Safe and Explainable
Critical Embedded Systems based on AI

D2.2 DL safety architectural patterns and platform

Version 1.0

Documentation Information

Contract Number	101069595
Project Website	www.safexplain.eu
Contratual Deadline	31.03.2024
Dissemination Level	PU
Nature	R
Author	Jaume Abella (BSC), Irune Agirre (IKR), Irune Yarza (IKR)
Contributors	Axel Brando (BSC), Martí Caro (BSC)
Reviewer	Carlo Donzella (EXI), Francesca Guerrini (EXI)
Keywords	Functional safety of AI, safety pattern, diagnostics, monitoring

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

Executive Summary

This document provides strategies and specific solutions to realize safety patterns for safety-relevant software systems that include Artificial Intelligence (AI) software components. In particular, this document provides some context and background on the characteristics of AI-based systems provided by relevant associations in the context of safety-critical systems, as well as the applicable standards. Then, this document proposes a number of safety patterns with increasing requirements and constraints for the AI software components, as well as specific strategies and solutions to realize those.

Note that, while that is the scope and general objectives of this deliverable, this is ongoing work in the project, which will be finalized in 12 months, and the complete and updated report of this activities will be then delivered as part of D2.3. Hence, the contents of this document are not complete with respect to the general objectives and, instead, reflect the status of progress of the work in tasks T2.3 and T2.4 as per m18.

1 Introduction

Systems with safety requirements follows a development process that leads to architectures that allow satisfying those requirements. Those architectures have a large set of commonalities across different systems with comparable safety requirements, and hence, a number of safety patterns have been developed in each application domain as a way to generate specific architectures reusing previous efforts to reduce costs and risks. One such example of safety pattern is the E-gas Concept (EGAS Workgroup, 2013) used in the automotive domain, which is the basis for the architecture of many automotive systems with limited integrity levels (e.g., ASIL-A or ASIL-B according to ISO 26262 (International Standardization Organization, 2018)).

However, AI-based software has a number of characteristics that clash with those expected for safety-relevant software, such as a stochastic behaviour with non-guaranteed correct outputs, and a data-based software design rather than being a purely control algorithm. Hence, traditional safety patterns cannot be applied as they are, and new ones need to be devised.

In this section, we provide some context and scope to the problem of developing AI-based software architectures with safety requirements and provide some background on the relevant safety regulations and the target platform where those software architectures have to be deployed.

1.1 Background

This section provides background on the main concepts used in this report: nomenclature on AI-based systems, the main features of NVIDIA Jetson AGX Orin platform where the safety patterns will be applied, and a recall of the safety standards used as reference and already introduced in D1.1 [3].

1.1.1 AI-based systems

In the same way as in Deliverable D2.1 [4], when referring to AI-based safety systems, this report considers the definitions of the European Aviation Safety Agency (EASA) concept paper for Machine Learning (ML) application (European Union Aviation Safety Agency (EASA), 2023), which makes the decomposition shown in Figure 1.

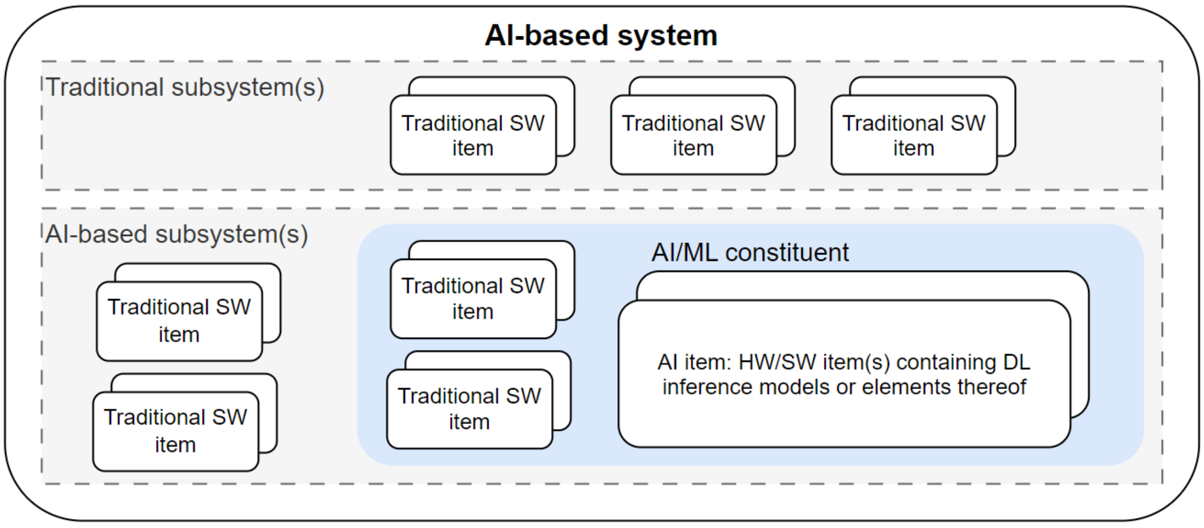


Figure 1: Artificial Intelligence (AI)-based system decomposition based on EASA concept paper (European Union Aviation Safety Agency (EASA), 2023)

Based on this decomposition, the EASA concept paper makes the following definitions:

- AI-based system: systems encompassing traditional subsystem(s) and incorporating at least one AI-based subsystem.
- AI-based subsystem: subsystem that involves one or more AI/ML constituents.
- AI/ML constituent: It is a combination of software and hardware items that include at least one specialized hardware or software item containing at least one ML model.
- AI/ML item: specialized hardware or software item that builds the ML model(s). In particular, we focus on Deep Learning (DL) models, a subfield of ML.
- Traditional subsystem: subsystem that does not include any ML model.
- Traditional SW/HW item: hardware or software items that do not include ML model(s).

This deliverable focuses on AI/ML constituents with the following features:

- Deep Learning (DL) algorithms based on supervised learning for visual perception classification tasks.
- Applications based on offline learning processes in which the model remains fixed at approval time, while excluding online learning processes.

1.1.2 NVIDIA Jetson AGX Orin

As detailed in D4.1, the target MPSoC platform in SAFEXPLAIN belongs to the NVIDIA Jetson AGX Orin family. In SAFEXPLAIN, the AGX Orin Dev Kit has been selected. The Orin comprises 3 clusters of 4 Arm Cortex-A78AE CPUs, a dual core Sensor Processing Engine (SPE) which can be configured in lockstep, a NVIDIA Ampere GPU, ad-hoc AI-oriented accelerators, and video encoder/decoder.

Clusters based on Arm cores provide general purpose computing capabilities suitable for control software and software components around AI-based software, but without needing AI-specific support or massively parallel hardware components. Each of the three clusters includes four cores that can be used as either 4 independent cores, or as 2 pairs of lockstep cores. Cores include local (L1 and L2) caches. There is a L3 cache shared across all cores in the cluster, but it includes partitioning support. We refer to the set of 3 core clusters as CCPLEX for short.

Computing devices like the GPU and the AI-oriented accelerators have a complementary nature to the core clusters and are suitable for AI software and other embarrassingly parallel software. In particular, the GPU has a number of Stream Multiprocessors (SMs) sharing a L2 cache, which, apparently, cannot be partitioned.

Core clusters and the GPU share an L4 cache that, so far, cannot be partitioned, and hence, both cores and the GPU share it. However, we note that, due to its limited size, does not seem to be intended for providing increased cache space for the L3 caches of the core clusters or the L2 cache of the GPU.

1.1.3 Standards and technical reports

SAFEXPLAIN deliverable D1.1 [3] provides an analysis of the relevant standards for the project, classified as: functional safety standards, AI and Autonomous Systems Standards, and guidelines for High-Performance Embedded Computing (HPEC) platforms. For each of them, a general overview was provided, as well as the baseline safety principles on which the architectural patterns of this report are based on. As the outcomes of this deliverable are meant to be applied in different domains, the following domain independent standards are taken as basis:

- **IEC 61508:2010 - Functional safety of electrical/electronic/programmable electronic safety-related systems [6]**: D1.1 provides an overview of "*Architectural safety patterns from traditional FUSA*" based on this standard, used as starting point in this report.
- **ISO/IEC TR 5469:2024 - Artificial intelligence — Functional safety and AI systems [7]**: D1.1 explains the definition of the different DL usage levels according to ISO/IEC TR 5469 which is used as the basis for the incremental strategy of this report (Section 2) and the "*Architectural safety patterns for AI*" on which the reference architecture of Section 3 relies on. As the first edition of ISO/IEC TR 5469 has been published in January of 2024, Section 3 complements the information of D1.1 with updated references to ISO/IEC TR 5469.

In addition to these domain independent standards, technical reports from different domains are also taken as reference:

- **CAST-32A and AMC-20-193 from the avionics domain for multicore processors [8]**: As introduced in D1.1, these two technical reports from the avionics domain provide guidance on the adoption of multicore processors for critical systems. Many of the principles described there are also applicable to HPEC platforms that integrate multicore processors together with additional accelerators and diverse computing resources.
- **E-Gas architecture concept from the automotive domain** (EGAS Workgroup, 2013): The E-Gas architecture concept, defined by the German Association of the Automotive Industry (VDA), has the aim to standardize the safety architecture for engine control systems. The defined architecture is in compliance with ISO 26262 and it can be applied as a well-trusted design principle. The E-Gas defines a 3-level monitoring concept that has been used as reference in this report (Section 3.2), adapting it to AI systems particularities and to ISO/IEC TR 5469.

1.2 Structure of the Document

The remaining of this document is structured as follows:

- Section 2 reviews the relevant sources of risk for AI-based solutions in safety-critical systems, including AI-related factors and traditional ones. Then, section 2 reviews the different usage and automation levels for DL solutions, and how they are tackled incrementally in the project, and also in this document.
- Section 3 provides the general structure of the safety patterns, as well as the techniques considered for diverse redundancy, diagnostics, monitoring, and supervision in the different safety patters.
- Section 4 instantiates the general pattern for specific usage and automation levels, reviews the relevant techniques to be used for diverse redundancy, diagnostics, monitoring, and supervision for each of the patterns, and provides guidance on how to map those solutions to the target platform of the project (NVIDIA Orin).
- Sections 5 and 6 provide the lists of acronyms and references respectively.
- Section 7 provides annexes with further details for some of the diverse redundancy, diagnostics, monitoring, and supervision mechanisms already developed in the project.

2 Sources of Risk and Incremental Strategy

In order to define the safety architecture patterns, first main problems arising from the use of AI in safety critical systems and their root causes are analysed and then an incremental strategy to deal with them is defined.

2.1 AI Risk factors

As for traditional safety related systems, the main risk factors related to AI-based systems can be originated either in its development or at operation. Figure2 depicts this classification criteria together with information sources used in this section to identify the different risk factors. During the development, there are several stages at which systematic faults can originate, like incomplete specifications, biased training data or wrong design decisions affecting the ML model. In order to reduce the probability of systematic faults in the AI-based subsystem, safety considerations shall be taken in the AI lifecycle. In SAFEXPLAIN this has been addressed by the AI-FSM described in deliverables D2.1 [4] and hence, this report does not focus on the risk factors originated in the AI lifecycle but on the AI Operation instead. However, the high complexity of AI systems makes it very difficult to mitigate all systematic faults through design and verification and validation processes, and it is assumed that residual systematic faults will manifest at operation and should be covered by the runtime mechanisms in the safety architecture too, as done for random faults.

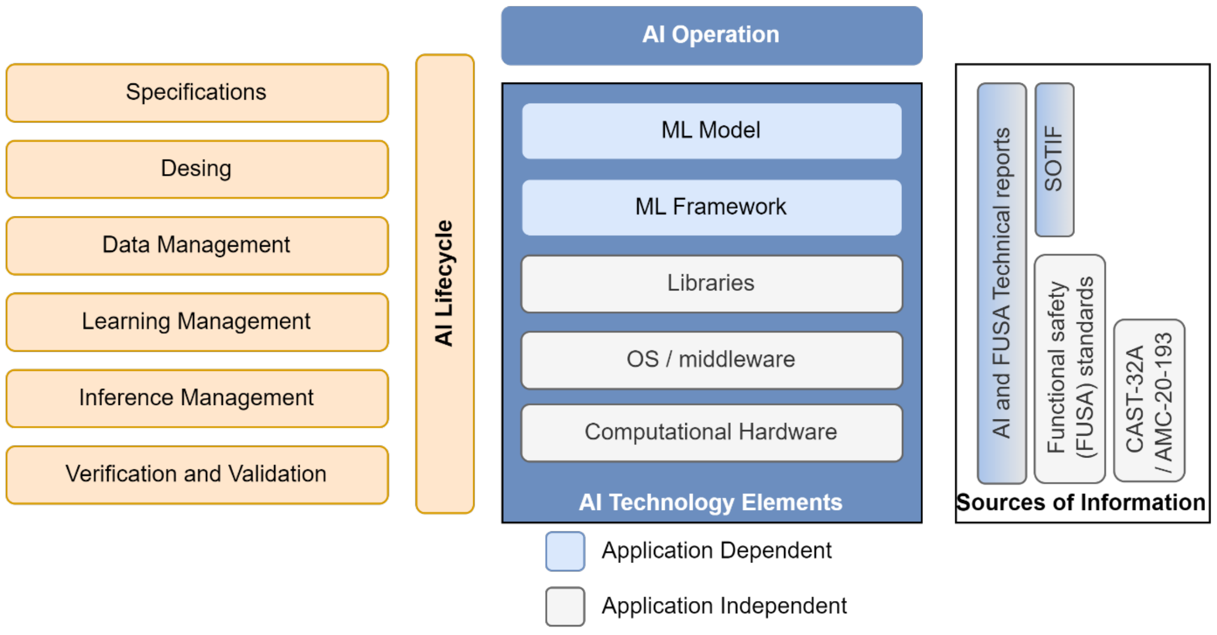


Figure 2: AI Risk factor classification and sources of information

At operation time, residual systematic faults and random faults can impact different elements of the AI-subsystem, this determines the architectural measures required for runtime error detection and monitoring. Based on ISO/IEC TR 5469 we classify these AI technology elements in the 5 groups shown in Figure 2:

- Application Independent: The lowest levels of the AI technology elements can be considered as application independent:
  - Computational Hardware: refers to the hardware platform and its processing units (e.g., CPU, GPU, Accelerators).

  - OS / middleware: This component is not included in ISO/IEC TR 5469, but many faults could also originate on this layer.
  - Libraries: ML components often require specific libraries for their inference (e.g., cuDNN). These libraries can also be implemented in different programming languages such as CUDA, C, C++, python…
- Application dependent: Higher levels of the AI technology elements are usually platform dependent:
  - ML framework: many different frameworks can be used depending on the application that is being implemented (e.g., TensorFlow, Pytorch, Keras).
  - ML model: the ML model will depend on the specific application and the design decisions taken for it (architecture, hyperparameters, number of layers, etc).

The analysed sources for identifying relevant risk factors on AI operation cover different AI technology elements as explained in next subsections.

2.1.1 Functional Safety Standards

Functional safety standards such as IEC 61508, ISO 26262 or EN 5012x only address the application independent technology elements from Figure 2 (computational hardware and software) as they do not address AI-specific topics. According to ISO / IEC TR 5469 these technology elements can be addressed through existing functional safety concepts following IEC 61508-2 for hardware and IEC 61508-3 for software. However, for elements containing AI technology (ML model and related tools) a set of new properties, listed in Section 2.1.3, are defined.

Traditional functional safety standards classify failures based on their origin as systematic or random:

- **Systematic failure**: "*failure, related in a deterministic way to a certain cause, which can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation or other relevant factors*" from IEC 61508-4.
- **Random hardware failure**: "*failure, occurring at a random time, which results from one or more of the possible degradation mechanisms in the hardware*" from IEC 61508-4.

Concerning systematic faults affecting AI SW development (lifecycle) it is an open research topic that has been explored by several research papers. Humbatova et al. introduce a large taxonomy of faults in DL systems' development using several frameworks. Moreover, Schnitzer et al. propose a framework for the systematic management of risks associated with AI. This framework builds upon an AI hazard list from a SoA analysis.

2.1.2 Technical reports on multicore integration (CAST-32A / AMC-20-193)

CAST-32A / AMC-20-193 focus on multicore processors and therefore mostly on the hardware computing element and allocation of resources to deal with interferences. The following main risk factors are considered:

- Contention for resources and interference between software applications or tasks even if there is no explicit data sharing among concurrent tasks, low level resources are shared (cache or interconnects) coupling exists on the platform level, which can cause interference between them.
- Interference caused by the arbitration of shared resources.
- Verification of the use of shared resources: demonstrate that the hosted software applications function correctly and have sufficient time to execute in the presence of the

interference that occurs when all the hosted software is executing on a multicore. The WCET of a software component or task may increase significantly when other software components or tasks are executing in parallel on the other cores.

- Exceedance of resource capabilities by software applications.
- Hardware dynamic features that can alter system behaviour (energy saving features, clock enable / gating, frequency adaptations, deactivating one or more cores, or dynamic control of peripheral access).
- Configuration settings: active cores, execution frequencies, priorities and allocation of shared resources (memory, cache, interconnect).

2.1.3 AI and FUSA Technical Reports (ISO/IEC TR 5469)

As previously stated, ISO/IEC TR 5469 refers to traditional functional safety standards for application independent AI technology elements, and defines the following 6 properties for AI application dependent ones:

- Degree of automation and control
- Degree of decision transparency and explainability
- Environmental complexity and vagueness of specifications
    - Operational Design Domain (ODD) complexity
    - ODD specification
    - Environmental changes: Data drift (training data does not match runtime domain), Concept drift (statistical properties of data change over time)
- Adversarial inputs
- AI Hardware issues
- Technological maturity

2.1.4 Safety of The Intended Functionality (SOTIF) (ISO/DIS 21448)

With the increase of advanced functionalities and automation, ISO/DIS 21448 acknowledges that many hazards are not covered by functional safety standards and defines the concept of functional insufficiencies, which can happen even when the system is free from systematic and random faults addressed in functional safety standards. Functional insufficiencies, which could be considered as a type of systematic faults, are defined as one of the following:

- Insufficiency of specification: This mainly affects to the AI lifecycle as it is defined as "*specification, possibly incomplete, contributing to either a hazardous behaviour or an inability to prevent or detect and mitigate a reasonably foreseeable indirect misuse*"
- Performance insufficiency: "*limitation of the technical capability contributing to a hazardous behaviour or inability to prevent or detect and mitigate reasonably foreseeable indirect misuse*"

2.1.5 Summary of risk factors

Table 1 summarises the risk factors described within Section 2.1 mapping them to the affected AI-technology elements the source of information.

Table 1: Summary of risk factors

Risk factor	**AI technology element**	**Source**
Traditional FUSA risk factors (systematic / random)	All AI technology elements (i.e., computational hardware, OS / middleware, libraries, ML framework and ML model).	IEC 61508 and derived FUSA standards (ISO 26262, EN 5012x, …)

Risk factor	AI technology element	Source
HPEC platform integration risk factors	Computational hardware and lower-level SW (i.e., OS / middleware).	CAST-32A / AMC-20-193 ISO 26262 part 11
AI performance insufficiency	ML model or framework.	ISO/DIS 21448 (SOTIF)
AI & FUSA risk factors (low/medium integrity level)	All AI technology elements (i.e., computational hardware, OS / middleware, libraries, ML framework and ML model).	AI and FUSA Technical Reports (ISO/IEC TR 5469)

2.2 Incremental strategy

As explained in D1.1, ISO / IEC TR 5469 describes different DL usage levels:

- **DL usage Level A1**: AI technology is used and it is possible to make automated decision of the system function using AI technology.
- **DL usage Level A2**: AI technology is used but it is not possible to make automated decision of the system function using AI technology (e.g., AI technology is present in the system for diagnostics).
- **DL usage Level C**: AI technology is not part of a safety function but can have an impact on it.
- **DL usage Level D**: AI technology is not part of a safety function and does not have an impact on it due to sufficient segregation and behaviour control.

Similarly, the EASA concept paper classifies AI systems based on the level of automation / assistance to the user:

- **EASA Level 1**: AI used for assistance to human (human augmentation or cognitive assistance in decision and action selection).
- **EASA Level 2**: Human-machine teaming (cooperation or collaboration).
- **EASA Level 3**: Autonomous AI-based decisions and actions.

Based on these classifications, in SAFEXPLAIN we have defined the incremental strategy Figure 3 for AI adoption in safety critical systems, which incrementally addresses the risk factors of Section 2.1. This strategi defines first a reference safety architecture (described in Section 3), which is instantiated to three safety patterns in order to address the different DL usage levels:

- Safety Pattern 1 (SP1) (described in Section 4.1) aims to address up to a DL usage level D or EASA Level 1. The AI/ML constituent is not part of the safety function, therefore, we set the focus on safety techniques and measures to detect and mitigate errors associated with traditional FUSA and HPEC platform integration risk factors.
- Safety Pattern 2 (SP2) (described in Section 4.2) aims to address up to a DL usage level C or EASA Level 2. The AI/ML constituent is not part of the safety function although it collaborates on the decision and can therefore have an impact on it, so we assume that the AI/ML constituent has a low/medium SIL. This SP considers the techniques and measures of SP1 and sets the focus on solutions to detect and mitigate errors associated with AI performance insufficiencies as well as AI and FUSA risk factors on the ML framework and DL model.
- Safety Pattern 3 (SP3) (described in Section 4.3) aims to address up to a DL usage level A1 or EASA Level 3, where the AI/ML constituent is part of the safety function. The AI/ML constituent provides autonomous AI-based decisions and actions, so it has a high

SIL. SP3 considers the techniques and measures from previous SPs and sets the focus on achieving higher integrity level.

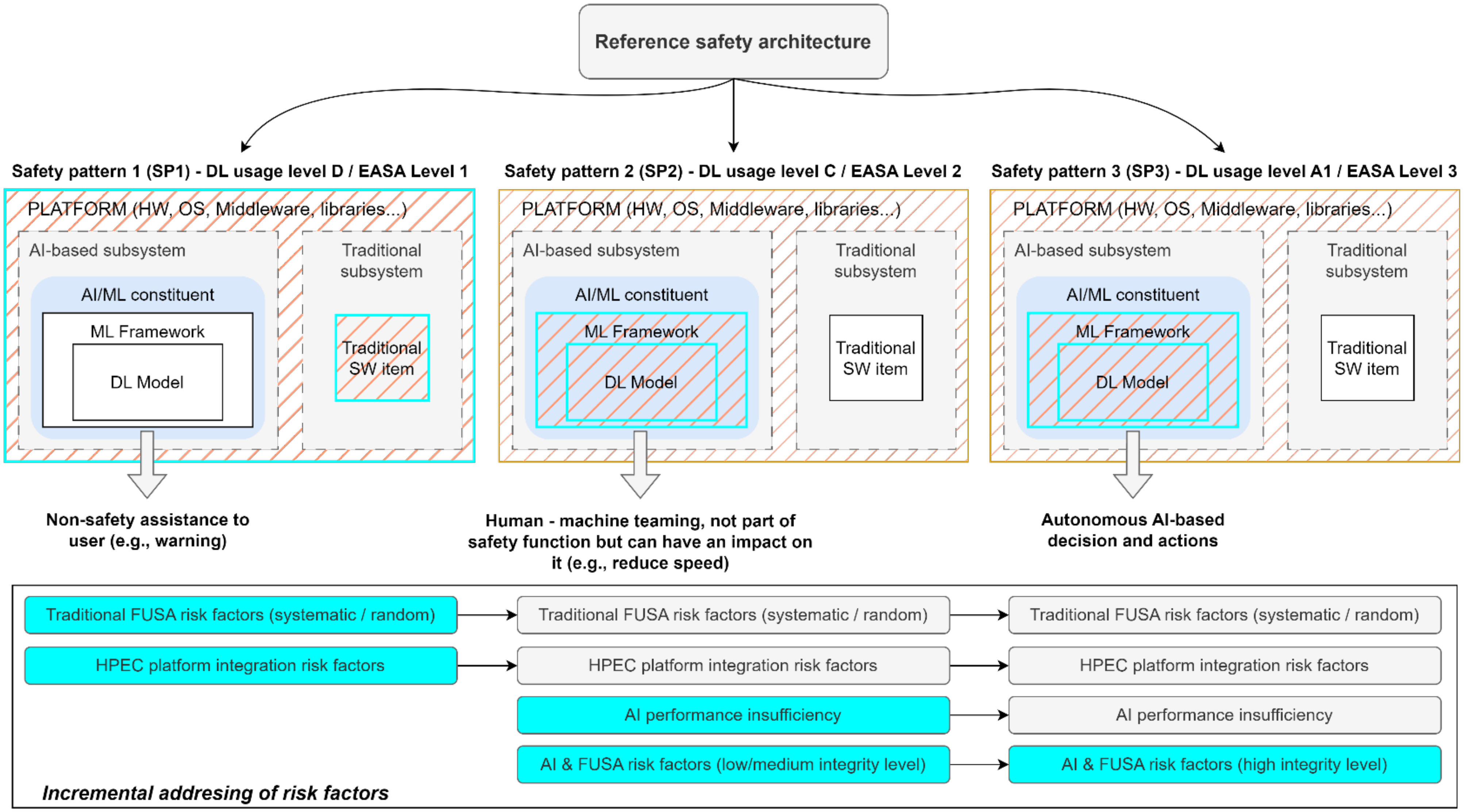

Reference safety architecture
Safety pattern 1 (SP1) - DL usage level D / EASA Level 1
Safety pattern 2 (SP2) - DL usage level C / EASA Level 2
Safety pattern 3 (SP3) - DL usage level A1 / EASA Level 3
PLATFORM (HW, OS, Middleware, libraries...)
AI-based subsystem
Traditional subsystem
AI/ML constituent
ML Framework
DL Model
Traditional SW item
Non-safety assistance to user (e.g., warning)
Human - machine teaming, not part of safety function but can have an impact on it (e.g., reduce speed)
Autonomous AI-based decision and actions
Traditional FUSA risk factors (systematic / random)
HPEC platform integration risk factors
AI performance insufficiency
AI & FUSA risk factors (low/medium integrity level)
AI & FUSA risk factors (high integrity level)
Incremental addresing of risk factors

Figure 3: Incremental strategy for AI adoption in safety critical systems

3 Reference architecture pattern

Previous deliverable D1.1 presented a high-level description of a generic architectural pattern for safe AI execution. In this section, we provide a more detailed view of such concept, which is later tailored to different DL usage levels in forthcoming sections. To this end, Figure 4 depicts a reference safety architecture pattern and this section describes its main safety elements, illustrated by means of enumerated rhombus symbols in Figure 4. These elements constitute the main building blocks to later define the specific safety architectural patterns for each DL usage level.

The goal of the reference safety architecture pattern is to show the main elements proposed in this report and to show an example on how they can be integrated together to build a safety critical system. This reference architecture shall be later tailored and adapted to each use case, and hence many different variants can be defined taking it as baseline.

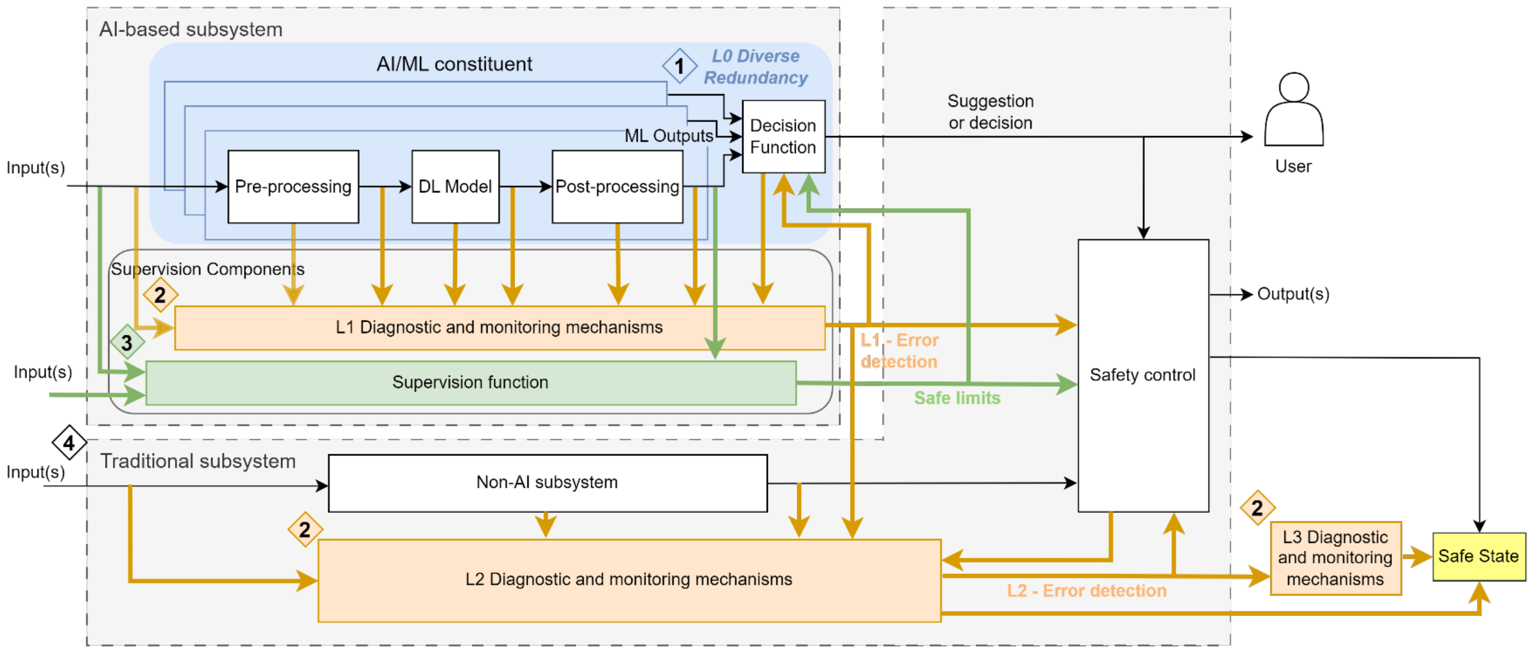


Figure 4: Reference safety architecture pattern for safe AI-based systems

In summary, the reference architecture pattern of Figure 4 integrates the following four safety mechanisms on it:

1. Diverse redundancy within the AI/ML constituent (Subsection 3.1). Note that redundancy and diversity are commonly also applied to traditional systems, however we consider such part out of the scope of this document, and we just focus on its application to the DL component.
2. Diagnostic and monitoring mechanisms (Subsection 3.2). A hierarchical diagnostic and monitoring concept consisting of three levels is defined.
3. Supervision function (Subsection 3.3). Its main goal is to check the appropriateness of the environment and to supervise the output of the ML constituent to identify unsafe situations and stablish the limits for safe operation, providing a safe envelope.

All in all, the reference safety architecture pattern shown in Figure 4, consists of two main subsystems:

- **AI-based subsystem**. Comprises all the AI-related elements in the architecture.

- **AI/ML constituent**. Consists of the ML model as well as the corresponding data pre-processing and post-processing components.
- **Decision Function**. The AI/ML constituent implements a diverse redundancy scheme (see Section 3.1) to enhance the performance of the DL model and increase the ration of AI subsystem errors that can be detected and controlled. Within this diverse redundancy scheme, the Decision Function (see Section 3.1.1), takes the output from all the redundant nodes and provides a single prediction. Then, according to the DL usage level, the Decision Function might provide a suggestion (e.g., warning message, status information) and / or decision (e.g., command on the actuators) to the user and / or safety control.
- **Supervision components**. Consists of the **L1 diagnostic and monitoring** (L1DM) mechanisms as well as the **supervision function**.

  The former, comprises the lower-level diagnostics and monitoring in the platform, which complements the DL model diverse redundancy for detecting runtime errors or model insufficiencies and anomalies on the AI/ML constituent (see Section 3.2.1). To this end, the L1DM mechanisms gathers information from the AI/ML constituent (i.e., inputs, pre-processing, DL model, post-processing, decision function) and provides information on the detected errors to the decision function and L2 diagnostics and monitoring (L2DM) mechanism.

  The latter, bounds the AI/ML constituent operations to work within a predefined safety envelope (see Section 3.3). To this end, it collects information from the inputs/outputs of the AI/ML constituent and provides a set of constraints or limits to the decision function.

* **Traditional subsystem**. Comprises the components usually present in a traditional (non-AI based) system (see Section 3.4).
  - **Non-AI subsystem**. Depending on the DL usage level, the non-AI subsystem can have different usage, it can either be the main safety element, or a fallback element that is activated whenever the supervisor identifies an unsafe operation of the AI-based subsystem.
  - **L2 diagnostic and monitoring mechanisms.** Comprises platform-level diagnostic and monitoring following traditional functional safety techniques (see Section 3.2.2). To this end, it gathers information from all the components within the traditional subsystem (i.e., inputs, non-AI subsystem, safety control) as well as the L1DM mechanism (i.e., L1 – Error detection) and provides information on the detected errors (L2 – Error detection) to the safety control and L3 diagnostics and monitoring (L3DM) mechanism. Considering the error information from lower-level diagnostic and monitoring mechanisms (i.e., L0 and L1), the L2DM is responsible of triggering the required reaction for error handling (e.g., take the system to a safe state).
  - **Safety control**. Collects the output information from the AI/ML constituent and the non-AI subsystem as well as safety related information from the diagnostic and monitoring components (i.e., L1DM, L2DM) and the supervision function (i.e., safe limits) to implement the system control and actuation logic (i.e., command system output(s) or take the system to a safe state). For instance, if the non-AI subsystem implements a fallback function for the ML constituent, in the event that a functional safety problem is detected in the AI-subsystem, the safety control might decide to discard the suggestion/decision from the decision function and operate according to the fallback function. The safety control might also limit the output(s) to meet

with the safe limits defined in the supervision function or even take the system to a safe state.

- **L3 Diagnostic and monitoring mechanisms (L3DM)**. Following traditional functional safety practices some diagnostic may be implemented on an external device such as an external watchdog or microcontroller. Considering the error information from the L2DM mechanisms (i.e., L2 – Error detection), the L3DM mechanism is responsible of triggering the required reaction for error handling (e.g., take the system to a safe state).

Elements with no colouring in the reference safety architecture pattern (Figure 4) are out of the scope of this deliverable since they are not directly related to safety techniques and measures for the AI execution and may vary depending on the system application area.

3.1 Diverse redundancy

The goal of redundancy and diversity on the DL model is to enhance the performance and to increase the portion of AI subsystem errors that can be detected and controlled. In addition, redundancy can also improve system availability, by maintaining system operation even when one instance of the redundant architecture is failing. Redundancy shall be complemented with diversity in order to deal with systematic faults and model insufficiencies. When used for improved diagnostics, diverse redundancy can be considered as a particular technique of the AI subsystem diagnostic mechanisms of Subsection 0.

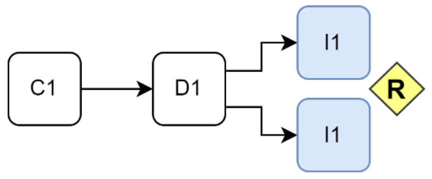
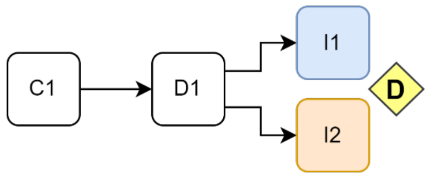
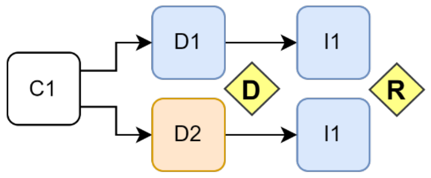
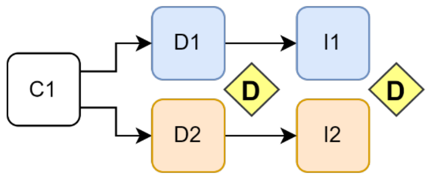
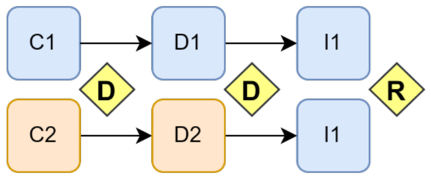
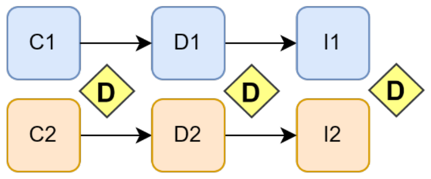
We base on the following three types of diverse redundancy that can be combined to build the Diverse Redundancy Schemes (DRS) proposed in Table 2 with varying degrees of diagnostic coverage:

- Inference Platform diversity (identified as “I” in Table 2) – different platform resources for running the inference. Inference Platform diversity can be applied to different platform elements. These are some example approaches:
    - Inputs (e.g., diverse cameras, sensors, input image flips…)
    - Processing resources (accelerators, CPUs…)
    - OS, execution framework (e.g., TF lite, pytorch, darknet…)
    - Temporal (e.g., different time instants)
    - Data precision (e.g., float vs double, short vs int vs long int)
    - Data rounding (e.g., for floating point data)
    - Sources for potential random parameters
- DL model Development diversity (identified as “D” in Table 2): different techniques or elements for developing the DL model. DL model development diversity can be applied to different development phases and elements. These are some example approaches:
    - Model Architecture
    - Hyperparameters
    - Prediction criteria (e.g., confidence threshold, bounding box overlapping threshold, single vs multi class prediction…)
    - Training datasets
    - Training process (including convergence thresholds, weighting predictions…)
    - Training platform
- Concept diversity (identified as “C” in Table 2): different problem formulation with same final goal. The problem formulation may vary depending on the application are. These are some example approaches:
    - Object detection vs object part detection

- Object detection vs obstacle free path detection

More specific application examples of each DRS are later proposed for each of the safety patterns (refer to Sections4.1, 4.2, and4.3).

Table 2: Diverse Redundancy Schemes

ID	Type of Diverse redundancy: Concept – Development – Inference	Diagnostic Coverage	Description
DRS_1		Low	Single DL model development, replicated in the inference platform without diversity (each replica uses same SW stack and processing elements (e.g., GPU)). Allows addressing traditional FUSA risk factors.
DRS_2		Low to medium	Single DL model development, replicated in the inference platform with diversity (each replica uses different resources (different inputs, AI framework, OS, processing elements (e.g., GPU and CPU), etc.)). Allows addressing traditional FUSA risk factors.
DRS_3		Low to medium	Development of two different DL models based on same concept (e.g., object detection) with diversity in the training process, training data, model architecture etc. Inference without diversity (each replica uses same SW stack and processing elements (e.g., GPU)). Allows addressing AI & traditional FUSA risk factors as well as AI performance insufficiencies.
DRS_4		Medium to high	Combination of DRS_2 and DRS_3 with DL model Development diversity and Inference Platform diversity. Allows addressing AI & traditional FUSA risk factors as well as AI performance insufficiencies.
DRS_5		Medium to high	Development of two different DL models based on different concepts (e.g., one for object detection and the other for obstacle free path detection). This requires DL model Development diversity. Inference without diversity (each replica uses same SW stack and processing elements (e.g., GPU)). Allows addressing AI & traditional FUSA risk factors as well as AI performance insufficiencies.
DRS_6		High	Combination of DRS_4 and DRS_5 with concept diversity, DL model Development diversity and Inference Platform diversity. Allows addressing AI & traditional FUSA risk factors as well as AI performance insufficiencies.

R Redundancy D Diverse Redundancy

Note that, differently to usual control algorithms where diverse redundancy is applied preserving bit-level precision (e.g., dual-core lockstep), some diversity schemes lead naturally only to semantic-level precision, meaning that we expect the same result in semantic terms[1] (e.g., same object detection in the same location), but results are very likely to be different at bit level (e.g., different confidence values, not fully identical bounding boxes, etc.). Therefore, mechanisms to produce the final (combined) prediction or report an error are needed, such as voting, averaging, and non maximum suppression (NMS).

3.1.1 Decision Function

Before introducing those decision functions, it is important to determine the criteria to use to decide what data needs to be combined to provide a single prediction. In some cases, DL models may be predicting a single feature, hence making obvious what should be combined. However, this is not always the case. For instance, in the case of camera-based object detection (CBOD), objects may come along a confidence level, a class and a bounding box. The usual case to determine whether two objects detected by diverse DL models correspond to the same object in this case is doing so only when both of them belong to the same object class (e.g., a car), and the bounding boxes overlap *enough*. The most common metric used to determine whether two bounding boxes overlap enough is the "intersection over union" (IoU) metric (Padilla, Netto, & da-Silva, 2020):

$$IoU = \frac{Area_A \cap Area_B}{Area_A \cup Area_B}$$

Where $Area_A$ and $Area_B$ correspond to the area of the two objects to be combined.

Note that, upon building on higher degrees of redundancy (e.g., three or more redundant instances), IoU can either be applied globally to the objects across the different redundant instances, or obtained using any other criteria to decide such as, for instance, using an object as reference, and computing the IoU of each other object against the reference one.

For those objects regarded as being the same object, different functions can be realized to combine the output of different predictions, each one with its own pros and cons, hence, being some of them particularly suitable for specific problems. While one could devise an arbitrary number of such functions, hereafter we describe some relevant examples that allow illustrating the different tradeoffs to take into account. Moreover, since predictions may include multiple data, those decision functions may come along with specific characteristics when applied to different data (e.g., when applied to confidence values or bounding boxes). Some key functions to consider are as follows:

- Averaging: this function can be applied to any number of outputs and different types of data. For instance, it can be used for dual (DMR) and triple modular redundancy (TMR), and can be applied to the confidence values and bounding boxes in the case of CBOD. Such a function avoids the risk of sticking to the worst prediction, but on the other hand guarantees that the worst prediction will influence the final prediction, since it will have a fair share of impact on the outcome.
- Voting: this function can be applied, typically when there are at least three redundant instances and, preferably, when the number of instances is odd. In this case, the majority

[1] Two results are regarded as identical in semantic terms when, despite not being identical at bit level, both of them can be considered correct, and the impact of using one or another at system level is negligible or, simply, undistinguishable.

decides (e.g., whether an object exists or not). Once the decision is taken, typically, one will select the parameters of the instance with a higher bias towards the decision taken. For instance, in the case of TMR, if two instances indicate that there is an object and one that there is not such an object, the decision function will determine that the object exists, and will resort to the prediction of the DL model with highest confidence on its prediction. This function cannot be applied in the case of DMR, but brings the advantage of using the data of the most confident model when the majority regards the prediction as right. This, however, is also a risk since the worst prediction can be eventually chosen.
If voting is applied over an even number of instances (e.g., DMR), a function is needed to break the draw. One may decide to consider that the prediction is right as long as one of the instances regard it as right, which is very similar to the case of NMS that we introduce right after. Alternatively, one may decide that, upon a draw, the prediction is dropped, which risks at missing many correct detections, but on the other hand, avoids many false positives.

- NMS: this function performs conceptually a union of the predictions just merging those predictions that correspond to the same object. Hence, it is a suitable scheme when the redundant instances to be combined tend to miss objects in the case of CBOD, or when it is more important finding true objects than filtering out false detections since as long as one instance detects an object, NMS will preserve it. For obvious reasons, this trend towards overdetection of objects exacerbates as we increase the number of redundant instances to be combined.

3.1.2 Diverse Redundancy techniques

This section presents different approaches to achieve diverse redundancy at the concept, DL model and inference platform levels. These techniques can be combined to achieve a higher diagnostic coverage.

3.1.2.1 Lockstep execution

The simplest redundancy scheme is achieved by replicating a single DL model on redundant inference platforms (see DSR_1 in Table 2). MPSoC platforms such as the NVIDIA Jetson AGX Orin family usually include multiprocessing units that can be configured in lockstep mode, where two or more processors run the same set of operations at the same time in parallel. In order to increases the temporal diversity needed for common cause failure mitigation, a usual approach is to set a delay as a fixed number of clock cycles (see Figure 5).

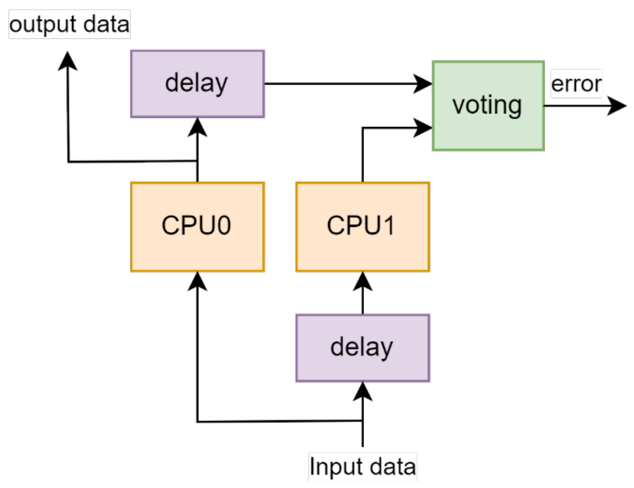


Figure 5: example lockstep execution

If the hardware platform does not support lockstep mode, it can also be implemented at software level, by creating a system where multiple instances of a process run the same set of operations simultaneously and in parallel. The lockstep execution might be used to increase system availability as well as for error detection and correction.

3.1.2.2 Diverse Input Image Transformations

So far, diverse redundancy has been implemented with bit-level correctness in mind. For instance, lockstep processors use 2 or 3 identical cores running the same software with just some time staggering so that their internal state differs at any point in time and, upon a fault affecting all of them simultaneously, the errors generated (if any) are expected to differ and, at least, be detected. In this case, any discrepancy in the results is regarded as an error even if such a difference is semantically innocuous for a specific application, since it is hard to tell whether it is or not innocuous. However, in the context of AI software, such as DNNs, many applications do not require bit-level correctness and, instead, perform stochastic processes where bit-level different outcomes can be regarded as providing identical semantic answers. For instance, two object detectors identifying the same object class as the most likely one, with confidence values above a detection threshold, and with highly overlapping bounding boxes, can be regarded as providing identical outputs even if confidence values and bounding boxes differ.

The input image transformation solution exploits this observation, i.e. redundant AI software may afford bit-level differences across semantically identical outcomes, to realize diverse redundancy in more efficient ways. In particular, we consider solutions where diverse redundancy can be used not only to deal with random hardware faults, but also to mitigate inaccuracies brought by AI models themselves (e.g., false positives and false negatives). Moreover, we do so containing system design and operation costs by preserving the original AI model. Otherwise, the cost of designing, training and verifying two models could be prohibitive, as well as the cost of fetching twice the amount of AI model parameters, which may be in the order of several GBs in the case of some DNNs for camera-based object detection (Caro, Tabani, & Abella, At-scale assessment of weight clustering for energy-efficient object detection accelerators, 2022).

Our solution builds upon applying semantically-neutral transformations to the input data of the AI model (e.g., images for camera-based object detection) that alter inference computations so that random hardware faults are mitigated due to physical redundancy, but also small AI model inaccuracies can be mitigated due to input data diversity. This is illustrated in Figure 6. For the sake of easing the explanation, we focus on a TMR scheme applied on the popular You Only Look Once v4 (YOLOv4) (Bochkovskiy, Wang, & Liao, 2020) object detection software for images, which is relevant for the case studies in SAFEXPLAIN.

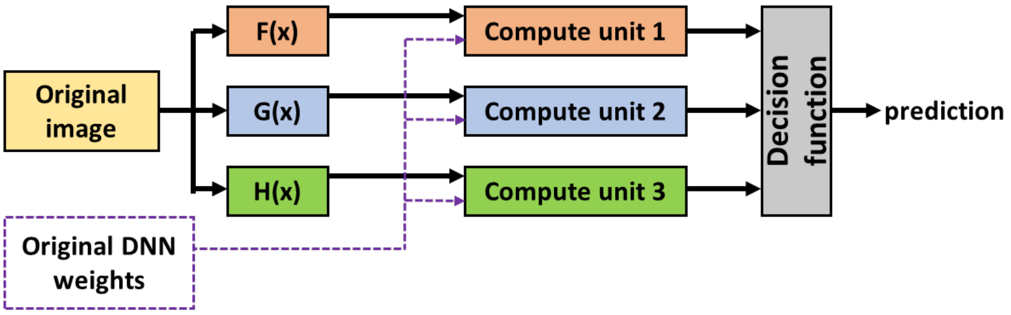


Figure 6. Overview of our proposed image transformation diverse redundancy scheme when applied with TMR for YOLOv4.

Efficient Diverse Redundancy

As shown in some recent work, DNNs such as the one in YOLOv4 may require fetching some GBs of weights to process each image (Caro, Tabani, & Abella, At-scale assessment of weight clustering for energy-efficient object detection accelerators, 2022), whereas images occupy up to few MBs. Hence, memory bandwidth is often saturated to fetch the DNN weights. Based on this fact, our redundancy scheme aims at preserving the same DNN model for all redundant inference processes with the aim of enabling appropriate execution approaches that allow sharing weights fetched across the redundant processes.

Therefore, if weights are fixed, the simplest way to introduce diversity with software-only means consists of altering the input data. Our goal is applying modifications to the input data so that it differs across the redundant processes, but they are semantically equivalent so that inference should lead to the same object detections. In particular, we have considered some of the image transformations provided by the CLoDSA open library (Heras, Accessed Oct-2023), which would map to F(x), G(x) and H(x) in the figure:

- Applying histogram equalization (EQ).
- Image sharpening (SH).
- Dropout (DR) by setting some pixels to zero.
- Applying Gamma correction (GC).
- Blurring the image applying a Gaussian filter (GB).
- Applying Gaussian noise (GN).
- Blurring the image applying the median filter (MB).
- Adding salt and pepper pixels (SP).
- Raising pixel values by a set amount (RV).
- Shifting the image some pixels to the right (RS).
- Shifting the image some pixels to the left (LS).
- Shifting the image some pixels to the top (TS).
- Shifting the image some pixels to the bottom (BS).
- Rotating the image clockwise at a certain angle (CR).
- Rotating the image anticlockwise at a certain angle (AR).
- Flipping the image horizontally (HF)
- Flipping the image vertically (VF).

To realize the TMR scheme, we select three different choices between the image transformations above and the original image and perform inference for the three of them using the identical

YOLOv4 DNN model. With this, we obtain three potentially different object detection outputs that need being combined properly.

Combining Predictions

Rather than attempting to identify wrong predictions – if any – across the diverse and redundant predictions, we merge those predictions with the aim of making correct predictions supersede erroneous ones (see Figure 6). We build on the fact that predictions, even if correct, may not be identical at bit level. Hence, we opt for combining predictions whose detection class matches (e.g., a pedestrian is detected), and whose bounding box overlaps enough (e.g., bounding box area overlaps more than 50%). However, it remains open how to set the final bounding box and confidence for the detection. Regarding the bounding box, note that model A can provide a bounding box overlapping above the threshold with the one from model B and the one from model C, but the ones from models B and C may overlap below the threshold. In this case, we still regard detections as the same detection and attempt to combine them.

Confidence and bounding boxes can be combined using any of the decision functions described before, namely voting, averaging or NMS. As shown in Annex 7.1.1, each function offers a different tradeoff. In general, we recommend the use of voting since it is the one providing higher true positives and lower false positives than a pure identical redundancy, but NMS is the one providing highest mAP (a key accuracy metric) values.

3.1.2.3 Diverse Data Types

As explained in the case of the image transformations, due to efficiency, it is important avoiding fetching DNN weights multiple times despite the DNN needs redundancy. In the previous case, we avoid it by introducing diversity in the image. However, there are other alternatives such as, for instance, introducing diversity in the weights *after fetching them*, so that they still need to be fetched once, as shown in (Caro, Fornt, & Abella, Efficient Diverse Redundant DNNs for Autonomous Driving, 2023). Next, we introduce that work and how it applies to our context.

Our goal is applying modifications to the DNN weights so that they differ across the redundant processes, but they are semantically equivalent so that inference should lead to the same object detections. In particular, we have considered building on different data types whose representations can be produced *on the fly*, which would map to F(x), G(x) and H(x) in the figure below. Note that, differently to Figure 7, the input of F(x), G(x) and H(x) is the set of weights rather than the image to be processed:

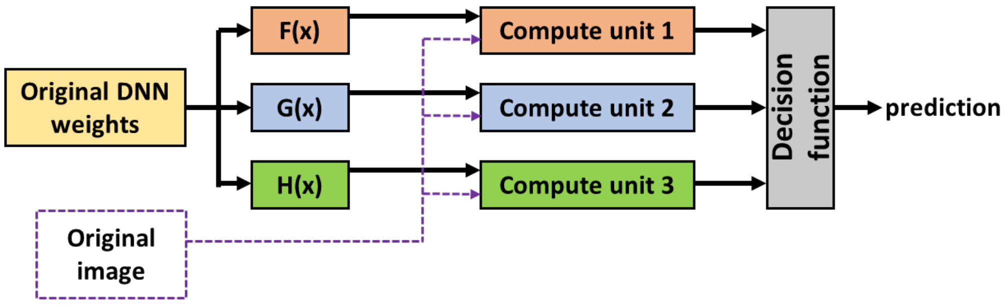


Figure 7. Overview of our proposed data type based diverse redundancy scheme when applied with TMR for YOLOv4.

Data Types

Example arithmetics (data types) satisfying the requirements indicated have been analyzed: for instance, one could consider IEEE754 floating-point 32-bit (fp32) for one of the instances if this is the data type used to store the DNN weights, and dropping the 3 lowermost bits of the mantissa to generate diversity in the input, mimicking some form of fp29 for redundant instance (see Figure 8), which we refer to as fp16 and fp13 respectively.

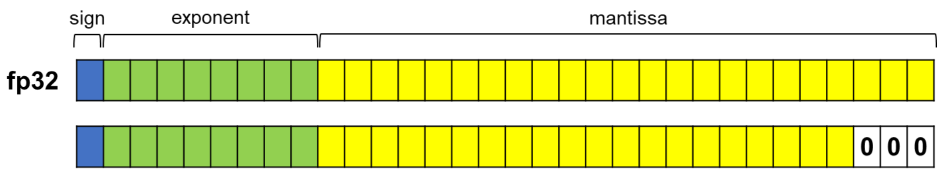


Figure 8. Example of diverse data types generated out of the same weight fetched.

Note that other potential realizations could be possible dropping a different number of bits of the mantissa, using fewer bits also for the exponent, or even using a different numerical representation (e.g., integer 16-bit and integer N-bit where N < 16).

Dominance across Data Types

As part of our analysis of different data types, we have observed that the instance using lower precision arithmetic is expected to provide lower confidence values. This relates to the fact that objects identified with high confidence have values close to 1, but strictly below 1 across layers in the DNN. When operating those values across layers, the distance to 1 naturally increases. Hence, since the highest value strictly below 1 – HVSB1 for short – is farther away from 1 for lower precision arithmetic, the resulting confidence also decreases further. Hence, if we consider fp16 (1 bit for the sign, 5 for the exponent, and 10 for the mantissa), and fp13 (as fp16 but dropping the 3 least significant bits of the mantissa), fp13 confidence values tend to be lower than fp16 ones. For instance, for fp16, the HVSB1 is 0.9995, whereas for fp13 it is 0.996. If, for instance, such value is multiplied 20 times (i.e., $HVSB1^{20}$), the resulting value for fp16 would be 0.990 and for fp13 0.925, hence much lower than for fp16.

Such relationship across different data types offers new opportunities to realize decision functions as we detail next.

Combining Predictions

While the same decision functions explained before can be applied in the case of diverse data types, the particular relationship among different data types detailed above allows building decision functions that trade such relation to correct errors, even in the case of DMR. In particular, if the confidence of the lowest precision arithmetic instance is higher than the one with higher confidence, then likely one of the instances is wrong. This could be used to decline to raise a prediction, but not to choose the right prediction reliably because we could not tell which one of the instances is wrong (i.e. whether the error lead to low confidence for the highest prediction instance, or to higher confidence for the lowest prediction instance). In these cases, DMR can build on temporal redundancy if such temporal redundancy exists. For instance, in the case of CBOD, we can build on the confidence levels for the previous frames to determine which of the instances is experiencing a larger change, and hence, opt for trusting the most stable one among those. An example is evaluated in Annex 7.1.2.

3.1.2.4 Diverse Framework [IKL]

This approach offers an inference platform diversity technique implemented by combining the use of diverse AI execution frameworks. As shown in Figure 9, it follows the DRS_2 diverse redundancy scheme according to Table 2. The DL model is replicated on several inference platforms with divers AI frameworks (e.g., Darknet and Pytorch). The output of running the inference of two redundant DL models on both inference platforms shall then be processes by the decision function to detect difference among them and react accordingly.

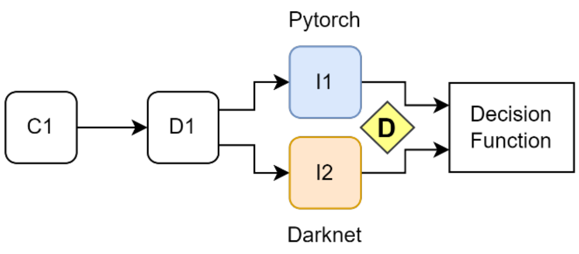


Figure 9: Inference platform diversity scheme using diverse redundant frameworks (i.e., Pytorch and Darknet).

Figure 10 shows the results obtained for running YOLOv7-tiny DNN model on the Orin platform using two diverse AI frameworks, Pytorch and Darknet. The figure shows the bounding boxes for the elements detected by both frameworks (e.g., car||Pytorch, car||Darknet, stop sign||Darknet, traffic light||Pytorch). Green bounding boxes are used to mark matching elements (i.e., objects that have been detected on both frameworks). Whereas red bounding boxes are used for object that do not have a match on both frameworks (i.e., objects that have been detected only on one of the frameworks, or the IoU of the object detected on both frameworks is bellow a predefined threshold (in this case the threshold was set to 0,5)).

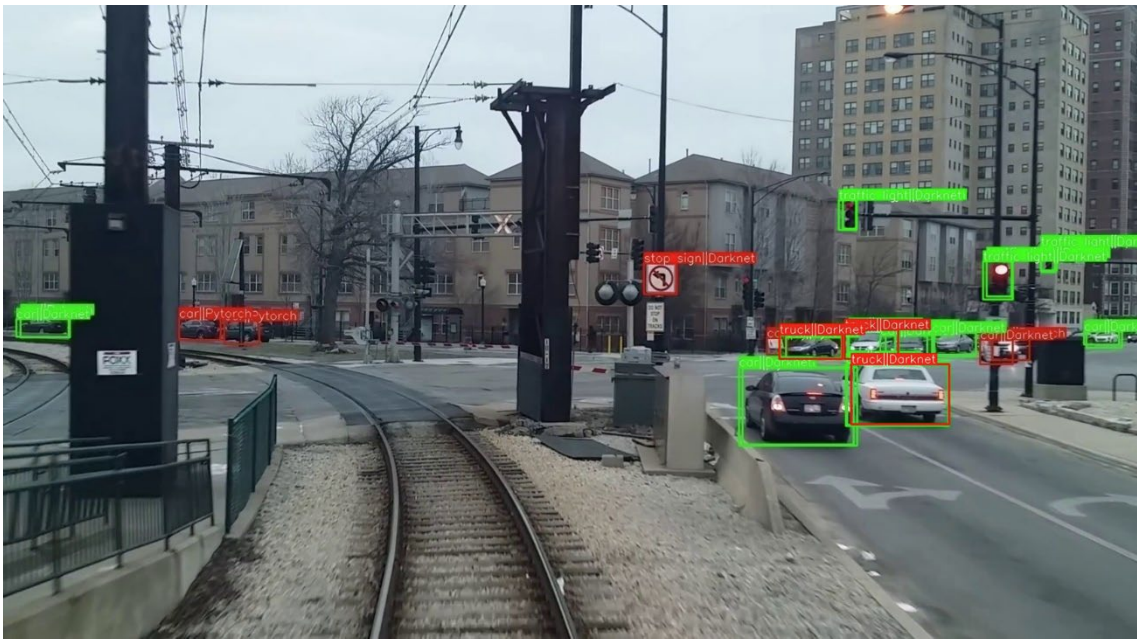
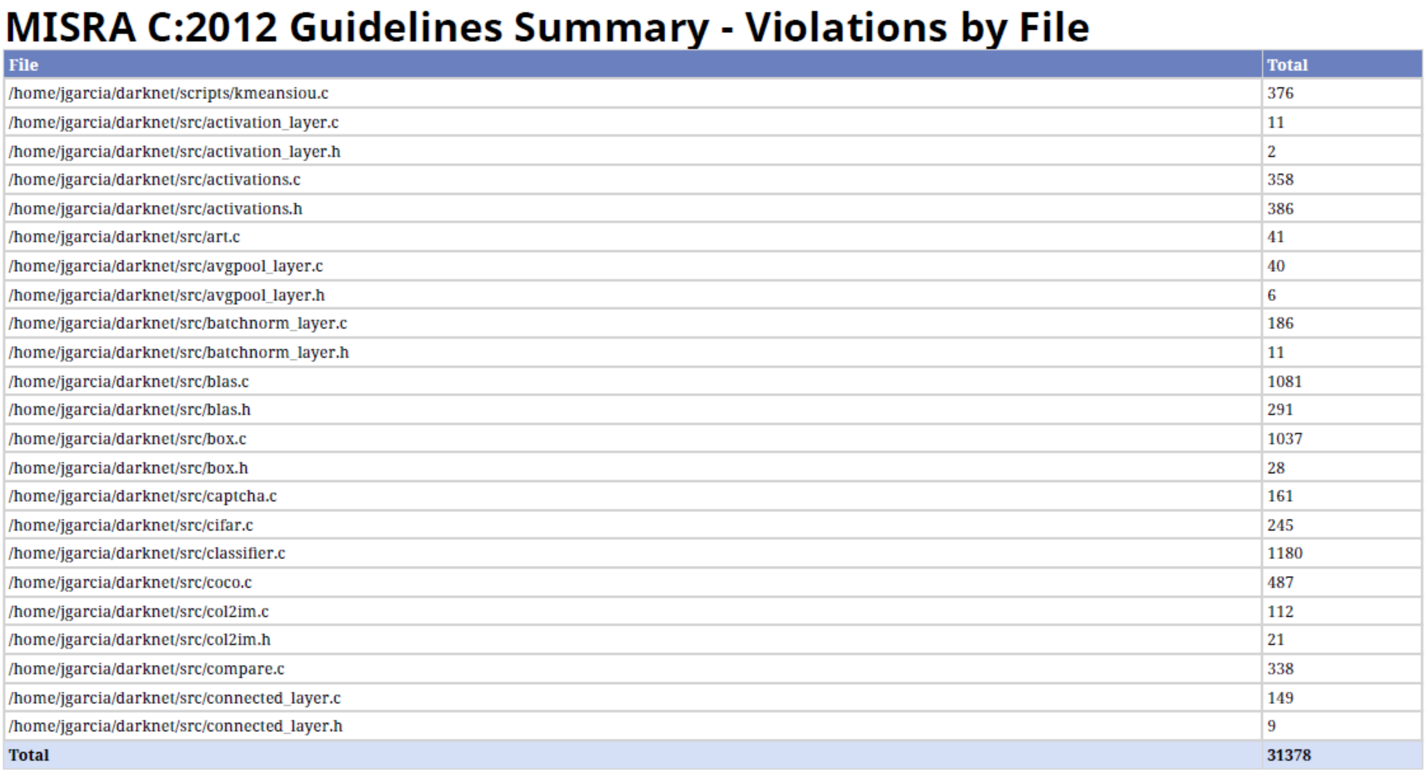


Figure 10: Results obtained for running two diverse AI frameworks (i.e., Pytorch and Darknet).

However, given their high complexity as well as their open-source nature, AI frameworks are not designed in compliance with functional safety standards. In fact, if we use Polyspace to detect the violations of coding guidelines (e.g., MISRA C) on AI frameworks (e.g., Darknet), we see that they are not compliant with coding standards. Figure 11 and Figure 12 show a caption of the results obtained from running Polyspace to Darknet. A total amount of 31378 MISRA C:2012 violations have been found, being the following ones the top five most frequent:

- D4.6: typedefs that indicate size and signedness should be used in place of the basic numerical types.
- 12.1: The precedence of operators within expressions should be made explicit.
- 10.3: The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category.

- D1.1: Any implementation-defined behaviour on which the output of the program depends shall be documented and understood.
- 10.4: Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.



MISRA C:2012 Guidelines Summary - Violations by File

File	Total
/home/jgarcia/darknet/scripts/kmeansiou.c	376
/home/jgarcia/darknet/src/activation_layer.c	11
/home/jgarcia/darknet/src/activation_layer.h	2
/home/jgarcia/darknet/src/activations.c	358
/home/jgarcia/darknet/src/activations.h	386
/home/jgarcia/darknet/src/art.c	41
/home/jgarcia/darknet/src/avgpool_layer.c	40
/home/jgarcia/darknet/src/avgpool_layer.h	6
/home/jgarcia/darknet/src/batchnorm_layer.c	186
/home/jgarcia/darknet/src/batchnorm_layer.h	11
/home/jgarcia/darknet/src/blas.c	1081
/home/jgarcia/darknet/src/blas.h	291
/home/jgarcia/darknet/src/box.c	1037
/home/jgarcia/darknet/src/box.h	28
/home/jgarcia/darknet/src/captcha.c	161
/home/jgarcia/darknet/src/cifar.c	245
/home/jgarcia/darknet/src/classifier.c	1180
/home/jgarcia/darknet/src/coco.c	487
/home/jgarcia/darknet/src/col2im.c	112
/home/jgarcia/darknet/src/col2im.h	21
/home/jgarcia/darknet/src/compare.c	338
/home/jgarcia/darknet/src/connected_layer.c	149
/home/jgarcia/darknet/src/connected_layer.h	9
Total	**31378**

Figure 11: Subset of MISRA C:2012 violations by file found by Polyspace in Darknet

MISRA C:2012 Guidelines Summary - Violations by Rule

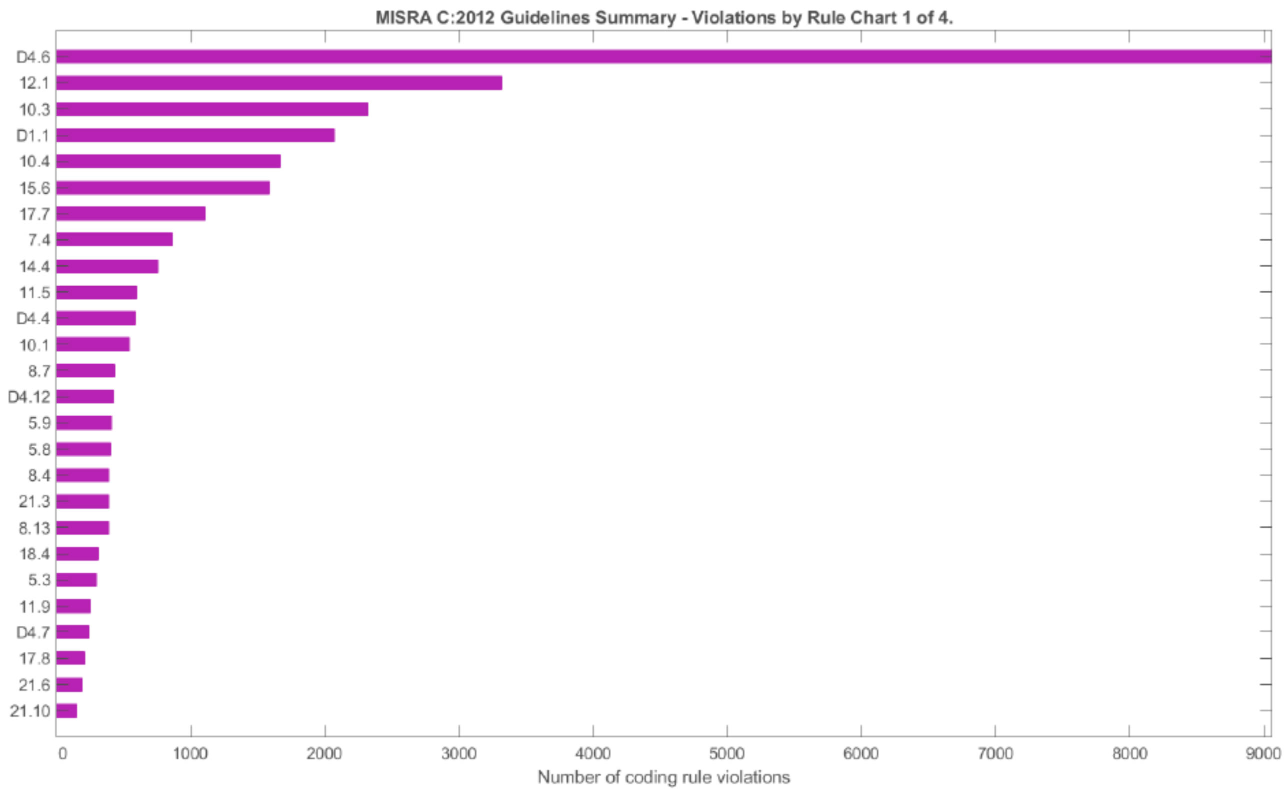


Figure 12: Subset of MISRA C:2012 violations by rule found by Polyspace in Darknet

Therefore, the diverse framework redundancy scheme shown in Figure 9 can be implemented using a compliant AI framework for one of the redundant nodes (e.g., Pytorch and MISRA C compliant Darknet).

The compliant framework is implemented in C following coding rules so it must be executed in the CPU, which is much slower that the execution in the GPU. In fact, running the inference of a pretrained YOLOv7-tiny model using Pytorch on the NVIDIA Orin GPUs takes an average of 55.421ms. Whereas running the inference of the same model using Darknet on the NVIDIA Orin CPUs takes an average of 278.759ms. So, the unchanged AI framework (running in the GPU) shall be combined with the coding guideline compliant AI framework (running in the CPU). However, redundant nodes are not executed with the same frequency, so the comparison in the decision function occurs once every 5 executions of the non-compliant replica (this ratio is application dependent and shall be fixed for each specific use-case).

This approach aims to detect random as well as systematic faults within the inference platform. In fact, when combined with the MISRA C compliant framework, systematic SW programming or design faults in the framework are identified and mitigated.

3.1.2.5 Diverse Concept – Object detection vs object part detection [IKL]

As shown in Figure 13 , diverse redundancy at the concept level, relies on the redundant execution of DL models implemented following different approaches (e.g., object detection, part detection, free path detection…).

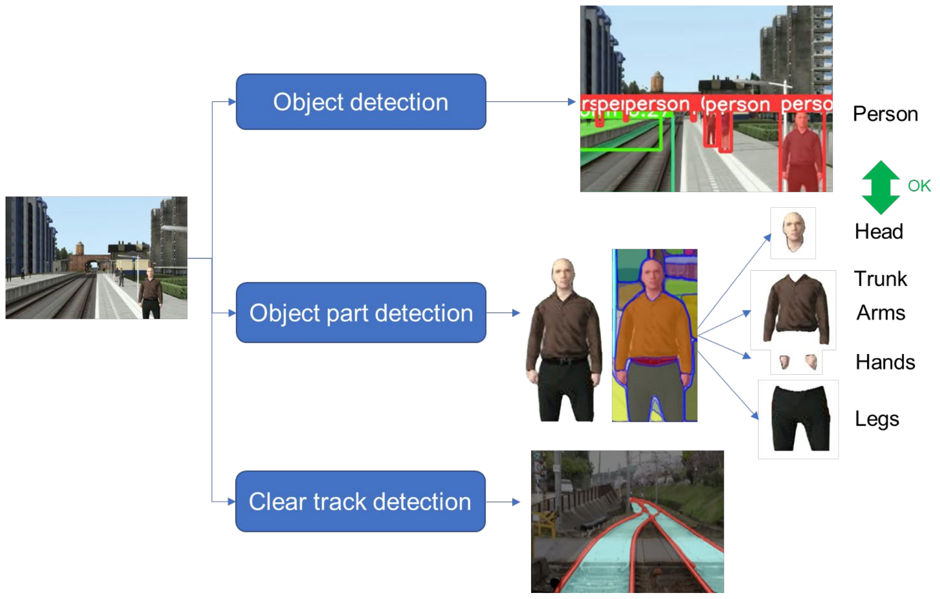


Figure 13: Example approaches for concept diversity.

One of the approaches for the implementation of diverse redundancy at concept level combines two diverse DL models based on different concepts, one for object detection (e.g., person, train, bicycle…) and the other for object part detection (e.g., head-trunk-arms-hand-legs (person)). The decision function receives the detections (objects and object parts) from the DL models and decides whether there are inconsistencies. In fact, if a person is detected but there is no detection of the person parts, there might be a false positive. Whereas if a person is not detected, but parts of a person are detected, there might be a false negative.

This technique implements diverse redundancy at concept and DL model development resulting on the diverse redundancy scheme shown in Figure 14 (DRS_5 according to Table 2).

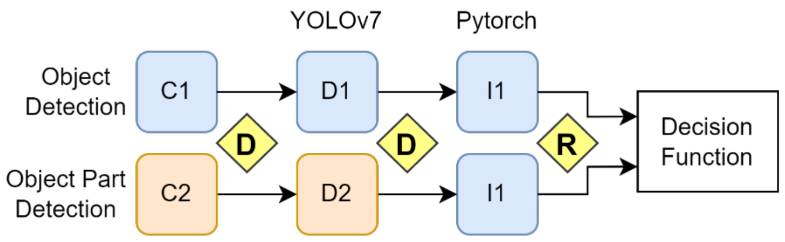


Figure 14: Concept and DL model diversity scheme using diverse concepts (i.e., Object Detection and Object Part Detection).

Figure 15 shows the results obtained for running two YOLOv7 models implemented following different concepts, one for object detection and the other for object part detection. The big bounding box shows the detected person, whereas smaller bounding boxes show the detected parts, for example, foot, leg, and arm.

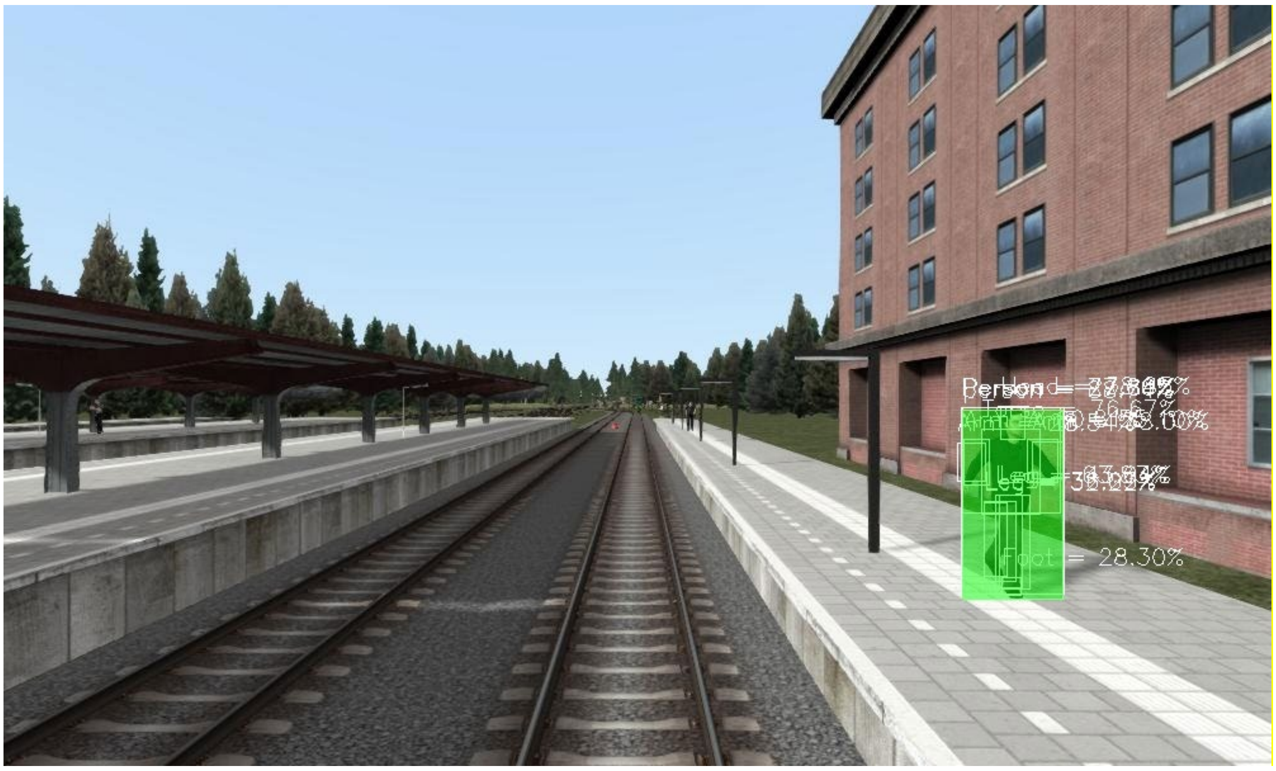


Figure 15: Results obtained for running two diverse concept models (i.e., Object Detection and Object Part Detection).

3.2 Diagnostic and Monitoring mechanisms

The reference safety architecture pattern is based on the hierarchical diagnostics and monitoring approach of Figure 16 . This strategy seeks to decouple AI specific techniques from traditional functional safety approaches and to ease, in this way, the later tailoring to incremental safety patterns.

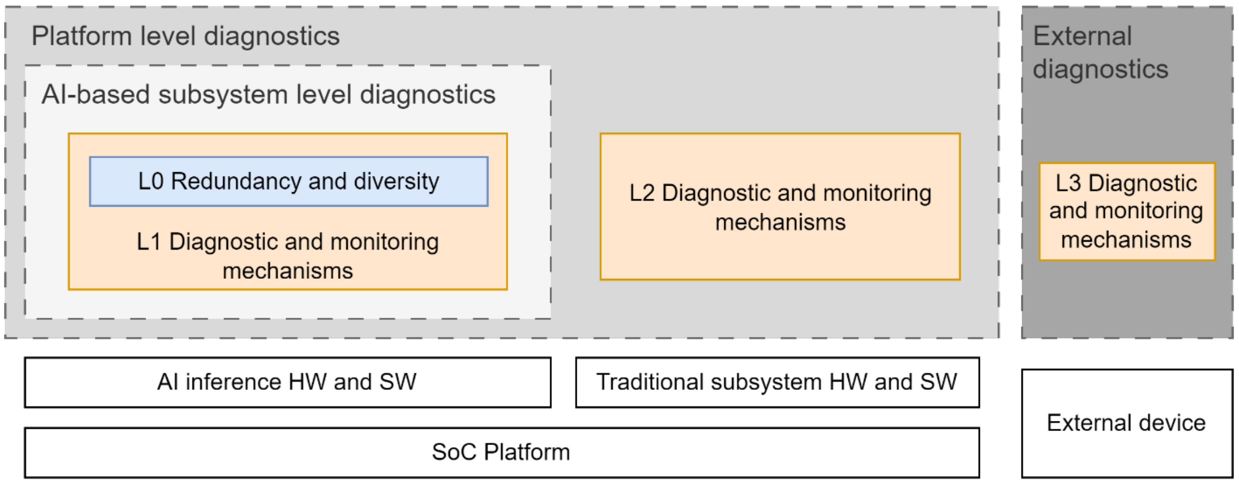

Platform level diagnostics
AI-based subsystem level diagnostics
L0 Redundancy and diversity
L1 Diagnostic and monitoring mechanisms
L2 Diagnostic and monitoring mechanisms
External diagnostics
L3 Diagnostic and monitoring mechanisms
AI inference HW and SW
Traditional subsystem HW and SW
External device
SoC Platform

Figure 16: Hierarchical diagnostics and monitoring approach

The proposed approach is based on four diagnostic levels (L0 to L3) to incrementally cover the errors on the following subsystems:

1. AI-based subsystem level diagnostics (L0 – L1): The redundancy and diversity (L0) described in previous subsection is complemented by additional diagnostic and monitoring mechanisms (L1) with the purpose of detecting runtime errors or model

insufficiencies and anomalies on the AI subsystem and the elements required for its execution (e.g., accelerators, AI frameworks, etc.). Note that L1 diagnostics and monitoring mechanisms may, in turn, include AI components (e.g., to assess suitability of the data used for inference, or monitoring abnormal results from specific DNN layers).

2. Platform level diagnostics (L2): This includes L0 – L1 diagnostics and additional L2 diagnostics to detect runtime errors on additional platform HW and SW components following traditional functional safety practices and diagnostics techniques (e.g., memory self-tests, freedom from interference at platform level…).
3. External diagnostics (L3): Following traditional functional safety practices some diagnostic may be implemented on an external device such as an external watchdog or microcontroller.

Each of these diagnostic and monitoring levels is further described in next subsections, except L0 that has been already described in Subsection 3.1.

3.2.1 L1DM mechanisms – AI subsystem

As stated above, L1 diagnostic and monitoring mechanisms aim to diagnose the AI subsystem and the platform resources required for its execution. All in all, the L1 mechanisms have the following purpose:

- Diagnose the AI model to detect runtime errors or model insufficiencies and anomalies.
- Diagnose the elements that participate in AI inference: complex HW and SW stack, black / grey box diagnostics.
- Monitor the use of resources at the AI subsystem level to guarantee freedom from interference at platform level (L2)

These mechanisms can be applied on different elements of the AI-subsystem. Bellow we present different L1 diagnostic mechanisms that may be applied to the different AI-subsystem elements.

- Inputs: diagnostic mechanisms for input correctness, data quality, data redundancy, temporal consistency…
  - Anomaly detectors
  - Input image comparison (e.g., consecutive input frames, input from redundant cameras).
- Model: diagnostic and monitoring mechanisms for execution errors, timing, program sequence, neuron activation patterns…
  - Generation of execution signatures.
- Outputs: diagnostic mechanisms for outputs, plausibility checks, input-output correlation, temporal consistency …
  - Output trajectory prediction
- Resource usage: monitoring mechanisms for resource usage (e.g., CPU/GPU usage, memory usage)

The following subsections provide further details on the implementation of the techniques for L1 diagnostic and monitoring.

3.2.1.1 Input image comparison

Within this section we propose different diagnostic techniques that can be applied in order to detect faults in the input of the AI-subsystem:

- **Input temporal consistency:** this approach computes the difference between two consecutive input frames. It allows detecting inconsistencies in consecutive input frames due to multiple errors (e.g., back frame, lost frame…). Figure 17 shows an example of the result obtained for the input temporal consistency check with lost frames. The figure shows on top of each of the consecutive input frames, the difference with respect to the previous one. As it can be seen in the figure, when some input frames are lost (top right image), the computed difference increases considerably. Therefore, the proposed solution is a feasible approach to detect input temporal inconsistencies.

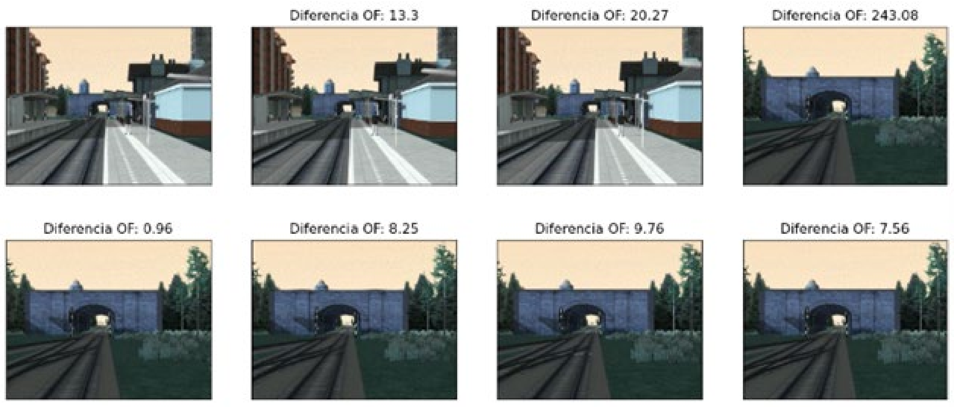
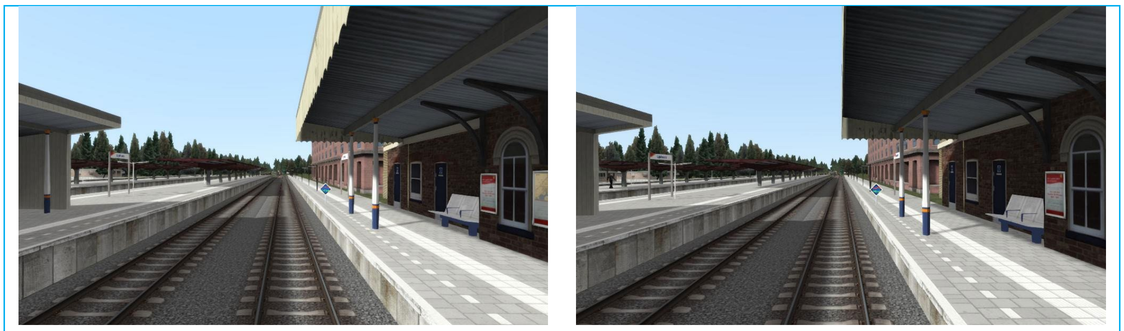


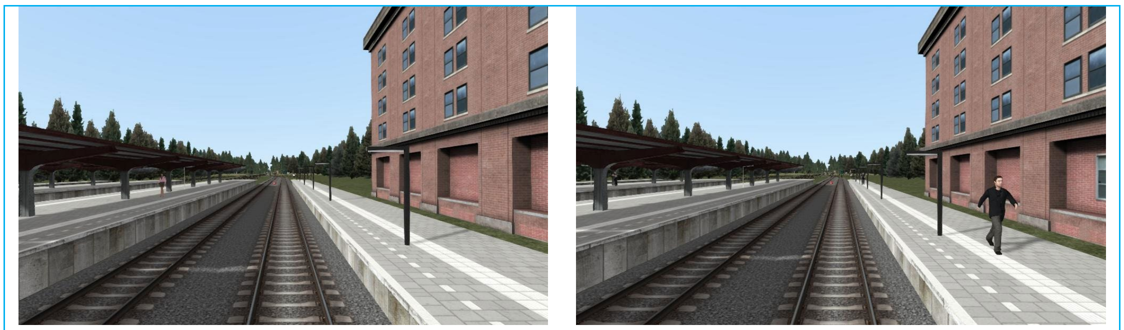
Figure 17: Results obtained for input temporal inconsistency check with lost frames.

Redundant input consistency: this approach consists of applying transformations to redundant input images and comparing them. Several solutions might be applied following this technique, for example, transforming redundant input images into histogram and computing the correlation between both, or transforming them into vectors and computing the cosine distance between both vectors to compare them. This approach aims to detect random and systematic faults on the input devices as well as on the image pre-processing.

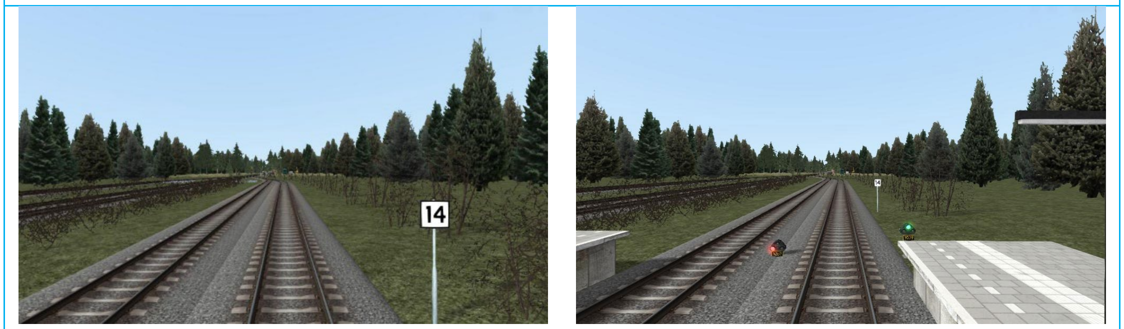
The following figures show the results obtained for the redundant input (i.e., left and right camera) consistency check using the histogram correlation and the cosine distance. The greater the difference between redundant input images, the lower the histogram correlation and the greater the cosine distance. For example, for the fourth test case, the histogram correlation decreases bellow 0.6 and almost reaches 0.4. In contrast, for the rest of the tests, the histogram correlation is always above 0.78 and the cosine distance bellow 0.15.



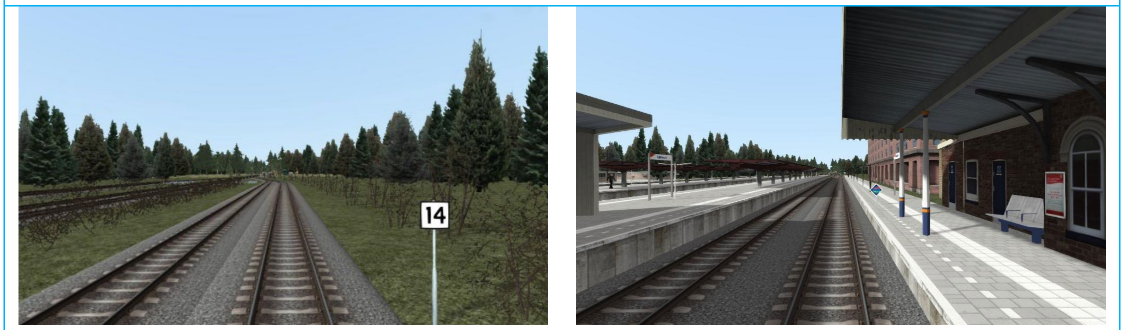
Histogram correlation:0.7800929836385664
Cosine distance: 0.03096352



Histogram correlation:0.7800929836385664
Cosine distance: 0.03096352



Histogram correlation:0.9228015960299797
Cosine distance: 0.14926167



Histogram correlation:0.5432720393166135
Cosine distance: 0.39401721

Figure 18: Results obtained for redundant input consistency check using histogram correlation and cosine distance.

3.2.1.2 Generation of execution signatures

Convolutional Neural Networks (CNNs) are algorithms allowing to perform complex functions, such as object detection, efficiently and accurately. This has led to their adoption in safety-related systems such as autonomous systems. One of the most time-consuming elements on these algorithms are the matrix-matrix multiplications (MMMs), which require high computational performance platforms to be deployed under. Therefore, this technique aims to protect the CNNs by adding a set of diagnostics techniques in the MMM [14].

The computation of the MMM lies on arithmetic operations based on additions and multiplications as it can be seen in Figure 19. We employ the safe catalogue of diagnostics proposed in [14] to generate the Execution Signatures (ESs) of the MMM computation and guarantee the bit-wise correctness of the data. Additionally, this approach ensures the proper program sequence with a reduced number of bits and diagnoses all internal platform components participating in the computation.

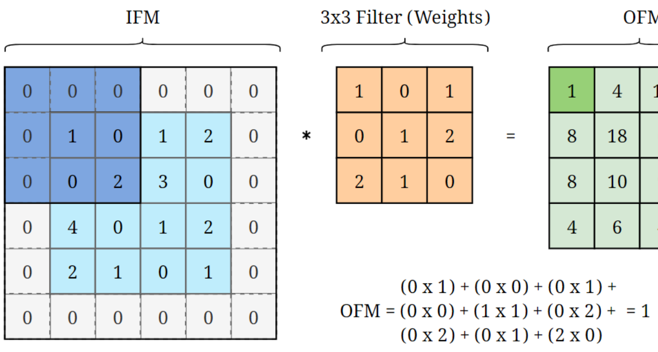

IFM
3x3 Filter (Weights)
OFM
(0 x 1) + (0 x 0) + (0 x 1) +
OFM = (0 x 0) + (1 x 1) + (0 x 2) + = 1
(0 x 2) + (0 x 1) + (2 x 0)

Figure 19. Example of a convolutional operation

These diagnostics techniques, composed of XOR, two's and one's complement addition, Fletcher, and CRC, are commonly used in the state of the art to ensure network message data integrity. Since MMM usually is computed through nested loops, we have included them individually in each of the loops or combining the diagnostics in the different loops to generate execution signatures of the variables involved in the MMM (C = A x B). As an example of how these diagnostics can be employed to protect data, we depict the CRC in Figure 20.

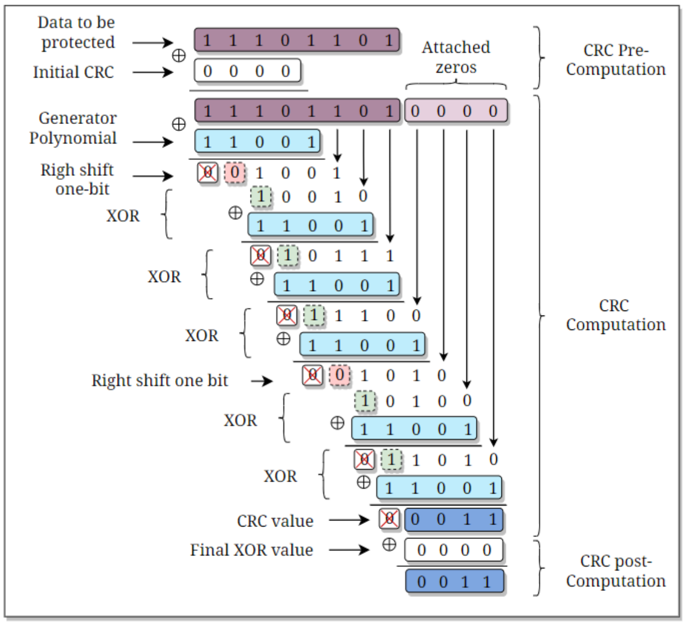


Figure 20. Example of how to perform CRC

However, the implementation of the CRC in MMM alone does not provide error detection capabilities; it must be integrated with safe architectural patterns such as redundancy (with or without diversity) or periodic diagnosis with design-time fixed data patterns. In Figure 21, there is included two commonly used safety patterns together with the pattern scheduling associated to each of them[2].

[2] Note that MMM refers to the matrix multiplication without diagnostics and "Safe MMM" to the MMM including the safety diagnostics techniques.

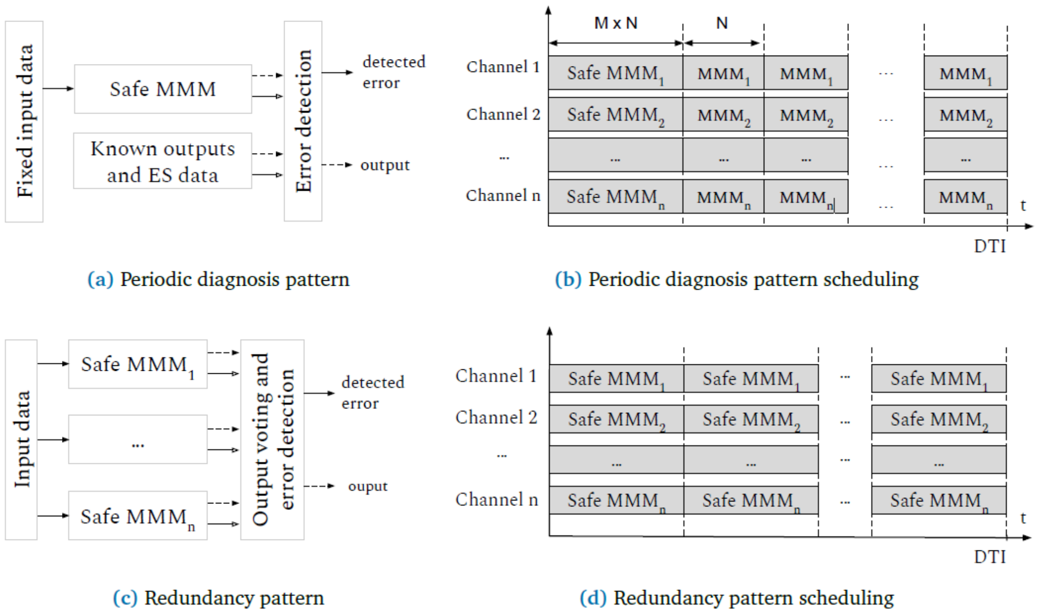


Figure 21.Candidates Safety Architectural Patterns [Jdez2022]

3.2.1.3 Output trajectory prediction

This approach aims to check the output temporal consistency. To this end, it implements a Kalmar filter to predict the trajectory of elements detected by the DL model. It allows detecting inconsistencies in the output of the DL model, such as and object that the model fails to detect in a particular input frame. Figure 22 shows an example of the result obtained for the output temporal consistency check where the algorithm fails to detect the object on a particular input frame. The green dot in the images shows the prediction of the Kalmar filter. As it can be seen in the image shown in the bottom-left of the figure, the proposed solution succeeds to predict the position of the person when the DL model fails to detect it.

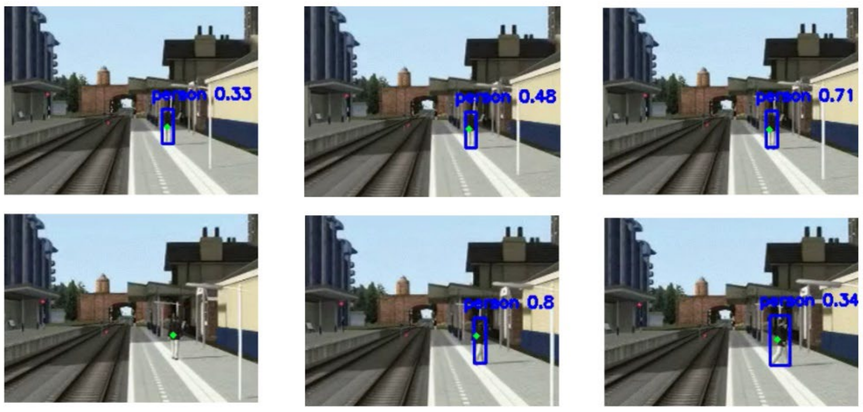


Figure 22: Results obtained for output temporal consistency check on object detection fail.

3.2.1.4 Input-Output correlation

This approach aims to detect the temporal consistency between the input image and the DL model output. To this end, this solution extends the input temporal consistency technique (described in Section 3.2.1.1) and correlates the output of this diagnostic (i.e., too high difference between subsequent input frames) with the DL model output (i.e., detected elements) to detect inconsistencies between the input frames and DL model output. In fact, if the input temporal consistency diagnostic technique detects a big difference between subsequent input frames, a big difference is expected on the elements detected by the DL model. Whereas, if no big difference is detected between subsequent input frames, similar elements should be detected by the DL model.

3.2.1.5 Information monitoring

Apart from diagnosing the correct behaviour of the AI/ML constituent, L1DM is also responsible for collecting low level behaviour information of the AI/ML constituent and transmitting it to L2DM for platform level diagnostics (e.g., freedom from interference as later explained in Subsection 3.2.2.2). This monitoring relies on the platform observability provided by WP4 and described in D4.1 deliverable. To this end, L1DM will use the PMUlib developed in SAFEXPLAIN WP4, which provides access to a number of events in the platform to capture diverse aspects of execution, such as resource usage and timing.

Similarly, L1DM provides the health information to L2DM, in such a way that if any of the mechanisms in the AI-subsystem (i.e., L0DR or L1DM) detect an error, the health management in L2DM can trigger the required safety reaction.

3.2.2 L2DM mechanisms – Traditional subsystem

L2 diagnostic and monitoring mechanisms aim to detect runtime errors on additional platform hardware and software components following traditional functional safety practices and diagnostics techniques required by standards (see Section 3.2.2.1). However, these standards do not reflect the increasing complexity of emerging systems, therefore, L2DM mechanisms shall integrate advanced diagnostic approaches for high-performance platforms as well as the for the software applications that run on them (covered in Section 3.2.2.2).

3.2.2.1 Traditional functional safety diagnostics

Traditional functional safety standards such as IEC 61508, ISO 26262 and EN 5012x define the faults or failures that shall be considered to reach different degrees of Diagnostic Coverage (i.e., Low (60%), Medium (90%) or High (99%)) for the different system components and proposes the applicable diagnostic techniques for each: electromechanical devices, discrete hardware, bus, CPU, memories, clock, communications, sensors, final elements, etc.. The defined diagnostic strategy will comprise among others, startup diagnostics, periodic diagnostics, as well as forced checks of the safety function. On the software side, functional safety standards propose techniques and measures to prevent systematic failures on the different stages of the development life-cycle (e.g., requirements specification, design and development, testing and integration…). For each technique or measure, the standard gives a recommendation (highly recommended (HR), recommended (R), not recommended (NR)) according to the target safety integrity level. Additionally, when the software implements safety functions of different safety integrity levels, it shall be demonstrated either that independence of execution is achieved both in the spatial and temporal domains, or that any violation of independence is controlled. These techniques have been widely applied over the years and are considered state-of-the-art so this report, and the SAFEXPLAIN project, do not focus on their application.

However, these standards do not reflect the increasing complexity of emerging systems, neither at hardware architectural design, nor for the software applications that run on them. Except for ISO 26262 that included a new part 11 with the topic of multicore processors in its second edition, the mechanisms described in functional safety standards focus on single-core architectures, with buses (no complex interconnects), simple memory hierarchies and no accelerators as those required to achieve the required performance in ML inference. For instance, the mechanisms defined in standards to achieve temporal independence do not generally contemplate the concurrent access to shared resources as they were not originally defined for high-performance embedded architectures with parallel access to such resources. Therefore, L2 diagnostics shall integrate traditional functional safety diagnostic mechanisms (out of the scope of this document) together with advanced diagnostic approaches for high-performance platforms explained in next subsections.

3.2.2.2 Techniques for multicore / high-performance platforms

Most HPEC platforms integrate multicore processors together with accelerators for ML inference, as it is the case of the NVIDIA Jetson Orin platform introduced in Section 1.1.2. ISO 26262-11 section dedicated to multicores, warns about the fact that multicores are subject to timing faults and it highlights the importance of independence of execution by dedicated analyses and countermeasures such as, the specification of timing constraints and detection of timing requirement violations, doing an upper estimation of resources, evaluating the influence of hardware and software interactions and evaluating timing and execution failure modes. Similarly, the Certification Authorities Software Team (CAST), an international group of certification and regulatory authority representatives from the Federal Aviation Administration (FAA), provide guidance for ensuring safe implementation of multicore processing in the avionics domain [8] [15]. Next table summarizes the objectives of CAST-32A and AMC 20-193:

Table 3: Summary of CAST-32A and AMC 20-193 Objectives [15]

ID		Description
Software Planning	PL_1	**Include MCP specific planning details in the SW plan doc.** Specific processor, number of active cores, software architecture, dynamic software features, whether it hosts an IMA-like system (with applications from different systems) or not, Robust Partitioning supported or not, methods and tools for development and verification.
Planning and Setting Resources	RU_1	**Determine configuration settings that enable to satisfy the functional, performance and timing requirements.**
	RU_2	**Critical configuration settings shall be static and protected against unintended modifications.**
	PL_2	**Include a high-level description of shared resource usage and active dynamic hardware features in the hardware and software planning documents.** Intended shared resource allocation and verification to prevent resource capabilities from being exceeded.
Inference Channels and Resource Usage	RU_3	**Identify interference channels and verify the chosen means of mitigation.** Interferences caused by shared memory, shared cache, interconnect, shared I/O or any other shared resource.
	RU_4	**Identify available resources in the intended final configuration, allocate them to the applications and verify that the demands do not exceed the available resources (under worst-case scenarios).**

Software Verification	SWV_1	**Verify that all software components function correctly and have sufficient time when all the software is executing in the intended final configuration.** Depends on the platform classification: 1. Platforms with Robust Partitioning: SW verification and WCET analysis can be done separately for each SW app. 2. All Other Platforms: If interference is mitigated for any software component or set of requirements, the verification of such components can be done separately. Otherwise, verification and WCET analysis shall be done with all software components executing together.

The objectives in Table 3 can be summarized into the following four high level principles:

1. Determining the final configuration. The designer shall determine which is the intended _final configuration that will enable to satisfy system requirements (PL_1, RU 1) and this configuration shall be protected against modification at runtime (RU 2).
2. Managing interference channels. It is required to identify interference channels in the intended final configuration and to define the means to either avoid interference by design or upper-bound it so that timing deadlines are not exceeded (RU 3). Upper-bounding interference involves analysing the use of shared resources and designing the means to control contention (PL 2).
3. Verifying the use of shared resources. Resource usage and data and control flows among cores shall be verified by guaranteeing that the software does not exceed the use of available resources even in worst-case scenarios (RU 4, SWV 1 and SWV 2).
4. Error Management. The system shall include features for multicore specific error detection and handling.

In order to achieve these four principles, within WP4, we define techniques and measures to ensure freedom from interference at platform level, with the support of the following L2 diagnostic and monitoring (L2DM) mechanisms:

- L2DM Configuration check: The configuration shall be defined and verified during system development process. Then, at runtime, L2DM checks that this configuration is kept (e.g., by a CRC check). Besides performance and functional correctness, also *Freedom from interference* can be highly dependent on this configuration, thus steering the focus of WP4 hardware analysis on platform support for interference mitigation, as reported in D4.1 [16]:
    - Regarding spatial independence, most platforms usually provide memory management support allowing to set memory regions with specific permissions, as well as hardware support, such as Memory Management Units (MMUs) allowing to enforce those permissions. Therefore, appropriate operating system support will allow realizing spatial partitioning.
    - Regarding temporal independence, the platform could also provide explicit support to realize it. In particular, we aim at bounding the maximum interference that a software component might cause on the execution of another, building on the existing platform support, which provides both, means to achieve some degree of partitioning (e.g., cache partitioning), as well as means to measure interference whenever partitioning is not enough. When robust partitioning cannot be achieved by the configuration, interference channels and shared resources shall be managed as required by CAST-32A / AMC 20-193. This is covered in next point.
- L2DM Interference mitigation and control: This L2DM mechanism addresses both "managing interference channels" and "verifying use of shared resources" principles from CAST-32A / AMC 20-193. L2DM is the responsible to check that all critical platform SW

components meet with their deadline and if this is the case, it refreshes an external watchdog (L3). In this way, if any critical software component violates a timing deadline or if the platform is not able to run L2 diagnostics, the L3 external device will be able to detect it. To this end, it is required that L2 knows the deadline of each critical software component, which shall be determined at design time by WCET analysis. The timing analysis strategy adopted in SAFEXPLAIN, which is explained in D4.1 [16], has specific implications on L2DM. Two main scenarios have been considered:

- WCET of each component takes into account worst possible interference for a given platform configuration. If at design time it is verified that under worst-case interference the WCET of the critical software component is below its deadline, then at runtime the L2DM is only checking that this condition is always preserved at runtime. Next Figure 23 represents this, assuming there is a critical software component 'A', at design time the $WCET_A$ is determined for a given design time configuration ($CONF_D$). Then at runtime, L2DM checks that the actual platform configuration ($CONF_R$) is consistent with the design time configuration and that execution time of SW component A (ET_A) is always below its deadline. This approach applies to all critical components in the platform. If all of them meet the deadline and all other traditional functional safety diagnostics of L2DM are passed, then L2DM refreshes an external watchdog that can be part of L3DM.

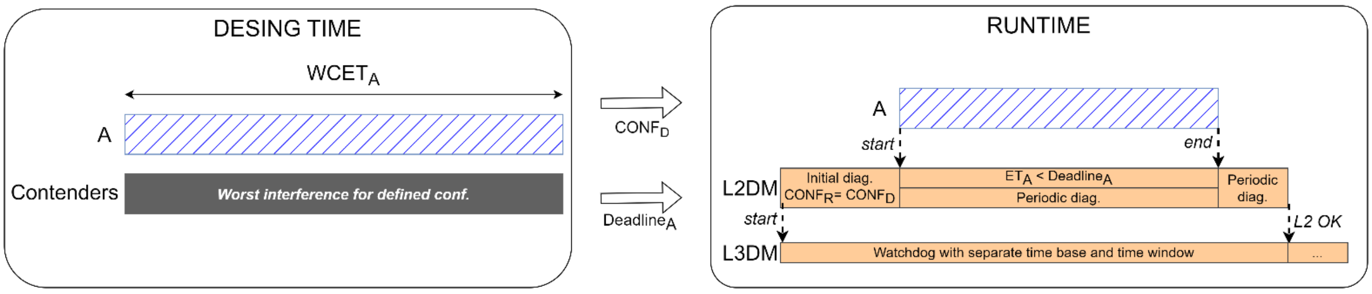


Figure 23: WCET of each component considers worst possible interference for a given platform configuration.

Following this approach, the resulting WCET estimates are usually over pessimistic, as the worst possible interference in such complex platform architectures can be very high. This can be reduced either by optimizing the configuration so that interferences are minimized or by the second approach explained below.

- WCET of each component depends on applications running concurrently for a given platform configuration, and is therefore only valid for a specific setup or scenario. This WCET estimation assumes an upper bound of the interference that the critical component can support when it is being executed. This is depicted in Figure 24. In this case, at design time, apart from the design time configuration ($CONF_D$) and the WCET of component A with limited interference ($WCET_{A\_Bounded}$), the interference limit of contenders used to estimate that WCET shall be determined too ($InterferenceLimit_A$). At runtime, L2DM follows the same approach as in the previous case and it additionally checks that contenders (e.g., SW component 'B' and 'C' on the right side of Figure 24 do not exceed the defined $InterferenceLimit_A$. To check this at runtime, different strategies defined in WP4 can be used, depending on the approach used to obtain the WCET estimate at design time. For instance, the number of active contenders can be monitored, or the specific usage (e.g., RU_B and RU_C in Figure 24) of given shared resources (e.g., caches, memory, interconnects…). To this end, WP4 analysed the monitoring capabilities in the Orin platform and elaborated an holistic approach for timing and residual (after configuration) interference characterization, as described in D4.1 [16].

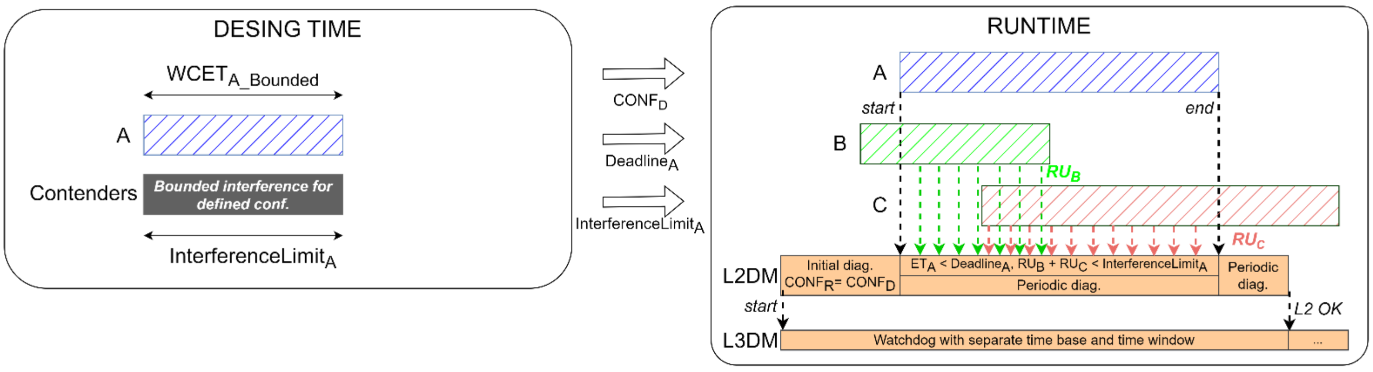

DESING TIME
WCETA_Bounded
A
Contenders
Bounded interference for defined conf.
InterferenceLimitA
CONFD
DeadlineA
InterferenceLimitA
RUNTIME
A
start
end
B
RUB
C
RUC
L2DM
Initial diag. CONFR= CONFD
ETA < DeadlineA, RUB + RUC < InterferenceLimitA
Periodic diag.
Periodic diag.
start
L2 OK
L3DM
Watchdog with separate time base and time window
...

Figure 24: WCET of each component depends on applications running concurrently for a given platform configuration (valid only for a specific setup or scenario).

- L2DM Health Management: L2DM includes an error management module to react to different platform level errors. To this end, L2DM collects the results from L0 and L1 diagnostic and monitoring mechanisms. Whenever L0, L1 or L2 diagnostics and monitoring detects an error in the platform, either by startup diagnostic tests or periodic checks, L2DM is the responsible of triggering the required reaction. This will depend on the specific error and the application itself. For instance, in a TMR architecture if L0 determines that one of the redundant instances is providing incorrect inputs, the system could keep working for a limited time in a degraded mode with the other 2 active redundant instances. Similarly, if the interference mitigation and control of L2DM determines that a contender is using a shared resource more than planned in the design, the action could be to force it to stop so that it does not cause uncontrolled interferences to the critical component. In many other cases, the reaction could be to move the system to the safe state. In addition, the health monitoring in L2DM shall provide an interface with L3DM (e.g., refreshing a watchdog), in such a way that if there is a problem in the platform that affects the correct execution of L2DM, the external L3DM will be able to detect it.

3.2.3 L3 mechanisms – External

L3 diagnostics and monitoring mechanisms comprise traditional external safety mechanisms required by functional safety standards IEC 61508 / ISO 26262 / EN 5012x (e.g., external watchdog, time and power monitoring, external monitoring unit…). Moreover, the L3 diagnostics and monitoring implements error management for the lower-level diagnostic and monitoring mechanisms (i.e., L0, L1 and L2 mechanisms).

The definition and implementation of traditional functional safety mechanisms is out of the scope of the project and will therefore not be covered within this deliverable.

3.3 Supervision function

The supervision function applies elements of control theory to minimally bound AI operations. All in all, it shall control that the AI system works within a predefined safety envelope. To this end, the supervisor function shall determine a safe subset of the action space or safe envelope (prior to the execution or at runtime) and compute a set of constraints or limits. The supervision function comprises techniques such as:

- A non-AI safety function that outputs an acceptable range of outputs for a given input and limits the intelligent control output (this approach might not be appropriate for systems with dynamics).
- Defining minimal bounds though control theory methods (e.g., barrier functions approaches).
- Checking functions such as metrological self-check or self-validation.

In addition, the supervision function implements explainability techniques to provide understanding on the DL model operation as well as on the decisions, recommendations, or predictions it provides.

Detailed techniques for the implementation of the supervision function are provided within deliverable D3.1 [17].

3.4 Traditional subsystem

By traditional subsystem we refer to a subsystem that does not include any ML model according to the system decomposition of Figure 1. In the scope of the reference architecture, this traditional subsystem can be either:

- Functions complementary to the ML constituent, required to perform the function required by it.
- Fallback subsystem: Following known proper functional safety precautions, a safe back-up function to the ML component might be implemented within the non-AI subsystem. This back-up decision system will take over the system in the event that a functional safety problem is detected, ensuring no harm is done by the AI system. The implementation of such a fallback function is use-case dependent and is therefore out of the scope of the project.

4 Safety Patterns

Following the incremental strategy described in Section2.2, this section presents the mapping of the reference safety architecture pattern (see Figure 4) to the different DL usages levels. In particular, three Safety Patterns are defined:

- Safety Pattern 1 (SP1) (for DL usage level D / EASA Level 1).
- Safety Pattern 2 (SP2) (for DL usage level C / EASA Level 2).
- Safety Pattern 3 (SP3) (for DL usage level A1 / EASA Level 3).

The following subsections present, for each Safety Patterns, the proposed system architecture and safety techniques, as well as its application on the project target platform (i.e., NVIDIA Orin). The description of the Safety Patterns follows an incremental approach; therefore, each Safety Pattern comprises also the safety techniques of the previous one.

4.1 Safety Pattern 1 (SP1)

On this safety pattern, the AI subsystem aids the driver or person responsible of the safety of the system. However, the AI subsystem is not part of the safety function.

On the following subsections we present the system architecture for SP1 as well as the corresponding diagnostic and monitoring mechanisms.

4.1.1 System Architecture

Figure 25 shows the adoption of the reference architecture pattern presented in Section 3 to the Safety Pattern 1. This pattern assumes the integration of safety related software (i.e., non-AI subsystem, safety control, safety diagnostics) and non-safety related software (i.e., AI/ML constituent, supervision function) within the same SoC on a High-Performance Computing platform. The main components of the reference safety architecture are applied as follows in SP1:

- Diverse Redundancy: In this safety pattern, the safety function is implemented by the non-AI subsystem and the safety control, therefore, the AI/ML constituent has no safety implications. Given that, diverse redundancy is not required in the AI/ML constituent.
- Diagnostic and monitoring mechanisms: Even if the AI/ML constituent is not safety related, in SP1 the main challenge rests on the integration of both the safety software together with the software involved in the inference of the DL model and to guarantee freedom from interference among them. Therefore, L1DM is the responsible for monitoring resource usage and timing of the AI/ML constituent, and L2DM / L3DM to apply the required diagnostic mechanisms on the safety subsystem and guarantee freedom from interference. The application of the diagnostic and monitoring strategy in SP1 is further elaborated in Subsection 4.1.2.
- Supervision function: the supervisor has no safety implications and its purpose is to improve the explainability of the AI-based subsystem. Different explainability techniques are proposed within deliverable D3.1, Section 2.4 [17].
- Traditional subsystem: in SP1 the safety function is performed by traditional SW components. No fallback subsystem is considered.

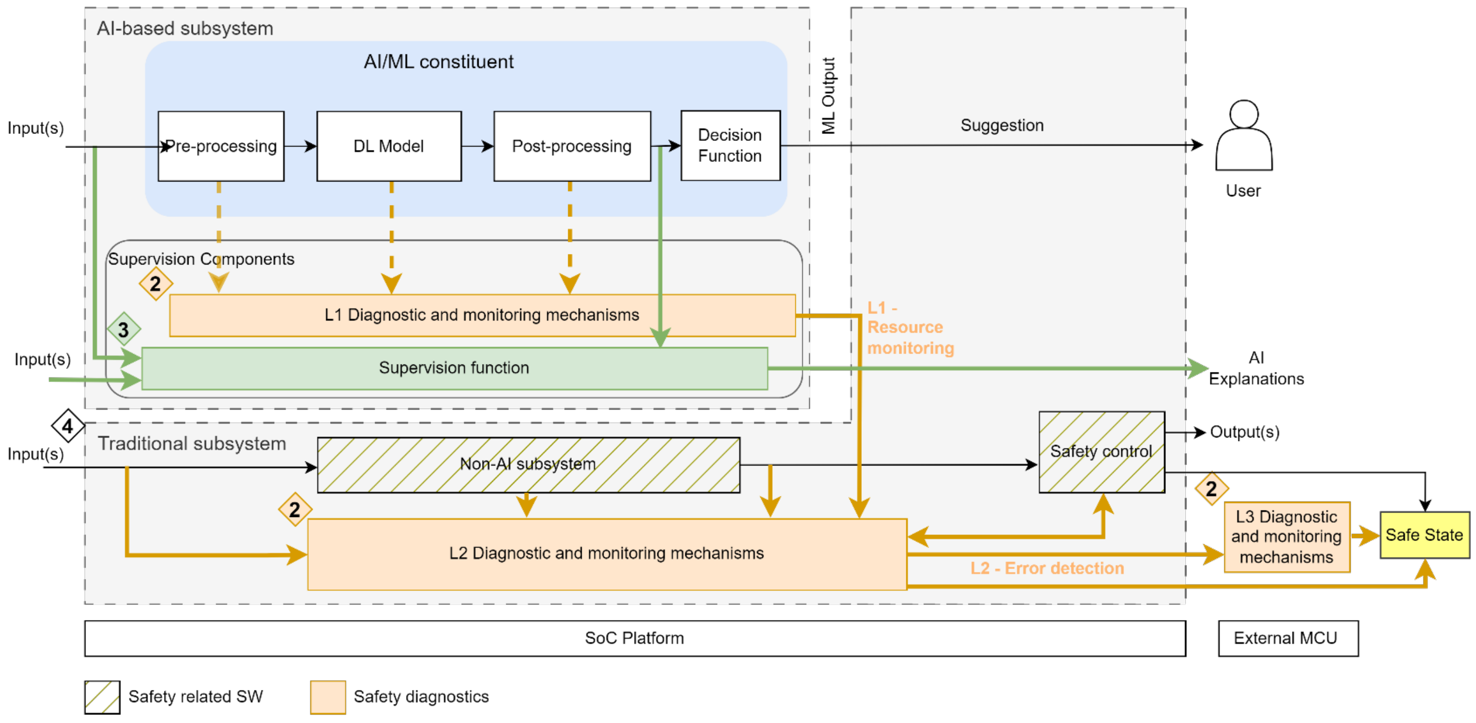


Figure 25: Conceptual System Architecture - Safety Pattern 1

4.1.2 Diagnostic and monitoring mechanisms

Next subsections explain the application of the hierarchical diagnostics and monitoring approach of Section 3.2 to SP1 on the SAFEXPLAIN platform.

4.1.2.1 L1DM – SAFEXPLAIN PLATFORM

As the AI subsystem is not safety related, there are no specific diagnostics for it. However, the L1 monitoring mechanisms are necessary to guarantee the required freedom from interference at platform level. The specific events that can be monitored in the SAFEXPLAIN platform are explained in D4.1 document. In this particular case, the monitoring of the use of shared resources among the safety and non-safety software components (i.e., memory, caches, interconnect) is of special interest. In SP1, L1DM will collect such events for the non-safety AI subsystem and provide the information to the L2DM.

4.1.2.2 L2DM – SAFEXPLAIN PLATFORM

The L2DM mechanism implements traditional functional safety techniques and measures according to the recommendations on functional safety standards. These diagnostics comprise start-up as well as periodic checks that can be applied to the different subsystems that participate in the execution of the safety function (e.g., sensors, CPU, memories, clock, electromechanical devices, etc.). To this end, NVIDIA provides support documentation as well as libraries for the implementation of safety diagnostics.

In addition, L2DM is responsible for checking system configuration and providing the mechanisms for interference mitigation and control. As introduced in Section 1.1.2, the NVIDIA Orin platform provides a number of diverse processing resources (CCPLEX core clusters, SPE, GPUs…) with multiple configuration options. The allocation of such resources to software components and their configuration play a crucial role in the mitigation of interference required by SP1. Figure 26 and Table 4 present different resources and configuration options that could be adopted for SP1.

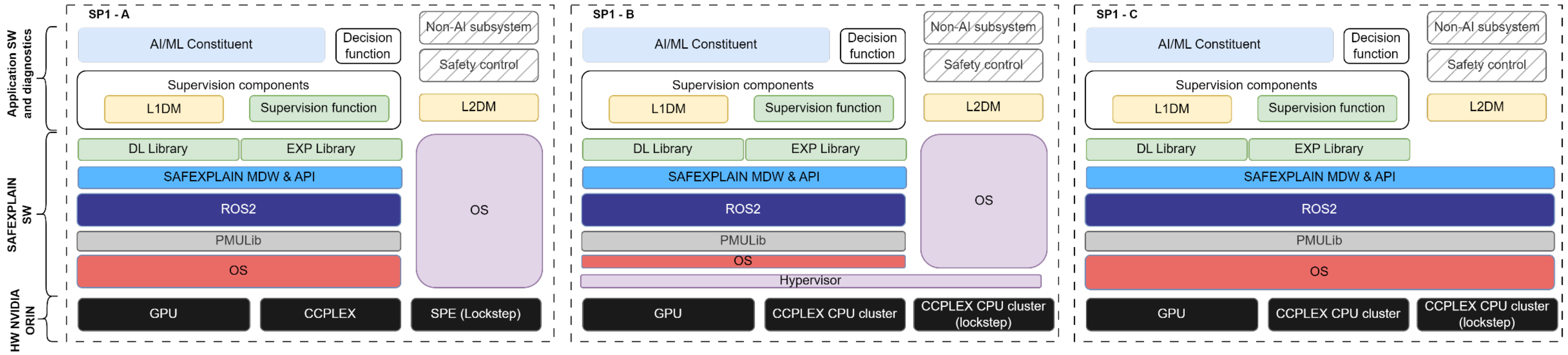


Figure 26: SP1 to NVIDIA Orin resource allocation and configuration options

Table 4: SP1 to NVIDIA Orin resource allocation and configuration options

SP1 Element	Safety / non-safety	SP1 - A NVIDIA Orin resources and configuration	SP1 - B NVIDIA Orin resources and configuration	SP1 - C NVIDIA Orin resources and configuration
AI/ML constituent	Non-safety AI based SW	CCPLEX CPU Cluster (Cortex A78) GPU for AI inference Memory controller fabric and traffic from CPU cluster to GPU MMUs for spatial independence SAFEXPLAIN SW Stack	CCPLEX CPU Cluster (Cortex A78) – different CPU cluster for safety SW GPU for AI inference Memory controller fabric and traffic from CPU cluster to GPU MMUs for spatial independence L4 cache partitioning or disabled SAFEXPLAIN SW Stack separated from safety SW by hypervisor	CCPLEX CPU Cluster (Cortex A78) – different CPU cluster for safety SW GPU for AI inference Memory controller fabric and traffic from CPU cluster to GPU MMUs for spatial independence L4 cache partitioning or disabled SAFEXPLAIN SW Stack
Supervision components	Non-safety traditional or AI based SW			
Decision function	Non-safety traditional SW			
Non-AI subsystem and safety control (safety function)	Safety traditional SW	SPE in lockstep configuration No sharing of caches with CCPLEX MMUs for spatial independence Safe RTOS on top of SPEs	CCPLEX CPU Cluster (Cortex A78) in lockstep – different CPU cluster for safety SW MMUs for spatial independence L4 cache partitioning or disabled Safe RTOS on top of hypervisor	CCPLEX CPU Cluster (Cortex A78) in lockstep – different CPU cluster for safety SW MMUs for spatial independence L4 cache partitioning or disabled SAFEXPLAIN SW Stack

L2DM	Safety traditional SW			
Pros		Highest independence: different processing elements, less shared resources, different OS and SW stack.	Better performance while preserving independence: different CPU clusters with partitioning approaches, different OS and SW stack.	Ease of integration, same SW stack for all platform elements.
Cons		Limited performance on SPEs.	Need of a hypervisor.	Less independence, required safety guarantees on the SAFEXPLAIN SW Stack.

The three SP1 configuration options rely on mechanisms provided by the platform, which include cache partitioning features in the CCPLEX, and performance monitoring counters in the CCPLEX allowing to monitor resource usage in the CCPLEX. So far, the monitoring capabilities of the shared L4 cache and the GPU are still under investigation in WP4, and freedom from interference in those resources may need to resort to software support such as, for instance, task scheduling constraints to avoid interference by construction as much as possible. Details on the different features can be found in D4.1 and will be further developed in forthcoming WP4 deliverables.

- **Independent computing components**: The NVIDIA Orin provides abundant computing resources, the safety critical tasks can be allocated either to the SPE or to a specific CCPLEX cluster, increasing independence and diversity among safety and non-safety tasks.
- **Spatial independence**. As explained before, the NVIDIA Orin platform provides explicit support to enable spatial independence in the form of MMUs. Therefore, spatial independence can be achieved by design with usual operating system support.
- **Temporal independence**. Safety related tasks needing temporal independence can be executed in the SPE or CCPLEX. The degree of independence achieved across tasks running in the same CCPLEX core cluster is relatively high if L3 cache partitioning is used, and even higher across tasks running in different core clusters as proposed in SP1 - B and SP1 - C. The remaining interference can be measured with existing event monitors so that safety measures can be built at software level if measured interference reaches system or application-specific thresholds. The specific strategy, monitors to use, and how to use them is part of D4.1 [16], and subsequent deliverables in WP4.
- **Support for lockstep redundancy**. As explained before, core clusters in the CCPLEX and the SPE provide lockstep support. Hence, safety related tasks requiring such redundancy can be deployed on the SPE or the CCPLEX by properly configuring the core cluster where they are run.

Following the approach described in Subsection 3.2.2.2, determining the configuration at system design time is crucial to perform the timing analysis and interference mitigation and control. Based on the configuration, the timing and interference bounds will be computed and at runtime, L2DM will check that the configuration corresponds to that defined at design time (e.g., through a CRC check).

4.2 Safety Pattern 2 (SP2)

For SP2 the AI-subsystem becomes a safety-critical component. Therefore, in addition to the mechanisms presented in SP1 to guarantee the independence among safety and non-safety software components, new mechanisms are required to ensure the correct behaviour of the AI-subsystem. In the defined incremental strategy, it is assumed that in SP2 the AI-based safety function has a low to medium integrity level (i.e., SIL 1 or SIL 2 according to IEC 61508).

4.2.1 System Architecture

SP2 consists of the same elements as those in the reference safety architecture, resulting in the same conceptual system architecture as that shown in Figure 4. However, the specific safety mechanisms that apply and the way of addressing them can vary according to the implications of the integrity level of the safety functions. As previously mentioned, for SP2 it is assumed that the safety functions executed by the AI-subsystem are assumed to be either SIL 1 or SIL 2. Next Table 5 introduced in D1.1. deliverable [3], shows the architectural implications of this integrity level according to IEC 61508-2:

Table 5: Maximum allowable safety integrity level according to IEC-61508-2 for Type B safety related elements (SP2)

Pattern	HFT	Safe Failure Fraction (SFF)			
		<60%	60% -90%	90% - 99%	≥ 99%
1oo1(D)	0	Not allowed	SIL 1	SIL 2	SIL 3
2oo2(D)	0	Not allowed	SIL 1	SIL 2	SIL 3
1oo2(D)	1	SIL 1	SIL 2	SIL 3	SIL 4
2oo3(D)	1	SIL 1	SIL 2	SIL 3	SIL 4
1oo3(D)	2	SIL 2	SIL 3	SIL 4	SIL 4

Therefore, SP2 requires either medium to high diagnostic coverage (60-90% for SIL 1 and 90-99% for SIL 2) or increasing the Hardware Fault Tolerance (HFT). However, achieving this HFT > 0 on the same chip has several implications on the hardware silicon level (IEC 61508-2 Annex E) which shall be guaranteed by the platform manufacturer. As the SAFEXPLAIN platform does not provide such guarantees, off-chip redundancy would be required, which is the case of SP3. For SP2, the focus is on HFT = 0 with low to medium diagnostic coverage.

Taking this into account, the main components of the reference safety architecture are applied as follows in SP2:

- Diverse Redundancy: On chip redundancy approaches are considered to improve the safe failure fraction and diagnostic coverage of the AI/ML constituent, further explained in Subsection 4.2.2.
- Diagnostic and monitoring mechanisms: Same mechanism as in SP1 apply, with additional diagnostics for the AI/ML constituent in the L1DM further elaborated in Subsection4.2.3.

- Supervision function: A supervisor function according to the description in the reference architecture shall be integrated too. The supervisor function does not have any particularities for SP2, so the main description in Section 3.3 can be checked.
- Traditional subsystem: depending on the application, a fallback subsystem implemented with traditional SW could be in place.

4.2.2 Diverse redundancy

From Table 2, diverse redundancy schemes with low or medium DC can be applied in this safety pattern, based on the specific requirements of the application (e.g., DSR_1, DSR_2, DSR_3, DSR_4). These techniques have been already explained in Section 3.1.

For the implementation of diverse redundancy in the SAFEXPLAIN platform, AI tasks often need the use of GPUs and/or AI accelerators. Depending on the diverse redundancy scheme, the redundant instances can either run redundantly on the GPU, or using diverse processing resources such as GPU and CPU, or GPU and other AI accelerators (DSR_2). The NVIDIA Orin platform allows multiple kernels to run simultaneously. However, the GPU does not provide explicit support for diverse redundancy, which needs to be built by external means using the techniques described before. Those techniques provide explicit support to manage random hardware faults affecting one of the redundant instances. However, faults potentially leading to common cause failures need additional support. In particular, if computation can be identical across redundant instances (e.g., some results are obtained applying the same operations on identical data), we may need mechanisms to enforce the use of separated computing resources and to make those computations not to occur simultaneously, as in the case of lockstep cores. We aim at realizing such features, if eventually needed, resorting to techniques we already devised in the past and realized in both, NVIDIA and Intel GPUs [18] [19]. In any case, we note that, as previous work noted, the GPU in the NVIDIA Orin platform includes some unique components, such as the hardware scheduler, which needs to be used by redundant tasks. Hence, with the information available of the platform, we have to assume that some permanent faults could potentially lead to identical errors for redundant kernels, and hence, they could not be directly detected with the solutions described here. In general, such type of errors should be managed at system level by, for instance, deploying diverse object detection mechanisms (e.g., based on multiple independent cameras, or on other sensors such as radars and LIDARs).

4.2.3 Diagnostic and monitoring mechanisms

Next subsections explain the application of the hierarchical diagnostics and monitoring approach of Section 3.2 to SP2 on the SAFEXPLAIN platform.

4.2.3.1 L1DM – SAFEXPLAIN PLATFORM

L1DM diagnostics mechanisms from Section 3.1.1 can be selected and evaluated to reach the required diagnostic coverage for SP2.

In addition, if the system includes application with different integrity levels, independence of execution shall be guaranteed following the same approach as in SP1.

Similarly, to reduce common cause failures among redundant instances in the AI/ML constituent, their independence shall also be guaranteed. Therefore, L1DM will monitor the use of resources and other relevant events for each instance of the redundant architecture and share the information with L2DM for a platform level monitoring of all system components.

4.2.3.2 L2DM – SAFEXPLAIN PLATFORM

Following the incremental approach, L2DM platform level diagnostics shall be applied in the same way as for SP1. In this case, the NVIDIA Orin platform resources used for the AI/ML constituent become safety relevant too, which would require additional diagnostics with respect to SP1 and the most suitable platform configurations shall be adapted too. Figure 27 presents a configuration option that could be adopted for SP2, which could be combined with other solutions shown in SP1 (see Figure) as described in Table 6.

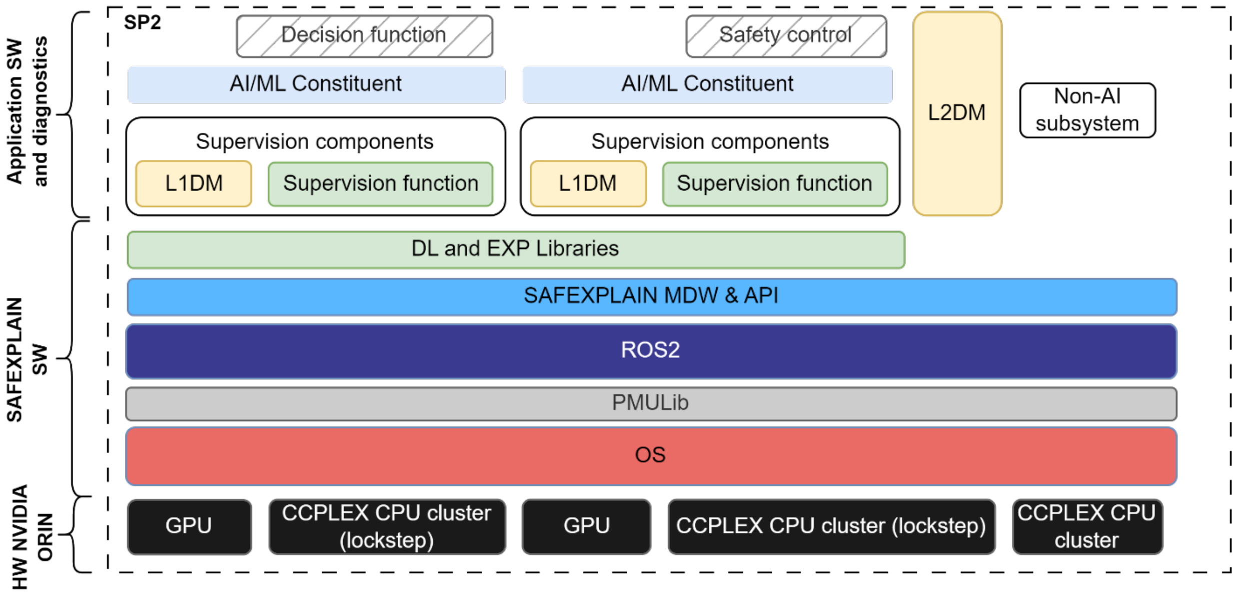


Figure 27: SP2 to NVIDIA Orin resource allocation and configuration options

Table 6: SP2 to NVIDIA Orin resource allocation and configuration options

SP2 Element	Safety / non-safety	SP2 - A NVIDIA Orin resources and configuration
AI/ML constituent	AI based safety SW	Two instances, each in one separate CCPLEX CPU Cluster (Cortex A78) in lockstep configuration GPU for AI inference (depending on the DRS CPU or other computing resources could also be used to improve diversity) Memory controller fabric and traffic from CPU cluster to GPU MMUs for spatial independence SAFEXPLAIN SW Stack
Supervision components	Traditional or AI based safety SW	Each AI/ML constituent has each own L1DM and optionally each own supervisor function (depends on user application). Depending on the implementation of the supervision component, it may need GPUs for improved performance (e.g., AI based supervision function). The supervision components can share same CCPLEX CPU Cluster (Cortex A78) in lockstep configuration as the AI/ML constituent. MMUs for spatial independence SAFEXPLAIN SW Stack
Decision function	Safety traditional SW	These SW components can run on any of the CCPLEX CPU Cluster (Cortex A78) in lockstep configuration used for the AI/ML constituent with the same configuration assuming they have the same integrity level.
Safety control	Safety traditional SW	
L2DM	Safety traditional SW	
Non-AI subsystem	Non-safety traditional SW	CCPLEX CPU Cluster (Cortex A78) or SPE (no need for lockstep configuration).

		MMUs for spatial independence L4 cache partitioning or disabled SAFEXPLAIN SW Stack or different OS on top of SPEs or hypervisor

The following mechanisms are provided by the platform:

- **Independent computing components**: In this case, all safety software can be run in CCPLEX CPUs in lockstep mode, using different clusters for the redundant replicas. Other accelerators or different computing resources could also be used to improve diversity.
- **Spatial independence**. The same support for SP1 addresses the needs of SP2.
- **Temporal independence**. The same support for non-AI tasks for SP1 addresses the needs of SP2. Regarding AI tasks, they are expected to use the GPU and/or the AI accelerators. Those accelerators have been devised to maximize throughput. However, either they do not support concurrency (e.g., AI accelerators) or support it without specific temporal independence support (e.g., GPU). Hence, AI tasks needing concurrency will have a high degree of temporal isolation if they use different accelerators. If they are deployed on the same one (i.e., the GPU) and made run simultaneously, timing interference can be arbitrarily high making its monitoring unfeasible. Therefore, those tasks need to be run sequentially if they need the very same accelerator.
- **Support for lockstep redundancy**. The same support for non-AI tasks for SP1 addresses the needs of SP2.

4.3 Safety Pattern 3 (SP3)

Safety pattern 3 follows the same approach as SP2 but it is assumed that the system has a higher degree of autonomy and therefore the safety implications of the AI-based subsystem are higher. Therefore, we consider a safety integrity level of SIL3 / SIL 4 according to IEC 61508 depending on the application.

4.3.1 System Architecture

In order to achieve higher integrity levels (with respect to SP2), SP3 involves either providing a high diagnostic coverage (≥99%) or increasing the HFT as shown in Table 7 according to IEC 61508-2.

Table 7: Maximum allowable safety integrity level according to IEC-61508-2 for Type B safety related elements (SP3)

Pattern	HFT	Safe Failure Fraction (SFF)			
		<60%	60% -90%	90% - 99%	≥ 99%
1oo1(D)	0	Not allowed	SIL 1	SIL 2	SIL 3
2oo2(D)	0	Not allowed	SIL 1	SIL 2	SIL 3
1oo2(D)	1	SIL 1	SIL 2	SIL 3	SIL 4
2oo3(D)	1	SIL 1	SIL 2	SIL 3	SIL 4
1oo3(D)	2	SIL 2	SIL 3	SIL 4	SIL 4

However, achieving a ≥99% DC in complex platforms such as the NVIDIA Orin considered in SAFEXPLAIN is very challenging and would highly depend on the guarantees provided by the platform manufacturer. The complexity of the SW stack and lack of transparency on the platform further exacerbates this. Therefore, we assume that for SP3 an HFT > 0 is required. To this end, off-chip redundancy shall be applied due to the restrictions of IEC 61508-2 Annex E as already mentioned in SP2. Figure 28 shows an example of SP3 implementation based on a TMR architecture, depending on the implementation HFT = 1 (2oo3) or HFT = 2 (1oo3) can be achieved as explained in D1.1 [3].

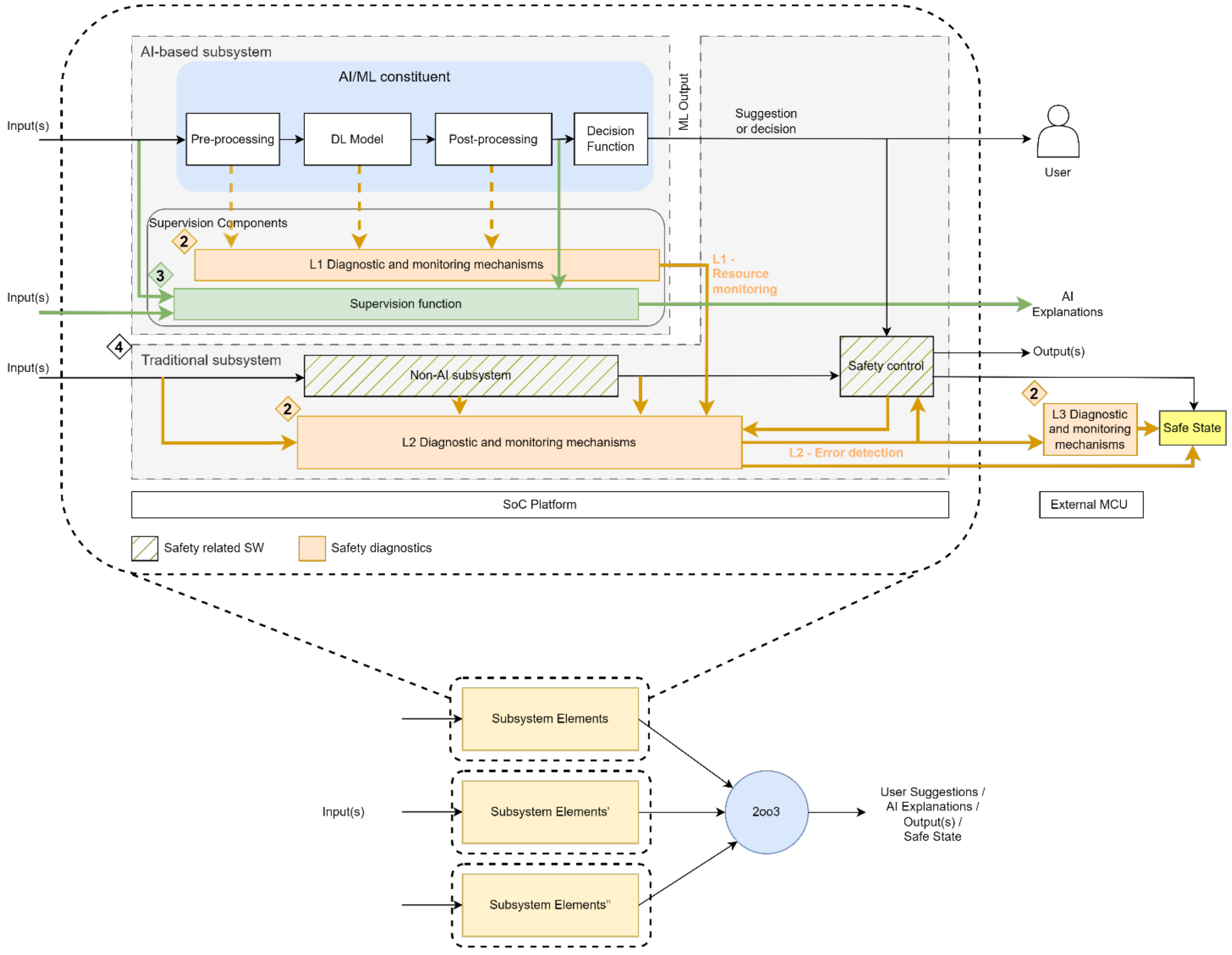


Figure 28: Conceptual System Architecture - Safety Pattern 3

If we look at each individual replica of the TRM implementation, each node could be implemented following SP2 safety pattern:

- Diverse Redundancy: In SP3 off-chip redundancy is applied. Therefore, the decision function should combine the outputs from different computing nodes. Each computing node can have its own decision function in such a way that if any node detects discrepancies, the safe state could be reached (1oo3 – HFT =2). Alternatively, the voting can be done in an external element (e.g., L3DM) and apply majority voting, meaning that at least 2 of the 3 redundant nodes shall provide the same output (HFT = 1). In order to apply this off-chip redundancy, diverse redundancy schemes with medium to high DC can be applied in this safety pattern, based on the specific requirements of the application (e.g., DSR_4, DSR_5, DSR_6). Off-chip redundancy can be combined with on chip redundancy approaches to improve the safe failure fraction and diagnostic coverage of each redundant node, following the same solutions as in SP2.

- Diagnostic and monitoring mechanisms: In each redundant instance the same solutions as for SP2 apply in SP3.
    - L3DM would now monitor the status of all the redundant replicas.
- Supervision function: Each redundant replica of SP3 can implement a supervisor function according to the description in the reference architecture in Section 3.3.
- Traditional subsystem: depending on the application, a fallback subsystem implemented with traditional SW could be in place.

5 Acronyms and Abbreviations

AI	Artificial Intelligence
CAST	Certification Authorities Software Team
CBOD	Camera Based Object Detection
CPU	Central Processing Unit
DC	Diagnostic Coverage
DL	Deep Learning
DMR	Dual Modular Redundancy
DNN	Deep Neural Network
DRS	Diverse Redundancy Schemes
EASA	European Aviation Safety Agency
FAA	Federal Aviation Administration
FSM	Functional Safety Management
FUSA	Functional Safety
GPU	Graphics Processing Unit
HFT	Hardware Fault Tolerance
HPEC	High-Performance Embedded Computing
IoU	Intersection over Union
L1DM	L1 Diagnostics and Monitoring
L2DM	L2 Diagnostics and Monitoring
L3DM	L3 Diagnostics and Monitoring
ML	Machine Learning
MPSoC	Multi-Processor System-on-Chip
NMS	non maximum suppression
ODD	Operational Design Domain
OS	Operating System
SIL	Safety Integrity Level
SoC	System on Chip
SM	Stream Multiprocessor
SP	Safety pattern
TMR	Triple Modular Redundancy
WCET	Worst Case Execution Time

6 References

[1] EGAS Workgroup, “Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units. Version 5.5,” 2013.

[2] International Standardization Organization, “ISO 26262. Road Vehicles - Functional Safety. Second edition,” 2018.

[3] European Union Aviation Safety Agency (EASA), “EASA Concept Paper: guidance for Level 1 & 2 machine learning applications,,” 2023.

[4] R. Padilla, S. Netto and E. da-Silva, “A Survey on Performance Metrics for Object-Detection Algorithms,” in *International Conference on Systems, Signals and Image Processing (IWSSIP)*, Niteroi, Brazil, 2020.

[5] M. Caro, H. Tabani and J. Abella, “At-scale assessment of weight clustering for energy-efficient object detection accelerators,” in *In Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22)*, 2022.

[6] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” in *ArXiv*, 2020.

[7] J. Heras, “Clodsa image augmentation library for object detection,” Accessed Oct-2023.

[8] M. Caro, J. Fornt and J. Abella, “Efficient Diverse Redundant DNNs for Autonomous Driving,” in *IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, Torino, Italy, 2023.

[9] C. Li, “YOLOv4 TensorFlow Keras implementation,” Accessed Oct-2023.

[10] Lin, TY. et al., “Microsoft COCO: Common Objects in Context,” in *European Conference on Computer Vision (ECCV)*, 2014.

[11] Everingham, M., Eslami, S.M.A., Van Gool, L. et al., “The PASCAL Visual Object Classes Challenge: A Retrospective,” *International Journal of Computer Vision,* no. 111, 2015.

[12] J. Hauser, “Berkeley SoftFloat,” 2018.

7 Annexes

7.1 Preliminary evaluation of L0 Diverse Redundancy solutions

7.1.1 Input image transformation

In this section, we first introduce the evaluation setup (YOLO implementation, dataset and metrics), and then analyze the results obtained with our scheme considering the different image transformations and results merging schemes.

7.1.1.1 Setup

We use a 32-bit floating point Tensorflow Keras implementation of YOLOv4 (Li, Accessed Oct-2023) with the pre-trained YOLOv4 parameters with the training subset of the Common Objects in Context (COCO) dataset (Lin, TY. et al., 2014). We evaluate our proposal with COCO, using the testing subset, with image sizes of 320x320 pixels, and keeping only those images that contain objects such as vehicles and pedestrians, which delves 880 images for evaluation. More extensive evaluations with other datasets, including larger image sizes will be provided in the future.

As evaluation metrics, we use the Accuracy (ACC) and Mean Average Precision (mAP). To obtain those, first we have to compute the Intersection over Union (IoU), i.e. the fraction of the intersection of the bounding boxes w.r.t. the union of those bounding boxes. In particular, we do so for the final objects detected by our proposal and the groundtruths. Since the detection process is stochastic and subject to some variation, whether an object is regarded as detected or not is done with a threshold, which in our case is 0.5, as set in other works . This leads to true positives (TP) if the IoU is above the threshold, false positives (FP) if the IoU is below the threshold for a predicted object, and false negatives (FN) if the IoU of a groundtruth is below the threshold.

The accuracy (ACC) is obtained as follows:

$$ACC = \frac{TP}{TP + FP + FN}$$

The mAP averages the ACC values across the existing object classes [4], and is often the reference metric for object detection evaluation.

7.1.1.2 Results

We have obtained the TP, FP, and FN counts, as well as the ACC and mAP values for all TMR configurations and decision function algorithms, namely voting, averaging and NMS. Since the number of combinations is too large to allow reporting data for all those configurations (around 1,000 per decision function), we show the mAP for all configurations in Figure 29, and the detailed results for the top-5 (in terms of mAP) for each algorithm in Table 8.

Results in the figure are sorted independently for each decision function, meaning that the nth best configuration for one function may differ for the other functions despite being aligned w.r.t. the X-axis. Hence, the X-axis is not labelled. However, showing the data this way allows us to get several conclusions: (1) NMS provides consistently the best results in terms of mAP across all decision functions; (2) Averaging is consistently worse than the baseline with the original image only in terms of mAP; (3) Voting is slightly better than the baseline in some cases; and (4) all decision functions provide a smooth degradation of the mAP across image transformation combinations, hence meaning that, while some transformations may delve better results than others, there is not a strong dependence on very few combinations, so the approach provides robust results.

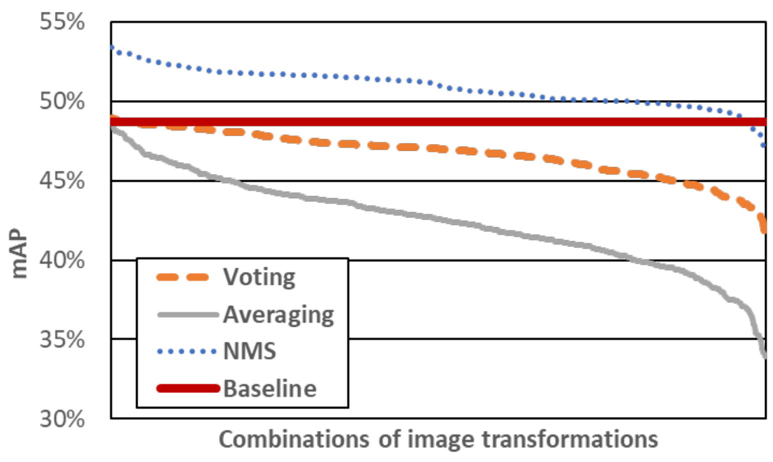


Figure 29. Sorted mAP for all TMR configurations for all merging algorithms.

Configuration	**TP**	**FP**	**FN**	**ACC**	**mAP**
BL	3087	373	3104	47.0%	48.69%
VOTING					
Configuration	**TP**	**FP**	**FN**	**ACC**	**mAP**
BL, RV, BS	3098	357	3093	47.31%	48.97%
RV, LS, BS	3091	290	3100	47.69%	48.95%
BL, LS, BS	3087	296	3104	47.59%	48.87%
BL, RV, TS	3093	362	3098	47.20%	48.85%
GC, LS, BS	3083	283	3108	47.62%	48.85%
AVERAGING					
Configuration	**TP**	**FP**	**FN**	**ACC**	**mAP**
BL, BL, BL	3087	373	3104	47.03%	48.69%
BL, GC, RV	3079	370	3112	46.93%	48.59%
BL, GN, RV	3057	369	3134	46.60%	48.24%
GC, GN, RV	3052	371	3139	46.51%	48.15%
BL, GC, GN	3050	363	3141	46.54%	48.15%
NMS					
Configuration	**TP**	**FP**	**FN**	**ACC**	**mAP**
EQ, LS, TS	3409	606	2782	50.15%	53.39%
EQ, LS, BS	3398	613	2793	49.94%	53.23%
RS, LS, TS	3405	617	2786	50.01%	53.20%
EQ, RS, TS	3403	626	2788	49.92%	53.19%
GC, RS, TS	3403	616	2788	49.99%	53.17%

Table 8. TMR results for the top-5 configurations of each decision function.

Results in the table offer a different angle to our analysis. In the case of voting, for instance, we note that the bottom shift (BS), left shift (LS), and raise value (RV) transformations, as well as the baseline (BS), are the most popular ones since they appear in 4, 3, 3, and 3 of the top-5 TMR combinations respectively, and only Gamma correction (GC) appears once. In the case of averaging, the best combination, despite not providing diversity, consists of using 3 times the baseline (BL). RV, GC, and Gaussian Noise (GN) are also quite frequent, and appear 3 times each one. Finally, in the case of NMS, we observe that slightly shifting the original image in one direction is a frequent choice since 11 out of the 15 choices correspond to top, bottom, left or right shift. Equalization (EQ) also appears 3 times and GC once.

Regarding TP, FP and FN results, we note that voting may increase a bit TPs and decrease a bit FPs. Hence, despite not providing the best ACC and mAP values, it provides an interesting tradeoff since it outperforms the baseline in all fronts. Averaging, instead, tends to decrease TPs while failing to decrease FPs, and hence, it is systematically worse than the baseline case. Finally, NMS tends to increase TPs and FPs, which is expected since any object being detected by at least one of the three redundant inferences is regarded as a detection, and hence, a TP or a FP. Still, the combined effect clearly increases ACC and mAP values w.r.t. the baseline. Overall, if we care only about mAP or ACC, NMS is clearly the best choice. If, instead, FPs are particularly problematic, voting is the best solution. Still, we plan to expand the evaluation to additional datasets in the near future.

7.1.2 Diverse data types

In this section, we first introduce the evaluation setup, and then analyze the results obtained with our scheme considering different data types.

7.1.2.1 Setup

We use YOLOv4, as in the previous section, which builds on native fp32 arithmetic. However, for some of our analyses we want to explore other non-native data types, such as, for instance, fp16 and fp13. To do so, we have integrated the SoftFloat library [12] in YOLOv4. With this library, all floating-point operations are emulated by software, hence enabling the evaluation of fp16 and fp13. Also, this framework allows us to perform fault injection campaigns by poisoning the results of floating-point operations with specific probabilities.

For our fault injection campaigns, we inject transient faults by randomly selecting floating-point operations of the DNN and flipping one bit of the exponent or the sign of the result (selected randomly). The probability of performing a bit flip is set to 10^{-10}, which leads to a 3-5% misdetection rate (either FP or FN) in our datasets. We note that faults can also be injected in the mantissa. However, the impact is often completely negligible, hence making the fault injection campaign highly ineffective since results mostly match those of the fault-free case. Instead, faults injected in the exponent or sign have a much higher impact and allow evaluating the effectiveness of TRUST. Finally, note that, when injecting faults, they are injected at most once every 5 frames. This ensures that any YOLO prediction is affected by a single fault injection since they combine results across 3 consecutive frames only.

We resort to the same metrics as before to determine whether two detections in the different instances correspond to the same object (IoU metric) and to assess prediction accuracy (mAP).

Regarding datasets, since we want to evaluate DMR, to bring a different example to that in the previous section, we use the COCO dataset for few experiments since it provides independent images for which we cannot use temporal redundancy. We lack labelled videos for autonomous operation, which YOLOv4 can process with high accuracy when using the pre-trained YOLOv4 model, trained with the COCO dataset. Hence, we have used unlabelled videos (see Table 9) using the fault-free fp16 implementation of YOLOv4 as the ground truth. This may lead to false mispredictions if a FP or FN in the baseline case is properly classified with our DMR scheme.

SET OF VIDEOS	train start time	eval start time	train start frame	eval start frame
https://youtu.be/hWCN1yV9TY	2:50	5:20	5094	9604
https://youtu.be/jJ08h2cgWjl	2:20	4:04	4214	7325
https://youtu.be/E7t3QyEfyLA	2:00	3:36	3639	6486
https://youtu.be/zfblxgasy-0	18:30	20:05	33237	36115
https://youtu.be/xj7abSp07w0	13:25	15:05	24142	27140
https://youtu.be/X9U5DafT0d4	1:10	3:05	2097	5554

Table 9. Set 1 of videos used.

For the videos, since we run YOLOv4 on a CPU emulating floating-point operations with a software library, which is time consuming, we study 100 consecutive frames per video (3.3 seconds given the 30 frames per second rate of the videos). The starting point has been selected arbitrarily, and then forwarded until relevant driving scenarios are found (e.g., skipping phases where no cars or signs are nearby).

Note that we use the set of videos for training purposes (SET1train). The second start time for Set 1 (SET1eval) is used for evaluation purposes and has been selected sufficiently far away from the first one to grant independence.

7.1.2.2 Results

As explained before, the lower the precision of a given data type, the lower the detection confidence is expected. We illustrate this fact with the example in Figure 30 where we show, for the COCO dataset (selecting only images with vehicles and persons) how many objects are detected within each confidence range using fp32, fp16 and fp13. As shown, differences between fp32 and fp16 are tiny since fp16 allows preserving values very close to 1. However, this is not the case for fp13, and the number of detections with confidence values above 0.5 drops by 11% w.r.t. fp16, and the ones above 0.95 drop drastically by 55%. Hence, the usual case is that $Conf^{fp16} > Conf^{fp13}$.

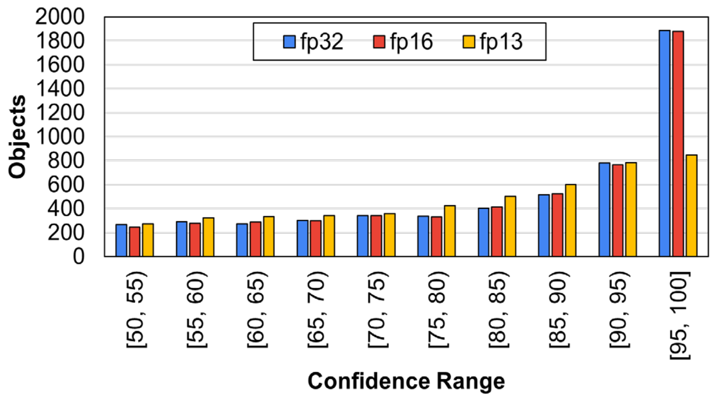


Figure 30. Objects detected across confidence ranges for varying precision arithmetic.

We have calculated the results of injecting a fault every 5 frames for different setups:

- Baseline: we use a single instance of YOLOv4, or two of them operating in DMR.
- DMR with diverse data types (DDDT for short): we use the prediction of the fp16 instance except when it varies by more than 0.2 w.r.t. the average of the last 2 frames, and the confidence of the fp13 instance for the given object is higher than for fp16.

If more than one instance is used, fault injection is performed in the very same operation of both instances to mimic the impact of a fault potentially leading to a common cause failure. Our results show that our DDDT scheme is able to correct around 70% of the errors, whereas the baseline cannot correct any.

A more detailed and complete evaluation will be provided in future releases of this deliverable.