electronics

MDPI

Article

Design and Assurance of Safety-Critical Systems with Artificial Intelligence in FPGAs: The Safety ArtISt Method and a Case Study of an FPGA-Based Autonomous Vehicle Braking Control System

Antonio V. Silva Neto , Henrique L. Silva, João B. Camargo, Jr., Jorge R. Almeida, Jr. and Paulo S. Cugnasca *

Safety Analysis Group (GAS), Department of Computer Engineering and Digital Systems (PCS), Escola Politécnica (Poli), Universidade de São Paulo (USP), São Paulo 05508-010, SP, Brazil; antonio.vieira88@usp.br (A.V.S.N.); henriquelefundes@usp.br (H.L.S.); joaocamargo@usp.br (J.B.C.J.); jorgerady@usp.br (J.R.A.J.)

* Correspondence: cugnasca@usp.br

Abstract: With the advancements in utilizing Artificial Intelligence (AI) in embedded safety-critical systems based on Field-Programmable Gate Arrays (FPGAs), assuring that these systems meet their safety requirements is of paramount importance for their revenue service. Based on this context, this paper has two main objectives. The first of them is to present the Safety ArtISt method, developed by the authors to guide the lifecycle of AI-based safety-critical systems, and emphasize its FPGA-oriented tasks and recommended practice towards safety assurance. The second one is to illustrate the application of Safety ArtISt with an FPGA-based braking control system for autonomous vehicles relying on explainable AI generated with High-Level Synthesis. The results indicate that Safety ArtISt played four main roles in the safety lifecycle of AI-based systems for FPGAs. Firstly, it provided guidance in identifying the safety-critical role of activities such as sensitivity analyses for numeric representation and FPGA dimensioning to achieve safety. Furthermore, it allowed building qualitative and quantitative safety arguments from analyses and physical experimentation with actual FPGAs. It also allowed the early detection of safety issues—thus reducing project costs—and, ultimately, it uncovered relevant challenges not discussed in detail when designing safety-critical, explainable AI for FPGAs.

Keywords: artificial intelligence; decision tree; explainable AI; field-programmable gate array (FPGA); high-level synthesis; random forest; safety; safety-critical system

check for updates

Citation: Silva Neto, A.V.; Silva, H.L.; Camargo, J.B., Jr.; Almeida, J.R., Jr.; Cugnasca, P.S. Design and Assurance of Safety-Critical Systems with Artificial Intelligence in FPGAs: The Safety ArtISt Method and a Case Study of an FPGA-Based Autonomous Vehicle Braking Control System. *Electronics* **2023**, *12*, 4903. https://doi.org/10.3390/electronics12244903

Academic Editors: Young-Ho Seo, José V Frances-Villora, Fei Yu and Jun Mou

Received: 8 November 2023
Revised: 24 November 2023
Accepted: 1 December 2023
Published: 6 December 2023

1. Introduction

The recent effervescence surrounding Artificial Intelligence (AI) has led to a significant increase, especially from the mid-2010s onwards, in research on its usage within the core of safety-critical systems [1]. One of the most relevant concerns on applying AI in safety-critical applications is the safety assurance of these systems, which comprises "(. . .) *the set of activities, means, and methods that shall be considered, throughout the lifecycle of a system, to produce results towards building arguments that confidently support the safety requirements/targets of such a system have been met*" [1–7]. This is taken into account not only at the design time but also throughout operation, notably when online learning is at play [1,8–12].

In the Internet of Things (IoT) era, safety-critical systems have also been growingly more distributed and based on embedded modules [13–15], such as those used in the control functions of Unmanned Ground Vehicles (UGVs) [16–24], Unmanned Aerial Vehicles (UAVs) [25–27], and in a new generation of distributed Communication-Based Train Control (CBTC) systems for metro and railway signaling [28,29]. Furthermore, the increased flexibility and processing capability provided by recent Field-Programmable Gate Arrays

(FPGAs) and other Programmable Logic Devices (PLDs) allows these elements to play an even more important role in cutting-edge safety-critical systems than today [30,31] while keeping design costs reasonable for achieving the needed efficiency and performance levels [32,33].

Consequently, the safety assurance of AI-based systems is a relevant theme for research, especially because means and methods within pre-existing standards and research papers feature gaps in relevant themes for their practical usage, as evidenced by a systematic literature review covering 5090 papers published up to 2022 [1]. These include, but are not limited to, the lack of detailed guidelines covering the dos and don'ts of safety-critical AI for safety practitioners [34,35], constraining safety practices to a subset of AI variants [36–41], and crafting safety practices only for specific application domains [8,9,37,40].

The safety assurance of AI-based systems becomes even more challenging when FPGAs are within the control loop of the safety-critical functions. This is justified by two aspects: (i.) the scarcity of research with this specific objective [1] and (ii.) specific safety constraints that are applicable to safety-critical systems with FPGAs. For instance, the majority of research involving safety, AI, and FPGAs altogether is solely based on presenting FPGA-based solutions for implementing safety-critical AI, notably in vehicle control automation [42–44] and general-purpose building blocks for FPGA design in safety-critical systems [45–47]. Therefore, these efforts lack safety assurance aspects, such as analyses that support whether safety requirements have indeed been met.

In addition to that, safety-critical FPGAs are inherently subject to design restrictions regardless of AI, as explored in well-established safety standards, such as IEC61508:2010 and CENELEC EN50129:2018 [4,6]. With the embedding of safety-critical AI onto FPGAs, a plethora of other design constraints emerges towards ensuring that the final AI programming is performed in such a way as to meet functional and safety requirements.

A remarkable FPGA-specific safety concern is the programming of the AI models. Since most AI reference models are sufficiently complex to be described from scratch for FPGAs but are readily available, with proven-in-use implementations, on several software libraries (e.g., Python's *scikit-learn* version 1.3.1), High-Level Synthesis (HLS) is a handy means to translate AI models coded in software to an equivalent model in a Hardware Description Language (HDL) [48]. The usage of HLS introduces the need to ensure that the resulting HDL-coded AI meets the needed functional and safety requirements. It is worth noting that solely ensuring that the original software-based solution meets these requirements is insufficient to take into account a multitude of FPGA-specific aspects, such as fault tolerance, circuitry synthesis, and signal routing, since these depend not only on the HLS per se but also on how the FPGA programming tools are configured.

In order to deal with the safety assurance of AI-based systems as a whole, a method called Safety ArtISt (Artificial Intelligence Structure) has been developed by the authors. Safety ArtISt has been conceived with the objective of guiding the lifecycle of safety-critical systems with AI with explicit recommended and not recommended practices that fill several safety assurance gaps identified in the papers reviewed in [1]. Safety ArtISt covers a wide range of AI models, including supervised learning, reinforcement learning, semisupervised learning, unsupervised learning, offline learning, online learning, and explainable AI. It has also been crafted bearing in mind concrete practice targeting the deployment of safety-critical AI on microprocessors and on FPGAs/PLDs.

The objectives of this paper are to provide an overview of the Safety ArtISt method, focusing on its FPGA-specific activities and practice, and to illustrate its application by means of a case study. This case study comprises the development and analysis of a Braking Control System (BCS) for autonomous vehicles whose safety core is based on two random forests fully deployed in FPGAs through HLS.

The main expected novelty and contribution of this research paper are as follows:

1. Providing, with Safety ArtISt, cost-effective guidance on how to structure and specify the lifecycle of a safety-critical system with AI, with a special focus on FPGA-implemented supervised learning;

2. Identifying design, verification, and validation activities that are critical for building and analyzing safety-critical AI on FPGAs, as well as illustrating how to carry them out and showing their importance to safety;
3. Presenting an approach for building quantitative safety models for safety-critical systems with supervised learning and exemplifying its application;
4. Providing general-purpose conclusions on the mission profile and safeguards that allow AI to be utilized in safety-critical systems.

The remainder of this paper is structured as follows. Section 2 covers two themes: it begins with a description of the Safety ArtISt method and continues with an overview of the case study's scope and justification. In Section 3, the Safety ArtISt-oriented lifecycle for the case study's BCS is explored in detail, including the workflow followed throughout its project and the results obtained in each Safety ArtISt activity. Section 4 is devoted to discussing the main results of this case study, as well as the role of Safety ArtISt in building a safety-critical system with FPGA-implemented AI and future work. Finally, Section 5 closes the paper with a summary of the research conclusions.

2. Materials and Methods

This section is divided into three subsections. The two first ones are devoted to detailing the Safety ArtISt method: they start in Section 2.1 with an overview of its lifecycle and progress onto Section 2.2 with FPGA-specific activities and techniques.

Finally, Section 2.3 aims to define the scope of the BCS case study. It includes not only the functionalities and constraints imposed on the AI- and FPGA-based autonomous vehicles' BCS but also how the Safety ArtISt method was applied in the case study.

2.1. Description of the Safety ArtISt Method

The Safety ArtISt method was created as a means to guide the entire lifecycle of safety-critical systems with at least one safety-critical function relying on AI. Its principle is based on a safety assurance process, which allows building safety arguments from the early design up to operation and maintenance in order to support whether the system is safe for revenue service.

Safety ArtISt augments classic safety assurance processes, such as the ones of safety standards CENELEC EN50129:2018 and IEC61508:2010 [4,6], with practitioner-oriented activities. tasks, and recommended practices stemming from AI's underlying technical aspects that aim to assess the safety-related properties of AI. These comprise fulfilling the objectives listed as follows, which stem from three sources: (i.) gaps from pre-existing research on the safety assurance of AI [1], (ii.) merging sparse efforts and practice towards the safety assurance of AI into a systematic, process-oriented approach [23,30,38,49–57], and (iii.) the expertise of the Safety ArtISt research team on safety and AI.

1. Specifying AI safety requirements and tracing safety arguments to them throughout the system lifecycle;
2. Providing justifiable means to define candidate AI variants for safety-critical functions, analyze their pros and cons, and effectively select the AI variants with the best potential to fulfill functional and safety requirements;
3. Providing systematic means to define AI failure modes and analyze their effects on safety at specification and design time;
4. Assessing the need for redundancy and diversity of AI modules for safety-critical functions;
5. Defining safety requirements and procedures for input datasets and evaluating how datasets impact the safety of AI designed with them;
6. Providing justifiable means to define, for each AI variant chosen for the system design, the domain of its hyperparameters/hyperfunctions and optimize them based on expert knowledge, optimization processes and/or cross-validation (CV);
7. Establishing guidelines for designing safety-critical AI with FPGAs and other PLDs;

8. Guiding the design of explainable AI and understanding its role in the safety assurance of AI-based safety-critical functions;
9. Defining means and methods to specify, choose, and analyze AI-related development tools and third-party libraries throughout the system lifecycle;
10. Adjusting and augmenting recommended practice for safety-critical systems without AI to deal with safety-critical AI.

Safety ArtISt is structured as a seven-step process following the workflow represented in Figure 1.

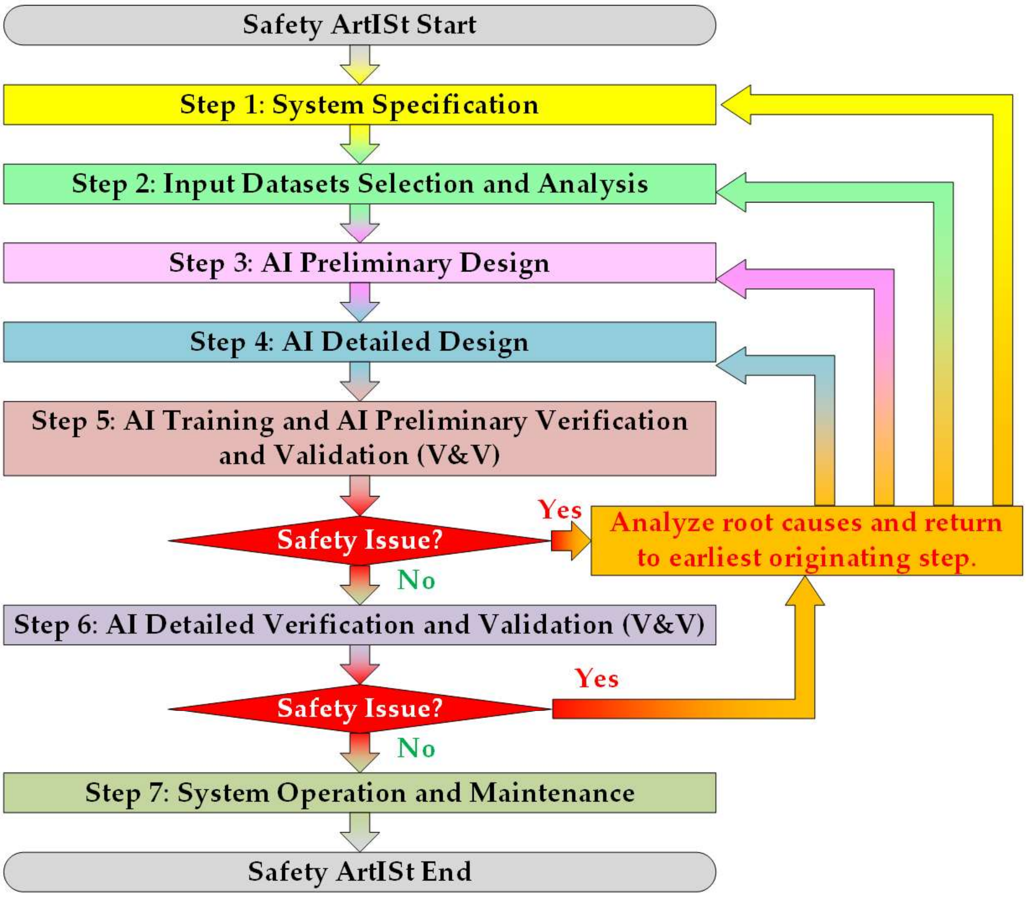

Safety ArtISt Start
Step 1: System Specification
Step 2: Input Datasets Selection and Analysis
Step 3: AI Preliminary Design
Step 4: AI Detailed Design
Step 5: AI Training and AI Preliminary Verification and Validation (V&V)
Safety Issue?
Yes
Analyze root causes and return to earliest originating step.
No
Step 6: AI Detailed Verification and Validation (V&V)
Safety Issue?
Yes
No
Step 7: System Operation and Maintenance
Safety ArtISt End

Figure 1. Workflow of the Safety ArtISt method.

Safety ArtISt starts with *Step 1—System Specification*—whose major aim is to build a systems-level specification comprising two major features: (i.) requirements specification and (ii.) system architecture. The dynamics of Step 1 include three major activities. The first one is eliciting systems-level requirements based on the project stakeholders' demands and constraints. Hazard and Risk (H&R) analyses at the systems level lead to the specification of Safety Requirements (SRs), which comprise qualitative features and the quantitative targets of each safety-critical function. These quantitative targets are expressed by means of the safety functions' Safety Integrity Level (SIL), which ultimately translates into a Tolerable Hazard Rate (THR) [4,6].

Afterward, a system architecture model is developed in such a way as to increase the system detailing with its subsystems and their interfaces. Based on these features, AI variants eligible for the needed functions are identified, and the SRs are ultimately refined with AI-specific characteristics. This can be achieved, for instance, by expanding the initial H&R analyses with a Hazard and Operability Study (HAZOP)-based approach for which the unary and binary HAZOP keyword operators are exhaustively associated with the potential failure modes of each candidate AI variant in order to systematically generate

H&R analyses scenarios. The AI Refined Safety Requirements (RSRs) are then elicited from the needed action to circumvent safety issues for each of the analyzed scenarios. It is also worth noting that, at the end of all forthcoming Safety ArtISt steps, safety arguments are iteratively built and traced to the SRs/RSRs on a traceability matrix.

Step 2—Input Datasets Selection and Analysis—is performed after finishing Step 1. Its objective is to ensure that all datasets utilized throughout the design of safety-critical AI have been properly specified, generated, and stored in order to mitigate potential systematic faults that can be introduced within the AI designed with the datasets. Step 2 includes a total of ten activities, labeled with the identifiers (IDs) from 2-A to 2-J and briefly described in Table 1.

Table 1. Activities of Safety ArtISt Step 2—Input Dataset Selection and Analysis.

ID	Description
2-A	Analyzing data origin and domain (e.g., end application, simulated)
2-B	Selecting and analyzing dataset processing tools
2-C	Analyzing missing and incomplete data
2-D	Analyzing safety-critical scenario representativeness (e.g., corner cases)
2-E	Performing general statistics (e.g., dataset variables' distributions)
2-F	Performing additional data pre-processing (e.g., normalization)
2-G	Updating statistical tests for the preprocessed dataset
2-H	Defining dataset partitioning strategy for AI design and validation
2-I	Analyzing underlying uncertainties on dataset variables
2-J	Updating safety arguments and SR/RSR traceability matrix with Step 2 results

The activities that are part of Step 2 start at 2-A by analyzing how a dataset has been built and to what extent datasets generated outside the target application scope (e.g., data from similar applications or generated in a simulated environment) are faithful enough for such an end application. It progresses to 2-B, which is related to selecting dataset processing tools and ensuring that all these tools are robust enough to avoid introducing potential safety issues—be they within the interpretation of the dataset per se or be they related to inadvertently changing the dataset in an undetectable and unsafe way.

Finally, all remaining activities (2-C to 2-I) span a series of analyses that allow not only providing a more in-depth knowledge of the datasets and their overall data quality but also adapting their contents, in a controlled way, to leverage potentially better performance and safety from safety-critical AI elements. Special attention is given to the representativeness of corner cases, given their relevance in establishing the borderline behavior of safety-critical AI, as well as to the underlying uncertainties of data records, which directly translate into how trustworthy AI built within these data is likely to be.

Once Step 2 is finished, Safety ArtISt progresses onto *Step 3—AI Preliminary Design*. This step has two major goals: (i.) deepening the detailing of the system architecture developed in Step 1 with the hardware and software components of each subsystem and (ii.) specifying tools, settings, and fault tolerance design guidelines for the remainder of the lifecycle. In order to meet these objectives, Step 3 is structured with the six activities presented in Table 2. Activities 3-Pre, 3-A, and 3-E are mapped onto objective (i.), whereas 3-B to 3-E are mapped onto objective (ii.).

Three relevant remarks on the Step 3 activities are worth highlighting. Firstly, activity 3-Pre is carried out whenever the analyses performed in Step 1 yield more than one viable AI variant for a safety-critical function. In activity 3-Pre, the candidate AI models are instantiated on a sandbox environment, following the applicable dataset-related design strategies of Step 2, and their performance is assessed. Based on the results of this analysis, the AI models with the most promising results are effectively incorporated into the system architecture on activity 3-A, whereas the other ones are discarded.

Table 2. Activities of Safety ArtISt Step 3—AI Preliminary Design.

ID	Description
3-Pre	Designing preliminary AI to choose the best-performing candidates
3-A	Refining system architecture with hardware and software components
3-B	Defining design tools and their safety-critical settings
3-C	Defining the strategy for the detailed design of safety-critical AI (e.g., dedicated AI modeling vs. reuse of third-party libraries)
3-D	Defining guidelines for fault tolerance of safety-critical components
3-E	Updating safety arguments and SR/RSR traceability matrix with Step 3 results

The second noteworthy highlight of Step 3 is related to activity 3-A. In order to leverage modularity and decouple application-specific AI instantiation from their underlying general-purpose models, it is highly recommended that all safety-critical AI be split into two different components. The first component, hereinafter referred to as "AI base model", encapsulates data structures and algorithms that implement an AI model (e.g., an artificial neural network (ANN) or a support vector machine (SVM)) and provide hyperparameters and hyperfunctions as interfaces. The second component, hereinafter called "AI instance", refers to each application-specific usage of the AI base model, along with its fine-tuned hyperparameters and hyperfunctions.

Finally, the last remark on Step 3 embraces the activities 3-C and 3-D. Since they depend on whether safety-critical AI is allocated to software or FPGAs and PLDs, the recommended practice that supports both activities is explicitly partitioned into two subsets: one for software- and microprocessor-based designs and another one for safety-critical AI targeted at FPGAs and PLDs.

Step 3 is followed by *Step 4—AI Detailed Design*—whose goal is to implement the AI base models and their corresponding instances. For this purpose, Step 4 comprises the activities listed and described in Table 3. Among them, activities 4-A, 4-B, 4-G, and 4-H are applicable regardless of the AI base models, whereas the remaining ones are performed only for specific AI types, as detailed in the descriptions in Table 3.

Table 3. Activities of Safety ArtISt Step 4—AI Detailed Design.

ID	Description
4-A	Designing and implementing AI base models
4-B	Defining and justifying hyperparameters/hyperfunctions and their domains
4-C	Additional modeling for reinforcement learning instances
4-D	Additional modeling for semisupervised and unsupervised learning instances
4-E	Additional modeling for explainable AI instances
4-F	Additional modeling of online learning instances
4-G	Designing and implementing the generation of AI processing logs
4-H	Updating safety arguments and SR/RSR traceability matrix with Step 4 results

Step 4 starts with activity 4-A, whose aim is the design and implementation of the safety-critical AI base models that were initially identified in Step 3. Once this activity is over, Step 4 progresses to detailing the design of each AI instance. Since the majority of AI base models require tuning by means of hyperparameters and hyperfunctions, activity 4-B allows the designers to formally choose and justify the hyperparameters and hyperfunctions to be exercised for each AI instance, along with their respective domains and a strategy for sweeping within them (e.g., steps and scales).

Activities 4-C to 4-F delve further into the design of AI instances for specific AI variants that require more than the definitions of activity 4-B. These include reinforcement learning (4-C), semisupervised and unsupervised learning (4-D), explainable AI (4-E), and online learning (4-F). Finally, activity 4-G targets the design of logs that allow monitoring AI during its operation and incorporating them into revised safety models.

Step 5—AI Training and AI Preliminary Verification and Validation (*V&V*)—closes the design of the safety-critical AI with the training of AI instances that require this procedure. It also launches the safety-critical V&V with activities that are qualified as "preliminary" because, despite allowing the early detection of safety issues by means of analyses and techniques that are less demanding of V&V resources, they are still incomplete for ensuring that a system is indeed safe. As a result, Step 5 aids in unveiling safety issues in advance and triggering their corrections despite not being sufficient to ensure safety per se, since its activities might still mask underlying safety issues whose detection depends on the exhaustive, resource-demanding activities of *Step 6—AI Detailed V&V*.

The activities that are part of Step 5 are identified and described in Table 4.

Table 4. Activities of Safety ArtISt Step 5—AI Training and AI Preliminary V&V.

ID	Description
5-A	Analyzing functional adequacy of AI base models and sampling their components for analyzing compliance to fault tolerance guidelines
5-B	Training AI instances (if applicable)
5-C	Assessing performance metrics and corner case behavior of AI instances
5-D	Updating safety arguments and SR/RSR traceability matrix with Step 5 results
5-E	Assessing potential safety issues, identifying root causes, and revising Steps 1–4

Step 5 starts with evaluating whether the AI base models designed and implemented in Step 4 are not only functionally accurate but also to what extent the fault-tolerant practices established in Step 3 have been followed in their design. The approach for the fault tolerance analysis in Step 5 is sampling-based instead of exhaustive, because this can allow earlier and quicker detection of inconsistent fault tolerance within the core components of safety-critical AI, thus leading to the full revision of the safety-critical AI with regard to fault-tolerant design prior to progressing with V&V activities.

Should all AI base models be deemed correct and credibly robust to tolerating faults, the AI instances that depend on them are then generated by means of training. This is carried out following the recommended practice for dataset partitioning and hyperparameter/hyperfunction sweeping defined in Steps 2 and 4, respectively. Once the training is finished, and optimized AI instances are obtained, their performance is then assessed by means of the applicable performance metrics in at least two conditions: (i.) regular operation and (ii.) worst-case scenarios defined by means of the corner cases identified in Step 2. The results of this activity might indicate potential safety issues in two main conditions: (i.) overly low performance, in which case the AI instances are not likely to meet the systems-level quantitative safety requirements, and (ii.) overly high performance, which might suggest overfitting and ultimately compromise safety should the operation context require behavioral generalization outside the variables' training domains.

If any issue is identified in a Step 5 activity, the remaining activities can be skipped to avoid unnecessary analyses of safety-critical items whose design is likely to be changed. The root causes of these safety issues are then analyzed and traced back to the Safety ArtISt steps in which they originated. Once this troubleshooting is finished, a new iteration of the Safety ArtISt method is performed, starting with its earliest step traced to a safety issue.

On the other hand, if Step 5 is finished without the detection of potential safety issues, *Step 6—AI Detailed V&V*—is triggered. Given the exhaustiveness of the V&V activities included in Step 6, it is considered that the safety arguments built at the end of this step are complete for deciding whether to start a system's revenue service should no safety issues be identified. The full set of activities that are part of Step 6 are presented in Table 5.

Table 5. Activities of Safety ArtISt Step 6—AI Detailed V&V.

ID	Description
6-A.i	Analyzing environmental compliance and hardware random failure modes
6-A.ii	Analyzing functional and fault-tolerant design adequacy for safety-critical AI
6-A.iii	Analyzing if FPGA/PLD tools guidelines and settings have been followed
6-A.iv	Analyzing if software tools guidelines and settings have been followed
6-A.v	Performing formal or semiformal AI reachability analyses
6-A.vi	Performing quantitative safety analyses
6-B	Updating safety arguments and SR/RSR traceability matrix with Step 6 results
6-C	Assessing potential safety issues, identifying root causes, and revising Steps 1–4

The activity 6-A, which comprises "safety and V&V analyses" as a whole, is split into six tasks, referred to as 6-A.i to 6-A.vi, in order to provide better insight into each procedure that is carried out within its scope. The process starts with two hardware-related analyses in 6-A.i: (a.) ensuring that the design fulfills the needed conditions for operating in its target environment (e.g., humidity, altitude, temperature) and (b.) exhaustively analyzing the plausible hardware failure modes and their systems-level safety effects. Item (a.) is fully detached from AI-specific features, whereas (b.) solely depends on AI specificities in describing the overall impacts of the hardware failure modes, since hardware failure modes themselves are also independent from AI. Fault injection techniques, including those for FPGAs [58], can be used at this point to check for the effectiveness of fault-tolerant design and increase the robustness of safety arguments.

Task 6-A.ii targets proceeding with the analyses of activity 5-A by following the same approach and expanding its scope to all safety-critical AI components. This allows ensuring that every safety-critical AI element (base models and instances) is not only functionally correct but also robust enough to tolerate and mitigate random faults.

Afterward, tasks 6-A.iii and 6-A.iv are devoted to analyzing if the guidelines for utilizing the design and V&V tools selected in Step 3 for FPGAs and software, respectively, have indeed been followed. These analyses aim to ensure that the tools have been used and configured in such a way to support that the as-built FPGA/PLD programming and software binary code remain faithful to their fault-tolerant design, since improper settings (e.g., optimizations) can void these mechanisms.

Task 6-A.v has the objective of performing at least semiformal analyses on the image set of the safety-critical AI instances in such a way as to strengthen the exhaustive understanding of their outputs given the domains of their inputs. For specific AI variants of higher opaqueness (e.g., deep supervised learning), these analyses might only be feasible in an overapproximate way by means of application-specific models built with expert knowledge and exercised with Satisfiability Modulo Theories (SMT) [30,49,59–61].

Finally, the objective of task 6-A.vi is to utilize the information collected throughout the remainder of the Safety ArtISt method to build quantitative safety models and analyze whether the safety targets of each safety-critical function have indeed been met. Special care has to be given to modeling the behavior-related failure rate of AI instances and merging it with the random failure rates of hardware components.

Like Step 5, if a safety issue is detected at any of the 6-A activity tasks, the forthcoming steps can be skipped for time-saving purposes. The root causes of the identified safety nonconformities are assessed and traced back to their originating Safety ArtISt steps so that the design is revised in a new Safety ArtISt iteration.

If Step 6 is finished with safety requirements having been accomplished, the system's revenue service is started, and Safety ArtISt moves to *Step 7—Operation and Maintenance*. In this step, the system is periodically monitored by means of the log-generating functions designed in Step 4, especially if the formal analyses covered in 6-A.v could not be performed in entirety due to difficulty in building the needed models.

Moreover, special care is required for systems with online learning and whenever maintenance actions are performed. Since systems with online learning change their own hyperparameters as they learn with operation-collected data, the safety arguments built up to Step 6 are periodically reviewed in order to guarantee that the system still adheres to its safety requirements as its learning progresses. Similarly, if maintenance actions affect safety-critical AI, the Safety ArtISt method is rerun so that the maintenance effects on safety are assessed.

2.2. Safety ArtISt Specificities for Safety-Critical Design with FPGAs

The Safety ArtISt method includes specific practices to guide the design and analysis of safety-critical AI embedded in PLDs and FPGAs. Such practices are herein detailed, along with the justification of their relevance for the safety assurance of FPGA-based AI.

In Step 1, the system architecture and the RSRs are crafted in such a way that they comply with known FPGA-related restrictions defined in current safety standards and remain valid regardless of whether AI is used. These restrictions comprise the following items:

1. No SIL 4 safety-critical function is exclusively implemented within a single FPGA [4];
2. On-chip redundancy is not used as the sole fault tolerance means for SIL 3 and SIL 4 safety-critical functions exclusively implemented with PLDs and FPGAs [6];
3. It is highly recommended to implement PLD and FPGA-based SIL 3 and SIL 4 functions with one of the two following approaches [6]:
    a. External redundancy with multiple programmable devices of different technologies (e.g., RAM-based FPGA along with Flash-based PLD);
    b. External redundancy or PLD/FPGA monitoring with other categories of hardware circuitry (e.g., microprocessors and nonprogrammable hardware).

In Step 3, the architectural refinement of activity 3-A reflects the same concerns detailed in Step 1. Moreover, specific safety principles and recommended practices are outlined in activity 3-C in order to guide the detailed design in Step 4 and the means to perform its V&V during Steps 5 and 6.

For instance, if the safety-critical AI relies on HLS, at least three different architectural levels of abstraction are created. The first level, depicted in Figure 2, corresponds to the software architecture that details the initial AI design and allows building the AI instances. It is then followed by the HLS architecture, which is illustrated in Figure 3 and represents how the software elements of the first block are processed towards generating their functionally equivalent modeling in HDL. Finally, the third and final architecture corresponds to the final PLD/FPGA-based infrastructure that is effectively deployed, as shown in Figure 4. It utilizes the HLS-generated HDL components arranged in such a way that the system functions are achieved.

HLS also plays an important role in defining the workflow and the recommended practice of Step 5. For instance, since HLS requires both the AI base models and their instances to exist prior to translating software into HDL, supervised-learning-based solutions are initially trained at the software level (activity 5-B) before the actual HDL implementations of AI base models are assessed in activity 5-A. Hence, the training of 5-B is performed in advance during Step 4 with HLS. Moreover, the strategy for the functional V&V activities of Step 5 (5-A and 5-C) might also depend on how complex the HDL-coded AI instances are. Two plausible alternatives are fully carrying out these activities in the PLD/FPGA domain or performing them at the software level and ensuring, via exhaustive tests, that their HDL translations are equivalent to their software-originating counterparts.

Special care also has to be given to data numeric representation. Since complex AI models with high fixed-point or floating-point precision might require costly FPGAs with a substantial amount of available resources (e.g., registers and logic modules), the design takes into account a trade-off involving cost, processing capabilities, performance, and safety requirements. As a result, sensitivity analyses on the resources for implementing an AI model and the safety-related AI performance metrics are performed in activity 5-C. A proven-as-correct AI implementation in software can be utilized as a baseline for checking to what extent an FPGA-based design deviates from it if a constrained numerical precision is needed to meet cost restrictions.

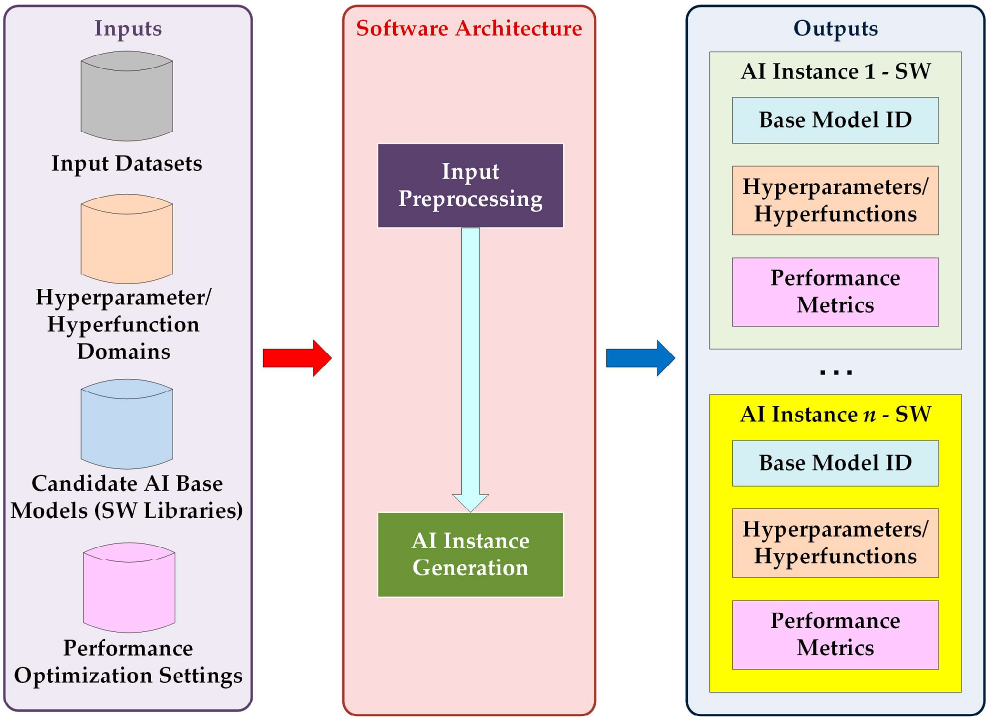


Figure 2. Software architecture model for safety-critical AI with HLS.

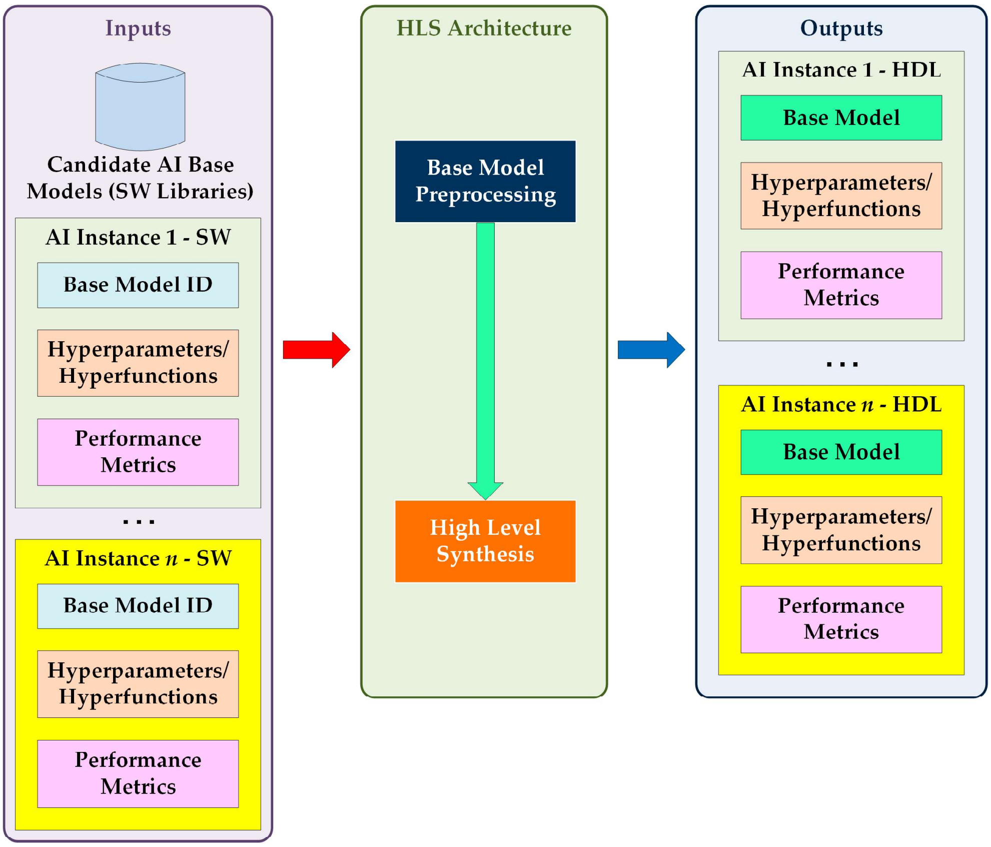


Figure 3. HLS architecture model for safety-critical AI with HLS.

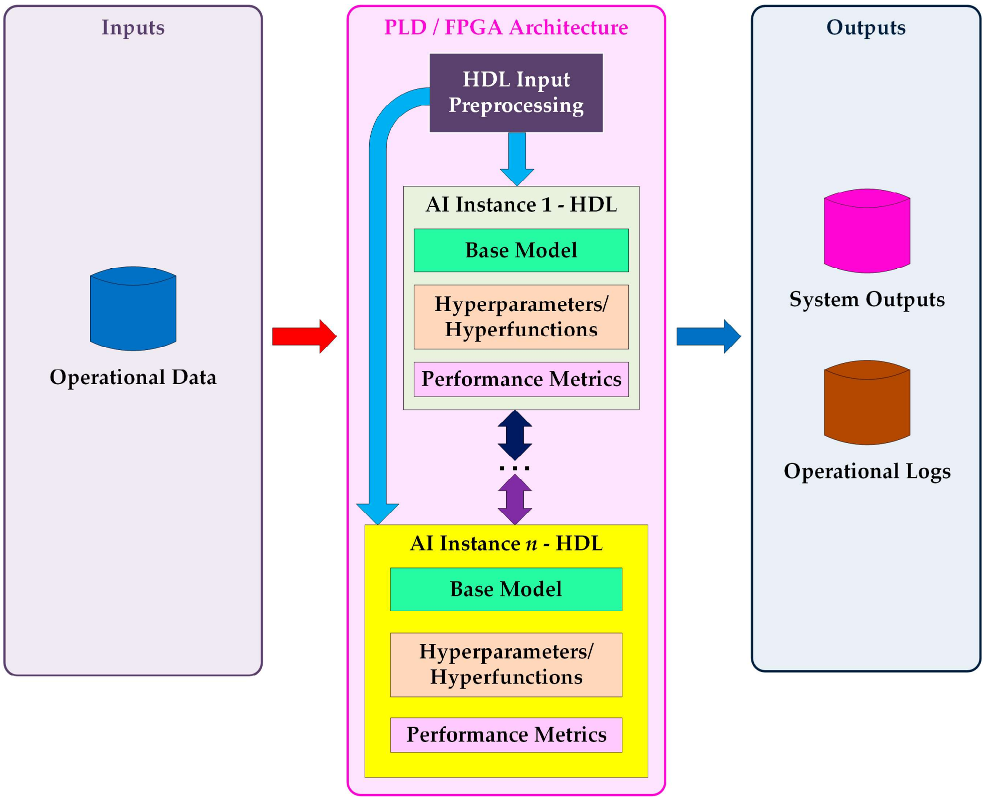


Figure 4. PLD/FPGA architecture model for safety-critical AI with HLS.

Other Safety ArtISt activities, which include FPGA and PLD specificities, are those involving fault-tolerant design and project tools. They start in Step 3 (3-D) by defining design guidelines and follow with their implementation in Step 4 and compliance checks throughout Steps 5 (5-A) and 6 (6-A.ii).

Since fault tolerance deals with detecting random hardware faults and mitigating them to prevent the impacts of Single-Event Upsets (SEUs), the techniques for this purpose are not directly affected by whether AI is utilized. As a result, the recommended practices for pre-existing safety-critical systems with FPGAs and PLDs still hold [31,62]. If HLS is used, manual changes to the HDL-generated code might be needed, since HLS tools might not support the native coding of fault tolerance directives.

As per the selection and the settings of the safety-critical design and V&V tools, which start with the specifications in Step 3 (3-B) and extend to their actual usage and verification in Steps 4 to 6, recommended practices for safety-critical systems without AI also remain applicable for safety-related AI functions:

1. Ensuring that timing restrictions have been observed in pessimistic environmental conditions in such a way as to prevent metastable behaviors [31,62]. This can be achieved by means of timing analyses (e.g., maximum clock frequency constraints) available in FPGA/PLD development suites, such as Intel Quartus Prime [63];
2. Disabling automatic optimizations during compilation, synthesis, and routing in order to ensure that the fault-tolerant design introduced in HDL coding is not eliminated to improve performance [31];
3. Guaranteeing that no automatic design decisions (e.g., latch inference, pinout changes) have been taken during compilation, synthesis, and/or routing [31].

2.3. *Contextualization of the Autonomous Vehicle Braking Control System Case Study*

A case study involving the full lifecycle of an FPGA-based BCS solely relying on supervised learning and explainable AI was selected to exercise the Safety ArtISt method and assess to what extent it reaches the research's presumed goals and contributions.

The choice of the BCS was justified to allow the Safety ArtISt method to be utilized and assessed in a real but not overly complex application for embedded systems. It was assumed that the autonomous vehicle moves on a noninclined plane, in a single direction, and along a single axis. Therefore, the vehicle can sense an obstacle ahead, anticipate a potential collision, and brake before reaching such an obstacle. These operational context constraints are depicted in Figure 5.

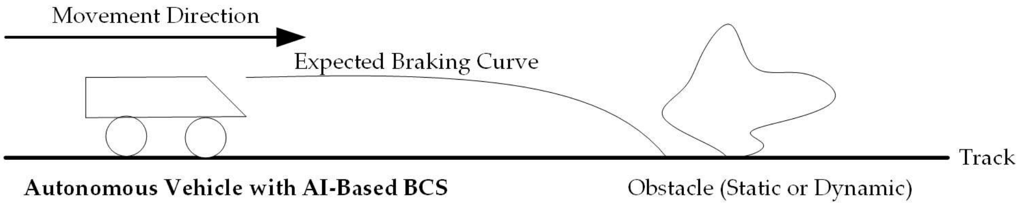

Movement Direction
Expected Braking Curve
Track
Autonomous Vehicle with AI-Based BCS
Obstacle (Static or Dynamic)

Figure 5. Operational context for the case study's BCS.

The high-level architecture of the BCS, in turn, is presented in Figure 6. It shows that the BCS has three inputs: (i.) the distance to its next obstacle (ahead), (ii.) the vehicle's current speed, and (iii.) the vehicle's Guaranteed Emergency Braking Rate (GEBR). The latter establishes the minimum deceleration that can be surely achieved by applying the brakes at a given operational condition (e.g., wheel–surface adherence and braking system integrity).

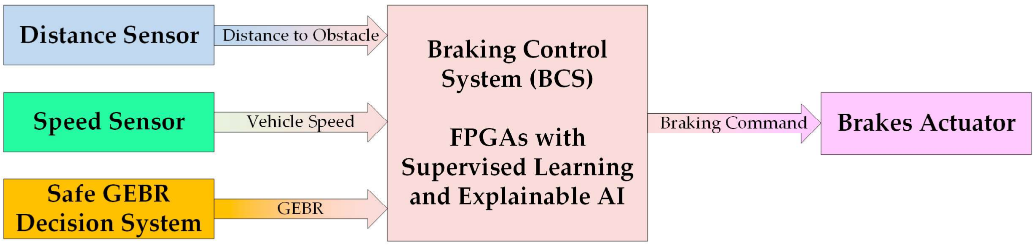

Distance Sensor
Distance to Obstacle
Speed Sensor
Vehicle Speed
Safe GEBR Decision System
GEBR
Braking Control System (BCS)
FPGAs with Supervised Learning and Explainable AI
Braking Command
Brakes Actuator

Figure 6. High-Level architecture of the case study's BCS.

The BCS periodically reads information from the distance and speed sensors and decides, considering the GEBR that is applicable to the vehicle's operation at that moment, whether the brakes are applied so that the vehicle does not collide with an obstacle ahead. It is assumed that the BCS receives GEBR-related information from an external safety-critical system, referred to as "Safe GEBR Decision System" in Figure 6, which ensures that the GEBR reflects the vehicle's braking capacity at the given environmental and braking integrity conditions.

Furthermore, even though BCSs are well implemented without AI by means of standardized Newtonian kinematics [64], the herein-explored AI-based solution was explored for two reasons. Firstly, it allows comparing how safe AI is in relation to closed-form solutions for the same problem. Secondly, since other types of AI variants (e.g., unsupervised learning, reinforcement learning, knowledge-based systems) and software implementations are covered in other ongoing case studies of the Safety ArtISt research project, focusing on supervised learning and explainable learning on FPGAs allows evaluating the contribution of the method towards building and ensuring the safety of systems with these AI variants.

Finally, the roles played by research team members in design and V&V activities were attributed in such a way as to maximize the independence between designers, verifiers, and validators, as recommended in current safety standards [4,6], while keeping each researcher's available workload within acceptable limits.

3. Results

The objective of this section is to present the main results stemming from the application of an iteration of the Safety ArtISt method on the BCS case study introduced in Section 2.3. Each of its subsections is mapped onto one of the Safety ArtISt steps.

Further information on the case study as a whole is available in its technical research report [65] and in the directory "*EC4-FPGA_Braking*" of the Safety ArtISt project's GitHub repository [66].

3.1. Remarks and Results of Step 1—System Specification

The initial H&R analyses of Step 1 were carried out with the quantitative risk analysis method of IEC61508:2010, leading to the four SRs listed in Table 6 for the BCS.

Table 6. List of Safety Requirements (SR) for the BCS.

SR ID	SR Description
SR 4.1	The BCS shall meet a SIL 4 target (THR < 10^{-8} failures per hour) in braking the vehicle when there is a potential collision with an obstacle ahead.
SR 4.2	The BCS shall meet a SIL 3 target (THR < 10^{-7} failures per hour) in not applying brakes when there is no potential collision with an obstacle ahead.

Afterward, the initial H&R analysis was refined with the HAZOP-based approach detailed in Section 2.1. It included six different base models typical of supervised learning with explainable AI: k-Nearest Neighbors (kNN), decision tree (DT), ANN, SVM, random forest (RF), and other ensembles. A total of 376 scenarios involving potentially unsafe situations from input processing and the features of each AI variant (e.g., hyperparameters and performance metrics) were analyzed, leading to a set of 62 RSRs. These RSRs are subdivided into eight groups:

1. Twenty-seven RSRs are applicable to the BCS regardless of the AI base model;
2. Six RSRs are applicable to the BCS only if at least one decision tree is utilized
3. Four RSRs are applicable to the BCS only if at least one kNN instance is utilized;
4. Five RSRs are applicable to the BCS only if at least one SVM instance is utilized;
5. Eight RSRs are applicable to the BCS only if at least one ANN instance is utilized;
6. Two RSRs are applicable to any ensemble (i.e., either RF or other ensemble);
7. Five RSRs are applicable to the BCS only if at least one RF is utilized;
8. Five RSRs are applicable to the BCS only if at least one ensemble except for RF is utilized.

Among the general-purpose RSRs, it is worth highlighting that the BCS updates its outputs at every 500 ms; otherwise, the BCS is led to a safe state. The response time of this RSR was determined in such a way that the BCS was able to process sensing data while still providing a fast enough response to trigger its brakes.

In addition to the RSRs, the HAZOP-based H&R analyses also allowed detailing the internal architecture of the BCS, represented by the "BCS" black box in Figure 6, in accordance with Figure 7. This diagram indicates that the BCS comprises a set of up to "m" different FPGAs, each of which containing up to "n_m" AI instances. Each AI instance generates a probability for applying brakes ($P(bcp(a, k))$ for every AI instance a = {1, . . . , n_k} and every FPGA k = {1, . . . , m}), and a decision element collects these probabilities and generates the FPGA's ensemble decision ($P(bcp(k))$ for every FPGA k = {1, . . . , m}). Afterward, the responses of all "m" FPGAs are combined by a majority voter, which ultimately generates the brake command prediction (bcp) that is sent to the brake actuator. The mandatory usage of multiple FPGAs is justified by the normative arguments and constraints discussed in Section 2.2.

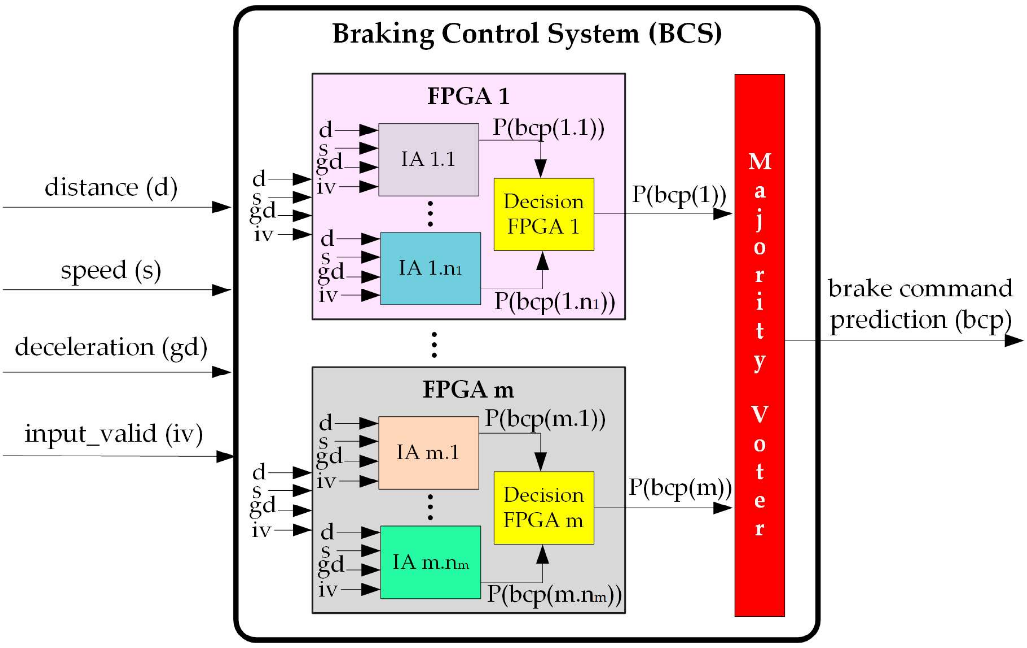


Figure 7. Detailing of the BCS systems-level architecture.

The diagram in Figure 7 also includes a new input, referred to as "*input_valid* (*iv*)", in addition to those already depicted in Figure 6. The objective of this input is to indicate to the BCS that the "*distance* (*d*)", "*speed* (*s*)", and GEBR "*deceleration* (*gd*)" inputs are stable enough to be read by the BCS at a processing cycle, thus preventing unstable data from being processed by the BCS. It is deemed that "*input_valid* (*iv*)" is obtainable by combining control signals generated by the distance sensor, the speed sensor, and the safe GEBR decision system in Figure 6.

3.2. Remarks and Results of Step 2—Input Datasets Selection and Analysis Specification

Given that Newtonian kinematics guide the braking of a vehicle, its equations were chosen as the source to generate synthetic datasets for usage throughout the remainder of the BCS lifecycle. The equations that were employed for that matter are based on the safe braking model of IEEE 1474.1:2014 [64] for the braking of trains in metro and rail domains.

The following guidelines were followed to produce the datasets that were used in the remainder of the case study:

1. The autonomous vehicle's parameterization (e.g., mass, moment of inertia, aerodynamic drag coefficient, acceleration, GEBR, and braking system behavior), as well as environmental features (e.g., wind speed and air density), were defined for rail and metro applications by considering the data of a train fleet of the São Paulo (Brazil) Metro, which feature five different GEBR (*gd*) conditions: 1.395 m/s^2, 1.1625 m/s^2, 0.93 m/s^2, 0.78 m/s^2, and 0.65 m/s^2 [65];
2. Distance and speed were constrained to typical operational conditions of the same application domain. It was assumed that the distance between the vehicle and its next obstacle remains within the interval [0 m; 2000 m] and that the vehicle's speed ranges from 0 m/s up to 29 m/s (104.4 km/h);
3. Commercial distance and speed sensors that match the domains defined in item "2" were selected so that their measurement tolerances are also taken into account in the dataset generation process [65].

The algorithm that describes the dataset generation is represented by the flowchart in Figure 8. It indicates that data records are tuples of the type (*d*, *s*, *gd*, *bcp*) generated by varying *d*, s, and *gd* within their domains, with steps of 10 m for *d* and 0.5 m/s steps for *s*, and exhaustively including all plausible combinations. Within the domains of *d*, s, and *gd*, 57,286 tuples are generated.

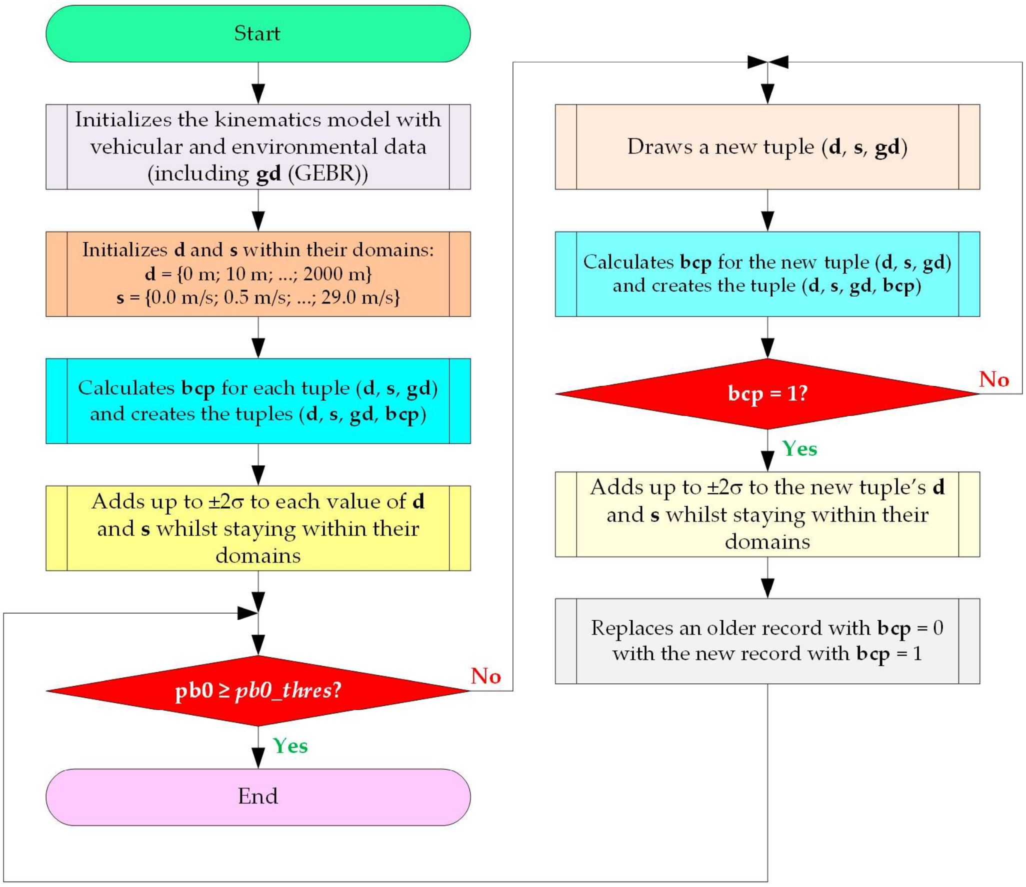

Start
Initializes the kinematics model with vehicular and environmental data (including **gd** (GEBR))
Initializes **d** and **s** within their domains:
d = {0 m; 10 m; ...; 2000 m}
s = {0.0 m/s; 0.5 m/s; ...; 29.0 m/s}
Calculates **bcp** for each tuple (**d**, **s**, **gd**) and creates the tuples (**d**, **s**, **gd**, **bcp**)
Adds up to ±2σ to each value of **d** and **s** whilst staying within their domains
pb0 ≥ *pb0_thres*?
No
Yes
End
Draws a new tuple (**d**, **s**, **gd**)
Calculates **bcp** for the new tuple (**d**, **s**, **gd**) and creates the tuple (**d**, **s**, **gd**, **bcp**)
bcp = 1?
No
Yes
Adds up to ±2σ to the new tuple's **d** and **s** whilst staying within their domains
Replaces an older record with **bcp** = 0 with the new record with **bcp** = 1

Figure 8. Flowchart of the dataset generator.

The brake command prediction (*bcp*) is then set to “1” (apply brakes) if the vehicle's braking curve reaches the obstacle at the given conditions of *d*, *s*, and *gd*, or to “0” (not apply brakes) otherwise. Afterward, positive or negative random noise of up to twice the standard deviations (σ) of distance and speed sensors' tolerances are added to the figures of *d* and *s* in order to represent their intrinsic uncertainties.

Finally, another aspect of the dataset generation algorithm is the possibility of configuring the proportion of data records with *bcp* = 0, hereinafter referred to as “*pb0*”. This proportion can be set by choosing a setting “*pb0_thres*” within the open interval [0%; 90%), The justification for the configurable “*pb0*” is to allow the fine-tuning of AI instances better specialized in braking, since data records with *bcp* = 1 are scarcer than those with *bcp* = 0. By decreasing *pb0*, data records with *bcp* = 0 are discarded and replaced by additional records with *bcp* = 1. The upper limit of “*pb0*”, which is no higher than 90%, has a twofold justification: (i.) a noise-free dataset has an 89.9% proportion of records with *bcp* = 0, and (ii.) given the abundance of records with *bcp* = 0, it was not deemed relevant to increase *pb0* even further for the BCS-related AI design.

A Python application was developed in accordance with the previous specification, and its results were redundantly verified by another member of the Safety ArtISt team who employed Microsoft Excel 2013 as a dual tool to perform the same processing. With this procedure, the developed dataset generator was considered functionally correct, thus leading the datasets generated with it to be adequate for the case study's end application.

Another aspect that is worthy of highlighting in Step 2 is the identification of potential corner cases in the datasets generated with the aforementioned tool. Two categories of corner cases were considered relevant for the case study:

1. *Corner Cases* (*0, 1*): This group includes data records classified with *bcp* = 0 based on noise-free *d* and *s*, and *gd* but that would shift to *bcp* = 1 with their actual noisy *d* and *s* figures. Data records of this corner case group are relevant in assessing whether the SR 4.2 is indeed met by the BCS;
2. *Corner Cases* (*1, 0*): This group is the opposite of the previous one. It comprises data records whose noise-free input data led to *bcp* = 1 and that would rather be classified with *bcp* = 0 based on the actual noisy *d* and *s* figures. As a result, the records of this group are important to evaluate the BCS compliance with the SR 4.1.

The percentage of records of both corner case groups varies with the proportion "*pb0*". For instance, datasets with *pb0* in its upper limit of 89.9% feature shares of circa 0.18% to 0.25% of their 57,286 records for each corner case group depending on vehicle-related settings (e.g., mass). These proportions shift to 0.94% for (1, 0) and to 0.11% for (0, 1) if *pb0* is reduced to 50%. The increase in corner cases of the group (1, 0) and the decrease in those of the group (0, 1) as *pb0* is reduced are indeed expected. Given that more records with *bcp* = 1 and fewer records with *bcp* = 0 will be within the dataset as *pb0* is lowered, the chance for meeting the criteria of the corner case group (1, 0) goes up whereas that for the corner case group (0, 1) diminishes.

At the end of Step 2, 19 different datasets were generated for the remainder of the case study. All of them share the same end application operational conditions (i.e., no changes on vehicle and environment-related parameters) and differ on two main aspects: *pb0* and data randomness due to the inclusion of noises to *d* and *s*.

Seven datasets were reserved for 10-fold CV and AI training purposes: two different datasets with *pb0* = 89.9% (*DatasetT&V* and *DatasetT&V2*), two different datasets with *pb0* = 80% (*DatasetStratified80v3* and *DatasetStratified80v2*), one dataset with *pb0* = 65% (*DatasetStratified65*), one dataset with *pb0* = 55% (*DatasetStratified55*), and one dataset with *pb0* = 50% (*DatasetStratified50*).

Three additional datasets with *pb0* = 89.9% were also generated to support the analysis of corner cases: *DatasetT1*, *DatasetT2*, and *DatasetT3*. This proportion was chosen for this analysis because it is deemed that, since *pb0* approximates the rate of BCS processing cycles at which brakes are applied, utilizing it would indicate that the BCS triggers the brake commands in circa 10% of its processing cycles. This allows a reasonably better approximation of an actual BCS for rail applications than with datasets with lower *pb0*.

The datasets *DatasetT1*, *DatasetT2*, and *DatasetT3* are also used along with a set of nine other datasets with other *pb0* rates for two other objectives: performing the *holdout* tests of the trained AI instances and providing supporting performance indicators for the quantitative safety analysis. The nine datasets that were generated and used in addition to *DatasetT1*, *DatasetT2*, and *DatasetT3* are as follows: *DatasetTestZ01* (*pb0* = 1%), *DatasetTestZ10* (*pb0* = 10%), *DatasetTestZ20* (*pb0* = 20%), *DatasetTestZ30* (*pb0* = 30%), *DatasetTestZ40* (*pb0* = 40%), *DatasetTestZ50* (*pb0* = 50%), *DatasetTestZ60* (*pb0* = 60%), *DatasetTestZ70* (*pb0* = 70%), and *DatasetTestZ80* (*pb0* = 80%). The objective of using datasets with different *pb0* rates is to assess to what extent the SRs are affected by different operational conditions, ranging from extremely pessimistic scenarios at which the brakes are frequently requested (*pb0* = 1%) up to scenarios at which the brake request rate is lower (*pb0* = 89.9%).

3.3. Remarks and Results of Step 3—AI Preliminary Design

Activity 3-Pre was initially performed to assess the most promising AI base models for the end application among all candidates raised in Step 1. For this purpose, these AI base models were initially exercised in software with the infrastructure provided by the Python-coded *scikit-learn* library (version 1.3.1) [67]. By means of CV and holdout tests with the datasets defined in Step 2 (Section 3.2), it was verified that DTs outperformed kNN, SVMs, and ANNs in accuracy, precision, recall, and specificity regardless of *pb0*. These performance metrics are utilized in this research as per their definitions by James et al. [68]. Moreover, since each DT was specialized in producing braking commands as per the corresponding *pb0* of its training dataset, building RFs by joining DTs trained with datasets of different *pb0* settings was deemed an appropriate way to leverage the overall BCS performance.

Activity 3-A started by assessing whether the direct HDL design or HLS would be used in the target implementation of DTs and RFs for FPGAs. Given that HLS tools such as Conifer [69,70] and LeFlow [71,72] are able to translate DTs and RFs coded in Python into equivalent HDL models targeted to FPGAs, an HLS approach was chosen for the case study. As a result, the initial BCS architecture of Figure 7 (Section 3.1) was refined considering the three-level abstraction model advocated in the Safety ArtISt method and detailed in Section 2.2.

Since the software and HLS levels are direct instantiations of the templates presented in Figures 4 and 5 (Section 2.2) with BCS-specific datasets and AI features, special attention is given to the final hardware architecture, achievable with the HLS byproducts and depicted in Figure 9. The results obtained in 3-Pre allow defining that the BCS comprises $m = 2$ independent FPGAs for AI processing and that the number of AI instances for both FPGAs (n_1 and n_2) is set to three, which means that each FPGA includes an RF with three DTs. In this setup, the majority voter, referred to as "*voting*" in Figure 9, averages the braking probabilities retrieved by both FPGAs, represented by "*prob_mean*", and sets *bcp* to the "active brakes" state whenever this probability is of at least 0.5. Given that "*voting*" tends to require a simple design, it better suits a Complex PLD (CPLD) than an FPGA.

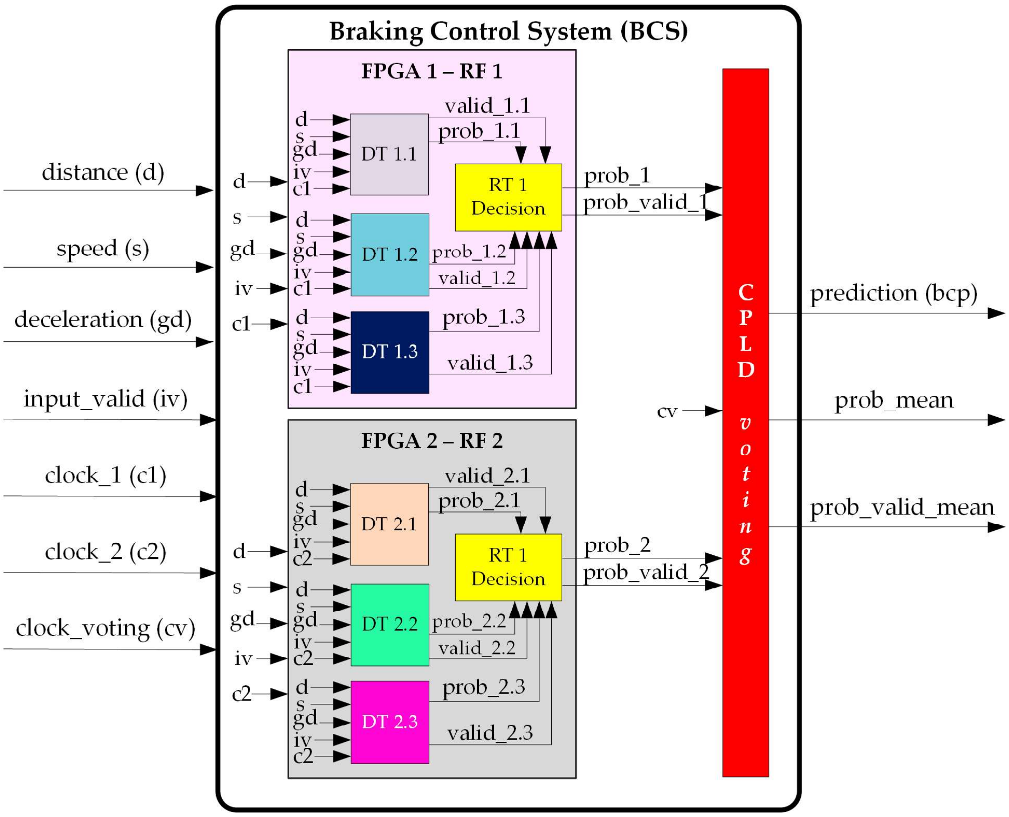


Figure 9. Refined BCS hardware architecture.

The architecture in Figure 9 also reveals the following refinements related to FPGA independence, processing stability, and trustworthiness:

1. Each FPGA/CPLD has its own clock signal ("*clock_1*", "*clock_2*", and "*clock_voting*");
2. In addition to *bcp*, the BCS has two other outputs: "*prob_mean*", which indicates the probability that supported the *bcp* decision, and "*prob_valid_mean*", which is detailed in item "4";
3. Each FPGA/CPLD enforces a safe state (by applying brakes) via Power-On Reset (POR) and reaches an absorbing safe state whenever faults related to loss of com-

munication with inputs (i.e., no periodic enabling of "*input_valid*") and/or partner FPGAs/CPLD. These directives exempt the need for a reset signal for all three programmable components in Figure 9, since only a cold restart is able to allow system recovery from an availability perspective.

4. In order to cascade control flow data from the inputs up to the final "*voting*" stage, all three programmable devices provide validity signals that enforce whether their outputs have been recently updated. These include "*valid_1.1*" to "*valid_1.3*" for each DT on FPGA 1, "*valid_2.1*" to "*valid_2.3*" for each DT on FPGA 2, "*prob_valid_1*" for the FPGA 1 RF as a whole, "*prob_valid_2*" for the FPGA 2 RF as a whole, and "*prob_valid_mean*" for "*voting*". These signals allow not only detecting and mitigating the lack of input activity on "*input_valid*" but also potential random faults that prevent the proper operation of both FPGAs and the "*voting*" CPLD.

In activity 3-B, the tools and resources that are applicable to the remainder of the BCS lifecycle are selected. These include the following items:

1. Anaconda 22.9.0 (with Python 3.9.15) to design the BCS DTs and RFs in software;
2. Conifer 0.3 to perform the HLS of DTs and RFs. Conifer was preferred over LeFlow because it has straightforward, native support for translating RFs coded as instances of Python's *scikit-learn* class *RandomForestClassifier* into VHDL;
3. AMD/Xilinx Vivado 2022.2 to provide the needed infrastructure for built-in tests and simulations performed by Conifer during the HLS process;
4. Intel/Altera Quartus Prime Lite 22.1 Std 0.915 for additional VHDL design (e.g., "*voting*" CPLD), timing analyses, and the programming of Intel/Altera devices for tests;
5. Intel ModelSim Starter Edition 20.1.1.720 for simulations. By using it along with AMD/Xilinx Vivado, the performance of the VHDL-coded AI generated on the HLS can be cross-checked, thus improving the confidence of the results;
6. Digilent AnalogDiscovery and Digilent Waveforms 3.20.1 to generate input signals during physical tests.

Other relevant settings and practices defined throughout activities 3-B to 3-D, in turn, comprise the following items:

1. Ensuring compliance with VHDL 2008 [31,62];
2. Ensuring that the final VHDL code, including the HLS-generated one, adheres to fault-tolerant design practice defined by Lange et al. [62] and Silva Neto et al. [31]. Since Conifer was not designed with fault tolerance in mind, manual code changes might be needed;
3. For Intel/Altera Quartus Prime, turning off optimizations at compilation, synthesis, and routing to avoid unwanted changes to the as-designed fault-tolerant design.

3.4. Remarks and Results of Step 4—AI Detailed Design

Based on the design decisions in Step 3, the AI instances were initially developed in software by using the *scikit-learn* Python library (version 1.3.1). Since its *RandomForestClassifier* only allows building DTs with the same depth, the approach that was used to build RFs with DTs of different depths followed two steps: (i.) designing each DT as an instance of *scikit-learn*'s *DecisionTreeClassifier* class and (ii.) creating instances of the *scikit-learn RandomForestClassifier* class with the DTs mapped onto each RF based on their respective performance metrics. The second step is needed to allow the HLS via Conifer, since Conifer does not support HLS straight from instances of the *DecisionTreeClassifier* class. A Python function called "*convert_tree_to_random_fores*" was developed for this purpose [65].

The design of the DTs was carried out with 10-fold CV with the corresponding datasets of Step 2 (Section 3.2) and by performing an exhaustive grid search on the domains of eight hyperparameters: *criterion*, *splitter*, *max_depth*, *min_samples_split*, *min_samples_leaf*, *min_weight_fraction_leaf*, *max_leaf_nodes*, and *max_features* [73]. Special attention was given to limiting the depth of the DTs to up to 10 in such a way as to convey DTs that remain explainable and with reasonable resource requirements for FPGA synthesis after the

HLS. Preliminary tests with Conifer also indicated that, on a computer with Intel i7-7700 (3.6 GHz × 4 cores; 8 threads) and 12 GB RAM, the HLS succeeded only for RFs with DTs no deeper than 15 levels; otherwise, Conifer crashed.

A total of eight different DTs, shown in Table 7, were designed with the previous workflow by advancing activity 5-B to Step 4. DTs 1 and 5 were iteratively discarded due to overly high depths and were ultimately replaced by DT 6, which had a similar performance [65] for the same "*pb0*" setting. Furthermore, Table 7 also indicates that, by means of software-based holdout tests with *DatasetT3* ("*pb0*" = 89.9%), the DTs designed with datasets of lower "*pb0*" have higher recall than those designed with datasets of higher "*pb0*". This indicates that the latter DTs tend to apply brakes more frequently than the ones with higher "*pb0*", which is expected given the higher abundance of data records leaning towards the need for brakes. On the other hand, and for a complementary reason, the DTs of higher "*pb0*" are significantly better in precision, which plays a relevant role in avoiding unneeded brake applications posed by the lower "*pb0*" counterparts.

Table 7. DTs, CV, and Training Datasets and Holdout Test Performance with *DatasetT3*.

DT ID	CV and Training Dataset ("*pb0*")	Precision	Recall	Depth
1	DatasetT&V (89.9%)	96.9%	96.9%	16
2	DatasetStratified80v3 (80%)	92.1%	97.8%	8
3	DatasetStratified65 (65%)	90.6%	99.0%	8
4	DatasetStratified55 (55%)	79.7%	99.5%	6
5	DatasetT&V (89.9%)	96.6%	96.3%	13
6	DatasetT&V2 (89.9%)	97.0%	96.7%	9
7	DatasetStratified80v2 (80%)	94.2%	98.5%	9
8	DatasetStratified50 (50%)	55.3%	99.6%	3

Based on the results presented in Table 7, the RFs of each of the two BCS FPGAs depicted in Figure 9 (Section 3.3) were designed with the principle of maximizing the overall recall while still achieving good precision. This was reached through experimentation and led to making each RF comprise one DT with high recall and low precision, one DT with average recall and average precision, and one DT with low recall and high precision. Therefore, the RF of FPGA 1, hereinafter referred to as "RF 1", includes DTs 2, 3, and 4, whereas the RF of FPGA 2, hereinafter referred to as "RF 2", comprises DTs 6, 7, and 8.

Holdout tests with *DatasetT3* yielded a precision of 90.4% and a recall of 99.4% for RF 1, as well as a precision of 94.2% and a recall of 98.8% for RF 2. These results indicate that the recall of both RFs is significantly close to that of the DTs with the highest recalls, whereas the precision approached that of the middling DT.

Step 4 also comprised the VHDL design of the "*voting*" CPLD in accordance with the functions and safety principles detailed for it in Step 3 (Section 3.3). However, since the design of *voting* is not based on AI nor contains any other cutting-edge techniques except for classic safety-critical design for CPLDs, it is not further detailed in this paper.

3.5. Remarks and Results of Step 5—AI Training and AI Preliminary V&V

Step 5 started with the HLS of the RFs generated in Step 4. The HLS was performed with the aid of Conifer's "*Python to VHDL*" module, which translates Python-coded RFs into VHDL and simulates the resulting VHDL code with AMD/Xilinx Vivado with a given input dataset so that both implementations can be compared.

In order to feed Conifer with the BCS RFs coded as instances of *scikit-learn*'s *RandomForestClassifier* class, one of the needed settings for Conifer is the fixed-point precision employed to translate the numeric inputs and outputs into VHDL numerical and logical types. The notation hereinafter utilized for the fixed-point precision is *ap_fixed<n,i>*, with *n* representing the sum of bits for both the integer and decimal parts and *i* depicting the integer part's bit width. As a result, the width of the decimal part equals *n-i*.

Conifer imposes two restrictions in setting *ap_fixed<n,i>*: the most significant bit of the integer part represents the number's sign (i.e., positive or negative), and the settings of *n* and *i* are the same for every variable. Based on these restrictions and on the domains of all BCS inputs and outputs, the integer part mandatorily features *i* = 12 bits, since 11 bits are needed to represent the upper range of the *distance* (*d*) measurements within [0 m; 2000 m] and an additional bit is needed for the sign. The decimal part *n-i*, in turn, was defined with the aid of a sensitivity analysis at which it was varied within the range [6; 12] in order to analyze the following performance indicators:

1. *Number of Wrong Predictions* (#WP): This indicator represents the number of predictions for which the HLS-generated VHDL produced an outcome that differs from the originating software implementation;
2. *Percentage of Wrong Predictions* (%WP): This indicator represents the percentage of the test dataset records that were added to #WP, as defined in Equation (1):

$$\%WP = \frac{\#WP}{\#\ Records\ on\ Test\ Dataset} \tag{1}$$

3. *The 75th Percentile Absolute Error* (*AE75*): This indicator represents the 75th percentile of the absolute errors in the braking output probability generated by the RF. It was chosen with the objective of assessing the degree to which the RF output probability is affected at a given *n-i* decimal bit width.

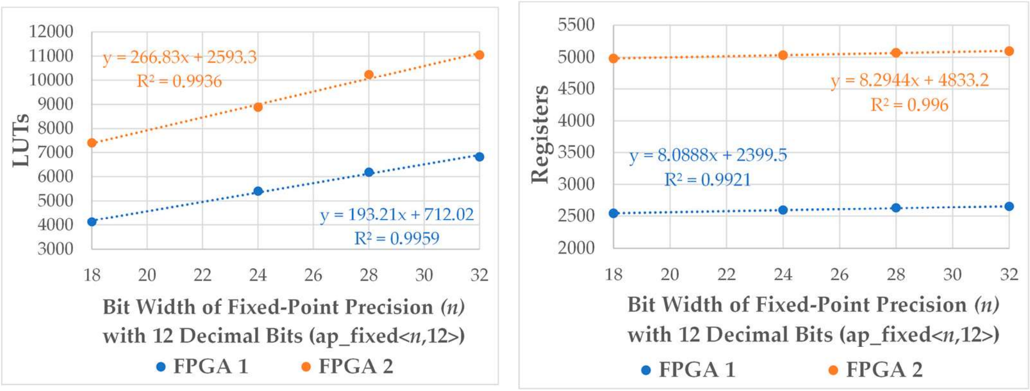
The results presented in Table 8 by exercising RF 1 with *DatasetT3* indicate that *ap_fixed<24,12>* provides no prediction errors at minimum AE75 ratings and, thus, represents an optimal setting in achieving the same performance of its software counterpart while keeping the needed resources for FPGA synthesis at a minimum for this performance. With all other *ap_fixed<n,i>* settings, #WP is not null, thus indicating that the HLS introduced additional errors with regard to the original software implementation.

Table 8. Performance Results to Optimize *ap_fixed<n,i>*—Tests with RF 1 and *DatasetT3*.

ap_fixed<n,i>	**#WP**	**%WP (%)**	**AE75**
ap_fixed<18,12>	44	0.077%	0.01346
ap_fixed<19,12>	25	0.044%	0.00727
ap_fixed<20,12>	19	0.033%	0.00346
ap_fixed<21,12>	6	0.010%	0.00157
ap_fixed<22,12>	5	0.008%	0.00078
ap_fixed<23,12>	2	0.003%	0.00042
ap_fixed<24,12>	0	0%	0.00020

The analysis of *ap_fixed<n,i>* also allowed identifying that the quantity of FPGA Look-Up Tables (LUTs) and registers linearly increases with *n* and *i* = 12, as shown in Figure 10 with synthesis data generated with AMD/Xilinx Vivado for the FPGA AMD/Xilinx Spartan 7 XC7S100FGGA484. Furthermore, Intel Quartus Prime also evidenced that the Intel/Altera FPGA Cyclone V 5CEBA4F23C7N, available at terasIC's DE0-CV experimental board, would suffice in fitting RF 1, RF 2, and *voting* with *ap_fixed<24,12>*, provided the occupancy rates presented in Table 9 are sufficiently low.

After the HLS was performed with *ap_fixed<24,12>*, the source code generated by Conifer was inspected in order to assess its functional adequacy and its robustness in dealing with fault tolerance. Each RF was structured in the seven VHDL entities depicted in Figure 11. In this diagram, an arrow directed from "A" to "B" indicates that "A" instantiates or refers to "B" at least once within its design.



(**a**) (**b**)

Figure 10. Relationship between the resources of the FPGA AMD/Xilinx XC7S100FGGA484 and the Bit Width of Fixed-Point Precision Numbers (*n*) with 12 Decimal Bits (*ap_fixed<n,12>*) for synthesizing RF 1 (blue) and RF 2 (orange): (**a**) LUTs and (**b**) Registers.

Table 9. Resource Occupation of Intel/Altera 5CEBA4F23C7N for the BCS with *ap_fixed<24,12>*.

BCS Component	Adaptive Logic Modules (ALMs)	Registers
RF 1	1526 (8%)	1823 (3%)
RF 2	2203 (12%)	3423 (5%)
voting	76 (<1%)	114 (<1%)

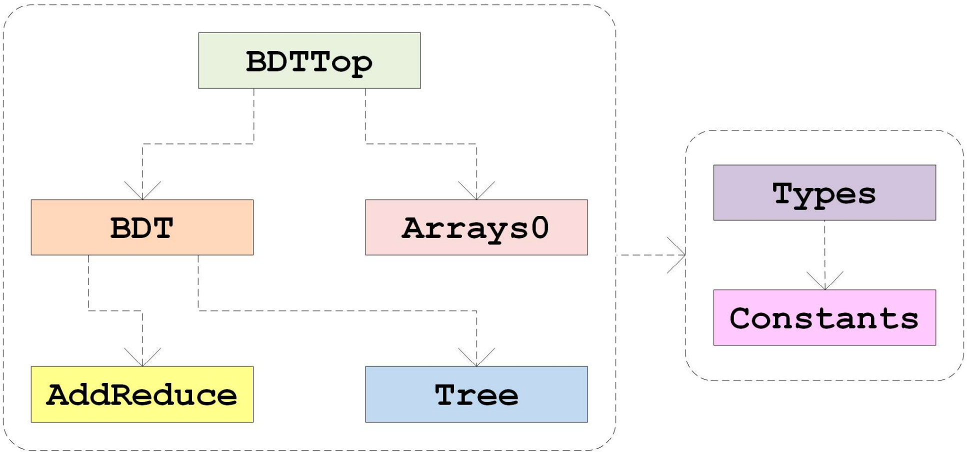


Figure 11. RF Architecture with VHDL entities generated via HLS.

The RF design's top entity is "*BDTTop*", which instantiates two other entities: "*BDT*" (Boosted Decision Tree) and "*Arrays0*". "*BDT*" describes the overall structure of the RF, whereas "*Arrays0*" comprises the numeric parameters that characterize each DT of the RF (e.g., interconnections among nodes, input thresholds for decision-making per node, and outputs per node). "*BDT*", in turn, depends on two other VHDL entities, called "*Tree*" and "*AddReduce*". "*Tree*" represents a DT and is then instantiated three times per BCS RF along with the corresponding parameterization obtained via "*Arrays0*". "*AddReduce*" combines the outputs of each DT and generates the RF ensemble outputs. Finally, all aforementioned entities also depend on the entities "*Constants*" and "*Types*". "*Constants*" provides the top-level hyperparameters of the RF, and both also contain the definition of data types.

By means of a binary comparison of both RFs' VHDL codes, RF 1 and RF 2 only differ in the parameterization within the entities "*Arrays0*" and "*Constants*". In addition to the overlapping of five out of seven entities, the VHDL code of all entities is not overly long or complex, thus allowing all safety assurance activities to be performed straight at the VHDL level, as opposed to adding an intermediate step at the software level.

In activity 5-A, code inspection allowed confirming the correctness of the VHDL code of both RFs and obtaining a set of 790 explainable decision rules that define the behavior of the BCS as a whole: 329 for RF 1 and 461 for RF 2. It is worth noting that these rules were only partially optimized and still have overlapping domains that give room for further summarization. Moreover, the compliance to fault tolerance [31,62] was checked in the entities "*Tree*" and "*AddReduce*", which were chosen due to their core roles within the RFs. The results indicate that, despite coding issues such as unconnected inputs/outputs, feedthrough, unused hardware components, and unnamed processes, these do not pose safety-critical concerns.

Since activity 5-B was performed in advance during Step 4, Step 5 was followed with activity 5-C. It started with analyzing the maximum clock frequencies for each component of Figure 9 (Section 3.3) for the Intel/Altera FPGA Cyclone V 5CEBA4F23C7N after turning off synthesis and routing optimizations on Intel Quartus Prime and considering the most pessimistic scenario of its Timing Analysis tool, called "*SLOW 1100 mV 85C*". The obtained clock frequencies were 101 MHz for *voting*, 38MHz for RF 1, and 24 MHz for RF 2 after full BCS integration. Since the VHDL code generated by Conifer requires a total of "*max_depth + 5*" clock cycles to update its outputs after reading a set of valid inputs, the lower bound constraint of RF 2 would still allow the BCS to meet its response time RSR.

Activity 5-C also included a series of experiments to assess the overall performance of the BCS by integrating both RFs and *voting*. These experiments were carried out in two major steps. The first one was by means of simulations aided by Intel ModelSim, for which the BCS is provided with *DatasetT1*, *DatasetT2*, and *DatasetT3* so that the overall BCS dynamics and performance ratings can be analyzed. The experiments revealed proper functional and safety dynamics, and the performance ratings presented in Table 10 show that, in comparison with the results of each RF (Section 3.4), the BCS reached an average recall that nears that of the highest scoring RF (RF 2), while achieving a precision that is the average of the indices reached by both RFs. Specificity is also significantly high and consistent, at 99.1%.

Table 10. BCS Performance in Holdout Tests with *DatasetT1*, *DatasetT2*, and *DatasetT3*.

Performance Indicator	Results			
	DatasetT1	**DatasetT2**	**DatasetT3**	**Average**
Accuracy	99.2%	99.1%	99.1%	99.1%
Precision	92.8%	92.5%	92.5%	92.6%
Recall	99.3%	99.3%	99.4%	99.4%
f1-score	0.960	0.958	0.958	0.959
Specificity	99.1%	99.1%	99.1%	99.1%

The impact of *pb0* was also assessed by repeating the same simulated experiments with the datasets *DatasetTestZ01* (*pb0* = 1%), *DatasetTestZ10* (*pb0* = 10%), *DatasetTestZ20* (*pb0* = 20%), *DatasetTestZ30* (*pb0* = 30%), *DatasetTestZ40* (*pb0* = 40%), *DatasetTestZ50* (*pb0* = 50%), *DatasetTestZ60* (*pb0* = 60%), *DatasetTestZ70* (*pb0* = 70%), and *DatasetTestZ80* (*pb0* = 80%). The results are plotted in Figure 12 along with the "average" results in Table 10, therein referred to as scenario *pb0* = 89.9%.

Figure 12 shows that recall achieved the smallest drifts with the different *pb0* settings, with a difference no higher than roundabout 0.1% across experiments. Specificity also showed differences of approximately 0.1% from *pb0* = 89.9% up to *pb0* = 20% and experienced an increase of circa 0.2% in the two lowest *pb0* settings (10% and 1%). These results indicate that the BCS has good confidence in effectively classifying data records that are distant from corner cases, i.e., scenarios that certainly require brakes to be applied (recall) and scenarios in which the brakes are certainly not needed (specificity).

On the other hand, accuracy, precision, and f1-score increased more than the other performance metrics as *pb0* decreased. The accuracy growth was fairly constant as *pb0* reduced, peaking at an improvement of roundabout 0.3% from *pb0* = 89.9% up to *pb0* = 1%.

The changes in precision, in turn, were not only greater in value, with an increase of approximately 7.3% from *pb0* = 89.9% up to *pb0* = 1%, but the growth rate also differed with *pb0*, as the precision had a steeper increase when transitioning between two high *pb0* settings than between two *pb0* low settings. Finally, the f1-score followed a trend that is similar to the precision because, since it depends on both precision and recall [68], and the latter remained virtually unchanged: precision dominated its change rate with *pb0*. The behavior of all three performance metrics is expected because, since datasets with lower *pb0* sport more records indicating the need for brakes, and these are known to be typically well classified as per the high recall ratings across all *pb0*, accuracy, precision, and f1-score are also likely to increase.

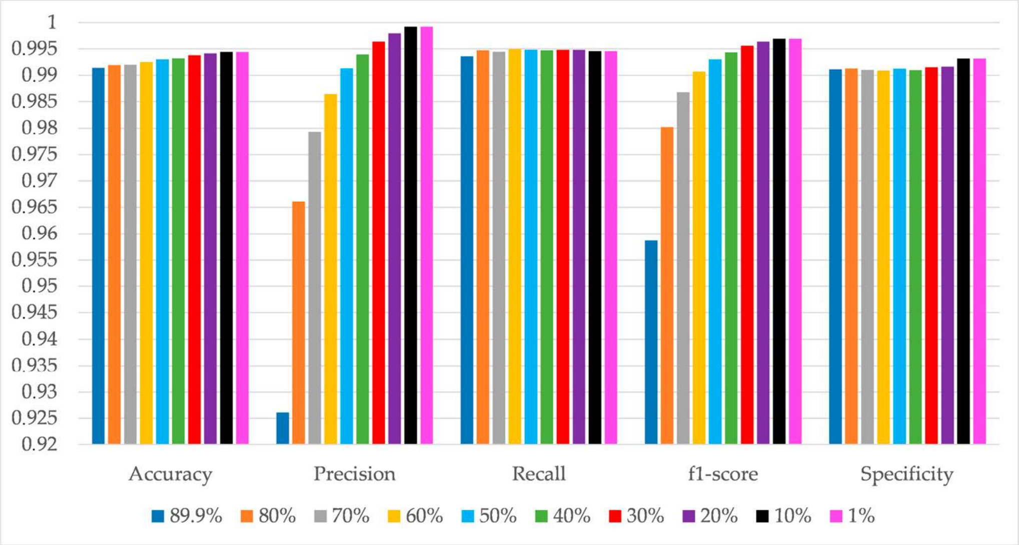


Figure 12. BCS performance in holdout tests with datasets of varying *pb0*.

The second experimental step involved a physical test session with two main objectives: (i.) assessing whether the simulated results are indeed observed in programmed FPGAs and (ii.) evaluating systems-level safety-critical mechanisms unrelated to AI, such as the partial and permanent loss of communication with at least one of the RFs. This was achieved with a built-in test bench whose test scenarios include not only a subset of *DatasetT3* but also fault injection means that emulate communication faults related to the signals *input_valid*, *prob_valid_1*, and *prob_valid_2*.

The physical tests were performed in a terasIC DE0-CV board (with Intel/Altera FPGA Cyclone V 5CEBA4F23C7N), whose programming included the entire BCS and the aforementioned automated test bench, along with Digilent AnalogDiscovery and Digital Waveforms as clock-generating tools. The physical tests' results corroborate those obtained in simulations. It is also worth highlighting that, given the objectives of the physical tests, it is not deemed that the usage of a single FPGA voids the validity of the collected evidence for the distributed architecture of Figure 9 (Section 3.3).

Finally, additional simulations were also performed with the objective of assessing how the BCS performance is affected when it is strictly subject to processing the corner case groups "*Corner Cases* (*0*, *1*)" and '*Corner Cases* (*1*, *0*)". For that purpose, subsets of *DatasetT1*, *DatasetT2*, and *DatasetT3* for each corner case category were generated and utilized as input stimuli for the same test benches utilized in the holdout tests' simulations.

The performance metrics of the corner case simulations are presented in Table 11, which is structured in three columns: the first one for the category "*Corner Cases* (*0*, *1*)", the second one for the category "*Corner Cases* (*1*, *0*)", and the third one for the merging of both categories. For readability purposes, the results therein presented correspond to the weighted averages of the corner cases from the three datasets (*DatasetT1*, *DatasetT2*, and

DatasetT3). Performance metrics signaled as “N/A” (not applicable) correspond to indices that cannot be mathematically calculated due to the features of the corner case categories.

Table 11. BCS Performance with Corner Case Partitions of *DatasetT1*, *DatasetT2*, and *DatasetT3*.

Performance Indicator		**Corner Cases Average Results**	
	(0, 1)	**(1, 0)**	**All**
Accuracy	9.4%	75.7%	43.8%
Precision	0.0%	100.0%	47.3%
Recall	N/A	75.7%	75.7%
f1-score	N/A	0.862	0.583
Specificity	9.4%	N/A	9.4%

The indices presented in Table 10 are highly suggestive of potentially unsafe behavior for the BCS when processing input tuples (*d*, *s*, *gd*) that are near the boundaries of either applying or releasing brakes. On the one hand, the low specificity with *Corner Cases* (*0*, *1*) indicates that brakes might be frequently applied when they are still not necessary. On the other hand, the recall with *Corner Cases* (*1*, *0*) suggests that the probability of not applying brakes immediately when they are needed (24.3%) is nearly 40 times higher than that of the overall scenarios of Table 10 (0.6%).

Taking into account that assessing whether the previous indices indeed translate into unsafe behavior depends on how frequently the BCS is requested, a quantitative safety analysis is performed in advance to assess whether the BCS is indeed unsafe. Since this analysis does not take into account the unsafe events stemming from hardware failure modes, as these would only be assessed in Step 6, the quantitative safety model herein explored the lower bounds of the unsafe failure rates of each SR with the systematic unsafe behavior of AI in misclassifying *bcp*.

Two unsafe failure rates are relevant for safety assurance purposes:

1. *Unsafe AI Failure Rate for Not Applying Brakes* ($\lambda_{U,AI,\ no\_brake}$): This failure rate, expressed by Equation (2), indicates how frequently the BCS misses applying brakes when needed. It is needed, thus, to evaluate the BCS compliance to SR 4.1;
2. *Unsafe AI Failure Rate for Applying Unnecessary Brakes* ($\lambda_{U,AI,unnec\_brake}$): This failure rate, described by Equation (3), quantifies how frequently the BCS applies brakes when they are not needed. It is important, thus, to evaluate whether SR 4.2 is met.

$$\lambda_{U,AI,\ no\_brake} = n_{O,h} \cdot (1 - recall)^{(n_{err,tol}+1)} \tag{2}$$

$$\lambda_{U,AI,\ no\_brake} = n_{O,h} \cdot (1 - specificity)^{(n_{err,tol}+1)} \tag{3}$$

In both equations, “recall” and “specificity” refer to the homonymous performance metrics at a given operational scenario. Moreover, the parameter $n_{err,tol}$ indicates the number of wrong brake control predictions (*bcp*) that is tolerated by the BCS at each operating hour. This parameter was added to Equations (2) and (3) in such a way that, if the failure rates $\lambda_{U,AI,\ no\_brake}$ and $\lambda_{U,AI,unnec\_brake}$ do not meet the safety requirements SR 4.1 and SR 4.2 if $n_{err,tol}$ = 0, it is possible to assess to what extent a higher $n_{err,tol}$ would be acceptable from a risk perspective. Finally $n_{O,h}$ indicates the number of outputs generated by the BCS per hour. It equals the inverse of its 500 ms processing cycle, i.e., 7200 outputs per hour.

Therefore, Equations (2) and (3) were initially exercised by making $n_{err,tol}$ = 0 and utilizing the recall and specificity figures of all *pb0* and corner case scenarios depicted in Figure 12 and Table 11. Neither case allowed achieving $\lambda_{U,AI,\ no\_brake}$ and $\lambda_{U,AI,unnec\_brake}$ lower than the corresponding target THRs of SR 4.1 and SR 4.2, respectively. This result corroborates the initial remarks raised during the corner case analyses.

The minimum figures of $n_{err,tol}$ that allow meeting the target THRs of both SR 4.1 and SR 4.2 are presented in Table 12, along with their corresponding unsafe failure rates

$\lambda_{U,AI,\ no\_brake}$ and $\lambda_{U,AI,unnec\_brake}$. They indicate that the BCS is potentially safe only if at least six classification errors of each type are tolerated per operating hour regardless of *pb0*. A minor exception occurs for SR 4.2 and *pb0* = 1%, for which the target THR is achieved by tolerating one less error.

Table 12. Final Results of $n_{err,tol}$, λ_{U,AI,no_brake}, and $\lambda_{U,AI,unnec\_brake}$ at each Operational Scenario.

Operational Scenario	**SR 4.1**		**SR 4.2**	
	$n_{err,tol}$	$\lambda_{U,AI,\ no\_brake}$	$n_{err,tol}$	$\lambda_{U,AI,unnec\_brake}$
pb0 = 89.9%—Only Corner Cases	20	3.69178×10^{-9}	253	9.62687×10^{-9}
pb0 = 89.9%—All Data Records	6	4.77109×10^{-10}	6	3.52228×10^{-9}
pb0 = 80.0%	6	1.48530×10^{-10}	6	3.18352×10^{-9}
pb0 = 70.0%	6	2.05412×10^{-10}	6	3.83331×10^{-9}
pb0 = 60.0%	6	1.15057×10^{-10}	6	4.10625×10^{-9}
pb0 = 50.0%	6	1.37022×10^{-10}	6	3.26103×10^{-9}
pb0 = 40.0%	6	1.48530×10^{-10}	6	3.91335×10^{-9}
pb0 = 30.0%	6	1.40232×10^{-10}	6	2.70740×10^{-9}
pb0 = 20.0%	6	1.41257×10^{-10}	6	2.49192×10^{-9}
pb0 = 10.0%	6	1.80814×10^{-10}	6	7.17302×10^{-10}
pb0 = 1.0%	6	3.11367×10^{-10}	5	2.85732×10^{-8}

Finally, in a pessimistic operational scenario in which the BCS is strictly subject to corner case data, there is a significant leap in $n_{err,tol}$. At least 20 hourly classification errors of not applying brakes when needed are tolerated to meet SR 4.1, whereas a minimum of 253 misapplications of brakes per hour are tolerated to accomplish SR 4.2.

Given that the thresholds of $n_{err,tol}$ are not deemed acceptable from a safety perspective, notably considering that Newtonian-based BCSs implemented in commercial train control systems that adhere to IEEE 1474-1:2004 [64] are able to achieve their safety goals without tolerating faults on braking commands, the AI-based BCS explored in this case study is deemed unsafe. This closes the iteration of the Safety ArtISt method and triggers the assessment of the safety issues' root causes so that the design is reviewed in a new iteration. This discussion is covered, along with other themes, in Section 4.

4. Discussion

The BCS case study allowed exercising the Safety ArtISt method's capability in guiding the lifecycle of a system with safety-critical AI embedded in FPGAs by means of a safety-assurance-oriented approach. The workflow and results detailed throughout Section 3 exemplify the importance of the Safety ArtISt method, along with the dynamics of its steps, activities, and recommended practices, in iteratively building evidence-based arguments that allow ascertaining whether a system meets its qualitative and quantitative safety targets with the aid of analyses, simulations, and physical experiments. Hence, it goes beyond pre-existing research efforts involving safety-critical systems with FPGA-based AI, whose focus is on detailing AI design and functional performance while falling short of demonstrating full compliance with safety requirements that take into account both operational and long-term fault tolerance aspects [42–47].

Specifically with regard to AI design with FPGAs, the Safety ArtISt method unveiled the importance of design tradeoffs that are usually not in hindsight for safety practitioners. Since AI base models are based on models that rely on numeric processing, the representation of these numbers within the FPGA-implemented AI plays an important role in how accurately the calculations are performed, thus affecting both performance and safety. As a result, the numeric precision is selected in such a way as to minimize negative collateral impact over safety while remaining within reasonable limits for synthesis at a target FPGA that satisfies the end application cost and performance constraints.

Given that software-based implementations boast significantly high numeric precision, they can be utilized as a reference for optimizing the settings for their FPGA-targeted versions. This becomes an easier and even more natural path if HLS is used, as well-established AI software models are mandatorily built beforehand. For instance, if fixed-point precision is chosen, the recommended practice is that the bit width is increased in such a way that calculation errors are minimized by comparing the results of holdout tests carried out both in software and in programmable hardware. In the BCS case study, the results presented in Section 3.5 (Table 8) evidence that the discrepancies among reference implementations at software and their VHDL-translated counterparts were sharply reduced by increasing the size of fixed-point precision decimal part from 6 to 12 bits while keeping the integer part at 12 bits due to the domain of variables. With 12 bits for decimal representation, the VHDL-coded AI led to the very same outputs predicted at the software level, and the underlying probabilities that back these results up deviated circa 0.2% from the software results at the 75th percentile of the entire holdout test dataset. The needed resources for synthesizing the BCS FPGAs experienced a linear increase with the better decimal resolution—still well within the limits for the target FPGAs.

The Safety ArtISt guidelines in HLS have also been evidenced as a relevant contribution towards building safety-critical systems with FPGA-based AI and justifying whether they are safe. In addition to the aforementioned benefits of supporting the fine-tuning of numeric representation on FPGAs, HLS also allows a quicker assessment of different AI base models that are preliminarily feasible for the project design, so that the most promising alternatives are indeed selected for an in-depth, hardware-oriented design. Once HLS tools such as Conifer, utilized in the BCS case study, translate projects coded in software with well-established AI libraries (e.g., Python's *scikit-learn*) to HDLs, prototyping the safety-critical can be performed in a rather straightforward way.

On the BCS case study, six different AI base models were initially considered for the design: kNN, DT, SVM, ANN, RF, and ensembles of components of different types. The HLS allowed exercising all alternatives in the early design, and DTs and RFs were chosen as the preferable candidates for the detailed FPGA-oriented hardware design due to their overall better performance and intrinsic explainability. The better performance achieved with DTs and RFs when compared with the other candidates translates into a smaller exposure rate of the BCS to the two types of unsafe scenarios the BCS is subject to, i.e., not applying brakes when necessary and applying brakes excessively when unnecessary.

Finally, the BCS case study also revealed a relevant tradeoff between the frequency at which an AI-based safety-critical function is requested and the performance that is achieved by the safety-critical AI. This relationship is that, given a target SIL/THR, the higher the request rate of a safety function is, the further the performance requirements of safety-critical AI are pushed.

Since the studied BCS is demanded rather intensively, at a rate of one new output per 500 ms, not even the high performance ratings of the BCS allowed for meeting its safety requirements. Despite the BCS boasting at least 99.4% recall, 99.1% specificity, 99.1% accuracy, and 92.6% precision, it would only meet its safety requirements if at least six braking classification errors of each type were acceptable per operation hour.

Furthermore, the situation becomes even more challenging considering the significant performance drop when the BCS is subject to corner case inputs. For these, the BCS reaches only 75.7% recall, 47.3% precision, 43.8% accuracy, and 9.4% specificity, leading to a pessimistic scenario at which 20 improper lacks of braking commands and 203 unduly brake applications per hour should be tolerated so as to accomplish the SRs.

The previous results indicate not only a subpar performance when compared with non-AI, Newtonian-kinematics-based solutions utilized for the same purpose in commercial train control systems [64] but also a major conclusion on using safety-critical AI—be it within FPGAs or processor-based.

The usage of AI for safety-critical functions with high SIL/THR targets and high demand might not be even technically feasible, given that an exceedingly high performance may either not be achievable, or may be attainable without proper generalization due to e.g., overfitting to specific scenarios. It is also worth noting that the BCS case study with Safety ArtISt allowed reaching this general conclusion with evidence collected early in the V&V process, thus saving resources in exhaustive and time-consuming activities that had not been performed up until the detection point (e.g., effects of hardware failure modes, exhaustive compliance to fault tolerance, and formal AI reachability).

The design and the safety analysis of AI-based BCSs were also covered in the recent literature. Most references focus on BCSs relying on neural networks verified by means of reachability analyses and miss the demonstration of compliance to long-term quantitative safety targets. In the research papers by Tran et al. [74] and Xiang et al. [75], BCSs relying on reinforcement learning and deep neural networks were formally analyzed and led to conclusions that supported the BCSs provided a safe-by-design response given a time window before the brakes were applied (e.g., 5 s). Socha et al. [76], in turn, utilized a safe envelope not based on AI to cover wrong braking commands generated by neural networks. Finally, Cleaveland et al. [77] identified that their neural network BCS remained formally safe only if specific mathematical properties were held in the target application.

Future work stemming from this research has two main avenues. The first one comprises potential attempts to circumvent the safety issues of the BCS designed in Section 3 while retaining the use of AI. These include, but are not limited to, utilizing deterministic safety envelopes [1], adding other AI elements other than the two current RFs, and, if successful, augmenting data collection and testing strategies with a physical prototype. This prototype would involve installing sensors on a vehicle and exercising forward movements as per Figure 5 with initial distances and speeds chosen at random within their domains. The second one is general to the Safety ArtISt method and involves exercising it with other AI base models and in other application domains (e.g., medical systems, safety-critical supervision, and other vehicle control functions).

5. Conclusions

The objectives of this paper were to present the Safety ArtISt method and illustrate its application in a safety-critical Braking Control System (BCS) for autonomous vehicles whose safety-critical functions rely on AI implemented in FPGAs. These objectives were defined based on four high-level goals that outline the expected contributions of the Safety ArtISt research project as a whole:

1. Showing the potential of the Safety ArtISt method in being a cost-effective, practitioner-oriented safety assurance lifecycle for AI-based systems that fills gaps of pre-existing safety assurance methods in the literature [1];
2. Defining safety assurance activities for AI-based systems and illustrating their application and relevance for safety-critical AI relying on FPGAs;
3. Building quantitative safety models for AI-based systems and analyzing them;
4. Drawing conclusions as to what extent safety-critical AI can be utilized with regard to AI specificities and the safety-critical application mission profile.

From a conceptual standpoint, contributions 1 and 2 stem from Safety ArtISt clearly identifying recommended design and V&V practices for several AI variant methods, thus going beyond technology-agnostic safety assurance approaches for AI-based systems, such as the ANSI UL4600:2020 [34], as well as research efforts focused on specific application domains [8,37,40] or AI variants [36–41]. Moreover, its cost-effectiveness is reflected in the subdivision of its V&V activities into two steps: "*AI Preliminary V&V*", which allows for identifying blocking safety issues with simplified yet important analyses of AI performance and fault tolerance sampling, and "*AI Detailed V&V*", whose costly, exhaustive analyses are triggered only if the "*AI Preliminary V&V*" is successful. This subdivision is not explored in other existing safety assurance methods [8,36–41].

The BCS case study also played a relevant role in supporting the novelty related to items 1 and 2. In order to build the BCS as a system with two FPGAs, each of which synthesized with HLS featuring an RF with three different DTs, along with a majority voting CPLD, several practices from the Safety ArtISt were instantiated in detail. These involved (i.) building, parameterizing, and analyzing datasets for training and testing; (ii.) analyzing and choosing AI variants for the safety-critical systems; (iii.) systematizing a framework for a safe HLS; (iv.) incorporating fault-tolerant design practice; and (v.) assessing AI performance in regular operation and in corner cases. Specifically for FPGAs, it was possible to analyze the role played by numeric representation (e.g., fixed-point precision) in improving safety while meeting FPGA cost and functional constraints, as well as the iterative refinement of FPGA design when using HLS.

The results obtained in the case study are directly mapped to contributions 3 and 4. The quantitative safety modeling of the BCS reveals that, despite overall high performance ratings (at least 99.4% recall, 99.1% specificity, 99.1% accuracy, and 92.6% precision), the BCS does not meet its quantitative safety requirements due to the high rate at which the safety-critical functions are requested (once every 500 ms). This result leads to a general conclusion that the usage of AI in safety-critical systems depends on a tradeoff between the safety-critical function targets and their mission profile, regardless of FPGA or processor-based AI implementations. For safety-critical systems of high demand, the performance goals for AI might be exceedingly high to the point of making its usage unattainable or at least severely restricting it for specific scenarios, given the potential lack of generalization caused by, e.g., overfitting to reach high performance. It is also worth noting that these conclusions were all reached during the BCS "*AI Preliminary V&V*", which reinforces the cost-effectiveness of the Safety ArtISt method in an early unveiling of safety issues, as per contribution 1.

Finally, future work related to improving the BCS design and applying the Safety ArtISt in case studies with different AI base models and application domains was also discussed.

Author Contributions: Conceptualization, A.V.S.N. and P.S.C.; methodology, A.V.S.N., H.L.S. and P.S.C.; software, A.V.S.N. and H.L.S.; validation, A.V.S.N., J.B.C.J., J.R.A.J. and P.S.C.; formal analysis, A.V.S.N. and H.L.S.; investigation, A.V.S.N. and H.L.S.; resources, A.V.S.N., H.L.S. and P.S.C.; data curation, A.V.S.N. and H.L.S.; writing—original draft preparation, A.V.S.N.; writing—review and editing, H.L.S., J.B.C.J., J.R.A.J. and P.S.C.; visualization, A.V.S.N.; supervision, A.V.S.N. and P.S.C.; project administration, A.V.S.N. and P.S.C.; funding acquisition, A.V.S.N., H.L.S., J.B.C.J., J.R.A.J. and P.S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This study was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil* (CAPES)—Finance Code 001 (PROEX scholarship—grant number 88887.513631/2020-00), in part by the *Fundação para o Desenvolvimento Tecnológico da Engenharia—Brasil* (FDTE) (grant number 1954.01.20), and in part by the *Conselho Nacional de Desenvolvimento Científico e Tecnológico—Brazil* (CNPq) (grant number 121228/2022-3).

Data Availability Statement: All data, codes, and detailed analyses referred to in this research paper are available in both the GitHub repository of the Safety ArtISt project [66] and the technical research report [65].

Acknowledgments: The authors thank the Digital Laboratory (*Laboratório Digital*) team of the Computer Engineering and Digital Systems Department of *Universidade de São Paulo* (USP)—*Escola Politécnica* (Poli-USP) for loaning the material for the physical experiments of this research. Special thanks go to the Digital Laboratory coordinator, Edson Toshimi Midorikawa, and to the technicians Maria de Fátima Salgueiro and Daniel Costa Ferreira.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Silva Neto, A.V.; Camargo, J.B., Jr.; Almeida, J.R., Jr.; Cugnasca, P.S. Safety Assurance of Artificial Intelligence-Based Systems: A Systematic Literature Review on the State of the Art and Guidelines for Future Work. *IEEE Access* **2022**, *10*, 130733–130770. [CrossRef]
2. McDermid, J.; Jia, Y.; Habli, I. Towards a Framework for Safety Assurance of Autonomous Systems. In Proceedings of the CEUR Workshop Proceedings, Macao, China, 11–12 August 2019; Volume 2419. Available online: https://ceur-ws.org/Vol-2419/paper_2.pdf (accessed on 20 October 2023).
3. Habli, I.; Lawton, T.; Porter, Z. Artificial Intelligence in Health Care: Accountability and Safety. *Bull. World Health Organ.* **2020**, *98*, 251–256. [CrossRef] [PubMed]

4. *ISO/IEC61508:2010*; Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems (7 Parts). IEC: Geneve, Switzerland, 2010.
5. *EN50126-1:2017*; Railway Applications—The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)—Part 1: Generic RAMS Process. CENELEC: Brussels, Belgium, 2017.
6. *EN50129:2018*; Railway Applications—Communication, Signalling and Processing Systems—Safety-Related Electronic Systems for Signalling. CENELEC: Brussels, Belgium, 2018.
7. RTCA. *DO-254—Design Assurance Guidance for Airborne Electronic Hardware*; RTCA: Washington, DC, USA, 2000.
8. Häring, I.; Lüttner, F.; Frorath, A.; Fehling-Kaschek, M.; Ross, K.; Schamm, T.; Knoop, S.; Schmidt, D.; Schmidt, A.; Ji, Y.; et al. Framework for Safety Assessment of Autonomous Driving Functions up to SAE Level 5 by Self-Learning Iteratively Improving Control Loops between Development, Safety and Field Life Cycle Phases. In Proceedings of the 17th IEEE International Conference on Intelligent Computer Communication and Processing, ICCP 2021, Cluj-Napoca, Romania, 28–30 October 2021; pp. 33–40. [CrossRef]
9. Koopman, P.; Ferrell, U.; Fratrik, F.; Wagner, M. A Safety Standard Approach for Fully Autonomous Vehicles. In Proceedings of the Computer Safety, Reliability, and Security, SAFECOMP 2019, Turku, Finland, 11–13 September 2019; Volume 11699 LNCS, pp. 326–332. [CrossRef]
10. Pedroza, G.; Adedjouma, M. Safe-by-Design Development Method for Artificial Intelligent Based Systems. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, Lisbon, Portugal, 10–12 July 2019; Volume 2019, pp. 391–397. [CrossRef]
11. Pereira, A.; Thomas, C. Challenges of Machine Learning Applied to Safety-Critical Cyber-Physical Systems. *Mach. Learn. Knowl. Extr.* **2020**, *2*, 579–602. [CrossRef]
12. Tarrisse, A.; Masse, F. Locks for the Use of IEC 61508 to ML Safety-Critical Applications and Possible Solutions. In Proceedings of the 31st European Safety and Reliability Conference, ESREL 2021, Angers, France, 19–23 September 2021; pp. 3459–3466. [CrossRef]
13. Čaušević, A.; Papadopoulos, A.V.; Sirjani, M.; Auevi, A. Towards a Framework for Safe and Secure Adaptive Collaborative Systems. In Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (CompSAC), Västerås, Sweden, 15–19 July 2019; Volume 2, pp. 165–170. [CrossRef]
14. Javed, M.A.; Muram, F.U.; Hansson, H.; Punnekkat, S.; Thane, H. Towards Dynamic Safety Assurance for Industry 4.0. *J. Syst. Archit.* **2021**, *114*, 101914. [CrossRef]
15. Schöning, J.; Pfisterer, H.J. Safe and Trustful AI for Closed-Loop Control Systems. *Electronics* **2023**, *12*, 3489. [CrossRef]
16. Kuutti, S.; Bowden, R.; Joshi, H.; de Temple, R.; Fallah, S. Safe Deep Neural Network-Driven Autonomous Vehicles Using Software Safety Cages. In Proceedings of the 20th International Conference on Intelligent Data Engineering and Automated Learning, IDEAL 2019, Guildford, UK, 14–16 November 2019; Volume 11872 LNCS, pp. 150–160. [CrossRef]
17. Zhao, H.; Zeng, X.; Chen, T.; Liu, Z. Synthesizing Barrier Certificates Using Neural Networks. In Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control (HSCC'20), New York, NY, USA, 22–24 April 2020. [CrossRef]
18. Wang, Z.; Huang, C.; Zhu, Q. Efficient Global Robustness Certification of Neural Networks via Interleaving Twin-Network Encoding. In Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe (DATE '22), Leuven, Belgium, 14–23 March 2022; pp. 1087–1092. [CrossRef]
19. Phan, D.T.; Grosu, R.; Jansen, N.; Paoletti, N.; Smolka, S.A.; Stoller, S.D. Neural Simplex Architecture. In Proceedings of the 12th International Symposium on NASA Formal Methods (NFM 2020), Moffett Field, CA, USA, 11–15 May 2020; Volume 12229 LNCS, pp. 97–114. [CrossRef]
20. Chen, S.; Sun, Y.; Li, D.; Wang, Q.; Hao, Q.; Sifakis, J. Runtime Safety Assurance for Learning-Enabled Control of Autonomous Driving Vehicles. In Proceedings of the 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 23–27 May 2022; IEEE Press: Piscataway, NJ, USA, 2022; pp. 8978–8984. [CrossRef]
21. Peng, Y.; Tan, G.; Si, H.; Li, J. DRL-GAT-SA: Deep Reinforcement Learning for Autonomous Driving Planning Based on Graph Attention Networks and Simplex Architecture. *J. Syst. Archit.* **2022**, *126*, 102505. [CrossRef]
22. Wang, Q.; Kou, G.; Chen, L.; He, Y.; Cao, W.; Pu, G. Runtime Assurance of Learning-Based Lane Changing Control for Autonomous Driving Vehicles. *J. Circuits Syst. Comput.* **2022**, *31*, 2250249. [CrossRef]
23. Salay, R.; Angus, M.; Czarnecki, K. A Safety Analysis Method for Perceptual Components in Automated Driving. In Proceedings of the 2019 IEEE 30th International Symposium on Software Reliability Engineering, ISSRE, Berlin/Heidelberg, Germany, 28–31 October 2019; pp. 24–34. [CrossRef]
24. Nahata, R.; Omeiza, D.; Howard, R.; Kunze, L. Assessing and Explaining Collision Risk in Dynamic Environments for Autonomous Driving Safety. In Proceedings of the 2021 IEEE Conference on Intelligent Transportation Systems, ITSC, Indianapolis, IN, USA, 19–22 September 2021; pp. 223–230. [CrossRef]
25. Lin, X.; Zhu, H.; Samanta, R.; Jagannathan, S. ART: Abstraction Refinement-Guided Training for Provably Correct Neural Networks. In Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design, FMCAD 2020, Haifa, Israel, 21–24 September 2020; pp. 148–157. [CrossRef]

26. Shukla, D.; Lal, R.; Hauptman, D.; Keshmiri, S.; Prabhakar, P.; Beckage, N. Flight Test Validation of a Safety-Critical Neural Network Based Longitudinal Controller for a Fixed-Wing UAS. In Proceedings of the AIAA Aviation 2020 Forum, Lawrence, MA, USA, 15–19 June 2020; pp. 1–15. [CrossRef]
27. Lazarus, C.; Lopez, J.G.; Kochenderfer, M.J. Runtime Safety Assurance Using Reinforcement Learning. In Proceedings of the 2020 39th AIAA/IEEE Digital Avionics Systems Conference (DASC), San Antonio, TX, USA, 11–15 October 2020; pp. 1–9. [CrossRef]
28. Doppelbauer, J. Command and Control 4.0. *IRSE News*. 2018, pp. 2–9. Available online: https://www.era.europa.eu/content/command-and-control-40_en (accessed on 20 October 2023).
29. Liu, J.; Zhang, Y.; Han, J.; He, J.; Sun, J.; Zhou, T. Intelligent Hazard-Risk Prediction Model for Train Control Systems. *IEEE Trans. Intell. Transp. Syst.* **2020**, *21*, 4693–4704. [CrossRef]
30. Zhu, Q.; Li, W.; Kim, H.; Xiang, Y.; Wardega, K.; Wang, Z.; Wang, Y.; Liang, H.; Huang, C.; Fan, J.; et al. Know the Unknowns: Addressing Disturbances and Uncertainties in Autonomous Systems. In Proceedings of the 39th IEEE/ACM International Conference on Computer-Aided Design, (ICCAD'20), Evanston, IL, USA, 5–8 November 2020. [CrossRef]
31. Silva Neto, A.V.; Vismari, L.F.; Gimenes, R.A.V.; Baraldi Sesso, D.; Almeida, J.R., Jr.; Cugnasca, P.S.; Camargo, J.B., Jr. A Practical Analytical Approach to Increase Confidence in PLD-Based Systems Safety Analysis. *IEEE Syst. J.* **2017**, *12*, 3473–3484. [CrossRef]
32. Mao, N.; Yang, H.; Huang, Z. A Parameterized Parallel Design Approach to Efficient Mapping of CNNs onto FPGA. *Electronics* **2023**, *12*, 1106. [CrossRef]
33. Yang, C. FPGA in IoT Edge Computing and Intelligence Transportation Applications. In Proceedings of the 2021 IEEE International Conference on Robotics, Automation and Artificial Intelligence (RAAI), Hong Kong, China, 21–23 April 2021; pp. 78–82. [CrossRef]
34. Ferrell, U.D.; Anderegg, A.H.A. Applicability of UL 4600 to Unmanned Aircraft Systems (UAS) and Urban Air Mobility (UAM). In Proceedings of the 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC), San Antonio, TX, USA, 11–16 October 2020; pp. 1–7. [CrossRef]
35. Dey, S.; Lee, S.-W. Multilayered Review of Safety Approaches for Machine Learning-Based Systems in the Days of AI. *J. Syst. Softw.* **2021**, *176*, 110941. [CrossRef]
36. Salay, R.; Czarnecki, K. Improving ML Safety with Partial Specifications. In Proceedings of the Computer Safety, Reliability, and Security, SAFECOMP 2019, Turku, Finland, 11–13 September 2019; Volume 11699 LNCS, pp. 288–300. [CrossRef]
37. Mock, M.; Scholz, S.; Blank, F.; Huger, F.; Rohatschek, A.; Schwarz, L.; Stauner, T. An Integrated Approach to a Safety Argumentation for AI-Based Perception Functions in Automated Driving. In Proceedings of the Computer Safety, Reliability, and Security. SAFECOMP 2021 Workshops, York, UK, 8–10 September 2021; Volume 12853 LNCS, pp. 265–271. [CrossRef]
38. Kurd, Z.; Kelly, T.; Austin, J. Developing Artificial Neural Networks for Safety Critical Systems. *Neural Comput. Appl.* **2007**, *16*, 11–19. [CrossRef]
39. Aslansefat, K.; Sorokos, I.; Whiting, D.; Tavakoli Kolagari, R.; Papadopoulos, Y. SafeML: Safety Monitoring of Machine Learning Classifiers Through Statistical Difference Measures. In Proceedings of the International Symposium on Model-Based Safety and Assessment (IMBSA), Lisbon, Portugal, 14–16 September 2020; Volume 12297 LNCS, pp. 197–211. [CrossRef]
40. Koopman, P.; Wagner, M. Toward a Framework for Highly Automated Vehicle Safety Validation. In Proceedings of the WCX World Congress Experience, Detroit, MI, USA, 10–12 April 2018; pp. 1–12. [CrossRef]
41. Douthwaite, M.; Kelly, T. Establishing Verification and Validation Objectives for Safety-Critical Bayesian Networks. In Proceedings of the 28th IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW 2017, Toulouse, France, 23–26 October 2017; pp. 302–309. [CrossRef]
42. Peng, J.; Tian, L.; Jia, X.; Guo, H.; Xu, Y.; Xie, D.; Luo, H.; Shan, Y.; Wang, Y. Multi-Task ADAS System on FPGA. In Proceedings of the 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Hsinchu, Taiwan, 18–20 March 2019; pp. 171–174. [CrossRef]
43. Hamdaoui, F.; Bougharriou, S.; Mtibaa, A. Optimized Hardware Vision System for Vehicle Detection Based on FPGA and Combining Machine Learning and PSO. *Microprocess. Microsyst.* **2022**, *90*, 104469. [CrossRef]
44. Li, T.; Ma, Y.; Shen, H.; Endoh, T. FPGA Implementation of Real-Time Pedestrian Detection Using Normalization-Based Validation of Adaptive Features Clustering. *IEEE Trans. Veh. Technol.* **2020**, *69*, 9330–9341. [CrossRef]
45. Du, B.; Azimi, S.; De Sio, C.; Bozzoli, L.; Sterpone, L. On the Reliability of Convolutional Neural Network Implementation on SRAM-Based FPGA. In Proceedings of the 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Noordwijk, The Netherlands, 2–4 October 2019; pp. 1–6. [CrossRef]
46. Mahesh, M.; Nalesh, S.N.; Kala, S.K. Bandwidth-Efficient Sparse Matrix Multiplier Architecture for Deep Neural Networks on FPGA. In Proceedings of the 2021 IEEE 34th International System-on-Chip Conference (SOCC), Las Vegas, NV, USA, 14–17 September 2021; pp. 7–12. [CrossRef]
47. Cheng, Q.; Huang, M.; Man, C.; Shen, A.; Dai, L.; Yu, H.; Hashimoto, M. Reliability Exploration of System-on-Chip with Multi-Bit-Width Accelerator for Multi-Precision Deep Neural Networks. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 3978–3991. [CrossRef]
48. Nagarale, S.D.; Patil, B.P. RTL Verification and FPGA Implementation of Generalized Neural Networks: A High-Level Synthesis Approach. In Proceedings of the International Conference on Mobile Computing and Sustainable Informatics (ICMCSI), Lalitpur, Nepal, 27–28 January 2022; Volume 126, pp. 447–462. [CrossRef]

49. Fazlyab, M.; Morari, M.; Pappas, G.J. Safety Verification and Robustness Analysis of Neural Networks via Quadratic Constraints and Semidefinite Programming. *IEEE Trans. Automat. Contr.* **2022**, *67*, 1–15. [CrossRef]
50. Gharib, M.; Zoppi, T.; Bondavalli, A. On the Properness of Incorporating Binary Classification Machine Learning Algorithms into Safety-Critical Systems. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 1671–1686. [CrossRef]
51. Hartsell, C.; Ramakrishna, S.; Dubey, A.; Stojcsics, D.; Mahadevan, N.; Karsai, G. ReSonAte: A Runtime Risk Assessment Framework for Autonomous Systems. In Proceedings of the 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2021, Madrid, Spain, 18–21 May 2021; pp. 118–129. [CrossRef]
52. Boulineau, J.F. Safe Recognition A.I. of a Railway Signal by on-Board Camera. In Proceedings of the 16th European Dependable Computing Conference—Workshops, EDCC 2020 Workshops, Munich, Germany, 7 September 2020; Volume 1279-CCIS, pp. 5–19. [CrossRef]
53. Jia, Y.; McDermid, J.; Lawton, T.; Habli, I. The Role of Explainability in Assuring Safety of Machine Learning in Healthcare. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 1746–1760. [CrossRef]
54. Aoki, T.; Kawakami, D.; Chida, N.; Tomita, T. Dataset Fault Tree Analysis for Systematic Evaluation of Machine Learning Systems. In Proceedings of the IEEE Pacific Rim International Symposium on Dependable Computing, PRDC, Perth, Australia, 1–4 December 2020; pp. 100–109. [CrossRef]
55. Ruchkin, I.; Cleaveland, M.; Sokolsky, O.; Lee, I. Confidence Monitoring and Composition for Dynamic Assurance of Learning-Enabled Autonomous Systems. In Proceedings of the Formal Methods in Outer Space—Workshop during the 9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, Rhodes, Greece, 17–29 October 2021; Volume 13065 LNCS, pp. 137–146. [CrossRef]
56. Schwalbe, G.; Knie, B.; Samann, T.; Dobberphul, T.; Gauerhof, L.; Raafatnia, S.; Rocco, V. Structuring the Safety Argumentation for Deep Neural Network Based Perception in Automotive Applications. In Proceedings of the Computer Safety, Reliability, and Security. SAFECOMP 2020, Lisbon, Portugal, 16–18 September 2020; Volume 12235, pp. 383–394. [CrossRef]
57. Rajabli, N.; Flammini, F.; Nardone, R.; Vittorini, V. Software Verification and Validation of Safe Autonomous Cars: A Systematic Literature Review. *IEEE Access* **2021**, *9*, 4797–4819. [CrossRef]
58. Ferlini, F.; Viel, F.; Seman, L.O.; Pettenghi, H.; Bezerra, E.A.; Leithardt, V.R.Q. A Methodology for Accelerating FPGA Fault Injection Campaign Using ICAP. *Electronics* **2023**, *12*, 807. [CrossRef]
59. Fazlyab, M.; Morari, M.; Pappas, G.J. Probabilistic Verification and Reachability Analysis of Neural Networks via Semidefinite Programming. In Proceedings of the 2019 IEEE 58th Conference on Decision and Control (CDC), Nice, France, 11–13 December 2019; pp. 2726–2731. [CrossRef]
60. Wu, H.; Lv, D.; Cui, T.; Hou, G.; Watanabe, M.; Kong, W. SDLV: Verification of Steering Angle Safety for Self-Driving Cars. *Form. Asp. Comput.* **2021**, *33*, 325–341. [CrossRef]
61. Zhao, Q.; Chen, X.; Zhang, Y.; Sha, M.; Yang, Z.; Lin, W.; Tang, E.; Chen, Q.; Li, X. Synthesizing ReLU Neural Networks with Two Hidden Layers as Barrier Certificates for Hybrid Systems. In Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control (HSCC'21), Nanjing, China, 19–21 May 2021; pp. 1–11. [CrossRef]
62. Users Group (Team Primary Author: Michelle Lange). Position Paper DO254-UG-001—Best Practice VHDL Coding Standards for DO-254 Programs. 2010. Available online: https://pt.scribd.com/doc/151230668/Best-Practice-VHDL-Coding-Standards-for-DO-254-Programs (accessed on 20 October 2023).
63. Intel. Intel Quartus Prime Pro Edition User Guide—Timing Analyzer—Rev. 2022.09.26. 2022. Available online: https://cdrdv2-public.intel.com/774741/ug-683243-774741.pdf (accessed on 20 October 2023).
64. *IEEE1474.1:2004*; IEEE Standard for Communications-Based Train Control (CBTC) Performance and Functional Requirements. IEEE: New York, NY, USA, 2014.
65. Silva Neto, A.V.; Silva, H.L.; Cugnasca, P.S. *Relatório Técnico de Pesquisa—Desenvolvimento do Estudo de Caso de Protótipo de Sistema de Controle de Frenagem de Veículo Por Aprendizado Supervisionado—Version 17*; GAS: São Paulo, Brazil, 2023. [CrossRef]
66. Silva Neto, A.V.; Silva, H.L.; Garcia, L.A. GitHub: Safetyartist: Safety ArtISt (Artificial Intelligence Structure). Available online: https://github.com/antoniovieira88/safetyartist (accessed on 20 October 2023).
67. *scikit-learn* Scikit-Learn: Machine Learning in Python—Scikit-Learn 1.3.1 Documentation. Available online: https://scikit-learn.org/stable/ (accessed on 20 October 2023).
68. James, G.; Witten, D.; Hastie, T.; Tibshirani, R. *An Introduction to Statistical Learning (with Applications in R)*, 2nd ed.; Springer: New York, NY, USA, 2008; pp. 1–607.
69. Summers, S.; Kreis, B.; Tran, N.; Brown, C.; Duarte, J.; Kreinar, E.; Damiani, A.; Guillaume-Bert, M.; Asif, F.; Wu, Z.; et al. GitHub—Thesps/Conifer: Fast Inference of Boosted Decision Trees in FPGAs. Available online: https://github.com/thesps/conifer (accessed on 20 October 2023).
70. Duarte, J.; Summers, S.; Di Guglielmo, G.; Harris, P.; Hoang, D.; Jindariani, S.; Kreinar, E.; Loncar, V.; Ngadiuba, J.; Pierini, M.; et al. Fast Inference of Boosted Decision Trees in FPGAs for Particle Physics. *J. Instrum.* **2020**, *15*, 1–12. [CrossRef]
71. Noronha, D.H.; Salehpour, B.; Wilton, S.J.E. LeFlow: Enabling Flexible FPGA High-Level Synthesis of Tensorflow Deep Neural Networks. *arXiv* **2018**, arXiv:1807.05317.
72. Noronha, D.H.; Salehpour, B.; Santacroce, M.; Pinilla, J.; Bragança, L. GitHub—Danielholanda/LeFlow: Enabling Flexible FPGA High-Level Synthesis of Tensorflow Deep Neural Networks. Available online: https://github.com/danielholanda/LeFlow (accessed on 20 October 2023).

73. Scikit-learn Developers Sklearn. Tree.DecisionTreeClassifier—Scikit-Learn 1.3.1 Documentation. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (accessed on 20 October 2023).
74. Tran, D.H.; Cai, F.; Lopez Diego, M.; Musau, P.; Johnson, T.T.; Koutsoukos, X. Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control. *ACM Trans. Embed. Comput. Syst.* **2019**, *18*, 1–22. [CrossRef]
75. Xiang, W.; Tran, H.-D.; Yang, X.; Johnson, T.T. Reachable Set Estimation for Neural Network Control Systems: A Simulation-Guided Approach. *IEEE Trans. Neural Networks Learn. Syst.* **2021**, *32*, 1821–1830. [CrossRef]
76. Socha, K.; Borg, M.; Henriksson, J. SMIRK: A Machine Learning-Based Pedestrian Automatic Emergency Braking System with a Complete Safety Case. *Softw. Impacts* **2022**, *13*, 100352. [CrossRef]
77. Cleaveland, M.; Ruchkin, I.; Sokolsky, O.; Lee, I. Monotonic Safety for Scalable and Data-Efficient Probabilistic Safety Analysis. In Proceedings of the 13th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS 2022), Philadelphia, PA, USA, 4–6 May 2022; pp. 92–103. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.