

Safety Assurance Objectives for Autonomous Systems

Version 4.0 (SCSC-153C)

Safety of Autonomous Systems Working Group (SASWG)

ISBN: 979-8879316780

SCSC Publication Number: SCSC-153C

Permanent URL: https://scsc.uk/SCSC-153C

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA. You are free to share the material in any form and adapt the material for any purpose providing you attribute the material to the Safety Critical Systems Club (SCSC) Safety of Autonomous Systems Working Group (SASWG), reference the source material, include the licence details above, and indicate if any changes were made. See the license for full details.

Cover photo by Markus Spiske, temporausch.com. Obtained from https://www.pexels.com/.

The Safety Critical Systems Club (SCSC) is the professional network for sharing knowledge about safety-critical systems. It brings together: engineers and specialists from a range of disciplines working on safety-critical systems in a wide variety of industries; academics researching the arena of safety-critical systems; providers of the tools and services that are needed to develop the systems; and the regulators who oversee safety. Through publications, seminars, workshops, tutorials, a web site and, most importantly, at the annual Safety-critical Systems Symposium (SSS), it provides opportunities for these people to network and benefit from each other's experience in working hard at the accidents that don't happen. It focuses on current and emerging practices in safety engineering, software engineering and product and process safety standards.

This document was written by the Safety of Autonomous Systems Working Group (SASWG), which is convened under the auspices of the SCSC. The goal of the SASWG is to produce clear guidance on how autonomous systems and autonomy technologies should be managed in a safety related context, throughout the lifecycle, in a way that is tightly focused on challenges unique to autonomy. The document was formally released at SSS'24, 13-15 February 2024.

Comments on this document are actively encouraged. These can be emailed to:

saswg-comments@scsc.uk

While the authors and the publishers have used reasonable endeavours to ensure that the information and guidance given in this work is correct, all parties must rely on their own skill and judgement when making use of this work and obtain professional or specialist advice before taking, or refraining from, any action on the basis of the content of this work. Neither the authors nor the publishers make any representations or warranties of any kind, express or implied, about the completeness, accuracy, reliability, suitability or availability with respect to such information and guidance for any purpose, and they will not be liable for any loss or damage including without limitation, indirect or consequential loss or damage, or any loss or damage whatsoever (including as a result of negligence) arising out of, or in connection with, the use of this work. The views and opinions expressed in this publication are those of the authors and do not necessarily reflect those of their employers, the SCSC or other organisations.



Safety Assurance Objectives for Autonomous Systems

The Safety of Autonomous Systems Working Group [SASWG]

February 2024

Change History

Version	By	Status	Date
SCSC-153	The SASWG Team	For publication at SSS'19	JAN-2019
SCSC-153A	The SASWG Team	For publication at SSS'20	JAN-2020
SCSC-153B	The SASWG Team	For publication at SSS'22	JAN-2022
SCSC-153C	The SASWG Team	For publication at SSS'24	FEB-2024

Changes Since the Last Version

The main changes since the last version of this guidance are:

- Appendices containing comparisons to other similar scoped assurance documents and academic papers are now contained in a separate document [89]. This is available on the Safety Critical Systems Club web repository.
- Further clarity on tailoring, integrity and criticality has been added (in Section 8).
- An additional Objective COM1-5 "Processing methods on inputs and outputs used during operation do not introduce errors." has been added to consider "processing" of data and signals to and from an algorithm. For example, this might include image processing for feature extraction prior to classification, or where output values are normalised or otherwise transformed prior to use by another element of the platform.

Contents

1 Introduction **1**

1.1 Document Aim and Scope 1

1.2 Frameworks, Projections and Objectives 1

1.3 Document Status 3

1.4 Terminology 3

1.5 Document Structure 5

2 Computation-Level Framework: Description **7**

2.1 Projections 7

2.2 Summary 9

3 Computation-Level Framework: Objectives **11**

3.1 Experience 11

3.2 Task 15

3.3 Algorithm 21

3.4 Software 24

3.5 Hardware 25

4 Autonomy Architecture-Level Framework: Description **27**

4.1 Projections 27

4.2 Summary 28

5 Autonomy Architecture-Level Framework: Objectives **29**

5.1 Tolerance 29

5.2 Information Provision 33

5.3 Adaptation 36

6 Platform-Level Framework: Description **41**

6.1 Projections 41

6.2 Summary 42

7 Platform-Level Framework: Objectives **45**

7.1 Behavioural Specification 45

7.2 Interacting Items 49

7.3 People 52

7.4 Environment 55

8 Tailoring **59**

8.1 Integrity/Criticality 59

8.2 Effect on Objectives 59

8.3 Summary 60

9 Summary **61**

9.1 Computation-Level 61

9.2 Autonomy Architecture-Level 62

9.3 Platform-Level 63

Appendix A An Illustration of Tailoring **65**

Appendix B Known Issues **69**

Appendix C Abbreviations **71**

Appendix D References **73**

Appendix E Contributors **79**

1 Introduction

1.1 Document Aim and Scope

This document aims to provide clear, practical, pan-domain guidance on the safety assurance of Autonomous Systems (AS). In particular:

- The document is intended to be of use to a wide readership, including: developers of autonomous systems; Artificial Intelligence (AI) and Machine Learning (ML) practitioners; safety engineers; regulatory authorities; and managers (at a range of levels).
- The guidance is intended to be widely applicable. It is not tied to any specific development approach, system lifecycle or safety argument structure. To achieve this wide applicability, terms like "appropriate" and "suitable" are occasionally used. In such cases, users of this document would be expected to describe, and justify, how these terms have been interpreted in their specific context.
- There is a deliberate focus on aspects directly related to autonomy, and enabling technologies such as AI and ML, rather than more general safety engineering or system engineering, where it is assumed that relevant general standards, guidelines and best practice will be applied. The intent is to avoid duplicating existing guidance relating to these general topics.
- Where one is available, this document could usefully supplement an existing Safety Management System (SMS). Consequently, there is intentionally very little mention of legal and / or regulatory requirements. Likewise, issues that are most appropriately addressed at an organisational, or enterprise, level (e.g. staff competencies) are not addressed.
- There is a deliberate focus on AS that use AI developed using ML. Although it is possible to envisage AS that do not use these technologies, AI and ML are considered to represent the greatest assurance challenges; they are also expected to be widely used.
- The objectives listed in this document would be expected to inform any discussions with regulatory authorities. However, since they are inevitably domain-specific, detailed certification-related considerations (e.g. timing and content of liaison with regulators) are not included.

1.2 Frameworks, Projections and Objectives

Three *frameworks* have been used to develop and structure the guidance: computation; autonomy architecture; and platform.

These frameworks are used purely as a tool for the specific purpose of identifying objectives that need to be addressed to achieve demonstrably safe autonomous systems. Different frameworks may be applicable to, or more appropriate for, the systems that are being developed by users of this document. In such cases there is no expectation that users structure their development efforts to directly align with the frameworks adopted in this document.

The frameworks used in this document are briefly described below:

- The computation-level framework addresses implementation at the software and computational hardware levels. It focuses on mapping an input to an output. Activities associated with this level typically relate to fault prevention. This is the lowest conceptual level considered.

- The autonomy architecture-level framework addresses how computations can be integrated into a system, or platform. Activities at this level typically relate to fault tolerance.
- The platform-level framework addresses what the final autonomous entity should (and should not) do and what effects it should (and should not) have on its environment. In essence, the focus is on requirements. This is the highest conceptual level considered.

In general, a platform may contain multiple autonomy architectures, each of which may contain multiple computations. This relationship is illustrated in Figure 1, where dashed grey lines represent optional items.

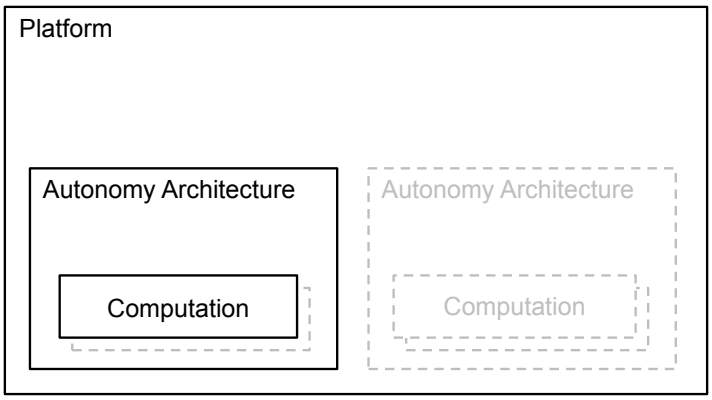


Figure 1: Relationship between Framework Levels

Table 1 includes two examples that illustrate the distinction between platform, autonomy architecture and computation within the context of this document.

Table 1: Example Platforms, Autonomy Architecture Components and Computations

Item	Illustration One	Illustration Two
Example Platform	Self-Driving Car	Medical Diagnosis Application
Example Autonomy Architecture Components	Sensor Health Checks, Sanity Checks on Generated Route	Integrity Checks on Supplied Image, Using Multiple Classifiers
Example Computation	Route Planning	Image Classification (Benign / Malignant)

A series of *projections* is associated with each framework. The projections provide different perspectives; they provide different ways of viewing each framework, in order to elicit associated objectives. Consequently, whilst each projection emphasises a particular aspect that is relevant to the framework level, the projections are not intended to be strictly independent, distinct or non-overlapping. Whilst they have been useful in developing this document, for example, by splitting the overall scope into manageable parts, the projections themselves are not intended to be of any great significance in their own right. As with the frameworks, there is no expectation that a user of this document will structure their development efforts around the specific projections used within this guidance.

A collection of *objectives* are listed against each projection. Even though the projection has been used as part of the process of eliciting objectives, an objective's scope may be wider than its parent projection; equivalently, the projection provides context, rather than bounds, for the objective.

Each objective is accompanied by a discussion that illustrates how the objective contributes to AS safety. This is followed by examples of approaches that could be taken to satisfy, or partially satisfy, the objective. These examples are not intended to be prescriptive; there may be other ways of satisfying an objective. Likewise, the examples do not necessarily represent a preferred way of satisfying an objective. They are included solely to demonstrate the feasibility of satisfying at least part of the objective.

As with frameworks and projections, there is no expectation that a user of this document will slavishly follow the way the objectives have been ordered and structured within this guidance. For example, a user may prefer to re-organise the objectives so that they are more closely aligned with, for example, their chosen system architecture, their development processes or their organisational structures.

However, it is expected that users of this document would provide evidence to demonstrate that the objectives have been satisfied. They may also provide evidence-based, structured arguments to justify why a particular objective needs only to be covered at a superficial level for their particular application.

1.3 Document Status

This document was authored by the Safety of Autonomous Systems Working Group (SASWG), which is convened under the auspices of the Safety Critical Systems Club (SCSC). The first version of the document, released in January 2019, only considered the computation-level framework. The subsequent version, released in January 2020, represented the first time that all three framework levels had been addressed. The next version, released in January 2022, contains revisions based on the experience of applying the previous version to a number of projects. This current version, released in February 2023, is an iterative update, and removes the comparison appendices. It also includes updates that reflect recent research, as well as changes that begin to address some of the known issues associated with the previous version.

Although they have been applied in a number of projects, the identified objectives (and associated frameworks) still have a much stronger justification from theoretical basis than from a practical one. Most notably, efforts have been made to check their validity. For example, where possible, the adopted frameworks have been compared against possible alternatives. The objectives have also been compared against relevant peer-reviewed documents. In addition, this document's contents have been reviewed from the context of the Organisation for Economic Co-operation and Development (OECD) Principles on AI[1]. These collected activities provide some confidence that the objectives are suitable for their intended use.

Despite their application, the objectives are not yet judged to be "proven through use". It is expected that the objectives and, especially, the associated example approaches will change as experience is gained. Consequently, feedback from the safety, AS, AI and ML communities is encouraged. This can be provided by emailing the address noted on the inner front cover.

1.4 Terminology

The Safety of Autonomous Systems Working Group (SASWG) has deliberately avoided defining the term *autonomous*, preferring to work from examples and assuming that, generally speaking, it is easy to identify whether a specific system is autonomous, even though a general definition is difficult to achieve. The desire to avoid protracted and largely uninformative debates about definitions extends across much of the SASWG's work. Nevertheless, it is helpful to provide outline descriptions for some terms used in this

[1] https://www.oecd.org/going-digital/ai/principles/.

document. For presentational reasons, it is helpful to split these terms into a number of categories, which build incrementally on each other.

Terms that are generally relevant include:

- An *assurance argument*, or *assurance case*, makes a claim about some property, for example, a system is acceptably safe (for a given application in a given operational domain), and provides arguments and evidence to support that claim [96]. Many regulatory domains require the use of assurance cases or, depending on their focus, safety cases.
- A *platform* delivers an end user capability during normal operation. It is, typically speaking, an individual vehicle rather than, for example, a swarm of cooperating vehicles or the control logic for vehicle navigation. The same general level applies to autonomous systems that are not vehicle-based: for example, a platform to support medical diagnosis may include patient records, a scanner and communication networks, as well as an autonomous decision-making algorithm. A key concept is that the platform can, and generally would, include elements that are developed using traditional approaches.
- An *autonomous system* can be viewed from multiple levels of abstraction. For example, it could be viewed as a platform (e.g. a self-driving car) or as a computation with an associated autonomy architecture (e.g. a component that provides the locations of pedestrians in an image). Depending on the domain, either use may be common. This is why, within this document, the highest-level framework is titled "platform" rather than "system". Nevertheless, in many cases the terms "platform" and "system" may be viewed as interchangeable.
- An *algorithm* is a well-defined procedure that implements, possibly indirectly, aspects of a system's behaviour. For the purposes of this document, generally speaking, an algorithm for autonomous aspects would be a single implementation developed using an ML technique, for example: a Neural Network (NN); a Support Vector Machine (SVM); or a random forest. This includes any bespoke data processing on inputs and outputs, such as feature extraction from images to be classified or normalisation of output values. A platform may include multiple algorithms.
- A *computation* is the physical embodiment of an algorithm. In some contexts, the words are largely interchangeable. A key distinction is that computation includes considerations related to supporting software (e.g. ML development frameworks) and to computational hardware; neither of these is included within "algorithm".

Terms that are most relevant to the computation-level framework include:

- For supervised and unsupervised learning, a *data set* is used to train, test and verify an algorithm using ML techniques. The part of the data set used to develop the algorithm is referred to as *training data*; the part used for testing is *test data*. Both training data and test data are used by the development team. A separate, possibly overlapping, data set, termed *verification data* may be used for assurance, independent of that team. Note that these definitions apply when the data is used as part of a pre-deployment training and development phase, as well as when there is continual learning.
- Reinforcement Learning (RL) involves an *agent*, which may be in a variety of different *states* within an environment. The agent can take a number of *actions*. Some actions may not be possible in some states. Taking a particular action, when in a specific state, yields a *reward*, typically by affecting the environment. The reward may be observed directly, or indirectly; it may be deterministic or stochastic in nature.

- Data sets are made up of a number of *samples* (e.g. images from a camera). Each sample comprises a number of *features* (e.g. the colour of a given pixel in that scene). The collection of features defines the *input domain*.
- Providing a computation with an *input* (or, during development, a sample) results in an *output*. This description includes cases where multiple samples as used (e.g. streaming data) and cases where the output is multi-dimensional (e.g. a vector of class-membership probabilities).
- A platform, or algorithm, is in *operational use* when it is being used for its intended purpose; that is, when its outputs have real-world consequences. Note that it is possible for learning to continue whilst a platform, or algorithm, is in operational use.
- There may be several *variants* of an algorithm, for example, to adapt to local legislation and practices. It may be possible to share some elements of assurance evidence across multiple variants.

Terms that are most relevant to the autonomy architecture-level framework include:

- After its initial operational use, an algorithm may undergo *adaptation*. This term also includes other types of change, made after initial operational use. Examples include changes to supporting software frameworks and computational hardware. It may be possible to use some elements of pre-adaptation assurance evidence to support post-adaptation arguments. The adaptation process itself would also be expected to be assured.
- An *adversary* is someone who deliberately aims to cause harm, by exploiting the autonomous nature of the platform, or by exploiting the associated autonomy-enabling technologies (like AI and ML).

Terms that are most relevant to the platform-level framework include:

- A platform operates in an *environment*. For example, a self-driving car might operate on public roads and a medical diagnosis application might operate within a clinical setting with links to other information systems (e.g. those containing patient records). Note that, within this document, "environment" is also used in a more general sense, for example, when referring to a software development environment.

1.5 Document Structure

The remainder of this document is structured as follows:

- Section 2 describes the computation-level framework.
- Section 3 discusses computation-level objectives.
- Section 4 describes the autonomy architecture-level framework.
- Section 5 discusses autonomy architecture-level objectives.
- Section 6 describes the platform-level framework.
- Section 7 discusses platform-level objectives.
- Section 8 discusses issues related to tailoring of the objectives.

- Section 9 contains a summary list of objectives.
- Appendix A provides an illustration of how tailoring may be applied to objectives in the computation-level framework.
- Appendix B provides a list of known issues, which are intended to be resolved in future versions.
- Appendix C provides a list of abbreviations.
- Appendix D provides a list of references.
- Appendix E provides a list of contributors.

2 Computation-Level Framework: Description

This section describes the framework adopted by the SASWG for computation-level considerations, which is based on the one presented by Faria in [28]. The justification for adopting this framework is provided in [89].

The framework consists of five projections, each of which views the computation's properties along a different axis. The projections are not intended to be strictly independent: they provide different ways of viewing the computation, in order to elicit associated objectives. To facilitate discussion, the projections have been arranged in an approximate hierarchical order, working from more abstract to more concrete considerations, specifically: experience; task; algorithm; software; and hardware. Each of these projections is considered, in turn, in the following subsections. The section concludes with a brief tabular summary of the entire framework.

2.1 Projections

2.1.1 Experience

This projection focuses on the data set (or the RL environment) used to train and develop the algorithm. When relevant, this also includes training that continues during operational use, based on the inputs provided by the system's experiences.

It includes consideration of how the data was generated, or collected, as well as the use of pre-existing data sets and the nature of any preprocessing activities (e.g. to synthesise missing values). It may also require consideration of wider issues, such as bias and privacy, and any safety implications of these. It also encompasses whether the training data is suitably representative of data that is observed (or expected to be observed) during operational use; this includes consideration of the environment(s) associated with the training data. The type of configuration management applied to the data is also relevant within this projection. Similar considerations apply to the RL training environment.

2.1.2 Task

This projection focuses on the performance of the computation. As such, it is mainly concerned with requirements, that is, what the system requires from the algorithm.

It includes the metrics that are used to measure performance, as well as the performance threshold required to allow the algorithm to be used safely within a system (which may depend on the intended operating environment). Items typically used to measure the performance of ML-based computations, like accuracy, precision and recall are relevant here although, by themselves, they may not be sufficient. There may, for example, be a need to provide confidence in an algorithm's output. Additionally, there may be a need to demonstrate some non-functional characteristics (e.g. an output will always be provided within a given time).

2.1.3 Algorithm

This projection focuses on the choice of algorithm type, for example, whether an NN, a SVM, a random forest, or some other approach is used. As such, it is mainly concerned with providing justification for

decisions relating to the chosen implementation.

It includes the choice of any hyper-parameters associated with the algorithm: for example, the structure of, and activation function used within, a NN, or the number of trees in a random forest. It also includes decisions related to the training process: for example, the number of training epochs that are used, or the stopping condition that is implemented.

2.1.4 Software

This projection focuses on the software instantiation of the algorithm; that is, the translation of a design, mathematics or pseudo-code into a form that can be directly executed on computational hardware. More specifically, this projection is concerned with whether the implementation is a valid representation of the algorithm.

The projection includes the choice of programming language. It also includes the choice of software libraries used to support the development and operational implementation of an algorithm. Tools used to support software development and verification are also captured in this projection.

The choice of, for example, programming language and supporting tools may be different during training than in operational use. Hence, it is convenient to consider this projection twice: once from a training and development perspective and once from the perspective of operational use of the algorithm.

Many of the considerations relevant for this projection are adequately addressed by existing software safety standards.

2.1.5 Hardware

This projection focuses on computational hardware. It includes consideration of the type of hardware, for example: Central Processing Unit (CPU); Graphical Processing Unit (GPU); Tensor Processing Unit (TPU); Field Programmable Gate Array (FPGA). It also includes whether this hardware is dedicated to one algorithm or whether it is used to support multiple algorithms (or multiple system features, including non-AI ones).

As with the previous projection, it is convenient to consider the hardware projection from both development and operational use perspectives.

Like the previous projection, many of the considerations relevant for this projection are adequately addressed by existing (computational) hardware safety standards.

2.2 Summary

Table 2 provides a brief summary of the five projections in the computation-level framework that has been adopted by the SASWG.

Table 2: Summary of Projections Within the Computation-Level Framework

Projection	Outline
Experience	Focused on the data (or environment) that is available to train (or develop) the algorithm
Task	Focused on the performance of the implemented algorithm; emphasizes requirements
Algorithm	Focused on the type of algorithm that is used; emphasizes implementation
Software	Focused on the software used to develop the algorithm and, separately, support its operational use
Hardware	Focused on the computational hardware that is used, both for development and for operational use

This page is intentionally blank

3 Computation-Level Framework: Objectives

This section lists the objectives associated with each projection of the computation-level framework. Supporting justification for these objectives is provided in [89].

3.1 Experience

The experience projection is focused on the training data (or in the case of RL the training environment) that is used to develop the algorithm. This data is crucially important because it encodes the requirements that the algorithm has to satisfy. Unfortunately, this encoding is implicit, in the form of the desired input-output relationship, so it cannot be directly examined. Hence, assurance that the algorithm's behaviour will be appropriate has to include aspects relating to the data.

There are five objectives associated with this projection.

COM1-1: **Data is acquired and controlled appropriately.**

Discussion: Data is a key part of an ML approach. Consequently, any assurance argument that addresses the ML-produced algorithm also has to address the data used to support its development. More particularly, if the source of the training, test and verification data cannot be adequately defined, or if this source is not appropriate for the intended use, then it will be difficult to produce a compelling assurance argument.

Examples: The first part of this objective relates to the way the data is acquired. For example, this could involve observing a natural process over which little control can be applied, or it could involve controlled trials; alternatively, it could involve the use of a synthetic environment from which training data is generated.

Ideally, the data would be acquired in a controlled manner using a documented process, which takes account of the prevailing environmental features (e.g. weather, system architecture) during collection. Changes to the acquisition method would be formally managed. Also, any software used to support data acquisition would be shown to be correct. In some ways, these considerations mirror those related to the use of Product Service History (PSH) in the aviation domain [15].

For RL, data is generated interactively, by taking actions in an environment and observing the associated reward. Consequently, in an RL context, this objective is focused on the link between the environment and the learning agent. More specifically, in RL the agent learns by exploring the rewards gained by taking particular actions when in specific states. Ensuring there is appropriate investigation of the combined space of states and actions is a notable challenge, especially when rewards are stochastic in nature, so that the same state-action pairing needs to be repeated multiple times. Approaches that provide guarantees about this investigation may be helpful, especially if learning continues during operational use [63].

If a complete data set is acquired from an external party then care should be taken to ensure that it has not been subject to "Data Poisoning"; for example, the addition of a small number of maliciously crafted samples can create a backdoor that could be exploited by an adversary [16]. The same techniques used to confirm the authenticity of information downloaded from the Internet (e.g. checksums) may be helpful here. When using data from an external party, care also needs to be taken to ensure it is not accidentally flawed, for example, because of translation issues (e.g, through different use of common terms like "speed").

Regardless of how the data is acquired there is also a need to analyse and, if possible, quantify uncertainty.

This may arise, for example, from sensor noise when measuring samples. Another potential source is labelling uncertainty: for example, should a person walking next to a bicycle be classified as a pedestrian or a cyclist? This can be an issue if labelling is conducted by a team of humans [21]. More generally, there may be benefit in using approaches that can identify data samples that are likely to be labelled incorrectly [70].

The second part of this objective relates to control of the data, or the training environment. More specifically, the data (or training environment) would be expected to be subject to some form of configuration management process, which protects it from accidental or unauthorised changes. Standard configuration management tools are likely to be suitable for this purpose, although the properties of training data (e.g. its volume and its change velocity) may mean they are not optimal.

Note that algorithms featuring online learning will continue to receive training data during operational use. This indicates safeguards may need to be included so that only suitable data is used for learning purposes. These could, for example, check that inputs are sufficiently similar to those that have been seen before, either because they were included in the original, pre-deployment training data or because they have previously been observed in operational use; these two conditions allow gradual expansion of the range of suitable data as the algorithm learns. The notion of "sufficient similarity" bears some relation to the concept of distribution shift[2] [67], but here it involves comparing a single sample with a distribution, rather than the more typical case of comparing two distributions.

COM1-2: Training data pre-processing methods do not introduce errors.

Discussion: Even if data has been collected in a controlled manner (as indicated by Objective COM1-1), it does not necessarily follow that the data is ready for training an algorithm. In many ML applications, there is a step between acquiring data and having data in a form suitable for algorithm development. This step generally involves pre-processing the data. It could include, for example, detecting missing data items and replacing them with suitable surrogate values; it could also include normalising features. Since pre-processing directly affects the data used to develop the algorithm, any errors in pre-processing could undermine a computation-level assurance argument.

Pre-processing is likely to occur during operational use as well. For example, raw sensor readings are likely to be processed in some way before being provided as inputs to an algorithm. Although it is important, this type of pre-processing is considered to be an autonomy architecture-level issue, rather than a computation-level issue.

Examples: Often, pre-processing would be expected to be achieved using traditional types of software. This means that the approaches used to provide assurance for traditional software are also applicable here (e.g. [82]). In addition, pre-processing software bears some similarities to tools used to support traditional software development. One way of achieving confidence that those tools do not introduce errors is the notion of Tool Qualification. Hence, concepts like Tool Qualification Levels (TQLs) [83] are also relevant. In that specific context, pre-processing software can be considered as a tool that can introduce errors into the operational software (rather than a tool that can only fail to detect an error).

Pre-processing may include techniques such as data imputation to create realistic missing data values, e.g. when using real world data for training, which may not cover all required scenarios. The impact of this on the behaviour of the ML should be considered [85].

In the case of RL, pre-processing of data may be used to simplify the observed environment into a suitable

[2] Distribution shift is also considered in Objective COM1-4.

representation. Using Convolutional Neural Networks (CNNs) to extract information from images is one example. This differs from the pre-processing typically used for supervised learning, which tends to use a combination of traditional software and statistical techniques. More particularly, if the CNN is an integral part of the RL implementation then, within the terms of this objective, it does not count as pre-processing. Instead, it is more appropriately considered as part of the algorithm.

Other, non-CNN-based, approaches may simplify observations of an environment to support RL. For example, a continuous variable may be reduced to a number of discrete bands. In such cases, this objective seeks confirmation that the reduction is conducted in an error-free manner.

The approaches used for traditional software should ensure that pre-processing software is under appropriate control, including archive retrieval, if necessary. Similarly, the control measures (discussed under Objective COM1-1) should ensure that the "raw" (i.e. not pre-processed) data remains available, should this be required.

COM1-3: Data captures the required algorithm behaviour.

Discussion: Even if the data is suitable for training *an* algorithm, it does not necessarily follow that it is suitable for training *a particular* algorithm. Fundamentally, training data encodes the requirements that the algorithm's behaviour is intended to satisfy, so a data set suitable for training an algorithm to recognise road signs will not be suitable for training an algorithm to recognise human emotions. Unfortunately, the data does not encode the requirements in an explicit manner. Consequently, these requirements cannot be directly reviewed by stakeholders or algorithm developers. This means an argument needs to be made as to why a particular set of data is appropriate for a specific algorithmic behaviour.

In cases where it is difficult to capture all the required domain knowledge in the training data, there may be value in using approaches that allow logical constraints to be incorporated during the training [30]. These considerations relate to Objective COM3-1.

Examples: Exploratory data analysis [99] would be a sensible first step in understanding the properties of a data set and, consequently, its applicability to a particular algorithm. This could include plotting marginal distributions of each feature, calculating two-way correlation coefficients and producing pairwise scatter plots for different pairs of features [44]. It can also be helpful to identify typical and outlier samples (possibly on a class-by-class basis, for classification problems) [10]. The concept of outlier samples can also be extended to include rare situations, the presence (or absence) of which is likely to be informative. Other automated techniques that allow the detection of rare subclasses (e.g. [73]) can also be useful. The insights gained from this work can inform discussions involving domain experts and ML specialists, as well as supporting an assurance argument.

In essence, part of this objective is about understanding the relationship between the training data and the algorithm's application domain. This could be informed by previous uses of the data [33]. In some cases, the relationship can be quite subtle. Consider, for example, an algorithm intended to recognise British traffic signs. Despite the restriction of the algorithm's domain to British traffic signs it may be appropriate to train it on British, Continental European, and worldwide, traffic signs. Along with apparent economic benefits (e.g. if the same algorithm could subsequently be employed in different markets), this approach could increase algorithm robustness.

A related, but more extreme, version of this general approach is transfer learning, where a pre-trained network is specialised, or fine-tuned, for a specific task. Given the potentially significant costs associated with training, this can offer significant benefits. It is often used for image recognition tasks. In this case,

the nature of the pre-trained network would be expected to be discussed in any assurance argument. This discussion would also be expected to address the possibility of the pre-trained network introducing a backdoor, or otherwise undesirable, behaviour [37].

There are three data-related aspects that govern the behaviour of an agent trained using RL but are address by other objectives: the possible states the agent can be in; the actions available to the agent; and the reward associated with conducting a specific action when in a particular state. States and actions are related to implementation and hence are addressed via objectives in the algorithm projection, specifically Objective COM3-1. The reward is related to requirements and hence is addressed via the task projection, specifically Objective COM2-3.

In the case of RL, this objective relates to the environment, specifically how it responds to agent actions. It is particularly relevant when a synthetic environment is used to train the agent. In particular, there is a danger that inaccuracies in the environment may yield inappropriate behaviour. Note that this objective is concerned with how the environment responds to actions. There is a distinction between the environment presenting an incorrect reward, which is addressed via this objective, and there being an inaccurate link between rewards and desired behaviour, which is addressed via the task projection.

COM1-4: Adverse effects arising from distribution shift are protected against.

Discussion: Distribution shift occurs when the operational inputs provided to the algorithm differ, in a statistically or semantically meaningful sense, from the samples used during development. This is important because, in addition to encoding requirements, training data also captures information relating to the domain in which the algorithm can safely be used.

Examples: There are a number of different types of distribution shift, including cases where the inputs change and cases where the input-output relationship changes [67]. The possibility of each type of distribution shift would be expected to be considered and appropriate protection provided. Any detection of distribution shift is statistical in nature. This means that a balance needs to be struck between the possibility of false alarms (i.e. false positives) and the possibility of false negatives; this balance may be a hard wired feature, or it may be tuneable.

Since the algorithm is meant to generalise the input-output mapping of the training data, there are dangers in taking too rigid a statistical approach. More specifically, the inputs seen by the algorithm during operational use are not expected to be precisely the same as those used during training. Consequently, the algorithm may be better suited to providing operational predictions for inputs that lie inside (i.e. within the convex hull of) the training data than to providing predictions for inputs that lie outside the training data.

There are other reasons why a naive comparison of training and operational distributions is likely to be inappropriate. For example, to increase robustness the training data may be supplemented with adversarial examples [36]. Additionally, the frequency of "rare but important" examples may be artificially increased within the training data by generating synthetic data.

Also note that, in some cases, data can be statistically similar, but semantically different. Consider a distribution with zero mean, that is symmetric about this value; swapping the sign on all samples would produce a data set that was statistically similar, but semantically different. This possibility should be considered and, if appropriate, protected against.

There is also the possibility that data is semantically similar but, statistically different. Obviously, this

depends on the specific nature of the statistical test that is being used. However, there are examples of statistical differences being found between training and test data, but these differences having no discernible effect on algorithm performance [78].

Conceptually, there is a similarity between detecting rare items in the training data and identifying distribution shift. In many cases, techniques related to rarity detection (e.g. [10], [73]) can be modified to help detect distribution shift.

In an RL context, differences between training and operational data may occur because operational use is in a different environment to the one used in training. This could be detected by platform-level or autonomy architecture-level functions that monitor the operational environment. Differences between training and operational data may also arise if the operational reward function is different to the training one. This could be detected by monitoring the operational reward matrix and determining if this is significantly different from the one established during training. Although the implementation details would differ, this approach could be used for cases where the reward matrix is fixed during operational use, as well as for cases where learning continues during operational use.

From a computation-level perspective, the focus is on detecting distribution shift. Appropriate responses are best enacted at other framework levels (see, for example, Objective ARC1-2 from the autonomy architecture level).

COM1-5: **Processing methods on inputs and outputs used during operation do not introduce errors.**

Discussion: During operation the algorithm may transform data and signals on input and output, for example, edge detection and filters on images to be classified or normalisation of output values. Assurance should be provided that methods used do not introduce errors.

Examples: Some classification algorithms, such as the image classifier You Only Look Once (YOLO) [48] use methods for processing inputs and outputs, such as for edge enhancement algorithms, de-noising or feature extraction. YOLO provides a confidence value which is a combination of confidence of a bounding box containing an object and what that object is.

Some algorithms require normalisation of data, e.g., [85] which normalises medical data on different scales between 0 and 1 in order to both train the algorithm and process medical data during operation.

Assurance of these methods in the context of the developed algorithm should be provided where appropriate. If the methods are implemented using traditional software, then traditional software assurance approaches can be used (see Objective COM4-1).

3.2 Task

This projection is focused on the performance of the algorithm; that is, whether it can be safely used within the intended system context. As with traditional safety-related software, requirements would be expected to be passed down from the system level. Whilst there are some similarities, there are also some differences between evidence that traditional software satisfies its requirements and the corresponding evidence for algorithms developed using ML techniques. This evidence is, obviously, an important component of an assurance argument.

There are seven objectives associated with this projection.

COM2-1: **Functional requirements imposed on the algorithm are defined and satisfied.**

Discussion: Ultimately, the algorithm is expected to be used as part of a system. In order to perform as part of that system, the algorithm will have to satisfy a number of functional requirements. For example, rather than returning a single prediction, it could be required to return a probability vector that expresses the likelihood of an input belonging to each of a collection of classes. Alternatively, or additionally, it may be required to provide some measure of confidence in its prediction.

Examples: Traditional software testing techniques may be helpful in demonstrating some of an algorithm's functional properties. Depending on the criticality of the algorithm, these may involve formal review of test cases and tests being independently conducted (and witnessed).

In ML approaches, functional requirements are not systematically decomposed into low-level requirements that can be unambiguously coded against [11]. This means that traditional software testing techniques should be supplemented by other types of testing. These could include the types of test that are more traditionally seen at the system level.

Note that the performance of the algorithm is considered in Objective COM2-3.

COM2-2: **Non-functional requirements imposed on the algorithm are defined and satisfied.**

Discussion: The algorithm will be embodied in a wider system. This means it will have to satisfy some non-functional requirements. For example, it may be required to produce an answer within a given time.

Examples: Similar to Objective COM2-1, traditional software testing techniques may be helpful in demonstrating some of an algorithm's non-functional properties, but they should be supplemented by other forms of testing. This could be informed by a set of standard scenarios [17]. The choice of scenarios would be expected to be described and justified as part of the computation's assurance argument. Care needs to be taken to ensure that the selection of scenarios is suitable for the intended use. However, experience from other areas suggests this may be possible: for example, a standard set of situations is used when testing an aircraft flight simulator [26]. The notion of situation coverage could also inform this decision [3].

However, AS often use novel technology, and it may be that there are no established techniques for measuring a given non-functional property. For example, there is a distinction between achieving human-like performance and analysing inputs in a human-like way. In particular, deep neural networks can often exploit features (specifically, shortcuts) that are not apparent to humans [34]. If human-like analysis (rather than human-like performance) is required then it should be explicitly stated as a non-functional requirement.

A key non-functional requirement for traditional safety-related software is execution time. Consequently, significant effort is often expended measuring (or, in some cases, calculating) the Worst Case Execution Time (WCET). Many algorithms developed using ML techniques will apply exactly the same computational process, regardless of the input; this is the case for NNs, for example. This means that establishing WCET for these algorithms may be no worse than is the case for traditional software.

There are, however, uses of AI for which this is unlikely to be the case; route planning is a possible example. In such circumstances, WCET estimates would be expected to be guided by both knowledge of the algorithm and the likely ways in which it will be used. There are also adversarial examples where carefully crafted inputs can significantly change the resources required by an algorithm to process an input [91].

COM2-3: **Algorithm performance is measured objectively.**

Discussion: Fundamentally, this objective is about *how* performance is measured. The question of what level of performance is required is, essentially, a system-level concern. The way that performance is measured should directly relate to the algorithm's requirements, passed down from the system level.

Typically, an algorithm would be expected to achieve at least a minimum level of performance. For classification algorithms, this often involves measuring properties like precision, recall or accuracy. These are measured using a validation data set, that has been withheld from the training process for this purpose. Note, however, that this implies a statistical, rather than an absolute, measure of correctness.

Examples: Although they can be useful, there are limits to what can be gained from measuring properties like precision, recall and accuracy. For example, the existence of adversarial inputs (i.e. inputs that are very close to a sample in the training data, but which are confidently predicted as belonging to a different class) for well-performing algorithms (e.g. [95]) indicates these measures are unlikely to capture all relevant

features of algorithm behaviour.

Special care needs to be taken if the data set is imbalanced; for example, if in a classification problem a large proportion of the data falls within a single class. In such cases poorly chosen performance measures can be dominated by the algorithm's performance on the large class [39].

Although it increases costs, perhaps significantly, there is benefit in using an entirely independent data set for at least part of the performance measuring process. This helps protect against shortcut learning, where training exploits features that are present in the training data but are unlikely to be present in the operational data [34].

Whilst they are not amenable to every type of algorithm, and every type of task, in situations where they can be used, approaches based on formal methods (e.g. [53], [74]) can be especially powerful.

In some cases, it may be appropriate, or necessary, to consider algorithm fairness. This could require changes to the training data (pre-processing), alterations to the model training approach (in-processing) or changes to baseline model outputs (post-processing). A variety of algorithms have been developed to help detect and protect against unintentional bias [13].

In the context of RL, desired performance is encapsulated by the reward function. This needs to encourage the desired behaviour without, for example, leaving any loopholes for reward hacking [5]. This is a particular concern if a synthetic environment is used to determine the rewards. In such cases, the training process may identify and exploit limitations in that environment, which will not be present in the real world. In theory, it may be possible to protect against this outcome by using multiple synthetic environments, for example, one for training and another for evaluation. This mirrors the use of separate data for training and verification in the case of supervised learning. However, developing and maintaining separate synthetic environments is likely to be expensive. Comparing results from the separate environments might also be a challenge. An alternative might be to ensure the trained behaviour is explainable, although this may require a task-specific approach [43].

COM2-4: **Performance boundaries are established and complied with.**

Discussion: Depending on the nature of the algorithm's input domain, there may be some combinations of features that do not represent a valid input. Consider, for example, the classic Iris data set that is available from the University of California, Irvine (UCI) Machine Learning Repository [23]. This relates information on specific Iris features (e.g. petal sizes) to the associated species of plant. In this case, there is some relationship between the length and width of an Iris petal. Hence, even though it would fall inside an algorithm's input domain, it would be unreasonable to expect the algorithm to predict Iris species for a very wide, very short petal (since this combination does not occur in nature).

More generally, as noted above (specifically, in the experience projection), the training, test and verification data encodes information about the region of applicability for the algorithm. Since this data covers the scope over which the algorithm has been developed and tested, this also establishes boundaries (albeit fuzzy ones) within which the measured performance may, in some sense, be expected.

Note that the question of what response should be provided if the algorithm receives an invalid input is best addressed at the autonomy architecture level. If appropriate, this response could also be extended to realistic, but very unlikely, points in the input domain.

Examples: The approach to this objective is similar to that of Objective COM1-4. However, the two objectives differ in that COM1-4 adopts a more theoretical, data-focused approach, whereas this objective

takes greater consideration of the wider system and application domain. Ensuring that the platform can tolerate operational inputs that are outside the algorithm's performance boundary is achieved via Objective ARC1-2 in the autonomy architecture-level framework.

COM2-5: **The algorithm is verified with an appropriate level of coverage.**

Discussion: Branch coverage, statement coverage and Modified Condition / Decision Coverage (MC/DC) [18] are well established measures of test coverage for traditional software [82]. These measures, and other related ones, allow judgements to be made regarding the sufficiency of a test set (typically, one based on the software requirements). More colloquially, in some sense they allow an informed decision to be made that sufficient testing has been achieved.

From the perspective of an algorithm developed using ML approaches there is a similar need to provide objective evidence that a sufficient level of testing has been completed.

Examples: Algorithm-related coverage measures would be expected to consider two perspectives: one focused on the input domain; and one focused on the internal features of the algorithm.

Approaches that address the former perspective (i.e. the input domain) are likely to be common across all ML approaches. These may consider the input domain it its raw form (i.e. as measured by system-level sensors); alternatively, they may consider a simpler representation of this data, for example, one developed using Principal Component Analysis (PCA). In some cases, they may also involve approach-specific characteristics, for example, considering the feature space represented by a particular layer in a neural network. However, since this space is dependent on the training data, by themselves these types of consideration would not be sufficient. One option may be to consider characteristic sets that categorise input scenarios (e.g. by weather, road type, traffic level) and then establish a form of combinatorial coverage across these sets [17].

Approaches that address the latter perspective (i.e. algorithm internal features) are likely to be specific to a particular type of algorithm, or family of algorithms. For example, in the case of random forests, a measure of how many branches are covered in each tree by the verification data may be informative. Understanding how this value varies across individual trees in the forest (and, especially, the minimum value) is also likely to be informative. In the case of neural networks, measures based on neuron activations may offer helpful insights [94], especially those based on activations of combinations of (rather than individual) neurons. However, these types of coverage are not directly equivalent to concepts like statement, branch and MC/DC coverage for traditional software [8]. In addition, hardware that is optimised for neural network training and inference may not allow easy access to the necessary information.

Another potentially useful approach to establishing test coverage is that of negation. Consider, for example, a pedestrian detection function. This could be tested with images that: should be classified as containing pedestrians; might be classified as containing pedestrians; should definitely not be classified as containing pedestrians. More generally, the latter class (which can be viewed as negating the requirement) can be an easy way of generating powerful test cases.

Within the context of RL, in a simple sense, coverage relates to the number of state-action pairs that have been attempted. Since actions describe available choices, this covers the input perspective. Since the reward matrix describes behaviour, and is structured around state-action pairs, this also covers the algorithm internals. Note that rewards may be stochastic, in which case a particular state-action pairing may need to be enacted multiple times.

Repeated state-action invocations may also be required if the environment changes over time. For fast-changing environments, this should be addressed as part of the training and verification approaches. For static, or slow-changing, environments, the effect of changes is best addressed at the autonomy architecture level, specifically via Objective ARC1-2.

COM2-6: The test environment is appropriate.

Discussion: Since test results will form part of the assurance argument, there must be confidence in the test environment that produced these results. This environment includes physical assets, software code and test cases. In addition, the test environment would be expected to be under configuration management, so that it could not be arbitrarily changed.

Examples: Much of this objective would be satisfied by traditional approaches to the development of safety-critical, or safety-related, software. However, it is possible, perhaps likely, that the test environment will include some representation of the real world, for example, because the test environment includes a representation of the system within which the algorithm is embodied or because it includes a representation of the real world process that generated the training data. In either case, there is a need to validate the representation of the real world entity, or process. This could be achieved using standard approaches for simulation validation [88], potentially supported by a standard set of scenarios (as discussed in Objective COM2-2).

From an RL perspective, there is likely to be a strong connection between the environment used for training and the environment used for testing. As discussed in Objective COM2-3, there may be benefits in using separate environments for these activities, although this may prove expensive. More generally, approaches taken to satisfy objectives that demonstrate the appropriateness of the training environment are likely to be applicable to this objective as well. Typical approaches to simulation verification [88] are also likely to be beneficial.

COM2-7: Each algorithm variant is tested appropriately.

Discussion: Algorithm variants provide a way to develop multiple computations that share many aspects but differ sufficiently to facilitate adherence to local legislation, or local practices. One approach might be developing a common computation, which is then specialised differently for each variant. This can save training time. It might also save testing time, as well.

The notion of variants might also be useful when considering adaptations (e.g. as part of maintenance and development). In particular, in some circumstances, later computations may be considered as variants of earlier ones. As before, this could save training time and, potentially, testing time.

Examples: The question of how much testing of one variant can be read across into another is, inevitably, situation specific. Nevertheless, the use of algorithm variants has some similarity to the notion of software product line development [76].

It is also related to the level of confidence that can be gained from, for example, a pre-trained network that has been subject to some testing. If the testing of the pre-trained network is closely related to the intended operational domain then it may be possible to gain considerable confidence. Conversely, if there are significant differences between the earlier testing and the intended domain then little confidence may be gained from earlier testing.

Similar considerations also apply to cases where analysis of operational inputs is used to create new, updated algorithm variants. Whilst some confidence may be gained from the testing of earlier variants, this will inevitably degrade as more variants are produced. It may be necessary to have a minimal set of test cases that are always run, to check that an update (to produce a new variant) has not undermined any critical properties of the algorithm. This has similarities to the notion of regression testing for traditional software.

3.3 Algorithm

Different types of algorithm have different strengths and weaknesses. Hence, the type of algorithm that is used has to be suitable for the task in hand. In particular, the choice of algorithm should be based on the requirements it has to satisfy and the application domain; it should not be an arbitrary choice, nor should it be based solely on developer familiarity.

There are four objectives associated with this projection.

COM3-1: **An appropriate algorithm type is used.**

Discussion: A variety of algorithm types are available, including NNs, random forests, SVMs and RL. There are further divisions within each type. For example, the NN family includes: Deep Neural Networks (DNNs), which feature hidden layers of neurons; Recurrent Neural Networks (RNNs), which have loops within the network structure; and CNNs, which have features designed for image classification.

In most cases, the ML process that produces these algorithms is controlled by hyper-parameters. These may include, for example: the way the available data is split between development and verification activities; the number of layers, and the number of neurons in each layer, of a NN; the neuron activation function; and dropout rates [93].

Examples: Any computation-level assurance argument would be expected to include justification for the chosen algorithm type and, also, any hyper-parameters that were used. This could include appropriately-referenced theoretical arguments, for example, arguing that the available literature demonstrates the utility of CNNs in image classification tasks [57].

Empirical arguments are also likely to be required; for example, the performance of a number of different algorithms could be investigated in order to justify the choice of the final algorithm. Likewise, a structured investigation of the effect of different hyper-parameter settings might be expected.

In cases where there is specific domain knowledge that cannot easily be captured via the training data there may be benefit in adopting an approach that allows logical constraints to be included in the training process [30]. More generally, the relationship between the intended behaviour and the loss function used in training should be considered.

The characterisation of states and actions (which structure an agent's understanding of the environment and its possible behaviours) is an important part of the appropriateness of an RL algorithm. If a CNN is used as part of a Deep Reinforcement Learning (DRL) process then this objective also covers the appropriateness of the parameters that define the CNN and associated training process.

COM3-2: **Typical errors are identified and protected against.**

Discussion: Broadly speaking, there are four different places where errors can arise: within the training (or verification) data; within the way individual steps are composed to form an algorithm; within a supporting library or tool kit; and within the execution environment [104]. These approximately map to the experience, algorithm, software and hardware projections, respectively. Consequently, this objective is concerned with issues relating to ML approaches in general, the class of algorithm and hyper-parameter choices.

Examples: Comparatively, there is much less experience as to what typical errors may be in algorithms trained using ML techniques than for traditional safety-related software. Nevertheless, there are some indications of things that should be avoided [40]. One example is over-fitting, where the algorithm learns the specific data rather than the generic relationship. Another is data leakage, where the algorithm has access to information that should not legitimately be available. Adversarial examples may also be a typical error for large-dimensional data sets [35]. Another typical error may be the under-representation of rare events in the training data [102]. For classification tasks, errors in labels may be another.

Whilst some indicative typical errors are beginning to emerge, it is less clear how these errors can be detected and corrected. In the specific case of over-fitting, it appears that groups of neurons that fire for a single class may be indicative of memorising the specific training data, rather than generalisation [66]. There is also some work that can help find incorrectly labelled samples [70].

Although supporting libraries and tool kits can simplify the use of ML approaches, their nature can make it difficult to detect errors. For example, many learning processes have stochastic features; this means that bugs are hard to reproduce and, furthermore, success criteria are statistical in nature (which means incorrect code can appear to be working) [104].

Bit flip errors in hardware may be considered a typical error for all types of computing. The way that network weights are stored can significantly alter the importance of these, rare but potentially important, events. In particular, networks where weights are quantised, with values stored as indices into a table appear more robust to random bit flips than networks where weights are stored using a traditional floating point representation [79].

In the context of RL, reward hacking, where the training process exploits loopholes in the environment is a typical error [5]: a large number of examples of this have been collected [56]. Another typical source of errors relates to exploration of the state-action space, for example, remembering state-action pairs that have significant negative effects on the environment [60].

COM3-3: **The algorithm's behaviour is explainable.**

Discussion: Algorithms developed using ML approaches do not feature the formal, traceable, hierarchical decomposition of requirements that is typical of traditional safety-related software [11]. This lack of traceable decomposition contributes to a lack of understanding regarding how a specific piece of algorithm behaviour contributes to the final output. Expressed another way, by following the mathematics it is easy to see *what* an algorithm is doing; it may be less easy to see *why* this results in desirable behaviour.

Examples: There are two main perspectives that should be considered when thinking about explaining behaviour [38]:

- Explaining a single output from an algorithm. A number of approaches have been proposed,

including: training simplified (human-understandable) models to represent the algorithm's behaviour for the input of interest [80]; and providing visual representations [41].

- Explaining algorithm behaviour in general. There is apparently less work in this area. It is notable that behaviour in general cannot be explained by looking at behaviour in even a large number of individual cases: the non-linear nature of many ML-developed algorithms means it is not appropriate to extrapolate from the specific to the general.

There is, in general, a tension between explainability and performance. The objective of explainable behaviour suggests a preference for low-complexity approaches but, in isolation, these approaches may not be able to achieve the required level of performance. Combining several algorithms, either in series or in parallel, may be a suitable way forward.

Depending on the way the algorithm is used, the need to explain the algorithm's behaviour, which is an important part of any computation-level assurance argument, may have to be balanced against the possible effects an explanation may have. Consider, for example, a medical diagnosis system that uses doctor's notes as one of many inputs. If the doctors were informed that using a particular word (e.g. "unusual") was a significant trigger for a particular decision from the algorithm, this may change the way they write their notes (which would be a form of distribution shift, so Objective COM1-4 is relevant). Whilst this is a platform-level consideration, it is informed by computation-level knowledge.

Techniques associated with explainable RL fall into two main categories, use of transparent algorithms and post-hoc explanations [43]. Examples of the former category include representation learning and hierarchical learning. Saliency maps are an example of the latter category. In addition, if a CNN is used as part of a DRL approach, then explainability of the CNN is also important.

Although the precise details are outside the scope of this document, it should be noted there may be ethical and legal factors that affect the extent to which an algorithm's behaviour has to be explained [100].

COM3-4: Post-incident analysis is supported.

Discussion: The process of air accident investigation is, arguably, one of the main reasons that air travel is comparatively safe. Given the relative immaturity of autonomous systems, analysis of incidents (including those that do not result in an accident) is likely to make a significant contribution to safety in this field. Consequently, the algorithm is expected to support post-incident analysis.

Examples: This objective is related to Objective COM3-3 in that explanation of a single result (or a small number of results) from the algorithm will be an important part of the post-incident analysis. However, sufficient information needs to be recorded to allow the algorithm's behaviour to be reconstructed after the incident. This may involve storing internal state information, including any data used to support non-deterministic choices within the algorithm.

Some aspects of this (e.g. provision of sufficient storage space and accessing recorded information) are autonomy architecture-level or platform-level issues. Other aspects may affect both the platform and the algorithm: for example, a requirement to support post-incident analysis for anything that has occurred sometime in the last 30 days may drive a different algorithm design to a requirement to support investigations over a 30-second period.

A computation-level assurance argument would be expected to demonstrate that post-incident analysis can be conducted. One way this may be achieved is by treating discoveries during development and testing

as pseudo-incidents and confirming that sufficient information was recorded to support post-incident analysis.

3.4 Software

Any algorithm will rely on software. Consequently, software needs to be considered in a computation-level assurance argument. The software associated with development of the algorithm is likely to be different to the software employed during operational use. Consequently, it is helpful to consider objectives in the software projection from both development and operational use perspectives.

There are two objectives associated with this projection.

COM4-1: **The software is developed and maintained using appropriate standards.**

Discussion: Even though supporting libraries, or tool kits, are available, at some point an algorithm will almost certainly rely on some traditional-style software (e.g. because this is what the supporting library is implemented in). Faults in this software have the potential to undermine an assurance argument.

Examples: Much of this objective is likely to be addressed through the use of an existing standard for safety-critical software development (e.g. [82]). This should help prevent typical errors (e.g. integer overflow) from being introduced. There are, however, a number of areas where an existing standard may not be straightforward to apply.

Firstly, generally speaking, supporting libraries and tool kits are not developed to such rigorous standards. There are several potential approaches to this challenge. For example, it may be possible to provide additional evidence that relates to the portion of the library or tool kit that is actually used. Alternatively, it may be possible to compare the results from different (independently developed) libraries [92]. It may also be possible to re-implement the ML algorithm from scratch (e.g. use a library to investigate multiple algorithms, then re-implement only the chosen one). Whatever approach is adopted, a computation-level assurance argument would be expected to provide a justification as to why any supporting library or tool kit is suitable.

Secondly, the pervasive nature of the library or tool kit means it is inappropriate to treat it as Software Of Uncertain Pedigree (SOUP) [50]. In particular, it is not possible to put the library or tool kit in a bounded, protected environment and carefully monitor the inputs and outputs to that environment.

Thirdly, rather than developing an algorithm from scratch, significant savings might be achieved by starting with a pre-trained model. However, there is a possibility that these models could include backdoors that cause the model to exhibit inappropriate behaviour in very specific circumstances [37]. The nature of these backdoors means it is unlikely that they will be discovered simply by running tests through the model. Consequently, any use of pre-trained models would be expected to be explicitly justified in any computation-level assurance argument. For example, pre-trained models should be obtained from trusted sources, using a distribution mechanism that provides strong guarantees on integrity.

COM4-2: **Software misbehaviour does not result in incorrect outputs from the algorithm.**

Discussion: Generally speaking, most safety-related systems that use software include protections against software failures or, equivalently, cases where the software does not behave as expected. This prevents

errors propagating through the system and allows restorative measures to be implemented (e.g. restarting an application). The key issue is that software misbehaviour is detected and responded to [77].

Examples: Algorithms developed using ML approaches do not fail (or misbehave) in the same way as traditional software. In particular, it is not apparent that all failures will be readily detectable from outside the algorithm. Hence, there may be benefit in including some form of Built-In Test (BIT) in the algorithm, which provides confidence that it is operating as expected [90].

Some algorithms may provide a measure of confidence associated with their output. That is, rather than simply classifying an image as a "cat", the information provided may be a 75% confidence the image is a "cat", a 13% confidence the image is a "dog", and so on. Whilst it may be helpful in some circumstances, this may not be sufficient to fully address this objective, not least because adversarial examples show NN can be confident in their output yet still wrong [95].

3.5 Hardware

In order to function, any algorithm will rely on computational hardware. Consequently, hardware needs to be considered in a computation-level assurance argument. The hardware associated with development of the algorithm may be different to the hardware employed during operational use. Consequently, it is helpful to consider objectives in the hardware projection from both development and operational use perspectives.

There are two objectives associated with this projection.

COM5-1: **Appropriate computational hardware standards are employed.**

Discussion: Similar to Objective COM4-1, ultimately, any algorithm will run on some form of computational hardware. This hardware needs to be considered in a computation-level assurance argument.

Examples: Again, similar to Objective COM4-1, much of this objective may be addressed by existing standards (e.g. [81]). In some cases this may be straightforward; in others, the specialist, complex nature of the hardware may pose challenges. For example, this hardware could include GPUs or TPUs, used for massively parallel calculations; alternatively, it may involve a complex System-on-Chip (SoC), featuring a combination of processor cores, GPUs (or TPUs) and bespoke components (e.g. video coders / decoders).

If novel, or complex, hardware is involved then it may be necessary to understand the extent to which the behaviour of this hardware is predictable. These considerations could be informed by recent experience with multi-core processors [27].

COM5-2: **Hardware misbehaviour does not result in incorrect outputs from the algorithm.**

Discussion: There are several reasons why computational hardware may not behave as expected; Single Event Upsets (SEUs) are one example. Another aspect, specific to algorithms developed using ML techniques, is differences in development hardware and operational hardware (which may mean the operational performance differs from what would be expected).

Examples: Any computation-level assurance argument would be expected to consider the possibility

of hardware misbehaviour and offer protections against it. This includes SEUs. It also includes the effect of different numerical precisions being used on development and operational hardware, as well as the possibility of non-deterministic behaviour on GPUs, even if the algorithm does not feature non-deterministic components [71].

Considerations relating to Size, Weight and Power (SWaP) may mean the computation hardware used to run the algorithm is also used for other purposes. In such cases, the assurance argument would be expected to demonstrate neither of these uses will interfere with the other. Standard approaches to partitioning are likely to be helpful.

Depending on the system, allocation of software to computational hardware may be fixed at design time or it may be dynamically allocated, possibly changing during operation. For example, some hardware can switch between 15 different types of operating mode [91]. In either case, a computation-level assurance argument would be expected to demonstrate that sufficient resources will be available to allow the algorithm to complete its processing within the expected amount of time. If a particular hardware configuration is required then steps should be taken to ensure this is initialised and maintained during operation.

4 Autonomy Architecture-Level Framework: Description

This section describes the framework adopted by the SASWG for autonomy architecture-level considerations. It is heavily based on the considerations discussed in [9]. The justification for adopting this framework is provided in [89]

The framework consists of three projections, each of which views the autonomy architecture's properties along a different axis. The projections are not intended to be strictly independent: they provide different ways of viewing the autonomy architecture, in order to elicit associated objectives. To facilitate discussion, the projections have been arranged in an approximate order, working from things more closely associated with an individual computation to more general considerations, specifically: tolerance; information provision; and adaptation. Each of these projections is considered, in turn, in the following subsections. The section concludes with a brief tabular summary of the entire framework.

4.1 Projections

4.1.1 Tolerance

This projection focuses on faults and failures, related to the computation, that the autonomy architecture must tolerate.

This covers situations where: the computation inputs are invalid (so, for example, they cannot be used for alternative, non-AI based approaches); the computation inputs are valid but they are outside the domain of the computation (so, limited confidence can be gained from previously-conducted testing and verification); and the computation output is invalid. It involves monitoring various aspects, including the health of sub-systems that provide computation inputs (e.g. vehicle-based sensors) and the internal properties of the computation.

4.1.2 Information Provision

This projection focuses on the way the autonomy architecture records and maintains information so it can be provided to relevant stakeholders.

This information may be used in a variety of ways, including: communicating with other entities in the operational environment (e.g. people and systems); supporting maintenance and further development of the computation algorithm; and facilitating post-incident analysis. Note that this projection is only concerned with making sure the required information is available and ready to be used. Actual use of the information is a platform-level responsibility.

4.1.3 Adaptation

This projection focuses on management and control of changes to the algorithm after its initial operational use. It also includes changes associated with, for example, supporting software frameworks and computational hardware. Specifically, it is concerned with updates that would not be produced by following the full engineering process associated with development of an algorithm using ML techniques. It includes, for example, considerations related to on-line learning and provision of regular (e.g. nightly) updates.

4.2 Summary

Table 3 provides a brief summary of the three projections in the autonomy architecture-level framework that has been adopted by the SASWG.

Table 3: Summary of Projections Within the Autonomy Architecture-Level Framework

Projection	Outline
Tolerance	Focused on tolerating faults and failures in computation-related items
Information Provision	Focused on recording and maintaining information for subsequent use
Adaptation	Focused on how updates to the algorithm (and associated software and hardware) are managed

5 Autonomy Architecture-Level Framework: Objectives

This section lists the objectives associated with each projection of the autonomy architecture-level framework.

5.1 Tolerance

In order to be used in a wider system, an autonomy-related computation (i.e. the software and hardware that embody an algorithm) needs to be able to cope with the consequences of real world usage. These may take the form of faults or failures, for example, of sensor systems that provide input data for the computation. Alternatively, they may occur if the autonomous system is being used outside its intended operational domain, through deliberate adversarial action, intentional user action or unanticipated environmental events.

Regardless of how they occur, these types of issue need to be handled in safe manner. Equivalently, they need to be tolerated. The autonomy architecture provides the means by which this is achieved.

There are five objectives associated with this projection.

ARC1-1: **Failures of sub-systems that provide computation inputs are tolerated.**

Discussion: This objective is concerned with failures in the wider system that may affect computation inputs, how these failures may be detected and how they may be responded to.

When talking about computation inputs, it can be helpful (as described in [9]) to distinguish between inputs received: when the model is used within the intended operational domain; when there are failures elsewhere in the system; or when being attacked by an adversary. This objective just relates to the second case: inputs under failure conditions. The first case is covered in Objective ARC1-2; the third case is covered in Objective ARC1-4.

Examples: An important part of tolerating failure-related inputs is knowing the health of the sub-systems that provide computation inputs, noting that these sub-systems could be quite complex. Such a sub-system could, for example, combine a physical sensor with some form of processing, possibly including some history-based features (e.g. a moving average). Health-related information would be expected to be provided by BIT, whether this be periodic or initiated in response to some external demand (e.g. from the autonomy architecture).

If an input-providing sub-system is known to be faulty then there are several options available to the autonomy architecture, including: using the last known good input value; using a constant "good" (or "safe") input value; using an input value derived from another sub-system; and declaring the computation output as incorrect (using one of the approaches discussed under Objective ARC1-5).

ARC1-2: **Operational inputs inconsistent with training inputs are tolerated.**

Discussion: This objective recognises that a computation is only valid for operational inputs that are, in some sense, consistent with (or supported by) the data that was used to develop it.

Note that this notion of consistency is distinct from that of distribution shift. For example, the training data

may have been supplemented with synthetic examples to address a class imbalance [61]. In such cases, the training data may exhibit roughly similar numbers of samples in each class, whereas the operational data may be highly biased towards a subset of classes. Consequently, the operational data would show an expected distribution shift in comparison with the training data. This shift need not affect the validity of a single computation, provided that computation's input is consistent with the data used to develop the computation (i.e. the training, test and verification data).

There are two ways that an operational input could be inconsistent with this data: firstly, the data could inadequately cover the operational domain; secondly, the autonomous system could be used outside its intended operational domain. Sometimes, the distinction between these two alternatives might not be clear. The first of these considerations relates to the experience projection of the computation-level framework (subsection 3.1). The second consideration is the focus of this objective.

Examples: The first aspect of meeting this objective involves detecting that an operational input is inconsistent with the Training, Test and Verification (TTV) data.

The simplest approach to this would be to store the minimum and maximum values for each feature present in the TTV data and to declare an operational input as being consistent if it is within the hyper-rectangle defined by these bounds. Whilst it would be easy to implement, this approach has some limitations. For example, there may be large parts of this hyper-rectangle that do not contain any samples [10]; a potential mechanism for detecting such regions is provided in [58]. Additionally, an operational input that was just outside these bounds would, perhaps unfairly, be declared as being inconsistent; this could be countered by extending the bounds by a suitably-chosen distance. However, justifying the use of a particular distance may be challenging: a short distance in one dimension may be of much greater import than a long distance in another dimension, for example. In addition, repeated extensions of this type could extend the bounds without limit, so they should be guarded against.

The approach discussed in the preceding paragraph relies on a summary of the TTV data, specifically, suitably-extended bounds of the TTV data and the location of any large empty hyper-rectangles within these bounds. Other ways of summarising the TTV could be used as the basis for alternative approaches. For example:

- Developing a model that given an operational input returns an estimate of the distance to the nearest TTV sample. In this case it may be challenging to determine what threshold distance should be used to decide whether a particular operational input is inconsistent with the TTV data; this could, possibly, be informed by statistics on the distances between nearest-neighbours within the TTV data. Alternatively, it may be informed by expert judgement.
- Developing a binary classifier that determines whether a given operational input is inconsistent. This would treat all of the samples in the TTV data as the "consistent" class. Samples defining the "inconsistent" class could be synthetically-generated. Additionally, it may be possible to train a Generative Adversarial Network (GAN) to represent the TTV data, similar to the way that a GAN can be used to augment training data [6]. The GAN could then be used to determine if an operational input was consistent with the TTV data.
- Developing a deep ensemble, which may be created by training multiple models (beginning with different initial parameters and pseudo-random number seeds), to provide uncertainty estimates [7]. High levels of uncertainty are typically associated with operational inputs that are inconsistent with TTV data. Creating and using the ensemble does, however, require more resources than would be the case for a single model.

Regardless of the approach that is used, there needs to be sufficient assurance that the detection of "inconsistent" operational inputs will be suitably accurate. Depending on the application, that may mean, for example: no false positives (i.e. points declared as being consistent when they are not); no false negatives; or neither too many false positives nor too many false negatives.

The second aspect of meeting this objective involves responding appropriately to the provision of an "inconsistent" input. In some cases, this may be achieved using the approaches discussed against Objective ARC1-1. Another option may be to map the operational input to the closest (or, more generally, an appropriate) input that is consistent with the TTV data; at least in some cases, it should be relatively easy to extend the detection approaches outlined above to also perform this function. Finally, as was the case for Objective ARC1-1, another option is to declare the whole computation invalid and rely on one of the approaches discussed under Objective ARC1-5.

Most RL involves significant training in a synthetic, or simulated, environment. In these cases, part of this objective relates to demonstrating that this simulation is an appropriate representation of the real world. Consequently, similar considerations to those discussed under Objective COM2-6 (which considers the appropriateness of the test environment) also apply here.

More generally, some techniques used to perform RL in changing environments incorporate mechanisms for detecting change points in the environment [72]. These change point detection techniques may prove helpful in satisfying this objective.

ARC1-3: Faults and failures internal to the computation are tolerated.

Discussion: There are a number of ways that internal faults and failures can be detected in traditional software. Examples include the use of defensive programming, exception handling (e.g. relating to floating point numbers) and hardware-based watchdog timers. These types of approach would be expected to be implemented, for example, as part of meeting the objectives associated with the software projection of the computation-level framework (subsection 3.4).

In addition, there are a number of related approaches that are specifically tailored to the use of autonomy-related computations. This objective is concerned with implementing approaches relevant to AI and ML.

Examples: A potential approach, specifically related to neural networks, is provided in [90]. This was developed to provide protection against random hardware errors, specifically, random bit flips. However, the underlying technique, which is based on anomaly detection in intermediate outputs of the neural network, may have more general applicability.

Other techniques exploit the general notion of tracking intermediate values within a neural network and comparing them against expectations. These expectations may be formed by analysing intermediate values and the associated results during the training process [62].

For particular types of hardware (e.g. GPUs), many faults may cause silent data corruption, which produces incorrect output with no obvious indication [59]. In such circumstances, applying checkpoints and recovery mechanisms at the GPU boundary can be helpful.

For particular types of network (e.g. CNNs) random bit flips in hardware may be likely to result in classification as one of a small number of classes [68]. If the bit flip is a temporary phenomenon, repeating the classification should resolve the issue. A defensive approach might be to repeat every classification

that results in one of these classes.

ARC1-4: **Adversarial attempts to disrupt the computation are tolerated.**

Discussion: In the ML community, the notion of adversarial examples is often restricted to small perturbations, generally undetectable by humans, that cause significant output differences (e.g. confident mis-classification) [95]. Whilst this is an interesting, and important, phenomenon this objective has a much wider scope. For example, it includes: adversarial poisoning of training data [16]; adversarial influence of pre-trained networks [37]; and adversarial impact via software libraries and tool kits.

Despite this wide breadth, it should be noted that this objective does not seek to address all cyber-related considerations associated with an autonomous system. For example, this objective does not consider protection of design information or staff screening. These, and other, wider objectives would be expected to be covered as part of the normal security engineering process. The Department for Transport (DfT) Guidelines for Cyber Security for Connected and Automated Vehicles [22], suitably re-interpreted for the relevant domain, may be helpful in this regard.

Examples: There is a large body of research aimed at providing protection against adversarial inputs (e.g. [54]). Likewise, there is considerable research aimed at demonstrating the robustness of ML approaches (generally of neural networks used for classification tasks) against adversarial inputs: this is typically expressed as the size of "ball" around each training sample within which inputs will be given the same class as the sample [103]. At the time of writing, there has been less work on run-time detection of adversarial inputs, although some examples are beginning to emerge (e.g. [62], [10]).

Most adversarial inputs are based on causing incorrect classification. That is, they affect the computation's output. However, other features of the computation can also be important, for example, time taken and energy used. It is possible for an adversary to craft inputs that cause undesirable behaviour from these perspectives [91].

As discussed in Objective COM1-1, obtaining data from known, trusted sources and applying strong guarantees on integrity during transmission can help protect against the risk of data poisoning; these approaches are also relevant to the use of pre-trained networks. Similarly, applying formal Configuration Management (CM) processes to data and pre-trained networks are likely to be beneficial, noting that traditional CM tools may be challenged by the volume and pace of change of these items. Applying techniques (e.g. [101]) specifically designed to detect and mitigate backdoors in pre-trained networks may also be helpful.

Conceptually, adversarial risks associated with software frameworks and tools may be protected against by using appropriate development methods during their creation (as discussed in Objective COM4-1). However, from a practical perspective, most ML pipelines make extensive use of open source libraries and tool kits which, generally speaking, do not provide the same type of assurance evidence as is delivered by development processes for critical software. Furthermore, traditional approaches to the use of SOUP can be difficult to employ, because libraries and tool kits cannot be placed in a tightly bounded sandbox and because their size prevents widespread reverse engineering of evidence artefacts. Possible options include using a library or tool kit to explore different approaches then re-coding the chosen one using appropriate development processes. Reverse engineering relevant evidence artefacts for a suitably-small part of the framework may be another option.

In some cases, software behaviour may be sufficiently predictable to allow for attack code to be detected as something that is "not normal" [20]. If such a situation is detected then it may be possible to reload original

software from a known good store. Of course, this assumes the known good store is suitably protected, the loss of processing whilst the reload occurs can be tolerated and false positives are sufficiently rare.

ARC1-5: **Incorrect computation outputs are tolerated.**

Discussion: If incorrect outputs are considered to be failures then even the best performing AI computation (developed using ML techniques) is likely to have an error density orders of magnitude greater than that of traditional safety-critical software. It follows that the autonomy architecture has to be tolerant to incorrect outputs.

Examples: There are several architectural approaches that could be used to support this objective [9]. Three examples are illustrated in Figure 2:

- The first architecture makes use of a validity checker, developed using traditional software techniques. If this detects an incorrect output then the computation is re-run, either using the same input (if the computation includes a random element) or a suitably adjusted input. The simplicity of this architecture is an attraction, but designing a checker that provides the appropriate protection without unduly constraining the computation may be challenging.
- The second architecture is inspired by multiplex avionics systems. It uses a number of distinct computations in parallel, combining their output with some form of voting or aggregation function. Although it bears some similarities to ensemble methods, the motivation is different: in particular, ensemble methods aim to improve general performance, whereas this architecture aims to provide protection against incorrect outputs (e.g. for "edge cases"). The nature of ML may make it easier to produce separate channels than would be the case for traditional software [12]. However, it may be difficult to achieve sufficient diversity between the different channels, especially if they share large amounts of training data. This may make it difficult to provide confidence that the multi-channel architecture has a sufficiently low error density.
- The third architecture features two channels, one of which uses AI, the other of which is implemented using traditional software. The intent is that the traditional software channel implements an always safe, but low (perhaps very low) performance behaviour, whereas the ML model offers higher performance with a concomitant risk of incorrect (i.e. unsafe) outputs. A safety switch, implemented in a similar manner to the validity checker in the first architecture, controls which channel provides the output to the wider system. Note that in this type of architecture it may be possible for much of the assurance burden to be borne by the traditional software and the (traditionally-implemented) safety switch, thus reducing the level of assurance required for the ML model.

5.2 Information Provision

The autonomy architecture is concerned with integrating computations into a platform, or system. This includes providing information related to these computations to the wider system, often for onward transmission to users, stakeholders or other systems. In order to support this information provision, there is an associated need to support information recording and retrieval. These topics are the focus for this projection.

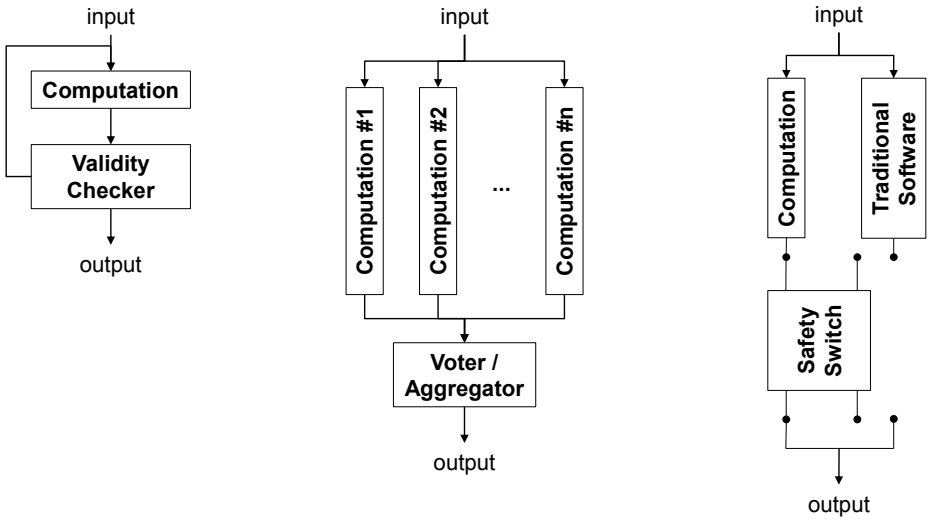


Figure 2: Example Architecture Options

There are four objectives associated with this projection.

ARC2-1: **Relevant information is presented to interacting parties.**

Discussion: In order to be safe an AS may need to provide information on what it is currently doing and what it is planning on doing, as well as the reasons for these decisions. This type of information may need to be presented to a range of parties, including other systems (both autonomous and traditional) and humans. In the latter case, this information may be provided purely to inform a passive user, alternatively, it may be provided to support teaming between the human and the AS, facilitating a collaborative decision making process.

Examples: An initial step towards satisfying this objective involves identifying the relevant interacting parties and understanding what information they require. Attributes of this information, for example, accuracy, timeliness and availability should also be considered: the data-related attributes listed in [97] may be helpful in this regard.

Part of this objective relates to the concept of "explainable AI", which is an active research area. There are, for example, established ways of explaining the result of a classifier [80].

It is important to recognise that the nature of any explanation needs to be tailored to the receiving entity and the prevailing situation. For example, in time-critical situations, a brief explanation that can be rapidly understood may be more valuable than a detailed justification. Similarly, when working alongside humans there may be a requirement to overcome a human's initial belief that a different course of action should be pursued [4].

ARC2-2: **Relevant information is available to support maintenance and future development.**

Discussion: Objective ARC2-1 focused on operational use. In contrast, this objective is concerned with information use during other phases within the system lifecycle.

There are several reasons why autonomous systems, and especially the associated ML-enabled computations, are likely to be updated more frequently than traditional systems. Firstly, it is difficult (if not impossible) to capture all relevant situations in TTV data; secondly, it is difficult (if not impossible) to address all potential edge cases during verification activities; thirdly, the dynamic nature of the operational environment means new requirements may emerge; fourthly, the relative ease (from a technological perspective) with which updates can be deployed may lower the threshold associated with update deployment.

This objective is concerned with providing information to support these updates; the update process itself is the focus of the adaptation projection. Also note that this objective relates to general maintenance and future development. Information (and actions) in response to incidents and accidents are covered in Objective ARC2-3.

Examples: Satisfying this objective should be relatively straightforward. Inspiration as to the type of information that is needed should be readily available from test and evaluation activities conducted during the system development lifecycle. For vehicles, this information would be expected to include data relating to sensors and control systems, as well as data about the vehicle's movement [14].

ARC2-3: Relevant information is preserved to support post-incident analysis.

Discussion: Learning from experience is an important part of a mature safety culture. This learning needs to occur at several levels, including platform-specific, within a domain (e.g. air, rail, road) and cross-domain. The former of these may be contained within a single organisation, but the latter two require information to be distributed across wider communities: air accident investigations are an example of this. This learning is predicated on suitable information being available.

Legislative and voluntary structures that allow "no blame" sharing of this type of information are important, but they are outside the scope of this objective (and this document). In particular, this objective is focused solely on the preservation of computation-related information.

Examples: Part of this objective relates to preserving information following a catastrophic accident. This preservation is likely to require crash-survivable recorders. The precise nature (including, but not limited to, time bounds) of the required information is, inevitably, application specific. In the case of road vehicles, an indication of the duration for which information needs to be recorded is provided in [14].

Another part of this objective relates to managing the immediate aftermath of an accident. In this case, suitable information needs to be made available to first responders. This needs to be presented in an intelligible manner without the need for complex analysis or interpretation techniques [14].

A key challenge in the context of autonomous systems is that it might not be readily apparent that the system has been involved in an accident. This could be the case if, for example, the accident resulted in damage to other entities but left the platform unscathed. It could also be the case if an incident was a near miss, which led to no damage. Providing sufficient information to support these types of analyses is likely to require complex trade-offs between: the volume of information recorded; the period for which the information is stored; and the location at which the analysis is conducted (e.g. on-platform or off-platform). It is also likely to require regular analysis of logged information [69].

ARC2-4: **Information is managed securely.**

Discussion: The preceding objectives in this projection have demonstrated the safety-related importance of a wide variety of information. It follows that this information needs to be managed securely.

This will require traditional approaches to cyber security, for example, those promoted by the National Cyber Security Centre (NCSC)[3]. However, there are aspects of ML that may warrant specific consideration. For example, both the trained model and the training data may need to be kept confidential.

Examples: In general, there is a trade between allowing use of a model and preventing it, or its training data, being reverse engineered.

Successful attacks that allow models to be recreated after a limited number of queries have been demonstrated [98]. There are several possible approaches that could be used to reduce, but not eliminate, this risk. Examples include: limiting the number of queries a single user can make of a model (which requires some way of identifying users); and adding noise to the model's output (which requires some way of identifying how much noise should be added). These approaches also adversely affect the model's performance. More generally, inspiration may be gained from privacy-preserving data mining [2].

An alternative approach may involve using the autonomy architecture to prevent potential adversaries getting direct access to the inputs and outputs from an ML model, for example, by passing them through pre-processing applications. This would require traditional cyber security approaches to protect these applications.

There are also attacks that can expose aspects of the training data (e.g. [31]). Again, protecting against these may involve using the autonomy architecture to protect against adversaries obtaining direct access to unfiltered model inputs and outputs.

5.3 Adaptation

In some cases, an instance of a computation may be left unaltered after it is deployed into operational use. Alternatively, all subsequent releases may progress through a full engineering development process. If either of these approaches is adopted then the adaptation projection is much less relevant to that application.

However, it is expected that many computations developed using ML techniques will be adapted in some way following their initial operational use. This could be achieved using a variety of mechanisms, including: online learning (where the computation continues learning and, consequently, adapts during operational use); and nightly over-the-air updates (which are released after a reduced amount of regression testing, rather than following the full engineering process).

There are two objectives associated with this projection.

ARC3-1: **Inappropriate or unauthorised adaptations do not occur.**

Discussion: Fundamentally, an adaptation changes some aspect of the computation's behaviour. This means adaptations have the potential to undermine an assurance argument and need to be managed

[3] https://www.ncsc.gov.uk/.

carefully.

For the purposes of this objective, an *inappropriate* adaptation would be one that did not achieve the intended aims. As such, the notion of what is inappropriate is, inevitably, application specific. Potential examples include an adaptation that: unintentionally reduces the computation's performance in common situations; unintentionally reduces the computation's performance in rare situations; or alters the computation's non-functional behaviour in a way that detrimentally affects interfacing items. An adaptation that was *incorrect*, perhaps because it did not correspond to the expected information format, would also be considered to be inappropriate.

Conversely, an *unauthorised* adaptation would be one that was made without appropriate authority having been granted. This could, for example, occur if a malicious third party, or a rogue employee, implemented an adaptation that was intended to cause harm. Alternatively, an adaptation that was released by the computation developers but which had not completed the necessary pre-release processes would also be considered unauthorised.

Examples: Since it uses a very distinct approach it is simplest to consider online learning as a special case. This is most commonly achieved via RL. There are a variety of approaches to ensuring adaptations via RL are appropriate, including: constraining the optimisation criterion; adopting a risk-sensitive optimisation criterion; having the computation ask for help; and using risk-directed exploration [32]. Alternatively, or additionally, it may be possible to provide a set of abstract policies that formally constrain the exploration of an RL agent [63] or including a representation of fear within the learning mechanism [60].

The nature of online learning is that it happens continuously, as a natural part of the computation's use. Hence, the mechanism by which adaptations are achieved forms part of the full engineering cycle associated with initial release to operational use. Consequently, the notion of an unauthorised adaptation does not apply in this case.

For computations whose behaviour is not altered by their use, an adaptation involves a deliberate act, typically loading new parameters (or hyper-parameters). For example, in the case of an NN, an adaptation may involve loading new network weights and biases. Considerations associated with Parameter Data Items (PDIs) are important, for example, the data being managed as a distinct entity and its effect on computation behaviour being understood [82]; this latter point may, for example, require some form of regression testing before the adaptation is deployed.

Some form of testing would be expected to be conducted before an adaptation was performed. This should be sufficient to prevent cases where the adaptation unintentionally reduces the computation's performance in common situations. One way of achieving this would be to define a collection of situations, along with a minimum level of performance in each. Adaptations would only be considered *appropriate* if at least the minimum level of performance (including safety and security) was achieved in each situation. This collection of situations can be viewed as being analogous to a minimal set of regression tests for traditional software. It can also be viewed as analogous to the criteria used to validate flight simulation training devices [26]. Note, however, that aviation is a well-understood domain. Determining an appropriate collection of situations is likely to be more difficult in many other domains. Also note that the collection may need to change, either in response to changes in the computation, or changes in the external environment in which the system is used.

Protecting against the case where the adaptation unintentionally reduces the computation's performance in rare situations is more difficult. In many cases a balance has to be found between enacting an adaptation that will demonstrably benefit computation performance in common situations against the possibility that the same adaptation could reduce performance (in a way that affects safety) in rare situations. An

evidence-based, structured argument is likely to be required to demonstrate that an appropriate balance has been achieved. Features of the autonomy architecture, especially those related to the tolerance projection (subsection 5.1), could protect against egregious safety failures; if present, these could also simplify the "balance" argument.

There are several aspects to understanding how an adaptation may affect interfacing items. Broadly speaking, three categories of interfacing item can be considered: items within the same platform as the computation; items within other systems; and interactions with humans. Platform-level testing ought to ensure the adaptation does not adversely affect interfacing items within the system. Likewise, this testing also ought to cover (planned) interactions with other systems. Interactions with humans are more subtle, especially if the human requires training or certification in order to use the computation. In this case, the impact of the adaptation on user training or certification needs to be considered. These considerations need to take account of not just the latest adaptation, but the cumulative effect of all adaptations that have occurred since the last training or certification event.

There are two main aspects to preventing unauthorised adaptations: cyber security; and management. Guidance on cyber security is available from a number of sources, including cyber security principles for connected and automated vehicles [22] (which can be generalised to cover a wide range of autonomous systems) and the NCSC. General engineering processes, especially those associated with safety-related systems, would be expected to contain safeguards that prevent unauthorised releases.

The preceding discussion has focused on changes to the computation. Changes to supporting software and computational hardware are also important, but these should be manageable within normal engineering processes.

ARC3-2: **Computation behaviour is appropriate before, during and after an adaptation.**

Discussion: This objective recognises a number of things, specifically: adaptations should be performed against a known baseline; a computation may be in use when an adaptation request (or command) is received; an adaptation cannot be applied instantaneously; and the process of applying the adaptation may fail.

Any of these factors could undermine an assurance argument. Some, like the finite amount of time taken to apply an adaptation, may only undermine an assurance argument for a relatively small amount of time. Others, like the consequences of a failed adaptation, may be persistent (unless appropriate action is taken).

This objective also recognises that many different types of computation behaviour may be appropriate. This is a consequence of requirements being implicitly expressed via the training data, rather than being formally decomposed (in a traceable manner) as is the case for traditional safety-related software.

Examples: If all objectives associated with the computation-level framework have been satisfied then the computation behaviour ought to be appropriate before an adaptation is applied.

In some cases it may be possible to instantiate two (or more) autonomy architectures. Such an arrangement would allow one instantiation to adapt whilst the other continues to respond to operational inputs; it would also have the additional benefit of increasing reliability in the context of hardware failures. If two copies are available then the system (or platform) can determine a suitable time to switch from the pre-adaptation computation to the post-adaptation one. This switch can be implemented in software, meaning it can be completed without a noticeable impact on the computation's ability to respond to operational inputs. More specifically, this arrangement provides a means of demonstrating

that computation behaviour remains appropriate during an adaptation.

If two (or more) architectures are not available then the computation is likely to have to stop processing operational inputs before allowing the adaptation to occur. This will require communication between the computation and the system to ensure the gap in processing can be accommodated safely. For example, in the case of an autonomous vehicle, an adaptation could be postponed until vehicle car is stationary, the parking brake is on, the engine is turned off and there are no people in the vehicle. An alternative may be to designate safe regions (e.g. the owner's garage, the dealer's service area) and only allow adaptations to occur when the vehicle is in one of these regions. Whatever approach is used, care needs to be taken to protect against the possibility of platforms not being in a situation where an adaptation is allowed for a prolonged period of time. Additionally, care needs to be taken to prevent removal of the system from the safe region until the adaptation is complete (and confirmed successful).

Some aspects of ensuring that behaviour is appropriate after an adaptation are covered by the *appropriate* part of Objective ARC3-1. The current objective includes cases where the adaptation process did not complete successfully. Failed adaptations should be detectable using standard approaches to data integrity and post-adaptation BIT. In many cases, the most suitable way of handling a failed adaptation is to revert to the previous "last known good" configuration. This requires storing the pre-adaptation computation parameters in some way (which happens naturally if there are multiple instantiations of the autonomy architecture).

In some cases (e.g. when the adaptation addresses a serious flaw in the computation) reversion to the previous configuration may not be desirable, regardless of whether this is readily available. It follows that every system ought to be capable of being put in a safe state that can be maintained for a considerable period.

This page is intentionally blank

6 Platform-Level Framework: Description

This section describes the framework adopted by the SASWG for platform-level considerations. The justification for adopting this framework is provided in [89].

The framework consists of four projections, each of which views the platform's properties along a different axis. The projections are not intended to be strictly independent: they provide different ways of viewing the platform, in order to elicit associated objectives. To facilitate discussion, the projections have been arranged in an approximate order, starting at the platform and working, in a sense, outwards: behavioural specification; interacting items; people; and environment. Each of these projections is considered, in turn, in the following subsections. The section concludes with a brief tabular summary of the entire framework and an illustration of how the framework's projections inter-relate.

6.1 Projections

6.1.1 Behavioural Specification

This projection focuses on the platform specification. Equivalently, it considers the system-level requirements that the platform would be expected to satisfy and the methods the platform developer might use to demonstrate satisfaction.

From the perspective of an autonomous system, the most challenging aspect of these requirements is defining the required platform behaviour. Hence, this aspect is emphasised in the projection's title. In addition, a key aspect of this projection is defining the scope of the autonomous aspects of the platform.

6.1.2 Interacting Items

This projection focuses on items that are intended or required to interact with the platform, but are not directly part of the platform under consideration.

The interacting items projection includes the case where there are multiple, notionally identical systems: for example, when an autonomous vehicle manufacturer has sold many vehicles, each of which is intended to be used individually. It also includes the case where the wider system includes optional items that may or may not be present: for example, additional roadside infrastructure that is only present in large cities. This projection may also include a central repository, which is used to support operation, maintenance and future enhancement of the platform.

6.1.3 People

This projection focuses on people associated with and affected by the platform, including those operating or using the system.

The projection adopts a whole lifecycle perspective, including design, manufacture, operation, maintenance and disposal. It includes: those operating the system; those associated with the system (e.g. maintainers); and bystanders in the operational environment of the platform. This latter class may be highly important for some autonomous systems (e.g. self-driving vehicles); conversely, it may be absent for other autonomous systems (e.g. medical diagnosis systems).

The projection also includes people behaving in an adversarial manner.

6.1.4 Environment

This projection focuses on items in the platform's operational environment that are outside the control of the platform developer or operator.

It includes, for example, other platforms and meteorological conditions, as well as things like terrain and infrastructure. It covers normal operating conditions and abnormal situations (e.g. extreme weather and unexpected terrain). It includes the potential for elements of the environment to be used in a hostile manner.

6.2 Summary

Table 4 provides a brief summary of the four projections in the platform-level framework that has been adopted by the SASWG.

Table 4: Summary of Projections Within the Platform-Level Framework

Projection	Outline
Behavioural Specification	Focused on platform-level specification; includes defining the scope of autonomous aspects
Interacting Items	Focused on things intended or required to interact with the platform, not directly part of the platform under consideration
People	Focused on how the platform interacts with people; adopts a whole lifecycle perspective
Environment	Focused on things in the operational environment that are outside the control of the developer or operator

Experience in writing this document has demonstrated the potential for confusion with regards to these projections and how they inter-relate. Firstly, it is important to remember the projections are a means to an end, rather than an end in themselves. Users of this document are not necessarily expected to directly align their development efforts with the projections. Prolonged discussions about whether a specific item (e.g. lane markings on a road) are part of the environment or an interacting item should not be of critical importance in establishing an assurance argument for an autonomous system.

To that end, Figure 3 provides an alternative illustration of the platform-level projections and how the inter-relate. For example, the people projection can, in some ways, be considered as a specialisation of the environment projection.

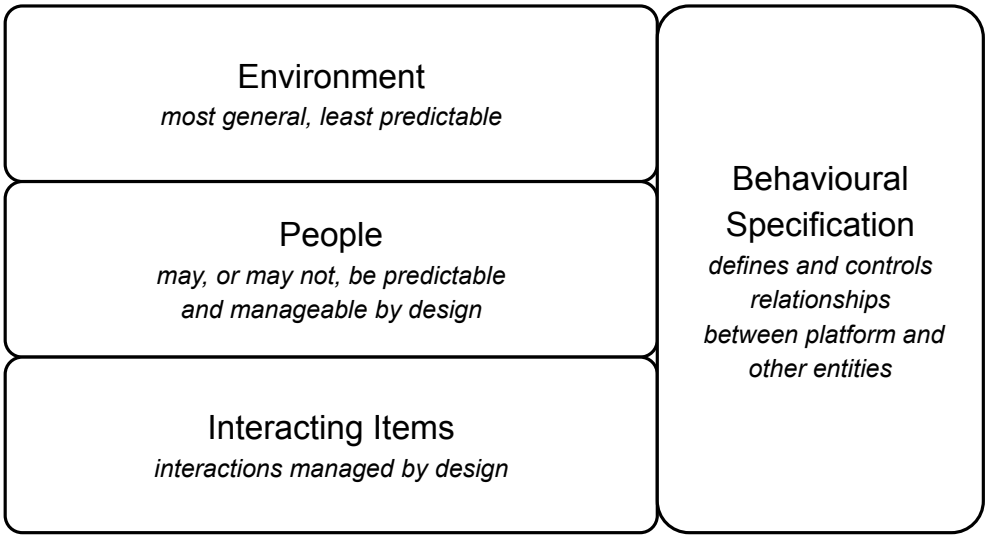


Figure 3: Platform-Level Framework Projections Inter-Relationships

This page is intentionally blank

7 Platform-Level Framework: Objectives

This section lists the objectives associated with each projection of the platform-level framework.

7.1 Behavioural Specification

The behavioural specification projection is concerned with what the platform should do and, also, what it should not do. These concerns are viewed largely from the perspective of an operator or user of the platform; that is, they are largely concerned with externally-observable effects, rather than the internal workings and structure of the platform.

There are six objectives associated with this projection.

PLT1-1: **All aspects of platform behaviour that are achieved using autonomy-enabling techniques are justified.**

Discussion: Ultimately, the platform has to be considered as an overall entity, which is demonstrably safe. However, the novelty of autonomous systems, and especially the AI and ML techniques that enable them, means these aspects need special attention. To focus that attention there is a need to identify such techniques, including how they contribute to platform behaviour.

Given the current low maturity of autonomy-related techniques, there is a strong safety-related argument that wherever possible more traditional techniques should be used [87]. Consequently, as well as identifying where autonomy-related technologies are used, their use should also be justified.

Examples: Before uses of autonomy-enabling techniques can be justified, they must first be identified. There are, broadly-speaking, two ways that the use of autonomy-related techniques can be identified, specifically, top-down and bottom-up. In a top-down approach, requirements can be tagged to indicate whether they are achieved through the use of autonomy-related techniques. This tagging could start at the system-level and be flowed down to lower levels as requirements are decomposed during a typical systems engineering process. Conversely, in a bottom-up approach, specific components (or items, or sub-systems, or algorithms) can be flagged as using autonomy-related techniques, with this information flowing up to functions and platform-level behaviours that use those components.

Regardless of which approach is used, there should be a clear link from computations, through autonomy architectures, to platform behaviour. These links enable evidence associated with objectives from the computation and autonomy architecture frameworks to support platform-level assurance arguments. They also allow platform-level requirements and constraints to provide the context for lower-level activities.

An argument to justify use of an autonomy-enabling technique could be based on many factors: performance, including an inability to achieve the associated function using traditional approaches, is often an important one.

PLT1-2: **Acceptably safe behaviour for the platform is defined.**

Discussion: In order to argue that a platform is "acceptably safe" there needs to be a definition of this term for that particular platform, within its intended operational domain. In traditional safety engineering, this definition may be comparatively simple, drawing upon contextual assumptions that significantly simplify the

operational domain. In contrast, a key feature of many autonomous systems is their intended use within complex, diverse and dynamic environments. Consequently, a key aspect of defining safety for autonomous systems is understanding, and suitably bounding, interactions with these types of environment.

Given the nature of these environments, it may also be necessary to consider what safe operation means should the platform be used outside the intended operational domain. This could, for example, involve finding a way to safely stop operation.

Examples: Traditional techniques for hazard assessment, for example Hazard and Operability Study (HAZOP) [25], may help generate an understanding of potential unsafe outcomes. However, to be productive these techniques may need to focus on platform-level outcomes (e.g. the vehicle stops too late) rather than component-level actions (e.g. the brakes are applied too late). They may also need to include types of hazard event that are directly related to autonomous systems (e.g. [75]). Considering safety-relevant characteristics across the three areas of internal characteristics, effects of the environment on the system and effects of the system on the environment may also be helpful [42].

In addition, the consequences of a vehicle stopping too late are critically dependent on both the nature of the operational domain and the specific situation pertaining at the time of the occurrence. Identifying specific situations that correspond to unsafe outcomes is one challenge. Providing some form of confidence that all relevant situations have been identified is another. The notion of situation coverage [3] may help, as might environmental hazard analysis [24]. Concentrating on requirements related to safety, rather than all behavioural requirements, may also be helpful [64].

Arguably, things that are unsafe can be viewed as stakeholder losses from the perspective of at least one stakeholder. This view can include, for example, accidents that cause significant environmental impact representing a loss from the perspective of stakeholders in the local community. In addition, many (but not all) autonomous systems exploit autonomy as part of a control loop. In such cases, accidents (i.e. stakeholder losses) can be viewed as a consequence of inadequate control. Given this discussion, there may be value in a Systems Theoretic Process Analysis (STPA) based hazard analysis [47].

As well as being safe during use, there may also be a need to define a safe state, in which the platform can be left for a prolonged period of time. Understanding this state (or collection of states), together with trajectories by which they may be reached, is an important part of maintaining safety in the presence of faults and failures (as discussed in Objective PLT1-4). Whilst it is a simple concept, defining and implementing a safe state is not trivial. For example, in the case of a self-driving car a safe state could be one where the car is stopped. However, this might not be safe if the car is stopped in a driving lane on a motorway, or if it is stopped on a level crossing, or if it is in any number of other dangerous locations. Rigorous, structured analysis of the intended operational domain and associated environment may help.

PLT1-3: **[Objective deleted.]**

Discussion: The original intent of this objective is now captured in other objectives associated with the behavioural specification projection.

PLT1-4: **The specified behaviour is safe in the presence of faults and failures, as well as foreseeable misuse and abuse.**

Discussion: This objective is concerned with platform behaviour in sub-optimal conditions, which occur when platforms are subjected to real-world use by humans. This includes, for example, deliberate misuse

that causes the platform to be used outside its intended operational domain.

The concepts of faults and failures can be directly related to those of misuse and abuse. For example, a fault in some part of the platform may cause an operator to deliberately misuse the platform in an attempt to complete their task. For this reason it is convenient to combine these items in the same objective.

Examples: This objective may raise the question of, "How many faults and failures should be tolerated?" In considering this question, it should be noted that current safety-related systems often require human intervention to handle cases involving multiple faults and failures. However, autonomous systems change the role of the human operator, which may increase the number of faults and failures that have to be handled by the platform, without human intervention. More generally, component failure rates could be used to inform the number and types of faults and failures that should be considered within this objective.

Traditional safety engineering techniques, for example, redundancy, could be used to provide for safe behaviour in the presence of faults and failures. Alternatively, or additionally, a Minimum Equipment List (MEL) could be defined, with the health of this equipment being monitored (as discussed in Objective ARC1-1). There may be a need to consider the case of a platform that has suffered too many faults and failures or, equivalently, a platform that is in operational use when the full MEL ceases to be available. To handle such circumstances there may need to be a defined safe state for the platform; there may also be a need for the platform to maintain an ability, or trajectory, to reach this safe state (which could be identified as part of Objective PLT1-2).

Detecting and preventing misuse and abuse is a difficult challenge. Simplistically, misuse could be construed as cases where a human operator tries to do something in opposition to the autonomous part of the platform. However, that view fails to account for the fact that, in some circumstances, the human might be correct: an illustration can be taken from traditional, non-autonomous software, which prevented access to engine thrust reversers even though the aircraft was on the ground, leading to runway excursion and loss of life [29]. Equally, there are cases that exhibit the converse, namely, the software is correct and the human is not.

A clearer view could perhaps be formed by looking for situations where the human operator tries to make the platform behave contrary to the safety-related requirements established via the previous two objectives. For example, this could be an effective way of protecting against misuse that aims to deliberately take the platform outside its intended operational environment. Requirements that relate to unintended behaviour (i.e. "the platform should not …") are especially helpful in this regard. Thinking about the potential for misuse and abuse can help establish requirements of this type.

PLT1-5: The behaviour of the platform is verified.

Discussion: Whereas other objectives in this projection have established requirements, this objective is concerned with verifying that these platform-level requirements have been satisfied. This includes providing artefacts to demonstrate (e.g. to a regulator) that verification has been successfully completed.

Examples: As with traditional systems, there are several ways to provide evidence that a given requirement has been satisfied. However, in comparison to traditional systems, the potential behaviour of autonomous systems is much less bounded. This is likely to require a gradual, progressive approach to Test, Evaluation, Verification and Validation (TEVV), beginning with simulation and real-world testing in controlled environments (e.g. closed roads).

Simulation, in particular, is expected to play a significant role in requirement verification. For example, it

allows testing of cases that would be too expensive or too dangerous to consider in the real-world; this is likely to be especially valuable for test cases involving platform faults and failures. Whilst simulations are undoubtedly valuable, demonstrating that the simulation is a suitable representation of the real-world may be a significant challenge. This is important as there are many examples of autonomy-related techniques exploiting flaws in simulations [19]. It follows that unsuitable simulations could fail to detect important errors in platform behaviour.

A key concept relating to verification of platform requirements for autonomous systems is the notion of coverage. Generally speaking, in operational use these requirements would be expected to be satisfied in a wide range of situations, this breadth resulting from the the complexity of the environment the platform has to operate in. Understanding how the verification evidence covers this variety is a key issue: analyses such as situation coverage [3] may assist. Other ways of structuring considerations related to coverage may also be helpful. One approach may be to consider the following categories: Operational Design Domain; Object and Event Detection and Response; Manoeuvres; and Fault Management [55].

The notion of coverage considered in this objective needs to be combined with notions of coverage relevant to computations (as discussed in Objectives COM1-3 and COM2-5).

PLT1-6: **Operational monitoring is sufficient to identify and support the mitigation of new hazards, including emerging cyber security threats.**

Discussion: Many safety-related systems include a form of monitoring; continuous, or periodic, with BIT being one example. The open nature of the operational domain associated with many autonomous systems means there is benefit in including this type of monitoring at the platform level. This should help capture cases where changes in the operational domain (or aspects of the operational domain that were not fully understood during development) result in new hazards.

The nature of autonomy-enabling technologies (including AI and ML) means they rely heavily on data, software and computational hardware. Consequently, cyber security is an important consideration. Hence, monitoring to spot the emergence of hazards from that domain is important, especially given the pace at which such hazards may emerge.

Examples: For the purposes of discussion, it is initially convenient to split operational monitoring into two main categories: monitoring of the environment; and monitoring of the platform.

Monitoring the environment is important as, in general, autonomous systems are used in environments that are less constrained, and less prescriptive, than is the case for traditional systems. This increases the likelihood of the platform being used outside the intended operational domain (either by accident or by deliberate intent). Conversely, most, if not all, of the platform verification will be based on the assumption that the platform is being used inside the intended operational domain. This apparent inconsistency may give rise to new hazards.

These hazards may need to be dealt with in two separate time scales. Firstly, if monitoring detects platform use outside the intended operational domain then appropriate action should be taken to maintain safety. This could, for example, involve achieving a defined safe state for the platform; alternatively, it could involve attempting to manoeuvre the platform back inside the intended operational domain. Secondly, if monitoring routinely detects use outside the intended operational domain then this is indicative of a need to change something about the platform (e.g. the system-level requirements, the assumptions made during verification, the way operators are trained).

Monitoring of the platform includes monitoring the sub-systems that provide inputs for the computation (as discussed under Objective ARC1-1). It may also include monitoring that decisions (based on autonomy-related technologies) have the expected consequences: for example, applying vehicle brakes should lead to a reduction in speed.

In addition to the separate cases discussed above, there are some aspects of operational monitoring that need to be considered more generally. For example, there may be a need to review information from operational use to identify near misses or anomalous behaviour. There is also likely to be benefit in capturing at least a selection of operational inputs, for example, to support future support and maintenance, as well as to monitor for potential distribution shift [67].

In these contexts there may be a need to make a decision on the extent of operational monitoring that is required. It is conceivable that a fleet of self-driving cars could cover thousands, or possibly millions, of miles a day. In such cases it may be neither feasible nor appropriate to analyse every single operational input seen by every single platform. Consequently, a reasoned argument as to what data will be analysed and why is likely to be required. In some cases a uniform, random choice of platforms may be appropriate; in other cases, there may be a specific desire to spread data collection across the operational domain as far as possible, resulting in a prioritisation of platforms that have experienced more exotic parts of the operational domain.

The final part of operational monitoring is specifically aimed at cyber threats. This may include specific on-platform monitors (e.g. to detect anomalous software behaviour [20]). An understanding of what aspects should be prioritised, from a monitoring perspective, could be gained by looking at the security perimeter (i.e. interfaces into the platform) and the security environment [84].

Depending on the specifics of the deployment, monitoring could also involve comparing multiple platforms within a fleet. The intuition is that unexpected, or unexplained, behaviour in the case of one platform, or a small number of platforms, could be indicative of a cyber-related issue. It could also be indicative of a number of other issues as well, many of which would require investigation from a safety-related perspective.

In addition to these platform-focused aspects of cyber-related monitoring, there is also likely to be a need to monitor general, cyber-related information. This should include monitoring for information on vulnerabilities associated with components and sub-systems used within the platform.

7.2 Interacting Items

The interacting items projection considers items that are are intended or required to interact with the platform, but are not directly part of the platform under consideration. Equivalently, this projection considers things that the platform is expected to interface with. In some cases, the platform developer may be able to exercise some degree of control over both sides of the interface (e.g. for platform test rigs); alternatively, it could be because the developer contributes to an open standard, as may be the case for roadside furniture.

There are three objectives associated with this projection.

PLT2-1: **Interacting items that affect the safe operation of the platform are identified and understood.**

Discussion: At least some of the interacting items are likely to have the potential to affect safe operation. The effect could arise if, for example, the item provided incorrect information and this was used without on-platform checking. Alternatively, it could arise if the item placed the platform in a special configuration, for example, to facilitate maintenance and this configuration was mistakenly left in place when the platform was returned to operational use. Given this potential impact, it is important that all interacting items are identified and their potential impact on safety is understood.

As well as being important in its own right, the understanding gained from satisfying this objective supports the other objectives associated with this projection.

Examples: There is a wide range of interacting items that could potentially affect safety. In addition to off-platform information sources and maintenance equipment, other examples include training information, technical publications and the process by which platform updates are achieved. Several of these items can be viewed as being safety-related data. Consequently, Data Safety Guidance may be helpful [97].

Several approaches could be used to gain the understanding necessary for this objective. For example, part of STPA [47] involves drawing a control structure, which illustrates entities, control actions and feedback information. This should highlight key interacting items as well as their potential effect on the platform, which is typically related to the information that is provided.

An alternative approach could involve looking at the platform from the perspective of a capability management framework, for example, the Defence Lines of Development (DLODs) used by the UK Ministry Of Defence (MOD), which are: training; equipment; personnel; information; doctrine and concepts; organisation; infrastructure; and logistics [65]. Considering each of these "lines" should help identify interacting items and, subsequently, their potential effects on safety.

PLT2-2: **Interactions preserve platform safety.**

Discussion: This objective is concerned with interacting items that are behaving according to their design intent. Since the platform developer has intentionally included these interactions, it is reasonable to assume they also have a sound understanding of this intent. Consequently, much of this objective would be expected to be satisfied via standard system and safety engineering approaches. However, there may be a number of autonomy-related "edge cases" that may warrant specific investigation.

For example, in some situations, it is possible (perhaps likely) that multiple examples of the same autonomous system will be present in the same environment. This would be the case, for example, if a manufacturer of self-driving vehicles had sold many vehicles within the same geographical region. Such vehicles are clearly under the control of the platform developer, but they may not naturally be highlighted by approaches used to identify interacting items.

This notion can easily be extended to interactions between different types of platform developed by the same organisation. If, for example, there is an industry-wide standard for sharing information then the notion can also be extended to cover platforms developed by other organisations; alternatively, or additionally, these platforms could be covered by the various objectives associated with the environment projection.

Examples: It may be the case that the techniques used to verify platform behaviour (i.e. to satisfy Objective PLT1-5) naturally extend, or can be easily extended, to cover the case of multiple platforms. Whilst this would make a significant contribution to the current objective, there remains a need to describe, and justify, the test cases that are considered (e.g. how many platforms, what initial states and configurations, and so on). These considerations could be informed by some notion of coverage, similar to those discussed in Objective PLT1-5, although the possibility of a combinatorial explosion in the number of situations would need to be managed..

Notions of coverage would also be required if new verification approaches needed to be developed in relation to this objective. These may be further complicated by the possibility of different software versions (including model variants) amongst the platforms.

If multiple instantiations (e.g. vehicles) are in operational use then each instantiation would be expected to comply with all objectives. Nevertheless, having multiple instantiations provides a potential opportunity to enhance fleet-wide safety. Consider, for example, a large multi-national organisation responsible for running many data centres. Every data centre would not be expected to be run at precisely the same software patch level. In this context, diversity in patch levels offers some protection against an unknown common mode failure affecting all data centres. In addition, it allows for changes to be tested in a small number of data centres before they are gradually rolled out.

Conceptually, diversity offers the same benefit for fleets of autonomous systems. However, in this case, a balance needs to be found between the known risks associated with older implementations and the potentially unknown risks associated with a newly-produced implementation. An incorrect balance would adversely affect platform safety.

It should also be noted that fleet-level diversity will naturally arise if the platform exhibits online learning. In this case, appropriate measures need to be in place to monitor and control this diversity. Otherwise, two apparently identical autonomous systems may exhibit very different behaviours; this could confuse people and other entities in the environment, with potentially unsafe results.

The first step in managing diversity within a fleet is to gain information on the individual platforms [12]. This could involve reporting from every autonomous system, or from a suitably-sampled subset of them. With this information, the developer could replicate platform behaviour in a synthetic environment (that has been demonstrated to be suitably representative).

This allows the developer to measure performance in a number of standard scenarios (or situations). The choice of scenarios would be expected to be described and justified as part of the computation's assurance argument. Care needs to be taken to ensure that the selection of scenarios is suitable for the intended use. However, experience from other areas suggests this may be possible: for example, a standard set of situations is used when testing an aircraft flight simulator [26]. The notion of situation coverage could also inform this decision [3].

Using standard scenarios provides a practical measure of the impact of diversity across the various instantiations. This may be supplemented by a theoretical measure of diversity, which could be calculated by sampling from across the input domain and comparing the results provided by different instantiations.

Verifying safe behaviour in the presence of autonomous systems developed by other manufacturers may be complicated, especially if manufacturers do not wish to share detailed, implementation-level information about the behaviour of their systems. This could, potentially, be overcome if a central regulatory authority received such information and, furthermore, investigated behaviours in multi-manufacture situations. This might be achieved using an appropriately verified and validated simulation environment.

Such an approach places a significant burden on the regulatory community, which they may be unwilling or unable to accept. From a manufacturer's perspective, an alternative would be to treat other autonomous systems as essentially unpredictable. Robustness testing of behaviours, and information passed via an agreed standard interface, would be helpful in this regard.

Conceptually, this approach moves other manufacturers' systems into the environment projection, so there would be value in considering the objectives associated with that projection.

PLT2-3: **Unavailability or unreliability of interacting items does not make the platform unsafe.**

Discussion: Although the distinction is not always rigid, generally speaking, it is easier to control things that are part of the platform, rather than interacting items. For this reason, there is a greater chance of interacting items being (or becoming) unavailable during operational use of the platform. Since the platform should remain safe at all times, it follows that unavailability of interacting items should not result in an unsafe platform.

Similarly, there is the possibility of interacting items being unreliable, for example, providing misleading information. Should it occur, this unreliability should not result in an unsafe platform.

Examples: Arguably, a good understanding of what constitutes safe behaviour (from the objectives associated with the behavioural specification projection) combined with a good understanding of potential impacts of interacting items (from Objective PLT2-1) should make a significant contribution to satisfying this objective (i.e. Objective PLT2-3).

There are, however, some subtleties. For example, there may be circumstances in which unavailability, or unreliability, of interacting items causes a gradual reduction in operational performance until a point is reached whereby the platform is no longer safe. This could occur if, for example, regular (but not necessarily frequent) communications with a central control entity are required to account for distribution shift. A suitable response to this gradual degradation could be, firstly, informing the operator and, subsequently, placing the autonomous system in a safe, potentially non-operational, state.

7.3 People

It is possible to imagine autonomous systems that interact with people in a very limited, very indirect way: an autonomous system that manages load balancing across computational resources in order to deliver communications functions could be one example. Despite this, most autonomous systems are expected to involve significant interaction with people. This is especially true for safety-related systems, which are the focus of this document. Consequently, interactions between the platform and people need to be considered. These considerations need to cover the whole system lifecycle, as well as people directly involved with the platform (e.g. operators and maintainers) and bystanders.

There are four objectives associated with this projection.

PLT3-1: **Safety-related demands on people interacting with the platform are reasonable.**

Discussion: Despite their autonomous nature, some platforms may require people to perform certain functions to maintain safety. For example, a person may be required to perform a monitoring function

and to take over control in off-nominal situations. Alternatively, a person may be required to perform regular maintenance activities, including sensor calibrations. Regardless of their nature, if human actions are necessary to maintain safety then these actions need to be reasonable; for example, they should not exceed the expected limitations of the relevant demographic. Equivalently, a system cannot be made safe by asking people to perform unreasonable actions.

Examples: Part of this objective relates to understand what safety-related demands are being placed on people. The various objectives associated with the behavioural specification projection should provide a good understanding of safety from a holistic perspective. Techniques such as task analysis or the Functional Resonance Analysis Method (FRAM) [45] may help identify those parts that rely on human actions.

There are likely to be domain-specific considerations in determining whether these actions are "reasonable". For example, a function that any member of the general public is expected to complete should be less demanding than a function that will only be completed by specially selected, trained and certified staff.

Human actions are likely to be repeated many times during the life of an autonomous system; they may be repeated many times during a single operational use of such a system. Consequently, generic information on human performance (e.g. reliability, timeliness, attentiveness) may help inform judgements on reasonableness.

PLT3-2: Suitable interfaces are provided for people that may interact with the platform.

Discussion: In many cases, people would be expected to interact with the platform through designed interfaces. Experience with traditional systems has demonstrated the importance of these interfaces. Arguably, in the case of autonomous systems, these interfaces become even more important: partly because of the unfamiliarity caused by the way autonomy changes the role of the human; and partly because there are examples of humans over-riding system behaviour because of poor situational awareness.

Although specifically-designed interfaces are of great importance, there may be cases where people interact with a platform in a more general manner. For example, in the case of self-driving vehicles, pedestrians may observe the vehicle's trajectory, as well as other entities in the local environment, to form an opinion of how the autonomous system will behave. In the longer-term, standards and norms that govern these interactions may be defined or emerge: the use of audible alarms when a vehicle is reversing may be one example. However, currently, these interactions need careful thought from the perspective of each autonomous system.

Examples: To understand what interfaces are required, and what characteristics would make these appropriate (or inappropriate), there is a need to understand the range of situations in which people will interact with the platform. Many of these situations should be identified as part of understanding the platform's behavioural specification, including what constitutes safe operation (Objective PLT1-2). In addition, there may be a need to search for other interactions with people that do not naturally arise from such considerations. This could possibly be achieved by "walking through" typical use cases for the platform, with the specific aim of identifying interactions with people. This should consider both pull-style interfaces, where the person has to request information, and push-style interfaces, where the information is always available.

As well as understanding the situations where interactions occur, there is also a need to understand

the different types of people that may be involved. For example, the interface provided to a child may be different from that provided to an adult. Similarly, the interface provided to a native speaker may be different from that provided to someone unfamiliar with that language. To give one final example, the interface provided to a trained (and certified) operator may be different from that provided to a lay person. Many of these considerations may be addressed by traditional Human-Machine Interface (HMI) approaches.

Provided it was sufficiently realistic (and demonstrated to be so), an interactive simulation could be a useful tool to help develop the understanding discussed in the preceding paragraphs.

These traditional approaches may need to be supplemented with techniques associated with "Explainable AI" [1]. In this context, there is a difference between local explainability, which relates to a single input, and global explainability, which relates to the input domain (or a large portion of it). As outlined in the preceding paragraph, there is also a need to account for the context within which the explanation is provided, or required.

PLT3-3: **Appropriate training is provided for platform users and maintainers.**

Discussion: A key aspect of many autonomous systems is that they change the role of the human. This change in role means that additional training is likely to be beneficial. A different form of training may also be helpful. More specifically, even though autonomy typically reduces the burden on the human there may still be a need to train operators: this could, for example, involve education about expected platform behaviour in both nominal and off-nominal situations. Similarly, some form of training may be required for maintainers, for example, being able to manage updates of ML-based models.

Examples: Training for users and maintainers is likely to be platform, or application, specific. As for traditional systems, there may be a need for formal certification and, potentially, re-certification.

The nature of autonomy enabling algorithms, especially those developed ML is such that autonomous systems may be updated more frequently than is typically the case for current systems. This raises the importance of considering the impact of individual changes and, especially, the cumulative impact of multiple changes. A time-based approach to re-training, combined with a bound on the scale and number of changes, could be adopted. Alternatively, a metric that estimates the cumulative effect of multiple changes could be developed, with re-training being invoked once that metric exceeds a threshold. In either case, care need to be taken to protect against a conscious or unconscious desire to minimise re-training (e.g. to minimise opertaional costs) to such an extent that platform operation becomes unsafe.

PLT3-4: **The platform is appropriately protected against harm from adversarial actors.**

Discussion: The possibility of people intending to deliberately cause the platform to behave in an unsafe manner needs to be considered. Given the extensive use of software and computational hardware in autonomy-enabling technologies, there is a significant cyber-related attack surface that adversaries may seek to exploit.

However, this objective is intentionally wider than cyber-related considerations; it also includes other activities that humans could take that affect platform behaviour, especially for effects that have the potential to lead to unsafe behaviour. One example activity could be pedestrians and cyclists positioning themselves to try and influence vehicle trajectories. Another example could be platform users implementing unauthorised modifications, perhaps with the aim of altering a vehicle's performance.

Examples: The potential scope of this objective is vast. From a cyber security perspective, the DfT Guidelines for Cyber Security for Connected and Automated Vehicles are likely to be useful [22]. Likewise, the combined development of a security case and a safety case may help [49].

From the specific perspective of autonomy-enabling technologies, protecting against the possibility of training abuse (e.g. via data poisoning or through tainted pre-trained models [37]) is important. Coupling this with monitoring of operational inputs and autonomous (or model) behaviour is likely to be beneficial. These concepts are also relevant to the autonomy architecture-level (e.g. Objectives ARC1-2 and ARC1-4).

An understanding of potential non-cyber effects may be gained, at least in theory, by a suitably broad red teaming exercise [52]. Like all red team exercises, this would require a group of innovative thinkers; it is also likely to require people with a good understanding of how the platform would respond to novel situations. Given the wide potential behaviour associated with many autonomous systems, this understanding may be difficult to obtain. It could, possibly, be replaced by appropriate tests (including simulations). Adopting this approach could result in a series of red team sessions, in which possible adversary actions are proposed, with the results of these being determined offline and fed back into a subsequent session.

If a suitable simulation was available, which included representations of the platform and adversarial actors, then there may be value in adopting some form of automated red teaming.

7.4 Environment

A key aspect of many autonomous systems is the complex, diverse and dynamic nature of the intended operational domain. These characteristics mean it is impossible to precisely define beforehand what entities will be present in the environment or how they will behave. The environment projection is concerned with these entities, which the platform developer and operator have little or no control over, as well as geographical and meteorological properties.

There are two objectives associated with this projection.

PLT4-1: **Elements of the environment relevant to the safe operation of the platform are identified and understood.**

Discussion: In order to make arguments about, and demonstrate, the safe operation of a platform it is necessary to understand the environment in which the platform may be used, noting that this is not necessarily the same as the intended operational domain. Part of this understanding is concerned with interacting items (Objective PLT2-1) although the most significant part comes from understanding entities that may be present in, and properties of, the environment.

Examples: In some cases, an understanding of the environment may be relatively easy to establish. This may be the case for an autonomous system that forms part of an application to support medical diagnoses. More generally, the approaches used to understand what constitutes safe behaviour (Objective PLT1-2) and what interacting items may affect safety (Objective PLT2-1) are likely to help identify elements of the environment that are relevant to this objective.

It is important that the environment is described in an inclusive manner, without jeopardising understandability. In some cases, it may be plausible to list all entities within a given class (or sub-class) that may form part of the environment. Existing classifications, for example, of road vehicles, may help in this endeavour. In other cases it may be appropriate to define parts of the environment by exclusion; that

is, the environment may contain everything apart from certain specified items.

The understanding of the environment needs to consider both nominal (i.e. expected) and off-nominal (i.e. unexpected) situations. The presence of emergency vehicles, or broken down vehicles may be examples of the latter class. It may be helpful to develop a range of situations that apply to an entire class of autonomous systems: for example, regardless of manufacturer, self-driving cars (operating in the same geographical region) are likely to experience the same, or very similar, environments.

In terms of geographical and meteorological properties of the environment, bounds are likely to be available from system-level requirements. These should, for example, identify minimum and maximum operating temperatures. If it is relevant, they may also specify bounds on properties like altitude, vibration, acceleration, gradient and so on.

The potential effect of environmental properties on redundancy needs to be understood. For example, suppose a platform is fitted with a range of sensors to provide redundancy, but some of the sensors do not function adequately in certain meteorological conditions (e.g. fog). In such circumstances, fog may not immediately create an unsafe situation, although it may reduce safety margins. This could create a case where a single failure resulted in unsafe operation, even though from a theoretical perspective the platform was equipped with redundant sensors.

PLT4-2: **Situational awareness of the platform's environment is maintained.**

Discussion: Situational awareness relates to the platform's understanding of the environment. This includes understanding what other entities exist, what courses of action these entities are likely to take and what courses of action they could potentially take in a worst case (from a safety perspective) scenario. This awareness is important as it forms the basis of decision making: making a correct decision based on incorrect data could lead to unsafe outcomes.

The importance of situational awareness is also emphasised in the first two components of common models for autonomous systems, for example: Monitor, Analyse, Plan, Execute (MAPE); and Sense, Understand, Decide, Act (SUDA).

Examples: For some platforms, maintaining situational awareness may be relatively straightforward. For example, in the case of a medical diagnosis system, the associated environment could include data about the image (e.g. the type of scanner used, the time and date at which the image was taken, an associated patient identifier); it could also include general healthcare information relevant to the patient (e.g. current medication). For other platforms, maintaining suitable awareness of the environment (in all operating conditions) could be a significant challenge.

Some aspects of situational awareness may come from off-platform sources: mapping information may be a good example of this. Ensuring that safety-related impacts of such data have been considered is important. The Data Safety Guidance may be valuable in this regard [97].

Other aspects of situational awareness may be more dynamic. Understanding the current meteorological conditions is likely to be important. As noted earlier these could, potentially, reduce safety margins. The impact of meteorological conditions on the environment also need to be considered: a wet or icy road surface will affect braking distances, for example.

To understand what level of situational awareness is required, analysis needs to be conducted to understand, for example, the accuracy and latency with which information is required. The control structure diagram (associated with STPA [47]) may help identify the relevant pieces of information;

subsequent analyses could provide details of the properties this information needs to exhibit.

In many cases, the environment changes relatively smoothly with time. Consequently, there may be a need to measure and propagate uncertainty about the environment and, especially, entities within it. This may allow the platform to adopt certain behaviours to either reduce this uncertainty (e.g. by changing the way sensors are used) or to reduce the potential consequences of it (e.g. by travelling more slowly).

This page is intentionally blank

8 Tailoring

Safety-related standards and guidance documents typically allow for some form of tailoring, so that more effort can be devoted to high-criticality parts, without introducing inappropriate assurance burdens on lower-criticality parts. The lack of any information relating to tailoring was an identified "Known Issue" in earlier versions of this document. This section addresses, as far as possible, this issue.

There are two main components to a tailoring scheme. Firstly, describing the level of integrity needed and, secondly, indicating how this affects what needs to be done. Each of these components is considered below, before the section concludes with a brief summary.

An illustration of tailoring applied to objectives in the computation-level framework is provided in Appendix A.

8.1 Integrity/Criticality

There are well-established ways of *describing* integrity or criticality, for example, Safety Integrity Levels (SILs) and Development Assurance Levels (DALs). Current methods to derive integrity levels should be applicable to autonomous systems. Similar to other application areas, different parts of a system may attract different levels: a low-criticality computation could be used within a high-criticality platform, for example.

There are well-established safety engineering techniques for *assigning* integrity levels to individual parts of a system. Since, in general, they have been designed to apply to a wide variety of systems, these techniques should be appropriate for use in an autonomous systems context. In particular, techniques that trace the effect of component-level failures to system-level behaviour ought to be applicable.

However, some autonomous systems components, for example, computations implemented using ML, may exhibit new types of failure mode or they may exhibit known failure modes with much higher frequency than is typically the case. Undetectable failures (e.g. when a classifier incorrectly, but confidently, mis-classifies an input), or erratic performance over time (e.g., classification on a video stream in the rain) may be a particular concern. These considerations were a significant motivation for including the tolerance projection in the autonomy architecture framework and, especially, objectives ARC1-3, "Faults and failures internal to the computation are tolerated.", and ARC1-5, "Incorrect computation outputs are tolerated."

Any approach to tailoring this document should separately, and explicitly justify the contribution to safety from each computation, autonomy architecture and platform, and hence their assigned integrity levels.

8.2 Effect on Objectives

There are at least two ways that tailoring can affect objectives. It could mean that lower-criticality systems do not need to satisfy some objectives (e.g. DO-178C [82]). Alternatively, it could mean that different techniques are used to satisfy objectives, depending on the integrity level (e.g. IEC 61508 [46]).

The way that this document's objectives were derived suggests that all objectives are relevant, regardless of integrity level. This suggestion is also supported by experience gained from undertaking a number of comparisons (documented in [89]). Consequently, tailoring approaches are expected to change the degree of effort associated with, and the amount of evidence produced for, each objective. They are not expected to remove any objectives from consideration.

For those objectives that map easily to traditional engineering activities, this tailoring could be relatively straightforward. For example, tailoring of COM1-2, "Training data pre-processing methods do not introduce errors." could be based on Tool Qualification Levels [83] if the tools are traditional software.

However, the relative immaturity of autonomous systems engineering as a discipline means that, for many objectives, tailoring is not straightforward. Therefore, a justification for techniques used or not used to provide confidence in safety assurance should be considered. This information would also assist independent reviewers in determining whether each objective has been satisfied to an appropriate level of confidence. The degree of confidence needed for high-criticality systems means that very strong justifications for the techniques chosen would be expected.

8.3 Summary

Current methods for defining integrity levels should be applicable. In addition, current safety engineering approaches that assign integrity levels to individual parts (or requirements for parts) of a system should be useful. We note that autonomy-enabling components may exhibit novel failure modes and may significantly change the frequency or nature of known failure modes. Integrity levels should be separately documented, and justified, for each relevant part.

Tailoring approaches should be based on the level of effort, and amount of evidence, associated with each objective. They should not be used to discount any objectives. Cases where assurance techniques have or have not been employed should be documented and justified.

9 Summary

For ease of reference, this section lists the objectives discussed earlier.

9.1 Computation-Level

Table 5: Computation-Level Objectives

Id	Objective	Projection
COM1-1	Data is acquired and controlled appropriately.	Experience
COM1-2	Training data pre-processing methods do not introduce errors.	
COM1-3	Data captures the required algorithm behaviour.	
COM1-4	Adverse effects arising from distribution shift are protected against.	
COM1-5	Processing methods on inputs and outputs used during operation do not introduce errors.	
COM2-1	Functional requirements imposed on the algorithm are defined and satisfied.	Task
COM2-2	Non-functional requirements imposed on the algorithm are defined and satisfied.	
COM2-3	Algorithm performance is measured objectively.	
COM2-4	Performance boundaries are established and complied with.	
COM2-5	The algorithm is verified with an appropriate level of coverage.	
COM2-6	The test environment is appropriate.	
COM2-7	Each algorithm variant is tested appropriately.	
COM3-1	An appropriate algorithm type is used.	Algorithm
COM3-2	Typical errors are identified and protected against.	
COM3-3	The algorithm's behaviour is explainable.	
COM3-4	Post-incident analysis is supported.	
COM4-1	The software is developed and maintained using appropriate standards.	Software
COM4-2	Software misbehaviour does not result in incorrect outputs from the algorithm.	
COM5-1	Appropriate computational hardware standards are employed.	Hardware
COM5-2	Hardware misbehaviour does not result in incorrect outputs from the algorithm.	

9.2 Autonomy Architecture-Level

Table 6: Autonomy Architecture-Level Objectives

Id	Objective	Projection
ARC1-1	Failures of sub-systems that provide computation inputs are tolerated.	Tolerance
ARC1-2	Operational inputs inconsistent with training inputs are tolerated.	
ARC1-3	Faults and failures internal to the computation are tolerated.	
ARC1-4	Adversarial attempts to disrupt the computation are tolerated.	
ARC1-5	Incorrect computation outputs are tolerated.	
ARC2-1	Relevant information is presented to interacting parties.	Information Provision
ARC2-2	Relevant information is available to support maintenance and future development.	
ARC2-3	Relevant information is preserved to support post-incident analysis.	
ARC2-4	Information is managed securely.	
ARC3-1	Inappropriate or unauthorised adaptations do not occur.	Adaptation
ARC3-2	Computation behaviour is appropriate before, during and after an adaptation.	

9.3 Platform-Level

Table 7: Platform-Level Objectives

Id	Objective	Projection
PLT1-1	All aspects of platform behaviour that are achieved using autonomy-enabling techniques are justified.	Behavioural Specification
PLT1-2	Acceptably safe behaviour for the platform is defined.	
PLT1-3	[Objective deleted.]	
PLT1-4	The specified behaviour is safe in the presence of faults and failures, as well as foreseeable misuse and abuse.	
PLT1-5	The behaviour of the platform is verified.	
PLT1-6	Operational monitoring is sufficient to identify and support the mitigation of new hazards, including emerging cyber security threats.	
PLT2-1	Interacting items that affect the safe operation of the platform are identified and understood.	Interacting Items
PLT2-2	Interactions preserve platform safety.	
PLT2-3	Unavailability or unreliability of interacting items does not make the platform unsafe.	
PLT3-1	Safety-related demands on people interacting with the platform are reasonable.	People
PLT3-2	Suitable interfaces are provided for people that may interact with the platform.	
PLT3-3	Appropriate training is provided for platform users and maintainers.	
PLT3-4	The platform is appropriately protected against harm from adversarial actors.	
PLT4-1	Elements of the environment relevant to the safe operation of the platform are identified and understood.	Environment
PLT4-2	Situational awareness of the platform's environment is maintained.	

This page is intentionally blank

Appendix A An Illustration of Tailoring

Tables 8 to 12, inclusive, illustrate a possible tailoring for objectives associated with the computation-level framework. These tables are included in an appendix, rather than in the main body, because there is less confidence in their content than there is in the objectives themselves.

For simplicity, this tailoring only includes two levels of criticality. The first level, "Basic", is likely to be the minimum required in practically every circumstance. Equivalently, "Basic" is relevant for situations of little consequence, where the computation is of very low criticality (e.g. when it is monitored by a high-integrity sub-system). In contrast, the second level, "Very High", is a plausible maximum requirement. Obviously, additional levels could be included between these extremes, as necessary.

For brevity, this tailoring provides only the titles of activities associated with each of these levels. More details on these activities should be accessible from the discussions in the main body of this document and the references therein.

Table 8: Possible Tailoring for Objectives Associated with the Experience Projection

Activity	Basic	Very High
Objective COM1-1: Data is acquired and controlled appropriately.		
Data acquisition process is documented.	✓	✓
Appropriate meta-data is captured and documented.		✓
Any software used for data collection is suitably verified.		✓
Externally generated data is checked for "data poisoning".		✓
Externally generated data is confirmed as semantically accurate.		✓
Where appropriate, uncertainty is reflected in the data.		✓
Data is stored in a system that protects against unintentional changes.	✓	✓
Data is stored in a system that protects against unauthorised changes.		✓
Objective COM1-2: Training data pre-processing methods do not introduce errors.		
Pre-processing software used on training data is documented.	✓	✓
Pre-processing software used on training data has defined requirements.		✓
Pre-processing software used on training data is shown to meet its requirements.		✓
Pre-processing software used on training data is protected against unintentional changes.	✓	✓
Pre-processing software used on training data is protected against unauthorised changes.		✓
Objective COM1-3: Data captures the required algorithm behaviour.		
Relationship between operating domain and data is documented.	✓	✓
Exploratory data analysis is performed, alongside domain experts.		✓
Typical samples are identified and checked.	✓	✓
Outlier samples are identified and checked.		✓
Inclusion of "rare but important" events is checked.		✓
Data is augmented to protect against adversarial examples.		✓

Activity	Basic	Very High
Objective COM1-4: Adverse effects arising from distribution shift are protected against.		
Data is summarised to allow shift to be detected for multiple inputs.	✓	✓
Data is summarised to allow shift to be detected for single inputs.		✓
Tests are conducted to confirm adequacy of detection methods.		✓
Objective COM1-5: Processing methods on inputs and outputs used during operation do not introduce errors.		
Processing methods are documented	✓	✓
Processing methods have defined requirements		✓
Processing methods are shown to meet defined requirements		✓

Table 9: Possible Tailoring for Objectives Associated with the Task Projection

Activity	Basic	Very High
Objective COM2-1: Functional requirements imposed on the algorithm are defined and satisfied.		
Testing confirms functional requirements are satisfied.	✓	✓
Testing is based on formally documented test cases.		✓
Testing is conducted by an independent party.		✓
Testing is conducted at the computation and platform levels.		✓
Objective COM2-2: Non-functional requirements imposed on the algorithm are defined and satisfied.		
Testing confirms non-functional requirements are satisfied.	✓	✓
Testing is based on formally documented test cases.		✓
Testing is conducted by an independent party.		✓
Testing is conducted at the computation and platform levels.		✓
Objective COM2-3: Algorithm performance is measured objectively.		
Common measures (e.g. precision, recall, accuracy) are used.	✓	✓
Performance measures are robust against unbalanced data.		✓
Performance measures consider all stakeholder perspectives.		✓
Performance measures are robust against reward hacking.		✓
Performance measures include the use of an independent data set.		✓
Performance measures include the use of formal methods based approaches.		✓
The algorithm is checked for potential bias.		✓
Objective COM2-4: Performance boundaries are established and complied with.		
Boundaries of the data are established and checked.	✓	✓
Boundaries of the data are confirmed with domain experts.		✓
Holes in the data are identified and confirmed with domain experts.		✓
Rare data items are identified and confirmed with domain experts.		✓
Single class regions are identified and confirmed with domain experts.		✓
Objective COM2-5: The algorithm is verified with an appropriate level of coverage.		

Activity	Basic	Very High
Verification suitably covers the operational input domain.	✓	✓
Verification suitably covers the theoretical input domain.		✓
Verification suitably covers the operational failure domain.		✓
Verification suitably covers the operational adversarial domain.		✓
Verification suitably covers algorithm internal features (e.g. neuron activations).		✓
Formal methods are used to justify behavioural claims.		✓
Sufficient repetitions are conducted in the case of stochastic rewards.		✓
Sufficient repetitions are conducted in the case of changing environments.		✓
Objective COM2-6: The test environment is appropriate.		
Test environment is suitably verified.	✓	✓
Test environment is suitably validated.		✓
Test environment is under configuration control.		✓
Test environment is protected against unintentional changes.	✓	✓
Test environment is protected against unauthorised changes.		✓
Objective COM2-7: Each algorithm variant is tested appropriately.		
A common set of tests is identified and applied to each variant.	✓	✓
Testing includes comparisons between different variants.		✓

Table 10: Possible Tailoring for Objectives Associated with the Algorithm Projection

Activity	Basic	Very High
Objective COM3-1: An appropriate algorithm type is used.		
The choice of algorithm type is justified using theoretical arguments.	✓	✓
The choice of algorithm type is justified using empirical arguments.		✓
The choice of hyperparameters is justified using empirical arguments.		✓
Objective COM3-2: Typical errors are identified and protected against.		
Over-fitting is protected against.	✓	✓
Data leakage is protected against.		✓
Adversarial examples are protected against.		✓
Under-representation of rare events is protected against.		✓
Loopholes in the test environment are protected against.		✓
Additional typical errors are identified and protected against.		✓
Objective COM3-3: The algorithm's behaviour is explainable.		
The approach to explainability is documented.	✓	✓
Behaviour on single inputs can be explained.	✓	✓
Behaviour on groups of inputs can be explained.		✓
Explanations can be tailored for different recipients.		✓

Activity	Basic	Very High
Objective COM3-4: Post-incident analysis is supported.		
Algorithm choice supports recording of the most recent computation.	✓	✓
Algorithm choice supports recording for an operationally relevant period.		✓
Post-incident analysis is conducted as part of computation development.		✓

Table 11: Possible Tailoring for Objectives Associated with the Software Projection

Activity	Basic	Very High
Objective COM4-1: The software is developed and maintained using appropriate standards.		
Equivalent to RTCA DO-178C, Level D (or similar).	✓	
Equivalent to RTCA DO-178C, Level A (or similar).		✓
Objective COM4-2: Software misbehaviour does not result in incorrect outputs from the algorithm.		
Input and expected output patterns are recorded for built-in test.	✓	✓
Software implementation includes checks (e.g. for numeric overflow).		✓

Table 12: Possible Tailoring for Objectives Associated with the Hardware Projection

Activity	Basic	Very High
Objective COM5-1: Appropriate computational hardware standards are employed.		
Equivalent to RTCA DO-254, Level D (or similar).	✓	
Equivalent to RTCA DO-254, Level A (or similar).		✓
Objective COM5-2: Hardware misbehaviour does not result in incorrect outputs from the algorithm.		
Hardware is suitable for the expected environmental conditions.	✓	✓
Implementation is robust against single event upsets.		✓
Implementation is robust against different workloads (e.g. use of shared resources).		✓
Implementation is robust against unintentional hardware configuration changes.		✓

Appendix B Known Issues

This appendix provides a list of known issues. These are items that were identified, but not resolved, during the creation of the current version of this document. Generally speaking, resolution of these issues needs a greater amount of knowledge and experience than is currently available. Occasionally, resolution requires more resource than was available to support the production of this document. In all cases, the issue was deemed sufficiently important to warrant specific capture and tracking.

Note that the following list is not intended to be complete.

- Whilst some experience has been gained from applying SCSC-153A and SCSC-153B, arguably, the objectives in this document still place a greater reliance on theoretical argument than on practical experience. This theoretical argument has been strengthened by comparisons with other safety assurance guidelines (see [89]). Despite this progress, further use would allow for greater confidence in this document's objectives.
- Currently, this document does not provide much information on integration of either Software (SW) and Hardware (HW) or SW-SW integration. However, the objectives associated with the autonomy architecture-level framework are helpful in this regard.
- The discussion section associated with each objective provides suggestions for how the objective may be at least partially met. However, at the moment, there are many cases where it is not clear how best to satisfy an objective and, equally important, how best to demonstrate this satisfaction. A larger amount of practical experience is needed before this specific type of guidance can be provided.
- The tailoring appendix is illustrative only (in Appendix A) and only addresses two different integrity levels at present. The list of proposed measures requires practical application for further validation.
- The relationship between adaptation objectives, and algorithms that continue learning during operational use is not addressed in detail in this version of the document.
- This version of the document does not address the use of AI based tools, such as Large Language Models (LLM), for the purposes of supporting the development process.
- The development of rigorous descriptions of operating environments, such as Operational Design Domain (ODD) [86, 51], and the relationship with requirements is not discussed in this version of the document.
- Requirements are mentioned numerous times within this guidance and are vital to being able to explain what an algorithm is intended to do and hence provide a base against which to both verify, if that is possible, and to validate (see COM2-1 "Functional requirements imposed on the algorithm are defined and satisfied."). There is ongoing debate specifically within the ML community about whether the 'data is the requirements' (see COM1-3 "Data captures the required algorithm behaviour.") or/and 'the requirements will only be known once the system has been trained'. However, neither position would be adequate for the rigour needed for high integrity system safety and safety architectures.

  Instead, there needs to be a set of functional behaviours desired (required) of a ML system to drive selection of data for training. For example, if an algorithm is required to actively manage a control system there has to be a defined behavioural set of requirements to show that the safety critical control system will always control the system. Or, if a ML based classifier needs to identify cancer in humans from X-Ray fiche, the training data needs to be of relevant pictures; there has to be a set of requirements for the data content and its distribution and variation. The question for the community, which cannot yet be adequately resolved in this publication, is to be able to adequately define requirements for ML.

For completeness, the following list briefly summarises known issues from the previous version that have, at least partially, been addressed in the current version.

- Appendices containing comparisons to other similar scoped assurance documents and academic papers are now removed to a separate document [89]. This is available on the Safety Critical Systems Club web repository.
- Further clarity on the notion of "Integrity Levels" and criticality has been included (in Section 8).
- Additional clarification has been added to consider "processing" of data and signals to and from an algorithm in COM1-5 “Processing methods on inputs and outputs used during operation do not introduce errors.”, for example, when output values are normalised or otherwise transformed prior to use by another element of the platform.

Appendix C Abbreviations

AI	Artificial Intelligence
AS	Autonomous Systems
BIT	Built-In Test
CM	Configuration Management
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAL	Development Assurance Level
DfT	Department for Transport
DLOD	Defence Line of Development
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
FPGA	Field Programmable Gate Array
FRAM	Functional Resonance Analysis Method
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit
HAZOP	Hazard and Operability Study
HMI	Human-Machine Interface
HW	Hardware
LLM	Large Language Models
MAPE	Monitor, Analyse, Plan, Execute
MC/DC	Modified Condition / Decision Coverage
MEL	Minimum Equipment List
ML	Machine Learning
MOD	Ministry Of Defence
NCSC	National Cyber Security Centre
NN	Neural Network
ODD	Operational Design Domain
OECD	Organisation for Economic Co-operation and Development
PCA	Principal Component Analysis
PDI	Parameter Data Item
PSH	Product Service History
RL	Reinforcement Learning
RNN	Recurrent Neural Network

SASWG	Safety of Autonomous Systems Working Group
SCSC	Safety Critical Systems Club
SEU	Single Event Upset
SIL	Safety Integrity Level
SMS	Safety Management System
SoC	System-on-Chip
SOUP	Software Of Uncertain Pedigree
SSS	Safety-critical Systems Symposium
STPA	Systems Theoretic Process Analysis
SUDA	Sense, Understand, Decide, Act
SVM	Support Vector Machine
SW	Software
SWaP	Size, Weight and Power
TEVV	Test, Evaluation, Verification and Validation
TPU	Tensor Processing Unit
TQL	Tool Qualification Level
TTV	Training, Test and Verification
UCI	University of California, Irvine
WCET	Worst Case Execution Time
YOLO	You Only Look Once

Appendix D References

[1] A. Adadi and M. Berrada, *Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)*, IEEE Access, 6 (2018), pp. 52138–52160.

[2] C. C. Aggarwal and S. Y. Philip, *Privacy-preserving data mining: a survey*, in Handbook of database security, Springer, 2008, pp. 431–460.

[3] R. Alexander, H. R. Hawkins, and A. J. Rae, *Situation coverage–a coverage criterion for testing autonomous robots*, University of York, (2015).

[4] E. Alves, D. Bhatt, B. Hall, K. Driscoll, A. Murugesan, and J. Rushby, *Considerations in assuring safety of increasingly autonomous systems*, tech. rep., NASA, 2018.

[5] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, *Concrete problems in AI safety*, arXiv, 1606.06565 (2016).

[6] A. Antoniou, A. Storkey, and H. Edwards, *Data augmentation generative adversarial networks*, arXiv, 1711.04340 (2017).

[7] F. Arnez, H. Espinoza, A. Radermacher, and F. Terrier, *A comparison of uncertainty estimation approaches in deep learning components for autonomous vehicle applications*, AI Safety 2020, 2020 (2020), p. 8.

[8] R. Ashmore and A. Banks, *The utility of neural network test coverage measures*, SafeAI, (2021).

[9] R. Ashmore, R. Calinescu, and C. Paterson, *Assuring the machine learning lifecycle: Desiderata, methods, and challenges*, arXiv, 1905.04223 (2019).

[10] R. Ashmore and M. Hill, *Boxing clever: Practical techniques for gaining insights into training data and monitoring distribution shift*, in First International Workshop on Artificial Intelligence Safety Engineering, 2018.

[11] R. Ashmore and E. Lennon, *Progress towards the assurance of non-traditional software*, in Developments in System Safety Engineering, Proceedings of the Twenty-fifth Safety-Critical Systems Symposium, Safety-Citical Systems Club, 2017. ISBN 978-1540796288.

[12] R. Ashmore and B. Madahar, *Rethinking diversity in the context of autonomous systems*, in Engineering Safe Autonomy, 27th Safety-Critical Systems Symposium, Safety-Citical Systems Club, 2019, pp. 175–192.

[13] R. K. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilovic, S. Nagar, K. N. Ramamurthy, J. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang, *AI fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias*, arXiv, 1810.01943 (2018).

[14] Centre for Connected and Autonomous Vehicles, *Code of practice: Automated vehicle trialling*, tech. rep., Department for Transport, February 2019.

[15] Certification Authorities Software Team, *Guidance for assessing the software aspects of product service history of airborne systems and equipment*, Tech. Rep. CAST-1, Federal Aviation Authority, June 1998.

[16] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, *Targeted backdoor attacks on deep learning systems using data poisoning*, arXiv, 1712.05526 (2017).

[17] C.-H. Cheng, C.-H. Huang, and H. Yasuoka, *Quantitative projection coverage for testing ML-enabled autonomous systems*, arXiv, 1805.04333 (2018).

[18] J. J. Chilenski and S. P. Miller, *Applicability of modified condition/decision coverage to software testing*, Software Engineering Journal, 9 (1994), pp. 193–200.

[19] P. Chrabaszcz, I. Loshchilov, and F. Hutter, *Back to basics: Benchmarking canonical evolution strategies for playing Atari*, arXiv, 1802.08842 (2018).

[20] A. Cui and S. J. Stolfo, *Defending embedded systems with software symbiotes*, in Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, RAID'11, Berlin, Heidelberg, 2011, Springer-Verlag, pp. 358–377.

[21] K. Czarnecki and R. Salay, *Towards a framework to manage perceptual uncertainty for safe automated driving*, in International Conference on Computer Safety, Reliability, and Security, Springer, 2018, pp. 439–445.

[22] Department for Transport, *The Key Principles of Cyber Security for Connected and Automated Vehicles*, HM Government, 2017.

[23] D. Dheeru and E. Karra Taniskidou, *UCI machine learning repository*, 2017.

[24] S. Dogramadzi, M. E. Giannaccini, C. Harper, M. Sobhani, R. Woodman, and J. Choung, *Environmental hazard analysis-a variant of preliminary hazard analysis for autonomous mobile robots*, Journal of Intelligent & Robotic Systems, 76 (2014), pp. 73–117.

[25] J. Dunjó, V. Fthenakis, J. A. Vílchez, and J. Arnaldos, *Hazard and operability (HAZOP) analysis. a literature review*, Journal of hazardous materials, 173 (2010), pp. 19–32.

[26] European Aviation Safety Agency, *Certification specifications for aeroplane flight simulation training devices*, tech. rep., European Aviation Safety Agency, 2012.

[27] FAA, *Assurance of multicore processors in airborne systems*, Tech. Rep. DOT/FAA/TC-16/51, FAA, July 2017.

[28] J. M. Faria, *Non-determinism and failure modes in machine learning*, in 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, 2017, pp. 310–316.

[29] F. M. Favarò, D. W. Jackson, J. H. Saleh, and D. N. Mavris, *Software contributions to aircraft adverse events: Case studies and analyses of recurrent accident patterns and failure mechanisms*, Reliability Engineering & System Safety, 113 (2013), pp. 131–142.

[30] M. Fischer, M. Balunovic, D. Drachsler-Cohen, T. Gehr, C. Zhang, and M. Vechev, *DL2: Training and querying neural networks with logic*, Internet, (2018).

[31] M. Fredrikson, S. Jha, and T. Ristenpart, *Model inversion attacks that exploit confidence information and basic countermeasures*, in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 1322–1333.

[32] J. Garcia and F. Fernández, *A comprehensive survey on safe reinforcement learning*, Journal of Machine Learning Research, 16 (2015), pp. 1437–1480.

[33] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumeé III, and K. Crawford, *Datasheets for datasets*, arXiv, 1803.09010 (2018).

[34] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, *Shortcut learning in deep neural networks*, arXiv, 2004.07780 (2020).

[35] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow, *Adversarial spheres*, arXiv, 1801.02774v2 (2018).

[36] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, in Proceedings of the 3rd International Conference on Learning Representations, 2015.

[37] T. Gu, B. Dolan-Gavitt, and S. Garg, *Badnets: Identifying vulnerabilities in the machine learning model supply chain*, arXiv, 1708.06733 (2017).

[38] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, *A survey of methods for explaining black box models*, ACM Computing Surveys (CSUR), 51 (2018), p. 93.

[39] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, *Learning from class-imbalanced data: Review of methods and applications*, Expert Systems with Applications, 73 (2017), pp. 220–239.

[40] B. Hamner, *Machine learning gremlins*. Presented at the 2014 O'Reilly Strata Data Conference in Santa Clara, 2014.

[41] L. A. Hendricks, Z. Akata, M. Rohrbach, J. Donahue, B. Schiele, and T. Darrell, *Generating visual explanations*, in Proceedings of the 14th European Conference on Computer Vision, 2016, p. 3–19.

[42] J. Hernández-Orallo, F. Martínez-Plumed, S. Avin, and S. O. hÉigeartaigh, *Surveying safety-relevant ai characteristics*, in Proceedings of 1st AAAI Workshop on Artificial Intelligence Safety, AAAI, 2019.

[43] A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez, *Explainability in deep reinforcement learning*, Knowledge-Based Systems, (2020), p. 106685.

[44] S. Holland, A. Hosny, S. Newman, J. Joseph, and K. Chmielinski, *The dataset nutrition label: A framework to drive higher data quality standards*, arXiv, 1805.03677 (2018).

[45] E. Hollnagel, *FRAM: the functional resonance analysis method: modelling complex socio-technical systems*, CRC Press, 2017.

[46] IEC, *Functional safety of electrical / electronic / programmable electronic safety-related systems*, Tech. Rep. IEC 61508, IEC, 2010.

[47] T. Ishimatsu, N. G. Leveson, J. P. Thomas, C. H. Fleming, M. Katahira, Y. Miyamoto, R. Ujiie, H. Nakao, and N. Hoshino, *Hazard analysis of complex spacecraft using systems-theoretic process analysis*, Journal of Spacecraft and Rockets, 51 (2014), pp. 509–522.

[48] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, *A review of yolo algorithm developments*, Procedia Computer Science, 199 (2022), pp. 1066–1073. The 8th International Conference on Information Technology and Quantitative Management (ITQM): Developing Global Digital Economy after COVID-19.

[49] N. Johnson and T. Kelly, *Safety-security assurance framework (SSAF) in practice*, 2018.

[50] C. Jones, R. Bloomfield, P. Froome, and P. Bishop, *Methods for Assessing the Safety Integrity of Safety-related Software of Uncertain Pedigree (SOUP).*, HSE Books, 2001.

[51] F. Kaakai, S. S. Adibhatla, G. Pai, and E. Escorihuela, *Data-centric operational design domain characterization for machine learning-based aeronautical products*, in Computer Safety, Reliability, and Security, J. Guiochet, S. Tonetta, and F. Bitsch, eds., Cham, 2023, Springer Nature Switzerland, pp. 227–242.

[52] M. Kardos and P. Dexter, *A simple handbook for non-traditional red teaming*, tech. rep., Defence Science and Technology Group Edinburgh, SA, 2017.

[53] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, et al., *The Marabou framework for verification and analysis of deep neural networks*, in International Conference on Computer Aided Verification, Springer, 2019, pp. 443–452.

[54] J. Z. Kolter and E. Wong, *Provable defenses against adversarial examples via the convex outer adversarial polytope*, arXiv, (2017).

[55] P. Koopman and F. Fratrik, *How many operational design domains, objects, and events?*, in SafeAI@ AAAI, 2019.

[56] V. Krakovna, *Specification gaming examples in AI*, Deep Safety, (2018).

[57] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105.

[58] J. Lemley, F. Jagodzinski, and R. Andonie, *Big holes in big data: A Monte Carlo algorithm for detecting large hyper-rectangles in high dimensional data*, in IEEE Computer Software and Applications Conf., 2016, pp. 563–571.

[59] G. Li, K. Pattabiraman, C.-Y. Cher, and P. Bose, *Understanding error propagation in GPGPU applications*, in SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2016, pp. 240–251.

[60] Z. C. Lipton, K. Azizzadenesheli, A. Kumar, L. Li, J. Gao, and L. Deng, *Combating reinforcement learning's sisyphean curse with intrinsic fear*, arXiv, 1611.01211 (2016).

[61] V. López, A. Fernández, S. García, et al., *An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics*, Information Sciences, 250 (2013), pp. 113–141.

[62] S. Ma, Y. Liu, G. Tao, W.-C. Lee, and X. Zhang, *Nic: Detecting adversarial samples with neural network invariant checking*, in NDSS, 2019.

[63] G. Mason, R. Calinescu, D. Kudenko, and A. Banks, *Assured Reinforcement Learning with Formally Verified Abstract Policies*, 2017.

[64] J. McDermid and Y. Jia, *Safety of artificial intelligence: A collaborative model*, AI Safety 2020, (2020), p. 7.

[65] MOD, *How defence works*, Tech. Rep. v4.2, MOD, 2015.

[66] A. S. Morcos, D. G. T. Barrett, N. C. Rabinowitz, and M. Botvinick, *On the importance of single directions for generalization*, arXiv, 1803.06959 (2018).

[67] J. G. Moreno-Torres, T. Raeder, R. Alaiz-RodriGuez, N. V. Chawla, and F. Herrera, *A unifying view on dataset shift in classification*, Pattern Recognition, 45 (2012), p. 521–530.

[68] A. Morozov, E. Valiev, M. Beyer, K. Ding, L. Gauerhof, and C. Schorn, *Bayesian model for trustworthiness analysis of deep learning classifiers*, AI Safety 2020, 2020 (2020), p. 7.

[69] NATS, *NATS system failure 12 December 2014 - final report*, tech. rep., NATS Independent Enquiry Panel, 2015.

[70] C. Northcutt, L. Jiang, and I. Chuang, *Confident learning: Estimating uncertainty in dataset labels*, Journal of Artificial Intelligence Research, 70 (2021), pp. 1373–1411.

[71] A. Odena and I. Goodfellow, *TensorFuzz: Debugging neural networks with coverage-guided fuzzing*, arXiv, 1807.10875 (2018).

[72] S. Padakandla, S. Bhatnagar, et al., *Reinforcement learning in non-stationary environments*, arXiv, 1905.03970 (2019).

[73] C. Paterson and R. Calinescu, *Detection and mitigation of rare subclasses in neural network classifiers*, arXiv, 1911.12780 (2019).

[74] C. Paterson, H. Wu, J. Grese, R. Calinescu, C. S. Pasareanu, and C. Barrett, *Deepcert: Verification of contextually relevant robustness for neural network image classifiers*, arXiv, 2103.01629 (2021).

[75] G. Pedroza and M. Adedjouma, *Safe-by-design development method for artificial intelligent based systems*, in SEKE 2019 : The 31st International Conference on Software Engineering and Knowledge Engineering, July 2019, pp. 391–397.

[76] K. Pohl and A. Metzger, *Software product line testing*, Communications of the ACM, 49 (2006), pp. 78–81.

[77] D. Powell, J. Arlat, Y. Deswarte, and K. Kanoun, *Tolerance of design faults*, in Dependable and Historic Computing, Springer, 2011, pp. 428–452.

[78] S. Rabanser, S. Günnemann, and Z. C. Lipton, *Failing loudly: An empirical study of methods for detecting dataset shift*, arXiv, 1810.11953 (2018).

[79] A. S. Rakin, Z. He, and D. Fan, *Bit-flip attack: Crushing neural network with progressive bit search*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 1211–1220.

[80] M. T. Ribeiro, S. Singh, and C. Guestrin, *Why should I trust you?: Explaining the predictions of any classifier*, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 1135–1144.

[81] RTCA, *Design assurance guidance for airborne electronic hardware*, Tech. Rep. DO-254, RTCA, April 2000.

[82] ——, *Software considerations in airborne systems and equipment certification*, Tech. Rep. DO-178C, RTCA, December 2011.

[83] ——, *Software tool qualification considerations*, Tech. Rep. DO-331, RTCA, December 2011.

[84] ——, *Airworthiness security process specification*, Tech. Rep. DO-326A, RTCA, August 2014.

[85] P. Ryan Conmy, B. Ozturk, T. Lawton, and I. Habli, *The impact of training data shortfalls on safety of ai-based clinical decision support systems*, in Computer Safety, Reliability, and Security, J. Guiochet, S. Tonetta, and F. Bitsch, eds., Cham, 2023, Springer Nature Switzerland, pp. 213–226.

[86] SAE, *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*, Tech. Rep. SAE J3016, SAE, April 2011.

[87] R. Salay and K. Czarnecki, *Using machine learning safely in automotive software: An assessment and adaption of software process requirements in iso 26262*, arXiv, 1808.01614 (2018).

[88] R. G. Sargent, *Verification and validation of simulation models*, in Simulation Conference (WSC), Proceedings of the 2009 Winter, IEEE, 2009, pp. 162–176.

[89] SASWG, *SCSC 153B appendices*, 2024. Available at http://scsc.uk.

[90] C. Schorn, A. Guntoro, and G. Ascheid, *Efficient on-line error detection and mitigation for deep neural network accelerators*, in International Conference on Computer Safety, Reliability, and Security, Springer, 2018, pp. 205–219.

[91] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins, and R. Anderson, *Sponge examples: Energy-latency attacks on neural networks*, arXiv, 2006.03463 (2020).

[92] S. Srisakaokul, Z. Wu, A. Astorga, O. Alebiosu, and T. Xie, *Multiple-implementation testing of supervised learning software*, in Proc. AAAI-18 Workshop on Engineering Dependable and Secure Machine Learning Systems (EDSMLS), 2018.

[93] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The Journal of Machine Learning Research, 15 (2014), pp. 1929–1958.

[94] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, *Concolic testing for deep neural networks*, arXiv, 1805.00089 (2018).

[95] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing properties of neural networks*, arXiv, 1312.6199 (2013).

[96] The Assurance Case Working Group, *Assurance Case Guidance: 'Challenges, Common Issues and Good Practice'*, no. v1.0, Safety-Critical Systems Club, 2021. SCSC-159.

[97] The Data Safety Initiative Working Group, *Data Safety Guidance*, no. v3.1, Safety-Critical Systems Club, 2019. SCSC-127D.

[98] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, *Stealing machine learning models via prediction apis*, in Proceedings of the 25th USENIX Security Symposium, 2016, p. 601–618.

[99] J. W. Tukey, *Exploratory data analysis*, vol. 2, Reading, Mass., 1977.

[100] S. Wachter, B. Mittelstadt, and C. Russell, *Counterfactual explanations without opening the black box: Automated decisions and the gdpr*, arXiv, 1711.00399 (2017).

[101] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, *Neural cleanse: Identifying and mitigating backdoor attacks in neural networks*, in 2019 IEEE Symposium on Security and Privacy (SP), IEEE, 2019, pp. 707–723.

[102] G. M. Weiss, *Mining with rarity: A unifying framework*, ACM Sigkdd Explorations Newsletter, 6 (2004), pp. 7–19.

[103] T.-W. Weng, P.-Y. Chen, L. M. Nguyen, M. S. Squillante, I. Oseledets, and L. Daniel, *Proven: Certifying robustness of neural networks with a probabilistic approach*, arXiv preprint arXiv:1812.08329, (2018).

[104] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang, *An empirical study on tensorflow program bugs*, in Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, New York, NY, USA, 2018, ACM, pp. 129–140.

Appendix E Contributors

This document has had the benefit of contributions from a large number of people, who work for a variety of organisations, which collectively span a range of different sectors. Note that contributions have been made on an individual basis and, in particular, the inclusion of an individual or organisation in the following list does *not* necessarily mean that individual or organisation agrees with the entire contents of the document.

Contributors to the current version include:

- Rasmus Adler, Fraunhofer Institute (Kaiserslautern)
- Rob Alexander, University of York
- Andrew Banks, LDRA
- Rajiv Bongirwar, Hemraj Consultants
- Ben Bradshaw, ZF
- John Bragg, MBDA UK Ltd.
- Lavinia Burski, AECOM
- Sarah Carrington, University of York
- Mike Ellims, Ricardo
- Jane Fenn, BAE Systems
- David Harvey, Thales UK
- Rob Harwood-Smith, TP Group
- Shakir Laher, NHS Digital
- Catherine Menon, University of Hertfordshire
- Will Onuba, Atkins
- Ganesh J. Pai, KBR / NASA Ames Research Center, USA
- Mike Parsons, CGI
- Davy Pissoort, KU Leuven
- Stewart Radcliffe, Thales UK
- Roger Rivett, Independent (ex-Jaguar Land Rover)
- Philippa Ryan, University of York
- Alan Simpson, EBENI
- Nick Tudor, D-RisQ
- Stuart Tushingham, Altran
- Bernard Twomey, Kongsberg

Contributors to all versions include:

- Rasmus Adler, Fraunhofer Institute (Kaiserslautern)
- Rob Alexander, University of York
- Hamid Asgari, Thales UK
- Rob Ashmore, Dstl
- Andrew Banks, LDRA
- John Birch, Horiba-MIRA
- Victor Bolbot, University of Strathclyde
- Rajiv Bongirwar, Hemraj Consultants
- Ben Bradshaw, ZF
- John Bragg, MBDA UK Ltd.
- Lavinia Burski, AECOM
- John Clegg, Independent (ex-QinetiQ)
- Timothy Coley, XPI Simulation
- Mike Ellims, Ricardo
- Jane Fenn, BAE Systems
- Chris Harper, Atkins/University of West of England
- David Harvey, Thales UK
- Rob Harwood-Smith, TP Group
- Richard Hawkins, University of York
- Nikita Johnson, University of York
- Neil Lewis, Dyson
- Catherine Menon, University of Hertfordshire
- Ken Neal, Ebeni
- Ashley Price, Raytheon
- Stewart Radcliffe, Thales UK
- Stuart Reid, STA Consulting
- Roger Rivett, Jaguar Land Rover
- Philippa Ryan, Adelard LLP
- Alan Simpson, Ebeni
- Rod Steel, Thales
- Mark Sujan, Human Reliability

- Nick Tudor, D-RisQ
- Stuart Tushingham, Altran
- Bernard Twomey, Kongsberg
- Michael Wong, Codeplay

This page is intentionally blank

Authored by the Safety of Autonomous Systems Working Group (SASWG), and building on previous editions, this document provides a coherent set of objectives that should be satisfied by any compelling safety argument for an autonomous system.

There is a deliberate focus on the novel safety assurance challenges that are associated with autonomy-enabling techniques, especially Artificial Intelligence (AI) developed using Machine Learning (ML) approaches. Consequently, this document is best employed within a wider Safety Management System (SMS).

The objectives are organised into three frameworks, covering issues related to computations, autonomy architectures, and platforms. Where possible, these frameworks and the associated objectives have been compared and contrasted to related documents. This provides some confidence that the listed objectives are representative of emerging consensus of what is required to support autonomous system safety arguments.

Changes introduced in this version include: additional clarity on proportional assurance and tailoring depending on integrity requirements and system criticality; updates based on recent experience of the authors and relevant research; an additional objective considering processing of data and signals to and from an algorithm (e.g., normalisation). This version addresses a number of known issues, including removing extra materials which were not part of the core assurance approach into a separate document available on the SCSC website.



The Safety Critical Systems Club (SCSC) is the professional network for sharing knowledge about safety-critical systems. It brings together: engineers and specialists from a range of disciplines working on safety-critical systems in a wide variety of industries; academics researching the arena of safety-critical systems; providers of the tools and services that are needed to develop the systems; and the regulators who oversee safety.

This document was written by the Safety of Autonomous Systems Working Group (SASWG), which is convened under the auspices of the SCSC. The goal of the SASWG is to produce clear guidance on how autonomous systems and autonomy technologies should be managed in a safety related context, throughout the lifecycle, in a way that is tightly focused on challenges unique to autonomy.