only the suggestions operational.

ROS Interfaces:

- Subscribers:
    - */ikerlan_distance_estimation/distande_warning* → Type: std_msgs.msg.Bool
    - */ikerlan_distance_estimation/distande_danger* → Type: std_msgs.msg.Bool
    - */train_speed_control* → Type: std_msgs.msg.Bool
- Publishers:
    - */train_reduced_speed_suggestion* → Type: std_msgs.msg.Bool
    - */train_stop_speed_suggestion* → Type: std_msgs.msg.Bool
    - */train_speed* → Type: std_msgs.msg.Float32

5.1.12 Simulation

Unreal Engine is the simulator chosen for video generation. It was decided that the best version for testing was Unreal Engine 5 (the newest available). The problem with this version is that no plugin has yet been developed to connect this simulator with ROS. However, we have managed to connect it with ROS through shared memory transport, which enables efficient communication between processes on the same machine by allowing them to access a shared memory segment directly, thereby reducing overhead and latency. We have developed two nodes to implement the shared memory mechanism: one for transmitting Float32 messages from ROS to Unreal, and another for transferring images. In the latter case, images generated in the simulation are deposited into shared memory, where they are subsequently retrieved by the ROS node. Using a ROS topic, we can control the train's speed and movement while also obtaining the camera images.

5.2 Testing

The chosen simulator for testing has been Unreal Engine 5, where a simple test environment has been created. This setup consists of two railway tracks, a train on one of them equipped with two stereo cameras, and various additional elements to enhance the simulation.

To complete the virtual world, different objects such as people, horses, and trees have been placed outside the tracks. This allows for validation that, even when these objects are near the train, the system does not mistakenly classify them as hazardous obstacles.

Simultaneously, in the ROS visualizer "RViz", a message is displayed using a *MarkerArray*, simulating a traffic signal.

Additionally, a car has been positioned on the tracks to test the system's ability to detect obstacles, calculate distances, and appropriately change the signal colours.

The following images illustrate the system's functionality through a structured visual layout: on the left side, the traffic signal and its different states. In the top right corner, the images captured by the two stereo cameras. In the bottom left corner, the calculated depth image:

- When the train is far from the car, the signal remains green, indicating safe passage, as can

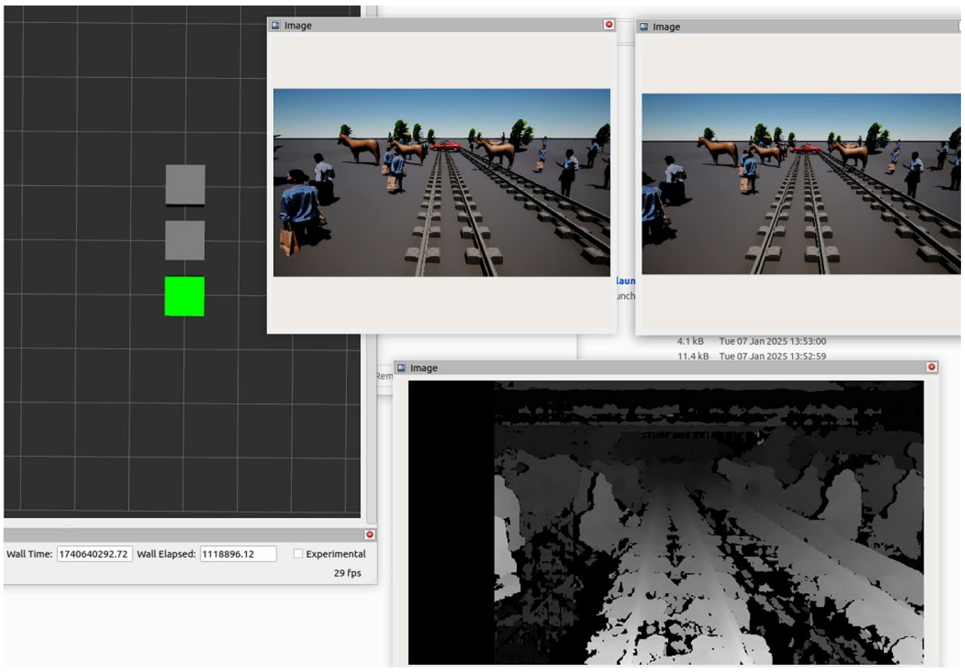
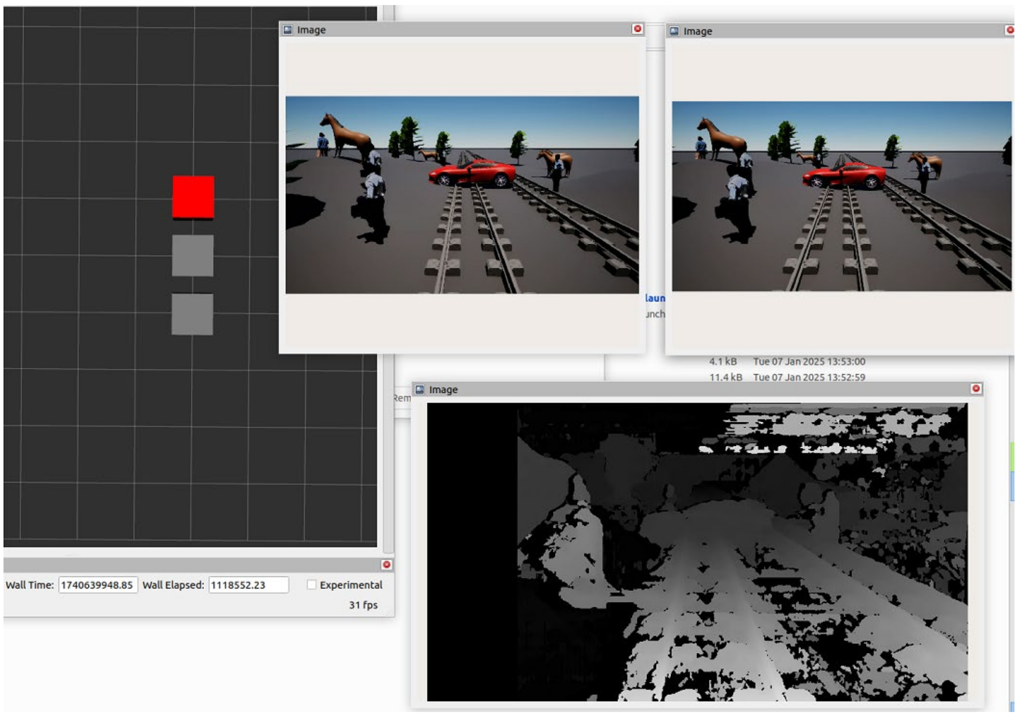


Image
Image
Image
4.1 kB Tue 07 Jan 2025 13:53:00
11.4 kB Tue 07 Jan 2025 13:52:59
Experimental
29 fps

Wall Time: 1740640292.72 Wall Elapsed: 1118896.12

Figure 29: Train warning visualization with "green" indication

  be seen on Figure 29.
- As the train approaches, the signal turns yellow, warning of a potential hazard, as can be seen on Figure 30.
- Once the train is near the obstacle, the signal switches to red, as can be seen on Figure 31, instructing the conductor to stop immediately.

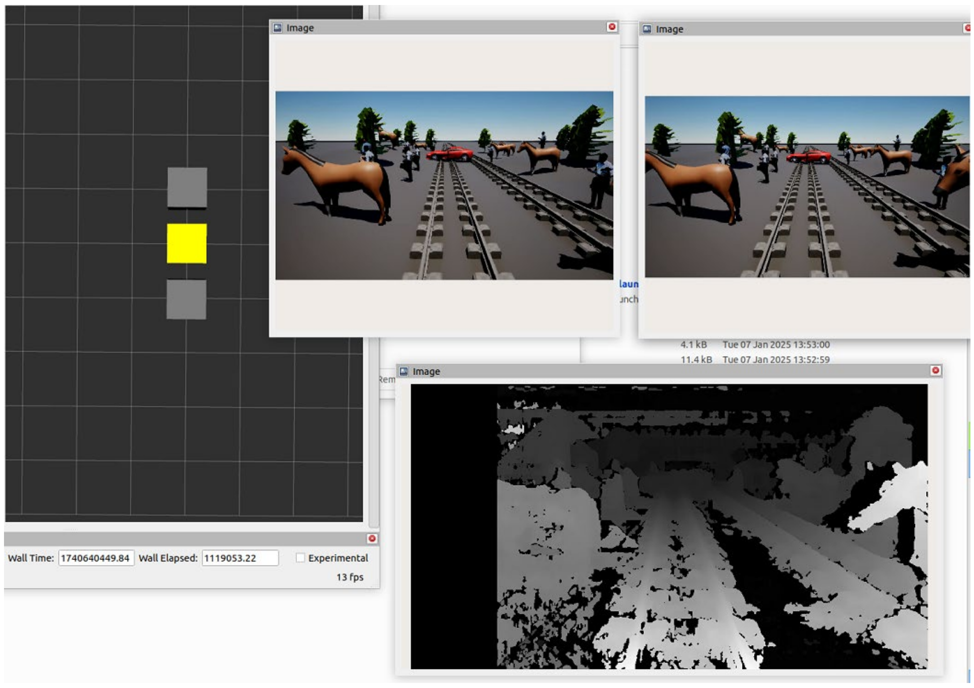


Wall Time: 1740639948.85 Wall Elapsed: 1118552.23 Experimental
31 fps

4.1 kB Tue 07 Jan 2025 13:53:00
11.4 kB Tue 07 Jan 2025 13:52:59

Figure 31: Train warning visualization with "red" indication

This setup ensures a realistic and effective validation of the system's obstacle detection and safety



Wall Time: 1740640449.84 Wall Elapsed: 1119053.22 Experimental
13 fps

4.1 kB Tue 07 Jan 2025 13:53:00
11.4 kB Tue 07 Jan 2025 13:52:59

Figure 30: Train warning visualization with "yellow" indication

response mechanisms.

5.3 Challenges and solutions

5.3.1 Challenge 1

One of the issues we encountered when implementing the decision function node in duplicate redundancy mode was that the similarity between bounding boxes had to be 100% for every pair, and that was not the case. We tried various algorithms, but none could correctly match the bounding boxes. It turned out that the problem was not with the algorithms but with the synchronization of the arrays received from both models. Without any synchronization mechanism, we were attempting to match objects from different images, causing discrepancies. This was resolved by implementing synchronization of the input arrays based on their timestamps.

5.3.2 Challenge 2

Another very significant challenge in implementing this use case was generating videos for testing:

Initially, with the generated videos, it was not possible to calculate an accurate depth image. One issue was the synchronization of image captures from the two cameras, as they had to be taken simultaneously so that the train remained in the same position. Otherwise, objects would appear further ahead in one camera than in the other, leading to errors when calculating the depth image. This problem was resolved by advancing the train only after both screenshots were captured.

Even after resolving the camera synchronization issue, the depth image still was not correct. We initially suspected a problem with the algorithm, but when the same algorithm was tested with other stereo images, it worked properly. We eventually concluded that the issue was due to the homogeneity of the scenario which means a very repetitive background to provide enough reference points for comparison, which is necessary to calculate the depth image for each object (this occurred when the simulation only featured the rails, the train, and a car in the middle of the tracks). The solution was to break the repetitiveness of the simplest scenario of the simulation by adding some objects so that the algorithm had more points for comparison.

5.3.3 Challenge 3

Another challenge, which remains unresolved, is the communication between Unreal Engine 5 and ROS. The issue is that the *"shm"* code is inefficient, making the transmission of images very heavy and costly. As a result, not all the train's images are received by ROS in time; they get queued up, and we are unable to process them all for real-time topic publishing. This is an area that will be addressed during validation. However, we have been able to generate videos by saving all the frames, which has been extremely helpful for integration testing.

5.3.4 Challenge 4

The integration of visual explainability using EigenGrad-CAM has proven computationally expensive on embedded hardware (Jetson Orin). While it provides valuable insights into model reasoning, it increases inference time drastically — far beyond acceptable thresholds for real-time applications.

Planned actions

- **Compute and store activations during a single forward pass.**
  Avoid multiple forward passes by saving the necessary intermediate activations when the input is first processed.
- **Compute and store gradients during a single backward pass.**
  Backpropagate once and cache the required gradients to prevent redundant computations.
- **Generate saliency maps in parallel for each target object using the stored activations and gradients.**
  Leverage parallel processing to compute maps efficiently across multiple objects of interest.

6 Conclusion

The MVP demo and all three case studies—space, automotive, and railway—have reached a mature stage of integration, with the core system components successfully implemented and validated in simulated environments. The SAFEXPLAIN architectural patterns and middleware have been effectively applied across domains, demonstrating modularity, redundancy, and diagnostic transparency.

Porting to embedded platforms, particularly the NVIDIA Jetson Orin AGX, has been achieved for the MVP, automotive and space demonstrators, ensuring real-time execution and system compatibility. Key challenges around synchronization, performance, and deployment have been addressed through iterative integration and targeted solutions.

The focus now shifts to comprehensive testing and scenario-based validation to assess system behaviour under nominal and degraded conditions. These upcoming activities will finalize the verification of DS3.1 and SP2 compliance and ensure readiness for external evaluation.

Acronyms and Abbreviations

- AEB – Autonomous Emergency Braking
- AI – Artificial Intelligence
- AIKO – AIKO Space Systems S.R.L (project partner)
- ARM – Advanced RISC Machine (refers to ARM64 architecture)
- BSC – Barcelona Supercomputing Center (project partner)
- CNN – Convolutional Neural Network
- D – Deliverable
- DLLib – Diagnostic Logic Library (project software library)
- DS3.1 – Driving Scenario 3.1
- EC – European Commission
- GMM – Gaussian Mixture Model
- GNC – Guidance, Navigation and Control
- IKR – Institute of Communication Networks and Computer Engineering, University of Stuttgart (project partner)
- IoU – Intersection over Union
- L0, L1, L2 – Layer 0/1/2 (diagnostic levels in system monitoring)
- LBP – Local Binary Pattern
- MCAP – Message Capture Format (used for ROS 2 data logging)
- MLP – Multilayer Perceptron
- MVP – Minimum Viable Product
- NMS – Non-Maximum Suppression
- PnP – Perspective-n-Point
- PoC – Proof of Concept
- RF – Random Forest
- ROS 2 – Robot Operating System 2
- RViz – ROS Visualization Tool
- SAFEXPLAIN – Safe and Explainable AI Architecture (project and middleware)
- SGBM – Semi-Global Block Matching
- SHM – Shared Memory
- SOTIF – Safety of the Intended Functionality
- SP – Safety Pattern
- SSD – Single Shot Detector
- SSDLite – Lightweight Single Shot Detector variant
- TTC – Time-to-Collision
- TTC AEB – Time-to-Collision for Autonomous Emergency Braking
- TTW – Time-to-Warning
- UE5 – Unreal Engine 5
- UI – User Interface
- VAE – Variational Autoencoder
- ViT – Vision Transformer
- WP – Work Package
- YOLO – You Only Look Once (object detection architecture)
- YOLOS-Tiny – Transformer-based YOLO object detector
- YOLOPv2 – YOLO Perception Version 2

References

[1] SAFEXPLAIN, “D2.2: SAFEXPLAIN DL safety architectural patterns and platform.” Deliverable of the HEU SAFEXPLAIN project, Grant Agreement No. 101069595, 2024.

[2] SAFEXPLAIN, “D4.2: Platform Technologies Report.” Deliverable of the HEU SAFEXPLAIN project, Grant Agreement No. 101069595, 2025.

[3] SAFEXPLAIN, “D3.3: Final proofs-of-concept, arguments, and DL components and libraries.” Deliverable of the HEU SAFEXPLAIN project, Grant Agreement No. 101069595, 2025.

[4] SAFEXPLAIN, “D5.1: SAFEXPLAIN Case study stubbing and early assessment of case study porting.” Deliverable of the HEU SAFEXPLAIN project, Grant Agreement No. 101069595, 2024.

[5] “SSDlite — Torchvision 0.22 documentation.” Accessed: May 19, 2025. [Online]. Available: https://docs.pytorch.org/vision/stable/models/ssdlite.html

[6] “Faster R-CNN — Torchvision 0.22 documentation.” Accessed: May 19, 2025. [Online]. Available: https://docs.pytorch.org/vision/stable/models/faster_rcnn.html

[7] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” Sep. 11, 2020, *arXiv*: arXiv:1905.11946. doi: 10.48550/arXiv.1905.11946.

[8] International Organization for Standardization, *Road vehicles — Safety of the intended functionality (ISO Standard No. 21448:2022)*, 2022. [Online]. Available: https://www.iso.org/standard/77490.html

[9] European Union, “UN Regulation No 152 – Uniform provisions concerning the approval of motor vehicles with regard to the Advanced Emergency Braking System (AEBS) for M1 and N1 vehicles [2020/1597],” *Official Journal of the European Union*, vol. L 360, pp. 66–89, Oct. 2020.

[10] Y. Fang *et al.*, “You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection,” *CoRR*, vol. abs/2106.00666, 2021, [Online]. Available: https://arxiv.org/abs/2106.00666

[11] C. Han, Q. Zhao, S. Zhang, Y. Chen, Z. Zhang, and J. Yuan, “YOLOPv2: Better, Faster, Stronger for Panoptic Driving Perception,” Aug. 24, 2022, *arXiv*: arXiv:2208.11434. doi: 10.48550/arXiv.2208.11434.