Assuring the Case: A Safety Engineering Approach to AI-Enabled Systems

Scot Davidson
Thales IAS
1 Alanbrooke Rd
Belfast BT6 9HB
United Kingdom
Scot.davidson@uk.thalesgroup.com

Oseghale Igene
CSIT
Queen's University Belfast
Queen's Titanic Quarter
Belfast BT3 9DT
United Kingdom
o.igene@qub.ac.uk

Paul Miller
CSIT
Queen's University Belfast
Queen's Titanic Quarter
Belfast BT3 9DT
United Kingdom
p.miller@qub.ac.uk

ABSTRACT

Given the increasing ubiquity of Machine Learning (ML), there is an urgent need for assurance frameworks, particularly in relation to ML-enabled safety critical systems. We begin by describing the process of assurance case construction for conventional safety-critical systems that has been developed by the safety engineering community. Goal Structured Notation (GSN), a graphical approach to assurance case construction, is then briefly described. In the sequel, we describe an existing ML assurance framework that employs GSN, Assurance of Machine Learning for use in Autonomous Systems (AMLAS), which is a six-stage assurance process. AMLAS is then applied to our use-case, namely Autonomous Vehicle (AV), with several of the AMLAS stages described in detail. In particular, the creation and testing activities of the ML assurance stage and the instantiation of the assurance argument are developed. Finally, we make the case for increased interaction between the ML and safety engineering community.

1. INTRODUCTION

The past decade has seen exponential growth in ML, particularly deep learning, due to advances in Neural Network (NN) architectures, computing hardware and the availability of huge amounts of data. This has led to the uptake of ML in many different sectors including life and health sciences, agri-food, finance, transport and defence and space. Of particular interest to this work is that of AV. ML models developed have allowed for the advancement of AVs, for example, when it comes to sense-understand and decide-act modules. It currently focuses on how ML models can be improved in terms of their efficiency and performance.

Many companies, ranging from startups to Small to Medium Enterprises (SMEs) to multinationals, working in these sectors have attempted to develop new products and services based on ML, giving rise to an ML-based innovation ecosystem where the level of development rigour, quality and performance can vary wildly. Of particular concern is the use of ML models in safety-critical systems such as AVs. Hence, there is a real need for an assurance framework for systems such as AV's containing ML models. Such a framework is required to provide trustworthiness in the ML models deployed by a system. When analyzing ML-enabled safety-critical systems, some challenges need to be considered. Firstly, due to the complexity and non-deterministic nature of these algorithms, applying traditional safety assessment techniques when determining component-level safety requirements is not adequate. Regulators and auditors lack the tools necessary to determine that ML will not violate intended safety requirements. Secondly, safety assurance arguments must be developed to confidently ensure the safety and performance of these ML-enabled systems. Existing International Safety Standard documentation, for example International Organization for Standardization (ISO) 26262 (Functional safety of Electrical and Electronic (E/E) systems in road vehicles), does not have guidelines associated with ML integration. This will be very important when developing a comprehensive safety case for ML-enabled services and specifically for AV systems.

The academic research community in AI safety typically focus on issues such as interpretability, transparency, bias, etc. This often involves the development of esoteric mathematical techniques addressing model characteristics that may have little relevance to safety. Furthermore, in all cases, they treat the model in isolation, without any system context.

Recently, several standards bodies have been developing standards/regulations for the secure development and use of Artificial Intelligence (AI). European Telecommunications Standards Institute (ETSI) recently published their Technical Specification and Implementation Guide on AI Security which has been adopted as a standard by the National Cyber Security Centre (NCSC) in the UK [14]. The ETSI specification is the first global standard that sets minimum security requirements throughout the AI life cycle for all stakeholders in the AI supply chain. More specifically, it will allow developers of AI systems to demonstrate to prospects and partners that they are adhering to a framework created by cross-disciplinary collaboration, with requirements that are both globally relevant and practically implementable.

In this paper, we discuss the challenges in assuring ML and making models trustworthy and present initial steps towards this for a specific use-case. We then describe several state-of-the-art assurance case construction tools and illustrate how one of these is used in a recently developed assurance framework for ML models. This is then applied to a computer vision-based use-case of AV navigation. Finally, we highlight the need for increased interaction between the ML and safety engineering communities.

2. ASSURANCE CASE CONSTRUCTION

Developing and certifying ML-enabled safety-critical systems requires applying evidence-based standards, process-based standards for interpretation, and increased levels of monitoring. The evidence-based approach requires developing a comprehensive and structured assurance case for such systems. An assurance case is a structured argument and reasoning that demonstrates how a system will meet its requirements. A system is thought to be assured if there is a clear argument (or assurance case) with convincing evidence that the system will work as intended for a given application in a defined environment [6; 8]. Assurance sits at the intersection of the designer and the user, providing assurance to both that the risks of the system have been identified, to what degree they have been mitigated, and under what circumstances the system can be operated. Inadequate requirements introduce evidentiary gaps. An assurance case pattern also documents a reusable argument structure, and evidence types can be instantiated to provide an assurance case instance that is specific [7].

An assurance case addressing safety is called a safety case. Safety cases are typically limited in scope to assurance that "a system is safe for a given application in a given operating environment" [5]. Security cases are used to gather material evidence in order to reason about the security posture of the system. It is to assure system operates securely, minimizing the risk of exploitation through weaknesses and vulnerabilities. Cybersecurity attacks can also have severe implications on safety. Safety-critical systems (Automotive, Traffic Services, Aviation, Railways, etc.) need to provide sufficient safety performance of systems, and current approaches require that manufacturers and operators employ a systematic process of proactively identifying risks and controlling them appropriately [13].

The core of the process of assurance case construction is defining a plan that will produce a system which is assured. Verification agents should be enabled to collect the best, comprehensive evidence possible in support of the argument and to reduce the system's operational risk to an acceptable level. For simple systems, the evidence is generally used to demonstrate that the system's requirements are met. Complex systems are often assured by establishing justified confidence in critical properties of interest such as dependability concerns around harm avoidance (for example, safety cases and security cases). The key terms for an assurance case are defined as follows:

- **Claim**: A high-level assertion or statement;
- **Argument**: Supports the claim and provides the link to the supporting evidence. It also allows the supporting justification to be broken up to aid visibility;
- **Evidence**: Facts and judgements that support the applicable argument(s).

Figure 1 illustrates the relationship between the safety engineer and the system engineer [15]. The system requirements feed into both the design model, including ML components, and the safety analysis. The latter leads to the identification of hazards and risks from which safety requirements are derived. These then feed into the design model. At the core of the process is the assurance case which controls every step of the process.

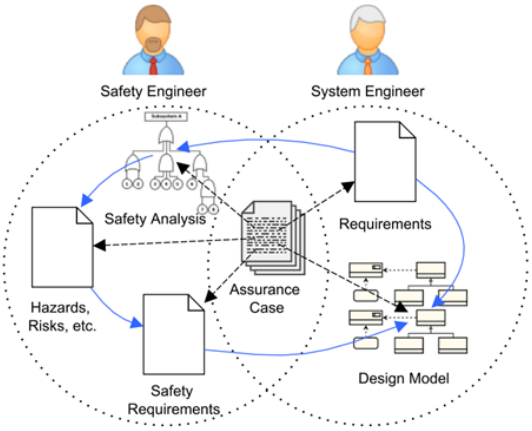

Figure 1: Illustrating the process of assurance case construction and the relationship between safety and system engineers [13].

Well-known notations can be utilised to develop assurance cases that are graphics-based, including GSN [2; 10; 3; 1], fault-tree analysis [4] and Claims Assurance and Evidence (CAE) [11]. These graphic-based approaches are the preferred option over text-based options because of the former's ability to simplify the construction and management of safety arguments and facilitate the presentation of these arguments to others. The GSN will be particularly useful as part of a broader methodology when developing a safety case for an ML-enabled system. GSN graphically provides links between arguments and evidence representing individual elements of any safety arguments and relationships between them in a top-down approach. Key elements of the GSN model include assurance goal or the claim; evidence that the goal is satisfied; and an argument linking evidence to the goal. These elements link together to form a goal structure. CAE is a structured approach which aids the visibility, review and assessment of safety case documentation.

3. METHOD

In this section we provide some background material on goal structured notation and introduce the Application of the AMLAS framework. The goal structure is hierarchical and is produced recursively with the overall goal for the system at the root. Other GSN elements include:

- **Strategies**: Decomposing a goal into additional goals;
- **Assumptions**: Taken without further argument or explanation;
- **Justifications**: Explaining why a solution provides sufficient evidence satisfying a goal;
- **Context**: Associated with another GSN element relevant to that element;
- **Solution**: Describes evidence that the stated goal has been met.

Figure 2 shows the various symbols used to represent the GSN elements along with a simple example. In the example shown above, claim C is justified by argument AS_1 based on

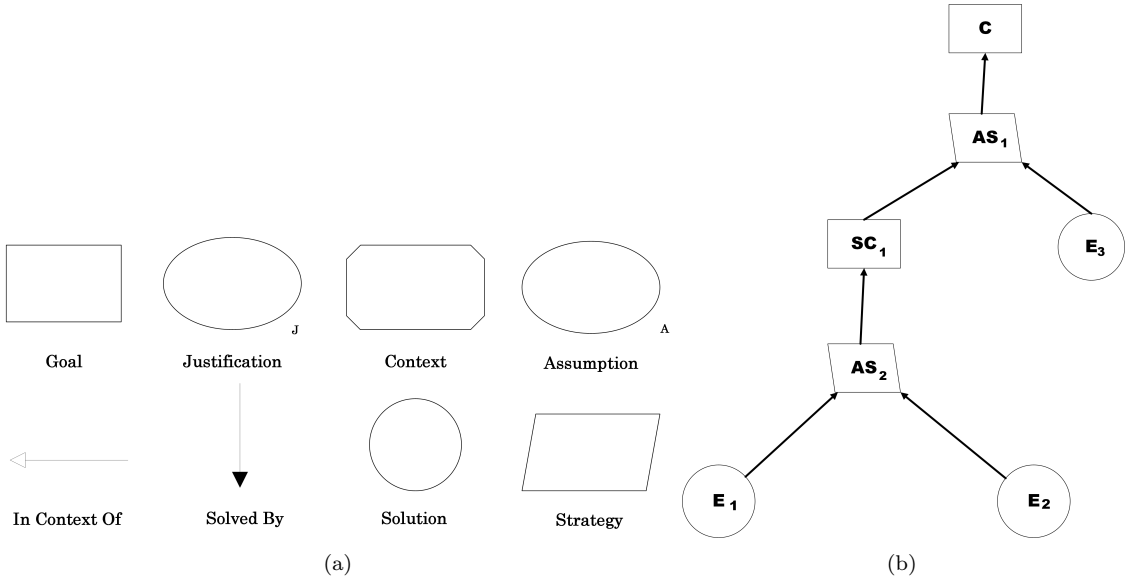

Figure 2: Symbols used to represent elements of goal-structured notation, (a), along with a simple example, (b).

evidence E_3 and subclaim SC_1, which itself is justified by argument step AS_2 based on evidences E_1 and E_2.

For the development of the safety case, the AMLAS methodology was chosen as it presents an iterative process involving machine learning-enabled components associated with our use-case [7]. As shown in Fig. 3, the AMLAS methodology provides six stages in which system safety requirements are first identified before narrowing to ML component safety requirements elicitation. Other stages include data management requirements associated with training and test datasets for the ML models, and formal validation and verification requirements.

Note that the process is iterative. At each stage, learnings from one stage can be fed back to previous stages to update it. To develop the safety case, the AMLAS approach will be applied to specific ML components, and each stage of the process is documented following each AMLAS step. Each of these is decomposed into a workflow of activities. In the next section, we will systematically go through some of the stages (due to space limitations) and illustrate what activities must be performed in the context of our use-case. For each stage and activity, inputs and outputs in the form of documentation have to be created. The output of each stage is an assurance argument pattern. Finally, at the completion of the process an assurance case is constructed for the ML model.

4. APPLICATION TO AUTONOMOUS VEHICLE USE-CASE

System

The application of interest is a navigation system as shown in Fig. 4 for an AV. The system for this is shown in Fig. 4. Inputs to the system are provided by RGB and infra-red cameras. C1-C5 and C7-C8 (red) are all ML components including: RGB image preprocessing for haze removal; image fusion; object detection and tracking; trajectory estimation; motion/pose analysis; and finally scene interpretation. As mentioned previously, assurance needs to be performed for all ML components, however, for the sake of brevity, we shall focus on C4 object detection.

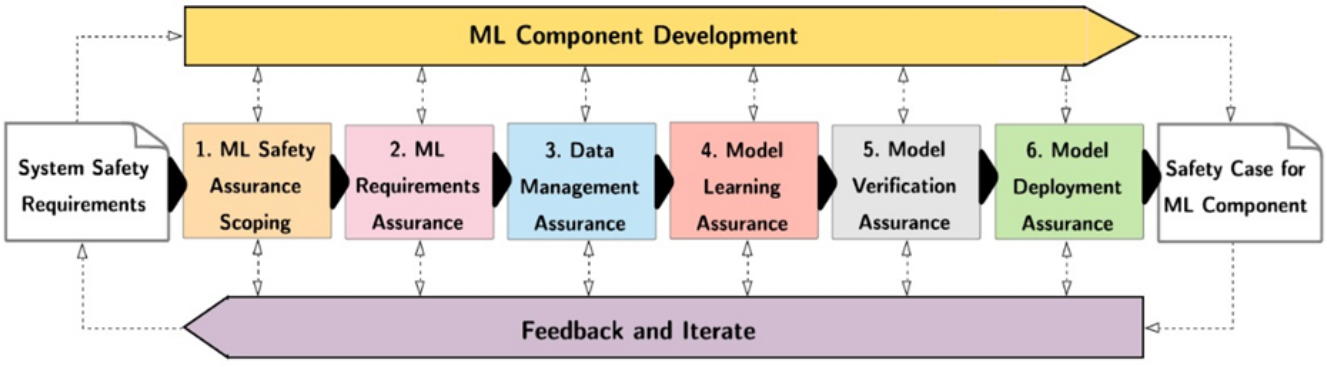

Figure 3: Overview of the AMLAS process [6].

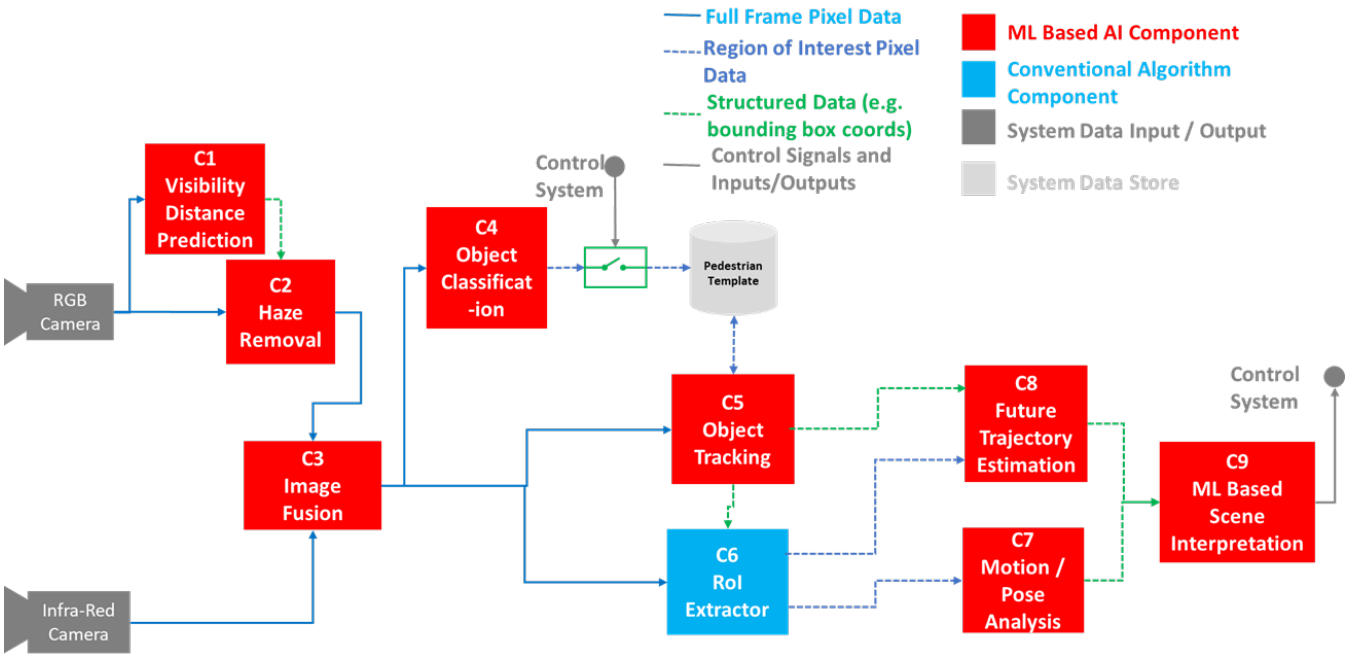

Figure 4: System architecture for navigation of the AV.

Environment

The system operates within the typical driving environment for human-driven vehicles, covering various road types including urban streets, highways, and suburban areas, under normal weather and lighting conditions. It is intended for non-industrial and non-commercial use, and must handle dynamic objects such as pedestrians, other vehicles, cyclists, and static infrastructure elements within the camera's field of view. Variability in traffic density, road layouts, and environmental factors such as rain, fog, or low light conditions influence system performance and safety requirements. The operational context also includes interaction with human users who may need to intervene or respond to system alerts, with the system designed to notify the user promptly in case of AI algorithm limitations or failures. These environmental characteristics and usage scenarios define the system's operational design domain and form a critical basis for tailoring safety assurance activities.

System Safety Requirements

When identifying system safety requirements, we first need to determine system-level hazards from which system safety requirements can then be developed. Applying risk assessment techniques such as HAZard and Operability Analysis (HAZOP) can be used to identify potential hazards that can be associated with safety-critical systems. An example potential hazard associated with autonomous vehicles that can arise depending on their operating environment is: Potential inaccuracies of "world models" (internal representation of the environment) generated by perception subsystems (Red Green Blue (RGB) camera) can lead to misinterpretation of data, compromising system safety [12].

In total, using the HAZOP methodology we identified five hazards. Based on the system architecture, Fig. 4, and potential system hazards identified, we derived six sets of safety requirements using requirements analysis [9].

ML Component Description - Object Detector (C4)

As mentioned previously, for the sake of illustration and brevity, we shall only consider ML component C4 Object Detector. This subsystem is responsible for detecting a single pedestrian within the vehicle's operational environment under all normal driving conditions, including varying lighting, weather, and traffic scenarios. The system utilizes a control module that maintains a pedestrian template to support recognition functions. This pedestrian detection subsystem operates as a component within a broader system-of-systems architecture typical of self-driving vehicles, which integrate numerous interdependent functional units such as perception, planning, control, and human-machine interfaces. Given this hierarchical structure, the subsystem's safety and performance requirements are derived not only from its internal function but also from its interaction and integration within the wider vehicle framework. The subsystem's functional scope, operational environment, and interfaces with other vehicle systems directly inform the safety assurance process, including hazard analysis, safety requirement allocation, and verification activities. Its role within the system of systems highlights the need for robust communication, interoperability, and adherence to safety standards to ensure reliable pedestrian detection as a critical element of the overall autonomous driving assurance case.

Fig. 5 shows the network architecture of YOLOV5 consisting of the backbone, PANet and outputs. The Object Detector (C4) takes a single image and classifies multiple objects (pedestrians) and outputs bounding boxes. This component implements an object detection algorithm that will be used to detect multiple persons who are crossing designated roadwalks. Furthermore, this ML component will handle and be exposed to both the In-Distribution and Out-of-Distribution

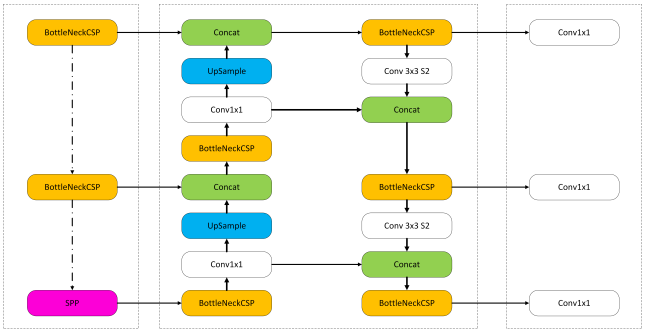

Figure 5: YOLOV5 network architecture (https://github.com/ultralytics/yolov5/issues/280).

(OOD) (Training and Test datasets) from environmental scenarios including weather conditions, background changes, object transformations and sensor anomalies. The component will receive live feed from the RGB Camera installed when in operation, which will also link with another ML component (C5).

4.1 Stage 1 - Machine Learning Assurance Scoping

Activity 1 - Defining the Safety Assurance Scope for the ML Component

For this activity, the system safety requirements associated with the C4 ML component need to be explicitly stated to build the safety assurance argument. Taking into account the defined system architecture and the operating environment, the system Safety Requirements (SRs) deemed within scope of C4 are:

SR4: "The system shall obey the highway code" which is further broken down into 6 requirements;

SR6: "The system shall be secure by design" which is further broken down into 2 requirements.

4.2 Stage 2 - ML Requirements Assurance

Activity 3 - ML Requirements Scoping

The scoping of Machine Learning Safety Requirements (MLSRs) involves deriving a dedicated set of requirements specific to machine learning components within a system. These requirements are critical because they relate directly to the component level rather than to the overall system. Establishing such requirements early in the development process ensures a coherent foundation, allowing subsequent engineering activities to be systematically traced back to the original design rationale. The proposed method employs traceability to illustrate the relationships between system requirements and the derived MLSRs. Moreover, these requirements serve as reference points for integrating documentation, reporting, and compliance evidence throughout the system lifecycle. In total, seventeen MLSRs have been defined, with two particularly notable requirements identified as:

MLSR6: "The ML component shall maintain data integrity throughout the development process" further broken down into 2 requirements;

MLSR7: "The ML system shall pass an in operation task, based on edge cases and normal operation" it is not further broken down.

Activity 4 - Validate ML Safety Requirements

Requirements are fundamental to understanding how a system and its components are expected to interact and perform. As the MLSRs represent high-level objectives, targeted validation activities are essential to confirm that these requirements are met in practice. These activities, denoted as VAL (Validation), are systematically mapped to one or more MLSRs. This structured mapping enables engineers to design and execute corresponding unit and system-level tests, while supporting safety and system engineers in validating the integrated system. In turn, this process underpins system acceptance and certification. Three notable VAL processes are defined as:

VAL6.1: Evaluate the model performance using adversarial attacks and provide a value for when the classification of objects defined in MLSR5 is obtained;

VAL6.2: Validate if adversarial training is sufficient for generating a robust model;

VAL 7: Assess the performance of the model and validate each of the given scenarios through simulations.

In total there are 13 validations ensuring that the MLSR's can be achieved. These are traceable to MLSR's using traceability analysis.

4.3 Stage 4 - Model Learning Assurance

In this section, the Activities 10-12 of the model learning assurance stage are briefly described. Fig. 6 shows the workflow.

Activity 10 - Creating the ML Model

Two MLSR safety requirements fed into Activity 10:

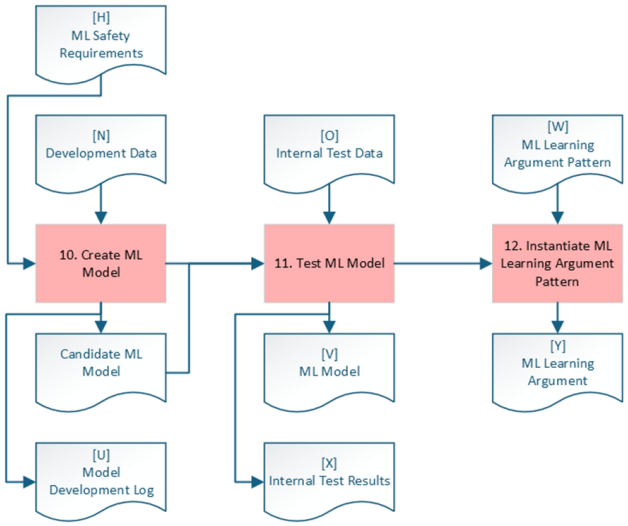

Figure 6: Activities associated with Stage 4 [6].

MLSR5: The system shall be able to recognise a person, car, sign, pedestrian crossing and all other objects defined in the highway code;

MLSR7: The ML system shall pass an in-operation task based on edge cases and normal operation.

Other inputs include the development data generated in Activity 7. MLSR7 enforces a real-time requirement which meant Vision Transformers were ruled out, whilst CNNs and a traditional ML recognition model were able to satisfy the requirement. MLSR5 further narrowed the model selection to the following off-the-shelf CNN architectures: YOLO, Faster-RCNN and SSD. From the literature it was discovered that only YOLO could satisfy MLSR7. The licensing arrangements for all nine versions of YOLO were then evaluated. YOLO5 was selected as its MIT license was deemed to be the least restrictive. The small and medium models were then put forward for adversarial training and possible deployment. The adversarial training involved fine-tuning the baseline YOLO model with adversarial samples containing patches. Part of the Model Based Systems Engineering (MBSE) approach means C++ conversion is a requirement for deployment of the neural network to the target hardware, Pytorch is converted using the libtorch package.

Activity 11 - Test of the ML model

Inputs to this include the internal test data developed under Activity 7. Four types of testing were then performed:

- **Test 1 – Baseline Test**: To establish the performance of the system without modifications such as augmentations and perturbations;
- **Test 2 - Environmental Test**: This refers to testing with augmented data that represents edge cases and environmental effects from the operational environment. Typically involves effects such as fog, rain and glare;
- **Test 3 - Adversarial**: This refers to testing the model utilising white-box and black-box attacks and adversarially trained patches. An example of the patch attack test is shown in Fig. 7.

These tests primarily involve measuring the attack success rate versus attack parameters such as patch size. This metric therefore provides quantitative evidence for the assurance process. The output is a document describing this evidence. This evidence is displayed in Fig. 8.

Activity 12 - Instantiation of the ML Argument Pattern

This is shown in Fig. 9. The Argument Strategy S4.1 is to argue over the internal testing of the model performed during development as well as the development approach adopted. The appropriateness of the development activities is considered within the context of creating a model that both satisfies the ML safety requirements as well as meeting the additional constraints that are imposed on the model, such as performance and cost. The model integrates YOLO for real-time object detection in an effort to detect all objects defined by the Requirement SR4.2 "The system shall maintain a safe distance from other objects as defined in the highway code". The development approach follows a structured lifecycle involving dataset curation; model training with augmented adversarial samples; adversarial robustness

Figure 7: Example of an adversarial patch placed on the target

evaluation; and validation under diverse driving scenarios. Internal testing emphasizes safety assurance by evaluating the model's resilience to adversarial and environmental variations.

These internal tests validate that the model maintains accurate perception under both nominal and adversarial conditions. The appropriateness of the development activities lies in their direct alignment with ML safety requirements Activity 5—robustness, interpretability, and reliability—while adhering to operational efficiency and hardware constraints relevant to automotive deployment. Cost is not assessed at this stage.

Goal 4.2 is that it must be demonstrated that the selected ML model satisfies the ML safety requirements when using internal testing data. This is demonstrated through a systematic traceability analysis linking each safety requirement to its corresponding verification activity, as defined in the use

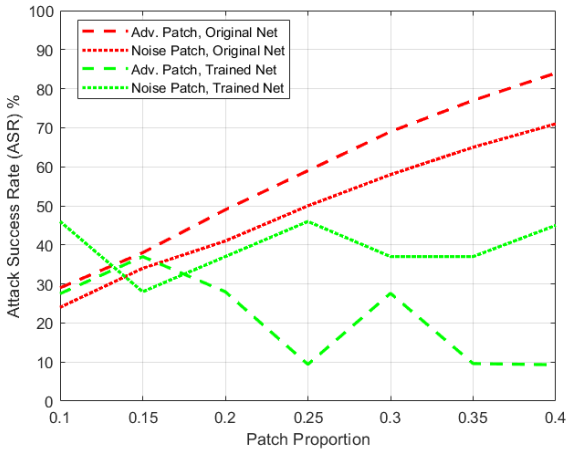

Figure 8: Attack Success Rate based on the patch proportion of the target. Tested against four neural networks red denotes the original net, and green for the adversarially trained neural network.

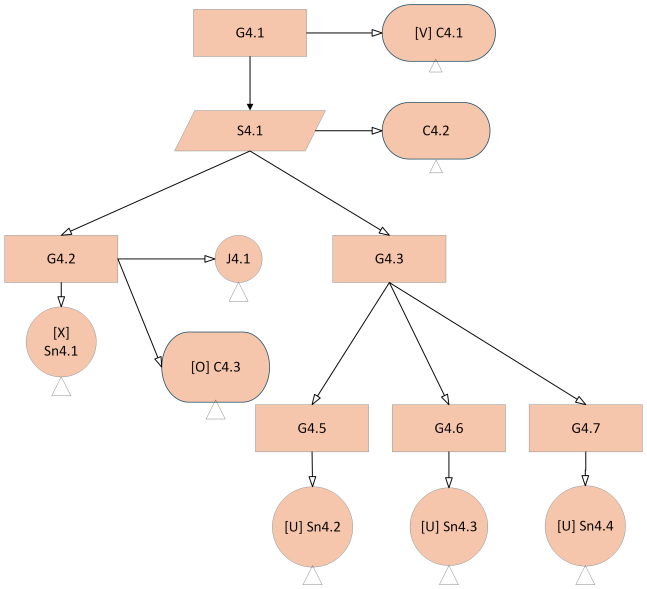

Figure 9: Assurance argument pattern given in for Stage 4 model learning assurance.

case illustrated in Fig. 4. This traceability ensures that all requirements are both testable and their specified verification strategies applied, linking the requirements of the original YOLO implementation with the robust YOLO model. The internal testing activities are coordinated within the AMLAS process, enabling stakeholders to observe and validate the relationships among diagrams, models, and associated code. The robustness of the YOLO architecture is assessed through controlled testing under both Operational Design Domain (ODD) and OOD conditions, ensuring the model maintains a reliable detection performance in realistic and adverse driving contexts. This integrated approach supports transparency and collective assurance across the creation and development lifecycle. The testing assess the requirements as well as the technical performances of the model. These technical performances found through the testing of the model are documented in the development log.

Evidence supporting the internal testing claim is documented comprehensively in the report generated from Activity 11, which details the test outcomes and validation processes. The sufficiency of these results in demonstrating compliance with ML safety requirements is further substantiated through the justification report (J4.1), providing a formal rationale that the internal testing methodology, environment, and outcomes collectively confirm that the ML model satisfies all relevant safety objectives.

Goal 4.3 addresses the development approach adopted for the creation of the ML model, supported by subordinate claims concerning model selection, parameterization, and the applied development process. For Goal G4.5, the model selection criteria are defined and justified in Activity 10, which establishes the linkage between the selected architecture and the system's functional and safety requirements. The choice of YOLO as the primary detection model is based on its proven capability for real-time object recognition and classification performance across a diverse range of categories, many of which correspond to objects and entities referenced within highway code standards.

Goal 4.6 incorporates standard retraining procedures designed to enhance robustness and generalization. This is performed during the training phase, model parameters are generated through iterative optimization to align detection accuracy and computational efficiency with safety and operational goals.

Goal 4.7 is that it must be demonstrated that the development process is appropriate. The development process has been designed to ensure methodological appropriateness and transparency throughout all iterative stages of model creation as part of Activity 10. As the approach employs a PyTorch based toolchain, it must convert to a C++ integration (MLSR 8); PyTorch is therefore selected over Tensorflow for its proven reliability, backward compatibility, and integration capabilities through the LibTorch package. The YOLO architecture is implemented using the established Ultralytics toolchain, which is widely documented and supported within the research community. This satisfies some of the testing that is required as part of Activity 11; although other tests still need to be performed for assurance. The retraining utilizes an open-source retraining script, facilitating transparent and reproducible model adaptation. The method aligns with established best practices in safety-critical ML development and supports continuous improvement through verifiable and explainable training updates. All of the required training and validation is found in the model development log. The justifications for each are documented in J4.1 and summarised in Fig. 9 and the following list for brevity.

C4.1: The ML model chosen is a detector tracker system utilising YOLO and Siamese tracker;

C4.2: Constraints are real-time compatibility, assured to be safe using verification, must run on GPU, Pytorch Implementation;

C4.3: The internal test data is described in O. This branches how the internal test data is linked to X Results;

G4.1: The development of the learnt model is sufficient;

G4.2: The selected model satisfies the ML safety requirements when using internal test data;

G4.3: The development approach adopted to create the model is sufficient;

G4.6: The model parameters are appropriate for meeting the defined ML safety requirements;

G4.7: The model development process is appropriate for meeting the defined ML safety requirements;

J4.1: The internal test results are representative of the components of the model;

S4.1: Argument over the sufficiency of the model development within the constraints of the target deployment;

Sn4.1: The internal test results are described in X. All of the internal tests are assumed to have passed in this scenario as failing requires redrafting the process until working;

Sn4.2: The model selection has been curated by utilising requirements analysis to ensure initial model selection meets the safety requirements;

Sn4.3: The off-the-shelf YOLO model has been appropriately trained for object classification. Fine-tuning is required to ensure the model is robust to adversarial attacks;

Sn4.4: The model development process is documented in [U] and all toolchains are assessed to comply with the MLSR's defined in Activity 3.

5. CONCLUSIONS

In this paper, we have described preliminary work on developing an assurance framework for image classifiers based on a deep learning model. We have modified an existing framework, AMLAS, and applied it to our use-case, namely AVs. AMLAS is a six-stage process, with each being decomposed into a series of activities that have inputs and outputs. Each stage completes with the development of an assurance argument pattern using GSN. At each stage, we have also described techniques that can be used to provide evidence for the various arguments; adversarial training, patch detection and mitigation.

Future work will involve investigating how we can use the evidence metrics to give a confidence score for each stage and then aggregating these using evidential reasoning networks. In addition, dynamic assurance cases are required for many systems, hence we propose to investigate this. Finally, the GSN approach used by AMLAS originated in the system safety engineering community. We feel that it is vital that the AI community reach out and collaborate with the former to achieve safety certification of system with AI models, rather than focusing on abstract theoretical concepts that have little relevance to safety.

6. ACKNOWLEDGEMENTS

This work is supported through the NICYBER2025 programme funded by Innovate UK. The project is a collaboration between CSIT and Thales. The views expressed are those of the authors and do not necessarily represent the project or the funding agency.

7. ADDITIONAL AUTHORS

Additional authors: Hamid Asgari (Thales RTSI, email: `Hamid.Asgari@uk.thalesgroup.com`), Chris Johnson (CSIT, email: `c.w.johnson@qub.ac.uk`), Nic Friedlander (Thales IAS, email: `nic.friedlander@uk.thalesgroup.com`.

8. REFERENCES

[1] Assurance Case Working Group (ACWG). Goal Structuring Notation (GSN) Community Standard, Version 3. Technical Report SCSC-141C, SCSC, May 2021. Accessed: 2024-10-22.

[2] A. Ayoub, B. Kim, I. Lee, and O. Sokolsky. A systematic approach to justifying sufficient confidence in software safety arguments. In *International Conference on Computer Safety, Reliability, and Security*, pages 305–316. Springer, 2012.

[3] H. Bourbouh, M. Farrell, A. Mavridou, and I. Sljivo. Integration and evaluation of the advocate, fret, cocosim, and event-b tools on the inspection rover case study. Technical report, 2020.

[4] K.-T. Chen, H.-Y. W. Chen, A. Bisantz, S. Shen, and E. Sahin. Where failures may occur in automated driving: a fault tree analysis approach. *Journal of cognitive engineering and decision making*, 17(2):147–165, 2023.

[5] DEF STAN 00-56. Safety management requirements for defence systems, part 1: Requirements. Technical report, Ministry of Defence, 2017. Retrieved October 24, 2024.

[6] E. Denney, G. Pai, I. Habli, T. Kelly, and J. Knight. 1 st international workshop on assurance cases for software-intensive systems (assure 2013). In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1505–1506. IEEE, 2013.

[7] R. Hawkins, C. Paterson, C. Picardi, Y. Jia, R. Calinescu, and I. Habli. Guidance on the assurance of machine learning in autonomous systems (amlas). *arXiv preprint arXiv:2102.01564*, 2021.

[8] L. Jöckel, M. Kläs, J. Groß, P. Gerber, M. Scholz, J. Eberle, M. Teschner, D. Seifert, R. Hawkins, J. Molloy, et al. Operationalizing assurance cases for data scientists: A showcase of concepts and tooling in the context of test data quality for machine learning. In *International Conference on Product-Focused Software Process Improvement*, pages 151–158. Springer, 2023.

[9] G. Kotonya and I. Sommerville. *Requirements engineering: processes and techniques*. Wiley Publishing, 1998.

[10] NASA. Advocate user guide 1.4. Technical report, NASA Technical Reports Server (NTRS), May 2022. Accessed: 2024-10-22.

[11] S. Porziani, F. Scarpitta, E. Costa, E. Ferrante, B. Capacchione, M. Rochette, and M. E. Biancolini. Caeup-update of cae models on actual manufactured shapes. *Procedia Structural Integrity*, 24:775–787, 2019.

[12] L. Steckhan, W. Spiessl, N. Quetschlich, and K. Bengler. Beyond sae j3016: New design spaces for human-centered driving automation. In *International Conference on Human-Computer Interaction*, pages 416–434. Springer, 2022.

[13] M. A. Sujan, F. Koornneef, N. Chozos, S. Pozzi, and T. Kelly. Safety cases for medical devices and health information technology: involving health-care organisations in the assurance of safety. *Health informatics journal*, 19(3):165–182, 2013.

[14] TS 104 223. Securing artificial intelligence (sai); baseline cyber security requirements for ai models and systems. Technical report, ETSI, 2025. Retrieved October, 2025.

[15] R. Wei, S. Foster, H. Mei, F. Yan, R. Yang, I. Habli, C. O'Halloran, N. Tudor, T. Kelly, and Y. Nemouchi. Access: Assurance case centric engineering of safety–critical systems. *Journal of Systems and Software*, 213:112034, July 2024.