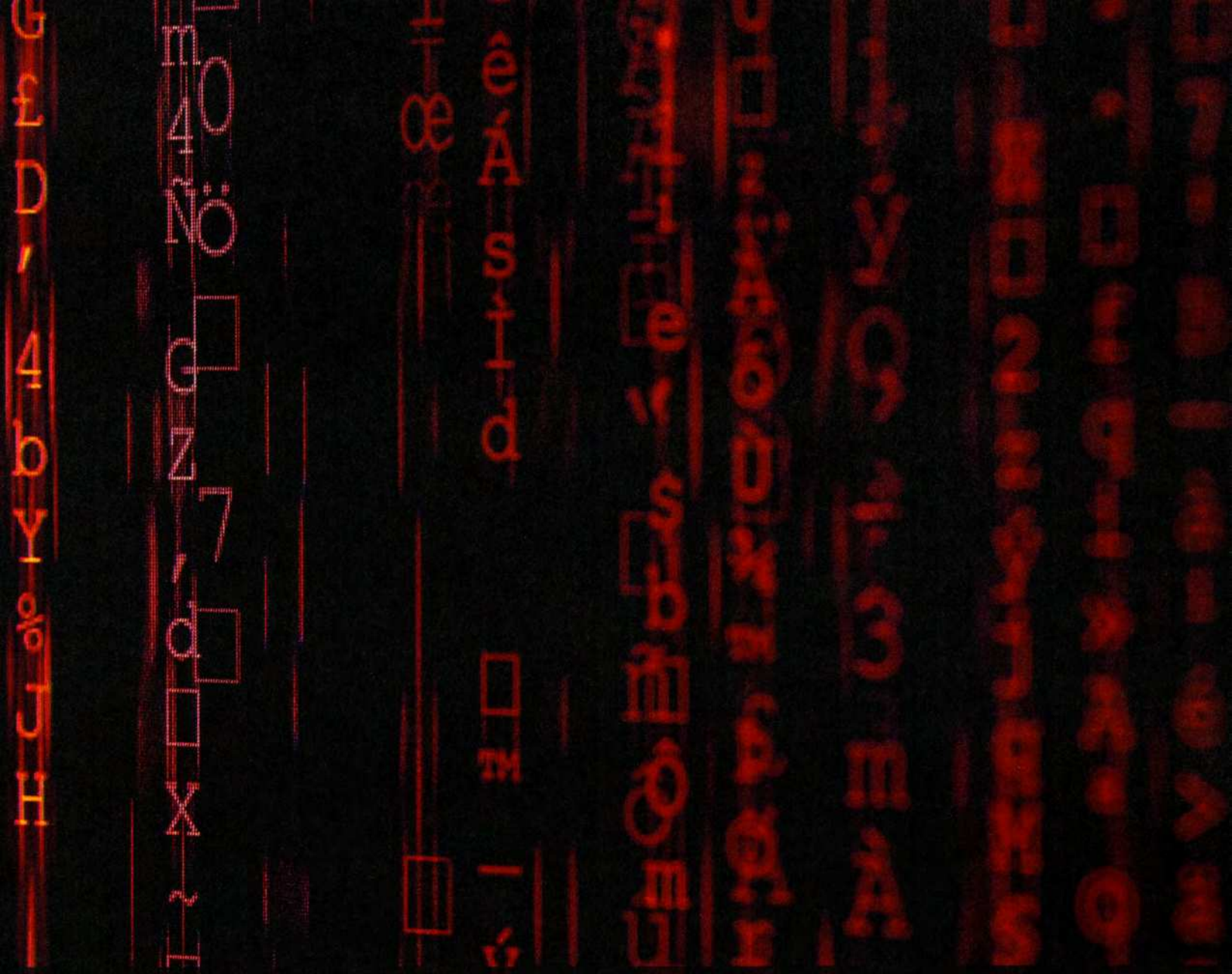

Safety Assurance Objectives for Autonomous Systems

Version 2.0 [SCSC-153A]

Safety of Autonomous Systems Working Group [SASWG]

ISBN: 978-1654029050

SCSC Publication Number: SCSC-153A

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

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

Distribution of hard copies of this document at the 2020 Safety-critical Systems Symposium was kindly supported by Rajiv Bongirwar of HEMRAJ Consultants Ltd UK (trading as HEMRAJ CONSULTING).

Web site: hemraj.co.uk.
Email: director@hemraj.co.uk.

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'20, 11-13 February 2020.

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

saswg-comments@scsc.uk

Safety Assurance Objectives for Autonomous Systems

The Safety of Autonomous Systems Working Group [SASWG]

January 2020

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

Changes Since the Last Edition

The most significant change since the last edition has been the population of the autonomy architecture-level and platform-level frameworks. As part of this change, the adaptation projection was moved from the computation-level to the autonomy architecture-level, where it is better placed.

Objectives are now written in a form that focuses on the intended outcome, rather than being expressed as a requirement. For example, the form "Data is acquired and controlled appropriately" is used in preference to "Data should be acquired and controlled appropriately". This change means the wording of the computation-level objectives has changed from the previous version. Despite this, the intent of the objectives remains the same.

The increased scope of the current version, combined with greater availability of other work (external to the SASWG), has allowed a larger number of top-level comparison activities to be conducted. These are recorded in a number of appendixes. Individually and collectively these provide increased confidence in the objectives listed in this document.

Contents

This page is intentionally blank

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).
- 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.
- 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 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 intentionally very little mention of legal and / or regulatory requirements. It is assumed that these will be identified and demonstrably complied with as part of normal practice.
- Issues related to staff competencies are deliberately excluded from consideration. Similarly, issues that are most appropriately addressed at an organisational, or enterprise, level are also excluded. These are expected to be covered by an existing Safety Management System (SMS), which could be supplemented by the considerations in this document.
- Issues related to domain-specific certification (e.g., liaison with regulators) are deliberately excluded from consideration. However, the objectives listed in this document would be expected to inform any discussions with regulatory authorities.
- This document makes no distinction based on criticality level, that is, there is no equivalent of Safety Integrity Levels (SILs) or Development Assurance Levels (DALs). These types of distinction may be included in future versions of this document.

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 do and what effects it should 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.

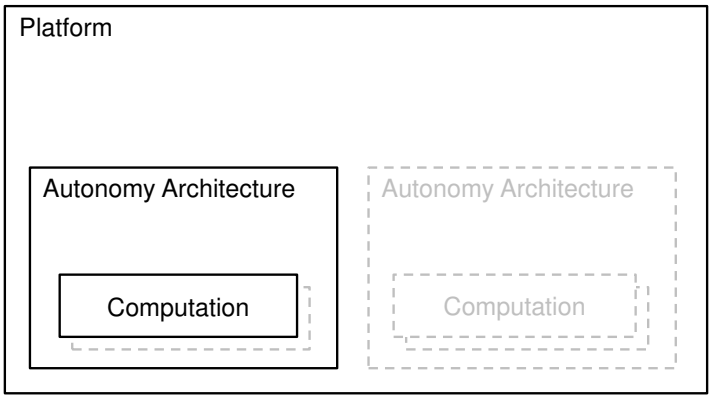

Figure 1: Relationship between Framework Levels

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 was the case 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 need not be considered for their particular application. Similarly, users may also argue why a particular objective needs only to be covered at a superficial level.

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 current version, released in January 2020, represents the first time that all three framework levels have been addressed.

The identified objectives (and associated frameworks) have been developed mainly from a theoretical basis. However, 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 those activities, it should be noted that, as yet, the objectives have not been subjected to practical use across an entire AS. In short, the objectives have not been "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] https://www.oecd.org/going-digital/ai/principles/.

1.4 Terminology

The 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 document. Specifically:

- 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. 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 and to computational hardware; neither of these is included within "algorithm".
- 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.
- 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.
- The data set is 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*. During operational use the algorithm is provided with *inputs*.
- 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).

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 contains a summary list of objectives.

- Appendix A provides justification for the computation-level framework.
- Appendix B provides additional justification (beyond that which is included in Section 3) for the computation-level objectives.
- Appendix C provides justification for the platform-level framework.
- Appendix D provides a top-level mapping between the argument structure in the Assuring Autonomy International Programme (AAIP) Body Of Knowledge (BOK) and the objectives listed in this document. This provides additional justification for all three frameworks adopted by the SASWG.
- Appendix E provides a top-level mapping between the key section headings in UL4600, "The Standard for Safety for the Evaluation of Autonomous Products", and the contents of this document (either objectives or projections). As with the previous appendix, this provides additional justification for this document's contents.
- Appendix F provides an illustration of how this document's objectives support the OECD Principles on AI.
- Appendix G provides a list of known issues, which will be resolved in future versions.
- Appendix H provides a list of abbreviations.
- Appendix I provides a list of references.
- Appendix J provides a list of contributors.

This page is intentionally blank

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 [30]. The justification for adopting this framework is provided in Appendix A.

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; 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 used to train and develop the algorithm. When relevant, this also includes training that continues during operational use, based on the data set 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 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.

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, 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 that is available to train (or develop) the algorithm
Task	Focused on the performance of the implemented algorithm; emphasizes requirements

Projection	Outline
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 Appendix B.

3.1 Experience

The experience projection is focused on the training data 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 four objectives associated with this projection.

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

Discussion: Data is obviously a very important 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 [16].

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 [17]. 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 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 [22].

The second part of this objective relates to control of the data. More specifically, the data 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 may mean they are not optimal.

Note that algorithms featuring online learning will continue to receive training data during operational use. This indicates there may be a need to include safeguards 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] [61], but here it involves comparing a single sample with a distribution, rather than the more typical case of comparing two distributions.

COM1-2: **Pre-processing methods do not introduce errors.**

Discussion: Just because data has been collected in a controlled manner (as indicated by Objective COM1-1), it does not necessarily follow that the data is suitable 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: Typically, 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., [69]). 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) [70] 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).

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 *specific* 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.

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

Examples: Exploratory data analysis [81] 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) [7]. The concept of outlier samples can also be extended to include rare situations, the presence (or absence) of which is likely to be informative. 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 [34]. 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. This approach 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].

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 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 [61]. 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 [66].

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).

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. 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 [18]. 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 [28]. The notion of situation coverage could also inform this decision [3].

This will depend, however, on the technologies in use. AS often use novel technology, and it may be that there are no established techniques for measuring a given non-functional property.

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.

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, which is withheld from the training process for this purpose. Note, however, that this involves a statistical 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., [78]) 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].

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 [11].

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 [24]. 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 (in the experience projection, subsection 3.1), 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 system 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) [19] are well established measures of test coverage for traditional software [69]. These measures, and other related ones, allow judgements to be made regarding the sufficiency of a test set (e.g., 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 [18].

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 are likely to be helpful [77], especially those based on activations of combinations of (rather than individual) neurons.

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.

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 [73], potentially supported by a standard set of scenarios (as discussed in Objective COM2-2).

COM2-7: Each algorithm variant is tested appropriately.

Discussion: For the purposes of this document, it is helpful to distinguish between instantiations, which *may* yield different behaviour, and variants, which are *intended* to yield different behaviour. For example, different algorithm instantiations would be expected in autonomous vehicles operating in the United States of America, whilst different variants would be needed to obey state-level driving laws. More generally, algorithm variants can facilitate adherence to local legislation, or local practices.

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 [64].

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 Reinforcement Learning (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 Convolutional Neural Networks (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 [76].

Examples: Any computation-level assurance argument would be expected to include justification for the chosen algorithm 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 [52].

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 would be expected.

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 framework; and within the execution environment [88]. 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 [85].

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 [60].

Although supporting frameworks 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) [88].

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 [8]. 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, it is easy to see *what* an algorithm is doing; it may be less easy to see *why*.

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 [67]; and providing visual representations [43].
- 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.

Although the precise details are outside the scope of this document, it should be noted there may be legal

(or ethical) factors that affect the extent to which an algorithm's behaviour has to be explained [83].

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) 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., [69]). 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 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 framework that is actually used. Alternatively, it may be possible

to compare the results from different (independently developed) libraries [75]. 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 framework is suitable.

Secondly, the pervasive nature of the framework means it is inappropriate to treat it as Software of Uncertain Pedigree (SOUP) [49]. In particular, it is not possible to put the framework 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 [65].

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 [74].

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 [78].

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 is likely to 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., [68]). 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 [29].

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 [63].

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. 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.

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 [6]. Unlike the computation-level framework (Section 2) and the platform-level framework (Section 6) there are no obvious comparator items that can be used to support the choice of this framework. However, some confidence can be gained by the overall comparison of objectives with contents of the AAIP BOK [41] (in Appendix D) and the main section headings in UL4600 [82] (in Appendix E).

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, as well as changes to computational hardware.

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 [6]) 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; declaring the computation output as incorrect (using one of the approaches discussed under Objective ARC1-5).

ARC1-2: **Operational inputs inconsistent with the training, test and verification data 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 [55]. 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 [7]; a potential mechanism for detecting such regions is provided in [53]. 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 determine 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 [5]. The GAN could then be used to determine if an operational input was consistent with the TTV data.

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.

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 [74]. 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 [56].

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) [78]. Whilst this is an interesting, and important, phenomenon this objective has a much wider scope. For example, it includes: adversarial poisoning of training data [17]; adversarial influence of pre-trained networks [37]; and adversarial impact via software frameworks and tools.

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 [23], 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., [51]). 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 [86]. 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., [56], [7]).

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.

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 frameworks and tools 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, especially in the case of frameworks, because their role prevents them from being placed in a tightly bounded sandbox and because their size prevents widespread reverse engineering of evidence artefacts. Possible options include using a framework 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" [21]. 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 [6]. 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 ML model 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 ML model may be challenging.
- The second architecture is inspired by multiplex avionics systems. It uses a number of distinct computations (or ML models) 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 [9]. 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 (implemented as an ML model), 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 algorithm, 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.

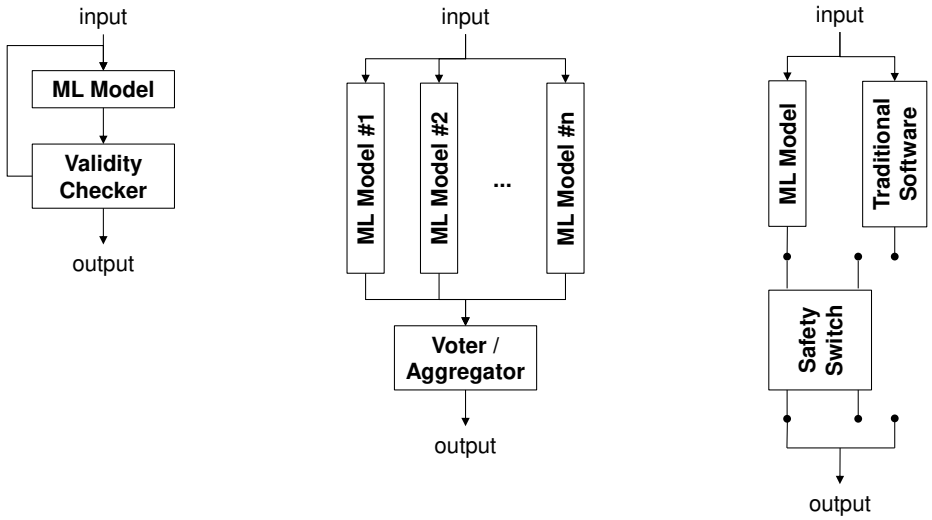

Figure 2: Example Architecture Options

5.2 Information Provision

The autonomy architecture is concerned with integrating computations into a platform, or system. Part of these considerations include 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.

There are four objectives associated with this projection.

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

Discussion: In order to be safe an autonomous system 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 [79] 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 [67].

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 (subsection 5.3). 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 earlier in 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 [15].

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, inevitablly, application specific. In the case of road vehicles, an indication of the duration for which information needs to be recorded is provided in [15].

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 [15].

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 [62].

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 [80]. 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 algorithms. This would require traditional cyber security approaches to protect these algorithms.

There are also attacks that can expose aspects of the training data, for example [32]. Again, protecting against these may involve using the autonomy architecture to protect against adversaries obtaining direct access to unfiltered model inputs and outputs.

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

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 not 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 case and need to be managed 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, context 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; 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 authorisation. 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 [33]. Alternatively, or additionally, it may be possible to provide a set of abstract policies that formally constrain the exploration of an RL agent [57] or including a representation of fear within the learning mechanism [54].

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 [69]; 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 [28]. 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.

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 [23] (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 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 platform ought to be capable of being put in a safe state that can be maintained for a considerable period.

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 Appendix C.

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; 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 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 may 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.

6.2 Interacting Items

This projection focuses on items that are intended or required to interact with the platform, but are not directly owned by the platform developer or operator.

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.

6.3 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.

6.4 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.

6.5 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 owned by the platform developer or operator
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 a safety 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.

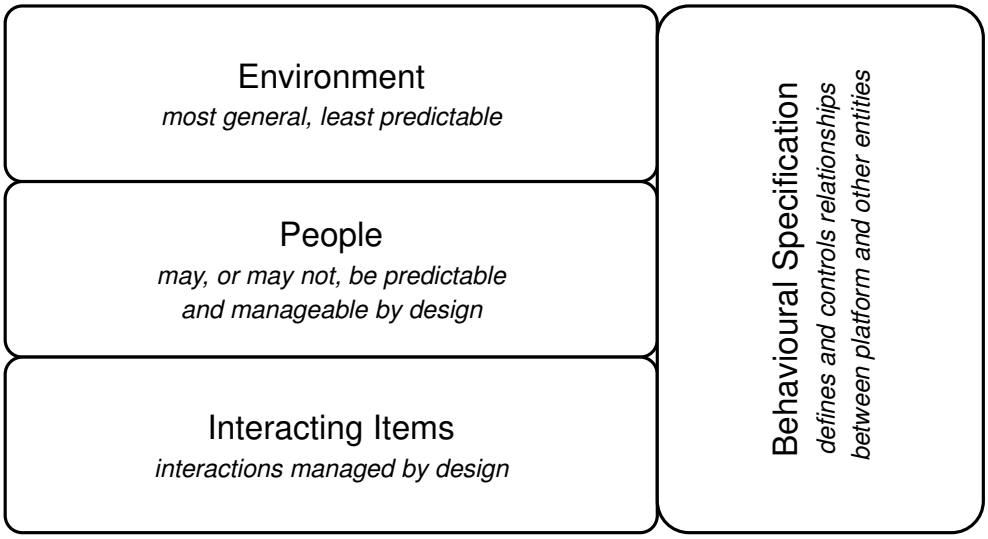

Figure 3: Platform-Level Framework Projections Inter-Relationships

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 [72]. 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 safety 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 operation 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) [27], 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).

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 [25].

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 [46].

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: **The specified behaviour of the platform is predictable, consistent and safe.**

Discussion: Specification of platform behaviour is needed for several reasons. For example: it is needed to provide the expected outcome for system-level tests; in addition, it facilitates operation alongside other entities within the intended operational domain. Predictability and consistency of platform behaviour are important for similar reasons. The need for safe behaviour is obvious, noting that the definition of safety is expected to be produced as part of the Objective PLT1-2.

Examples: Some aspects of behavioural specification may be addressed by traditional systems engineering techniques. However, one advantage of autonomy-enabling technologies is they do not need a detailed, low-level behavioural specification that has been decomposed from platform-level requirements in a traceable, hierarchical manner. For such technologies, it is helpful to consider both intended platform-level behaviour (i.e., "the platform should …") and unintended, or undesirable, behaviour (i.e., "the platform should not …"). These considerations may be usefully informed by iteratively specifying behaviour and observing the results in specific scenarios (both hand-crafted and automatically-generated).

The qualities of predictability and consistency do not necessarily require precisely identical, repeatable behaviour in apparently similar situations. Demanding that level of determinism is deemed inappropriate for at least two reasons. Firstly, minute changes in the autonomous system's perception of the situation could cause a flip between two different but equally acceptable behaviours: more particularly, for an autonomous system used in a domain of even moderate complexity, the chance that identical situations will be presented is negligible, and this chance reduces as the domain gets more complex. Secondly, there may be good reasons why a form of pseudo-randomness is included within the autonomy-enabling algorithm. Whilst precise repeatability is not required, meaningful bounds need to be placed around expected platform behaviour. Scenario-based sensitivity analysis may help inform this consideration.

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

Discussion: Objective PLT1-3 is focused on a fully-functioning platform that is being used as intended. Unfortunately, neither of these criteria can be assumed for platforms subjected to real-world use by humans. Consequently, this objective is concerned with platform behaviour in sub-optimal conditions. 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, where algorithms prevented access to engine thrust reversers even though the aircraft was on the ground, leading to runway excursion and loss of life [31]. 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 [20]. 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. This notion of coverage 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 a need to include 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 [61].

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, [21]). 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 [71].

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 owned by the platform developer or operator. 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 [79].

Several approaches could be used to gain the understanding necessary for this objective. For example, part of STPA [46] 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 [59]. 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.

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 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 [9]. 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 [28]. 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 qualified 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 provided.

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; furthermore, 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-specific. As for traditional systems, there may be a need for formal certification and, potentially, re-certification.

The nature of ML-based models 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 time-based 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 [23]. Likewise, the combined development of a security case and a safety case may help [48].

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 [50]. 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 and understood 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 [79].

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 [46]) 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 Summary

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

8.1 Computation-Level

Table 5: Computation-Level Objectives

Id	Objective	Projection
COM1-1	Data is acquired and controlled appropriately.	Experience
COM1-2	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.	
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.	

8.2 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 the training, test and verification data 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.	

8.3 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 operation for the platform is defined.	
PLT1-3	The specified behaviour of the platform is predictable, consistent and safe.	
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 Computation-Level Framework: Justification

This appendix summarises the process used to develop the computation-level framework adopted by the SASWG. In doing so, it provides some justification for the choice of framework. It also provides some confidence that the framework covers all relevant topic areas.

Initially, a small-scale survey of existing computation-level frameworks was conducted. This identified the items listed in Table 8.

Table 8: Computation-Level Frameworks Considered

Section	Framework
A.1.1	Modified Software Safety Assurance Principles
A.1.2	The "Faria Stack"
A.1.3	Douthwaite and Kelly's "Viewpoints"
A.1.4	Google's Machine Learning Rubric
A.1.5	Ethical and Safety Principles
A.1.6	Burton's "Making the Case" Argument

Each computation-level framework is briefly summarised (subsection A.1) and a preferred framework is selected. A top-level mapping between frameworks is completed, to confirm that the chosen framework incorporates all relevant parts of the other computation-level frameworks (subsection A.2). Similar, top-level mappings from the chosen framework to, firstly, a typical software development approach and, secondly, a generic approach to ML-based development are conducted; these demonstrate the framework provides appropriate coverage of typical development activities (subsection A.3).

A.1 Computation-Level Frameworks

A.1.1 Modified Software Safety Assurance Principles

This computation-level framework is described in a paper presented at the 2017 SSS [8]. The paper considers the "four plus one" software safety assurance principles [42] from the perspective of non-traditional (e.g., ML / AI) software. A slightly revised and extended set of six (or "four plus two") principles are proposed:

- Principle One: Software safety requirements shall be defined to address the software contribution to system hazards;
- Principle Two-Primed: The software detailed design shall embody the intent of the software safety requirements;
- Principle Three: Software safety requirements shall be satisfied;
- Principle Four: Hazardous behaviour of the software shall be identified and mitigated;
- Principle Four plus One: The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk;

- Principle Four plus Two: Software required to produce behaviour not predictable at design time should consider the consequence of behavioural adaptations on its environment.

A.1.2 The "Faria Stack"

This computation-level framework is based on the information presented in a paper at the International Symposium on Software Reliability Engineering (ISSRE) Workshop on Software Certification (WoSoCer) [30]. This framework comprises five projections:

- Experience, which is focused on the data that is available to train a machine learning algorithm;
- Task, which is concerned with the performance of the implemented computation;
- Algorithm, which considers the type of algorithm (e.g., neural network, random forest, etc.);
- Software, which includes considerations such as the language in which the computation is implemented;
- Hardware, which relates to the computational hardware that is used.

When using this framework it may be helpful to consider, at least, the Software and Hardware projections from two perspectives, specifically training and operational use. For example, it is likely that the computational hardware used for training will be different to that used during an operational deployment.

A.1.3 Douthwaite and Kelly's "Viewpoints"

This computation-level framework was presented at the 2018 SSS [26]. Building on the concept of distinct viewpoints used in systems engineering, this paper identifies six viewpoints. Although they were developed from the perspective of Bayesian Networks, the paper suggests the viewpoints are applicable to many types of artificial intelligence software. The viewpoints are:

- Model, which relates to the structure and parametrisation of the model underlying the learnt algorithm;
- Data, which covers all data acquisition, processing and storage concerns (including knowledge engineering and expert elicitation);
- Computational, which includes the properties of all algorithms used for learning and reasoning tasks within the system, their selection process, and the associated assumptions and design decisions;
- Operational, which focuses on the evolution and maintenance of the system after deployment;
- Technology, which covers the necessity, properties, constraints and assumptions of modelling frameworks used in the system;
- Implementation, which addresses all "conventional" software and hardware engineering concerns, including "normal" function allocation, requirements and associated verification and validation activities.

As with the "Faria Stack" considered above, there may be advantages in considering some of the above viewpoints from both training and operational perspectives.

A.1.4 Google's Machine Learning Rubric

This computation-level framework [13] includes a scoring mechanism that is intended to measure how suitable a machine learning approach is for deployment. It is based on computations used in a web-like environment, but may be of relevance to wider autonomous systems.

The framework includes four categories, each of which includes several considerations:

- Tests for Features and Data:
  - Test that the distributions of each feature match your expectations;
  - Test the relationship between each feature and the target, and the pairwise correlations between individual signals;
  - Test the cost of each feature;
  - Test that a model does not contain any features that have been manually determined as unsuitable for use;
  - Test that your system maintains privacy controls across its entire data pipeline;
  - Test the calendar time needed to develop and add a new feature to the production model;
  - Test all code that creates input features, both in training and serving.
- Tests for Model Development:
  - Test that every model specification undergoes a code review and is checked in to a repository;
  - Test the relationship between offline proxy metrics and the actual impact metrics;
  - Test the impact of each tunable hyper-parameter;
  - Test the effect of model staleness;
  - Test against a simpler model as a baseline;
  - Test model quality on important data slices;
  - Test the model for implicit bias.
- Tests for ML Infrastructure:
  - Test the reproducibility of training;
  - Unit test model specification code;
  - Integration test the full ML pipeline;
  - Test model quality before attempting to serve it;
  - Test that a single example or training batch can be sent to the model, and changes to internal state can be observed from training through to prediction;
  - Test models via a canary process before they enter production serving environments;
  - Test how quickly and safely a model can be rolled back to a previous serving version.
- Monitoring Tests for ML:
  - Test for upstream instability in features, both in training and serving;
  - Test that data invariants hold in training and serving inputs;
  - Test that your training and serving features compute the same values;
  - Test for model staleness;

- Test for Not a Number (NaN) or infinities appearing in your model during training or serving;
- Test for dramatic or slow-leak regressions in training speed, serving latency, throughput, or Random Access Memory (RAM) usage;
- Test for regressions in prediction quality on served data.

For each item above, one point is awarded for manual tests (including documenting and distributing the results). A second point is awarded if tests are run automatically and repeatedly. A score is calculated for each of the four categories by adding the scores for each of the listed items. The overall score is then the minimum of these four category scores.

A.1.5 Ethical and Safety Principles

This computation-level framework identifies a perspective on the ethics governing decisions around safety-critical autonomous systems [58]. It aligns with the Modified Software Safety Assurance Principles (discussed above) and is applicable to ethics only so far as these affect safety.

- Principle One: Ethics requirements governing the autonomous system behaviour shall be defined.
- Principle Two: The intent of the ethics requirements shall be maintained throughout decomposition.
- Principle Three: Ethics requirements shall be satisfied.
- Principle Four: The autonomous system shall continue to be safe, and emergent behaviour of the autonomous system which conflicts with the ethics requirements shall be identified and mitigated
- Principle Four plus One: The degree of rigour required to address these ethical principles shall be commensurate with the contribution of the autonomous system to system risk.

A.1.6 Burton's "Making the Case" Argument

This computation-level framework comes from a paper presented at the 2017 International Conference on Computer Safety, Reliability, and Security [14]. The paper outlines an assurance case structure for a highly automated driving system, which could possibly be extended to cover a wide range of autonomous systems. A Goal Structuring Notation (GSN) approach is used; key features include:

- GOAL G1: The residual risk associated with functional insufficiences in the object detection function is acceptable;
- CONTEXT C1: Definition of functional and performance requirements on the object detection function;
- ASSUMPTION A1: Assumptions on the operational profile of the system's environment;
- ASSUMPTION A2: Assumptions on attributes of inputs to the machine learning function;
- ASSUMPTION A3: Assumptions on the performance potential of machine learning;
- STRATEGY S1: Argument over causes of functional insufficiencies in machine learning;
- SUBGOAL G2: The operating context is well defined and reflected in training data;

- SUBGOAL G3: The function is robust against distributional shift in the environment;
- SUBGOAL G4: The function exhibits a uniform behaviour over critical classes of situations;
- SUBGOAL G5: The function is robust against differences between its training and execution platforms;
- SUBGOAL G6: The function is robust against changes in its system context.

A.2 Framework Mappings

Following discussions[4], the SASWG selected the "Faria Stack" as the basis for the computation-level framework. The following paragraphs briefly discuss each projection of the "Faria Stack", taking into account the other frameworks outlined in the preceding subsection. Within these discussions:

- For reasons of brevity, only the top-level of Google's Machine Learning Rubric is considered.
- Due to their similarity to the Modified Software Safety Assurance Principles, the Ethical and Safety principles are not explicitly considered.
- For simplicity, only the goals and subgoals are considered from Burton's "Making the Case" Argument.

The discussions also include a "Not Addressed" pseudo-projection, which captures considerations that do not readily relate to any of the projections. By checking the contents of this pseudo-projection, and confirming that it contains nothing significant, confidence can be gained that the adopted framework covers all relevant topics.

A.2.1 Experience

Consideration of the data used to develop the algorithm directly relates to Douthwaite and Kelly's Data viewpoint, and also to the Tests for Features and Data category from Google's Machine Learning Rubric.

The way the data reflects the operating context directly relates to Subgoal G2 from Burton's "Making the Case" Argument.

A.2.2 Task

Understanding the task should also include understanding the way it contributes to the wider system and, also, any associated computation (or software) safety requirements. This consideration relates to Principle One of the Modified Software Safety Assurance Principles.

Performance measurement against the intended task ought to include explicit measures against requirements (including safety requirements). It also ought to consider whether the computation has introduced any new hazards. These considerations relate to Principles Three and Four of the Modified Software Safety Assurance Principles. They also relate to Goal G1 from Burton's "Making the Case" Argument.

More generally, performance management relates to the Tests for Model Development category from Google's Machine Learning Rubric.

[4] SASWG 7, 17 April 2018, York.

The properties of the operationally-fielded computation relate to Douthwaite and Kelly's Computational viewpoint.

A.2.3 Algorithm

The link between choice of algorithm and intended task mirrors the link between requirements (including safety requirements) and detailed design. This relates to Principle Two-Primed of the Modified Software Safety Assurance Principles.

Part of choosing a specific algorithm also includes choosing hyper-parameters (e.g., number of nodes and layers in a neural network). This relates to Douthwaite and Kelly's Model viewpoint. More general algorithm-related choices relate to Douthwaite and Kelly's Computational viewpoint.

A.2.4 Software

The choice of software (for both development and operational use) is part of detailed design. This relates to Principle Two-Primed of the Modified Software Safety Assurance Principles. It also relates to Douthwaite and Kelly's Technology and Implementation viewpoints, and also to the Tests for ML Infrastructure category from Google's Machine Learning Rubric.

A.2.5 Hardware

The choice of hardware (for both development and operational use) is part of detailed design. This relates to Principle Two-Primed of the Modified Software Safety Assurance Principles, to Douthwaite and Kelly's Implementation viewpoint, and also to the Tests for ML Infrastructure category from Google's Machine Learning Rubric.

The possibility of different behaviour on development (training) and operational (execution) platforms relates to Subgoal G5 from Burton's "Making the Case" Argument.

A.2.6 Not Addressed

The chosen computation-level framework does not readily address Principle Four plus One of the Modified Software Safety Assurance Principles: "The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk". This is not a significant concern as this principle is a cross-cutting issue for all assurance, and thus not something that has to be specifically addressed at the computation level.

Likewise, the framework does not readily address Principle Four plus Two: "Software required to produce behaviour not predictable at design time should consider the consequence of behavioural adaptations on its environment.". This is not a significant concern as adaptation is considered at the autonomy architecture-level (Section 4).

From the perspective of Douthwaite and Kelly's "Viewpoints" the chosen computation-level framework does not readily address the Operational viewpoint. This is more readily addressed at the autonomy architecture-level and the platform-level (Section 4 and Section 6, respectively).

Similarly, the Monitoring Tests for ML category from Google's Machine Learning Rubric are addressed at other framework levels, as is Subgoal G3 from Burton's "Making the Case" Argument.

Finally, the chosen computation-level framework does not readily address Subgoal G4 of Burton's "Making the Case" Argument: "The function exhibits a uniform behaviour over critical classes of situations". It is not immediately clear whether this, especially the "uniform behaviour" part, is a generic requirement that should be satisfied by *every* computation. If it is a requirement for a particular application then it should be addressed by the Task projection (via the relationship to Principle One of the Modified Software Safety Assurance Principles).

A.2.7 Relationship Summary

For ease of reference, the relationships outlined above are summarised in Table 9. Note that this presentation is deliberately simple and top-level.

Table 9: Relationships between Computation-Level Frameworks Considered

Stack Level	Modified Software Safety Assurance Principles	Douthwaite and Kelly's "Viewpoints"	Google's Machine Learning Rubric	Burton's "Making the Case" Argument
Experience	-	Data	Tests for Features and Data	Subgoal G2
Task	Principles One, Three and Four	Computational	Tests for Model Development	Goal G1
Algorithm	Principle Two-Primed	Model and Computational	-	-
Software	Principle Two-Primed	Technology and Implementation	Tests for ML Infrastructure	-
Hardware	Principle Two-Primed	Implementation	Tests for ML Infrastructure	Subgoal G5
Not Addressed	*Principle Four plus One, Principle Four plus Two*	*Operational*	*Monitoring Tests for ML*	*Subgoal G4*

Overall, the preceding analysis indicates that, based on the selected comparator frameworks, there are no significant omissions from the chosen computation-level framework.

A.3 Software and ML Development Mappings

Table 10 maps the framework's projections to the activities involved in a generic software development [87].

This mapping shows that the chosen computation-level framework is sufficiently complete to address typical software development activities.

Table 10: Mapping Projections to Typical Software Development

	Experience	Task	Algorithm	Software	Hardware
Plan		Y			
Requirements		Y			
Design	Y	Y	Y		
Implement	Y	Y	Y	Y	Y
Test		Y			
Transition				Y	Y

To provide further confidence, Table 11 maps the projects to the steps that are required to produce a useful ML-based computation [84]. This mapping demonstrates the framework fits well with development in an ML context, with most development steps mapping to a single projection.

Table 11: Mapping Projections to Typical ML Development

	Experience	Task	Algorithm	Software	Hardware
Frame the question		Y			
Collect data	Y				
Select features	Y				
Choose algorithm			Y		
Choose metrics		Y			
Conduct experiment				Y	Y
Interpret results		Y			

Appendix B Computation-Level Objectives: Justification

This appendix provides some additional justification for the computation-level objectives listed in the main body. This is achieved by mapping those objectives to separately published material, specifically:

- A suggested list of requirements for a standard to support the use of NNs in safety-critical applications [10]. This source dates from 1996. Consequently, it provides a sound theoretical basis, independent from recent trends, against which computation-level objectives can be compared. However, its considerations do not encompass the latest research directions. In addition, whilst many of its requirements are applicable to a number of ML approaches, they have been derived in the specific context of NNs.
- An analysis of gaps in a current automotive standard with regards to the use of ML approaches [72]. This source dates from 2018, so it encapsulates recent research. However, the chosen standard, specifically International Organization for Standardization (ISO) 26262 [47] is a functional safety standard; that is, it only addresses unsafe behaviours caused by system malfunctions. For ML approaches, there is also a need to consider the Safety Of The Intended Function (SOTIF).

For the reasons outlined above, the computation-level objectives derived by the SASWG would not be expected to directly match the contents of either reference. Nevertheless, the objectives would be expected to cover all relevant issues raised in the reference material.

It is emphasised that the mappings established below are top-level and approximate. This is considered appropriate as the mappings are intended to justify (or, if necessary, refine) the computation-level objectives. More specifically, the mappings discussed in this appendix were not a key part of the process by which the computation-level objectives were derived.

B.1 Requirements for a NN Standard

Table 12 lists the requirements noted in [10]. Note that these requirements use the term Artificial Neural Network (ANN), rather than NN, which is preferred in the current document. Where appropriate, relevant computation-level objectives are highlighted. If no objectives are relevant, justification for this is provided.

Table 12: Computation-Level Objectives Compared against Requirements for a NN Standard

Standard Requirement	Relevant Objectives
Specify how the high-level goals of, or requirements for, the ANN module are to be obtained	COM2-1, COM2-2
Specify what should be done to ensure that the training data adequately represent the attainment of the high-level goals	COM1-3
Specify what type of networks can be used, and how each type is to be unambiguously designated	COM3-1
Specify how the input-output characteristics are to be unambiguously designated	COM1-1, COM1-2
Specify how the developer must describe the way in which the performance function for the network operates during training	COM2-3

Standard Requirement	Relevant Objectives
Specify what details the ANN developer must provide regarding the way in which the ANN module interfaces with the rest of the system	*Out of scope: Autonomy architecture-level*
Specify the extent of knowledge, relating to neural networks, required of management and development team personnel	*Out of scope: Staffing*
Specify what development model is to be used for the ANN module	COM5-1
Specify any outputs which the ANN module is required to produce in addition to its primary functional output	COM2-1
Specify whether formal methods or rigorous argument are to be used to develop the software which implements the neural network	COM4-1, COM4-2
Specify what methods are to be used for quality assurance in the trained network	COM1-1, COM4-1, COM5-1
Specify that the Verification and Validation (V&V) team should use generalisation tests on the trained network to verify that it has learned the principles implicit in the training data	COM2-3, COM2-6, COM2-7
Specify that the V&V team should validate a Safety-Critical Artificial Neural Network (SCANN) by investigating the behaviour of the SCANN over the whole of the input space	COM2-5
Specify how the developers should check that the initial safety assessments made for the system are not affected by the ANN module and how failures in other modules would affect the system, given the intended operation of the ANN	*Out of scope: Platform-level*
Specify that developers establish possible failure modes of the ANN module itself and the consequences	*Out of scope: Autonomy architecture-level (supported by COM1-4, COM2-4, COM3-2, COM5-2, COM5-2)*
Specify how HAZOP is to proceed, regarding the operation of network	*Out of scope: Platform-level*
Specify the brief and form of the HAZOP committee, as well as guide words for their use	*Out of scope: Platform-level*
Specify that a certification standard should insist that the developers build the network is such a way that the necessary data are available so that it is possible to do Failure Mode and Effects Analysis (FMEA) and HAZOP	COM3-3, COM3-4

It is apparent that all relevant requirements established by [10] are covered by one or more of the computational objectives derived by the SASWG. This provides further confidence in the identified computation-level objectives.

B.2 ML-Related Gaps in an Automotive Standard

The analysis of ISO 26262 identified a number of impacted or new Process Requirements (PRs). The associated phase and description are reproduced (from [72]) in Table 13. Relevant computation-level objectives are then highlighted; if there are no such objectives then justification is provided.

Table 13: Computation-Level Objectives Compared against Impacted or New PRs

Phase	Description	Relevant Objectives
(5) Initiation	Best practices: coding guidelines	COM3-2, COM4-1
(5) Initiation	ML decision gate	*Out of scope: Autonomy architecture-level*
(6) Software safety requirements	Requirements specification	COM1-3, COM2-1, COM2-2
(6) Software safety requirements	Requirements verification	COM2-3, COM2-5
(7) Architectural design	Fault tolerance	*Out of scope: Autonomy architecture-level*
(8) Software unit design, implementation	Best practices: notations	COM3-2, COM4-1
(8) Software unit design, implementation	Best practices: design principles	COM3-2
(8) Software unit design, implementation	Best practices: data set collection and verification	COM1-1, COM1-2
(8) Software unit design, implementation	Best practices: model selection	COM3-1
(8) Software unit design, implementation	Best practices: feature selection	COM1-3
(8) Software unit design, implementation	Best practices: training	COM3-2, COM4-1, COM5-1
(8) Software unit design, implementation	Best practices: data set splitting	*Out of scope: Approach-specific*
(8) Software unit design, implementation	Best practices: validation	COM2-3, COM3-2
(8) Software unit design, implementation	Best practices: testing	COM2-3, COM2-7, COM4-2, COM5-2
(8) Software unit design, implementation	Best practices: testing structural coverage	COM2-5
(8) Software unit design, implementation	Best practices: test vs. operating environment	COM1-4, COM2-6, COM4-1
(8) Software unit design, implementation	Best practices: test result explanation	COM3-3, COM3-4
(8) Software unit design, implementation	Best practices: verification	COM2-3, COM2-4, COM2-5, COM2-6, COM3-4

It is apparent that all impacted or new PRs established by [72] are covered by one or more of the computational objectives derived by the SASWG, or are intentionally outside the scope of this framework. This, again, provides further confidence in the computation-level objectives.

This page is intentionally blank

Appendix C Platform-Level Framework: Justification

This appendix provides a brief, outline survey of platform-level frameworks that could be used to structure thinking about the safety (or assurance) of autonomous systems. For ease of reference, Table 14 lists the frameworks that are considered.

Table 14: Platform-Level Frameworks Considered

Section	Framework
C.1.1	Waymo's System Safety Report
C.1.2	The Twelve Safety Elements from the NHTSA
C.1.3	HORIBA MIRA Framework
C.1.4	Uber Advanced Technologies Group
C.1.5	AAIP BOK
C.1.6	AI Safety Landscape Categories

Each platform-level framework is briefly summarised (subsection C.1) and a preferred framework is developed. A top-level mapping between frameworks is completed, to confirm that the chosen framework incorporates all relevant parts of the other platform-level frameworks (subsection C.2).

C.1 Platform-Level Frameworks

C.1.1 Waymo's System Safety Report

This platform-level framework comes from Waymo's System Safety Report[5]. This report establishes five distinct safety areas:

- Behavioural safety, which is about the behaviour of the vehicle on the road, including the decisions it makes. This is the most novel of the safety areas.
- Functional safety, which considers how the system operates in the presence of faults and failures. This appears to be standard system safety, including the use of redundant sub-systems, for example.
- Crash safety, which is about protecting people in the event of a crash. This appears to be normal automotive crash safety.
- Operational safety, which covers the interaction between Waymo vehicles and their passengers. This seems to be mainly focused on the user interface, which includes helping the passenger understand what the vehicle is perceiving and what it is doing on the road.
- Non-collision safety, which considers how the vehicle could harm those it interacts with in non-crash situations (including passengers, first responders and bystanders).

[5] https://storage.googleapis.com/sdc-prod/v1/safety-report/waymo-safety-report-2017-10.pdf.

C.1.2 The Twelve Safety Elements from the NHTSA

This platform-level framework comes from "Automated Driving Systems 2.0: A Vision for Safety", published by the US National Highway Traffic Safety Administration (NHTSA)[6]. This introduces twelve "safety elements":

1. System Safety;
2. Operational Design Domain;
3. Object and Event Detection and Response;
4. Fallback (Minimum Risk Condition);
5. Validation Methods;
6. Human Machine Interface;
7. Vehicle Cybersecurity;
8. Crashworthiness;
9. Post-Crash Automated Driving System (ADS) Behaviour;
10. Data Recording;
11. Consumer Education and Training;
12. Federal, State and Local Laws.

C.1.3 HORIBA MIRA Framework

This platform-level framework is described in an HORIBA MIRA presentation [12]. The presentation describes an Autonomous Driver (AD); the following bullets summarise the high-level, generic features of the framework.

- STRATEGY: Argument split according to functionality that is intended, unintended and due to malicious intent.
- CLAIM: Intended Behaviour - The absence of unreasonable risk associated with the intended behaviour of the [autonomous system] has been achieved.
  - STRATEGY: Argument structured by the rationale for, and satisfaction of, specified requirements (REQs).
  - CLAIM: Requirements Rationale - Meeting the REQs yields the absence of unreasonable risk associated with the intended behaviour of the [autonomous system].
  - CLAIM: Requirements Satisfaction - The [autonomous system] behaves according to the REQs. (In this area, the framework also introduces: virtual testing; physical testing; and testing diversity and number.)
- CLAIM: Malfunctioning Behaviour - The absence of unreasonable risk associated with malfunctioning behaviour of the [autonomous system] has been achieved.
- CLAIM: Malicious Intent - The absence of unreasonable risk associated with malicious attack of the [autonomous system] has been achieved.

[6] https://www.nhtsa.gov/manufacturers/automated-vehicles-manufacturers.

C.1.4 Uber Advanced Technologies Group

This platform-level framework is the safety case framework developed by Uber Advanced Technologies Group[7]. This framework is intended for use with self-driving vehicles, especially passenger-carrying cars on public roads. This is a narrower scope than the current document, which addresses all autonomous systems. For reasons of brevity only the five top-level goals associated with this framework are listed below:

- G1 - Proficient: The Self-Driving Vehicle is acceptably safe during nominal operation.
- G2 - Fail-Safe: The Self-Driving Vehicle is acceptably safe in presence of faults and failures.
- G3 - Continuously Improving: Any anomaly that could affect the safety of the Self-Driving Vehicle is identified, evaluated, and resolved with appropriate corrective and preventative actions.
- G4 - Resilient: The Self-Driving Vehicle is acceptably safe in case of reasonably foreseeable misuse and unavoidable events.
- G5 - Trustworthy: The Self-Driving Enterprise is trustworthy.

C.1.5 AAIP BOK

This platform-level framework is part of the structure of the AAIP BOK [41]. This has a vast scope: it aims to be cross-domain, cross-technology and cross-application, covering all aspects of assurance and regulation of Robotics and Autonomous Systems (RAS). The current document has similar aims with regards to breadth of domains, technologies and applications: however, regulation-specifics are not a focus. For reasons of brevity only the top-level items are listed below:

- Defining required behaviour.
- Implementation of an RAS to provide the required behaviour.
- Understanding and controlling deviations from required behaviour.
- Gaining approval for operation of RAS.

Note that a more detailed, objective-level comparison between this document and the AAIP BOK is provided in Appendix D.

C.1.6 AI Safety Landscape Categories

This platform-level framework is based on work associated with the AI Safety 2019 conference. This presents a series of seven categories[8], one of which is underpinning and one of which is overarching. These are illustrated in Table 15.

[7] https://uberatg.com/safetycase/.
[8] https://www.ai-safety.org/landscape-categories.

Table 15: AI Safety Landscape Categories

Safety-related Ethics, Security and Privacy				
Specification and Modelling	Verification and Validation	Runtime Monitoring and Enforcement	Human-Machine Interaction	Process Assurance and Certification
AI Safety Foundations				

Most of the categories are self-explanatory; the exception is AI Safety Foundations. This includes concepts such as uncertainty and generality, as well as characteristics like levels of autonomy and safety criticality. More generally, this category collects concerns in AI safety that span multiple other categories.

C.1.7 Choice of Framework

Whilst they provide a useful structure against which a chosen framework can be benchmarked, none of the preceding frameworks are suitable for use by the SASWG: they are either too focused on a specific type of platform, often a self-driving car, whereas the SASWG's work aims to cover all types of autonomous system; or they adopt a balanced view of system safety, whereas the SASWG's work deliberately targets aspects related to autonomy.

Consequently, having been informed by the frameworks listed above (and related items) a four-projection framework has been developed. These projections are described detail in Section 6. For ease of reference, a summary is provided below:

- *Behavioural Specification*, which is about what the platform is required to do (and not do).
- *Interacting Items*, which is about things intended or required to interact with the platform, not directly owned by the platform developer or operator.
- *People*, which is about how the platform interacts with people.
- *Environment*, which is about things in the operational domain outside the control of the platform developer or operator.

C.2 Framework Mappings

The following paragraphs discuss the relationship between the projections in the adopted framework and the aspects of the other frameworks outlined in the preceding subsection. Given the complexity of the items in the various frameworks, only top-level relationships are described.

The discussions also include a "Not Addressed" pseudo-projection, which captures considerations that do not readily relate to any of the projections in the adopted framework. By checking the contents of this pseudo-projection, and confirming that it contains nothing significant (from the perspective of the current document), confidence can be gained that the adopted framework covers all relevant topics.

C.2.1 Behavioural Specification

This projection relates to the Behavioural Safety, Functional Safety and Crash Safety elements from Waymo's System Safety Report.

It also relates to five of the NHTSA Safety Elements, specifically: Object and Event Detection and Response; Fallback (Minimum Risk Condition); Crashworthiness; Post Crash Automated Driving System Behaviour; and Federal, State and Local Laws.

Two of the top-level elements from the HORIBA MIRA framework are also relevant, specifically: Intended Behavior and Malfunctioning Behaviour.

The majority of the Uber Advanced Technologies Group framework is relevant to this projection. In particular, Proficient; Fail-Safe; Continuously Improving; and Resilient are all relevant.

Likewise, the majority of the top-level items from the AAIP BOK are relevant, specifically: Defining Required Behaviour; Implementing Required Behaviour; and Understanding and Controlling Deviations.

Finally, two of the AI Safety Landscape Categories are relevant: Specification and Modelling; and Verification and Validation.

C.2.2 Interacting Items

No elements from any of the frameworks are directly relevant to this projection. This is because none of the frameworks explicitly highlight the off-platform elements of the wider system. Instead, considerations of this type are implicitly included within discussions relating to the platform. However, as indicated by the objectives in Section 7, interacting items can have significant safety implications. Consequently, they are deemed worthy of separate identification.

C.2.3 People

Two elements from Waymo's System Safety Report are relevant: Operational Safety; and Non-Collision Safety.

There are three NHTSA Safety Elements that are relevant: Human Machine Interface; Vehicle Cybersecurity; and Consumer Education and Training.

A single top-level element from the HORIBA MIRA framework is relevant, namely, Malicious Intent.

None of the top-level items in the Uber Advanced Technologies Group framework are relevant to this projection.

Likewise, none of the AAIP BOK top-level items are relevant either.

Two of the AI Safety Landscape Categories are relevant, specifically: Safety-related Ethics, Security and Privacy; and Human-Machine Interface.

C.2.4 Environment

The Operational Design Domain safety element from the NHTSA directly maps to this projection, as does the Runtime Monitoring and Enforcement AI Safety Landscape category.

None of the other frameworks have items that directly map to this projection. This does not mean that these frameworks ignore the environment; it just means that these types of consideration appear at a lower-level, having been brigaded in a different fashion to the framework adopted by the SASWG.

C.2.5 Not Addressed

All elements from Waymo's System Safety Report are directly addressed by the collection of projections used in the framework adopted by the SASWG.

There are three NHTSA Safety Elements that are not directly addressed: System Safety; Validation Methods; Data Recording. The first of these is addressed by all three SASWG frameworks (i.e., by the entirety of the current document); the other two are addressed by the Computation-Level framework (Section 2).

All elements from the HORIBA MIRA framework are directly addressed.

There is a single top-level element from the Uber Advanced Technologies Group framework that is not directly addressed: Trustworthy. This element relates to the trustworthiness of the self-driving enterprise, which is a much wider consideration than the safety assurance objectives of the current document.

There is also a single top-level element from the AAIP BOK that is not directly addressed: Gaining Approval. This relates to liaison with certification authorities, which is outside the scope of the current document.

There are two AI Safety Landscape Categories that are not directly addressed, specifically: Process Assurance and Certification; and AI Safety Foundations. The former of these is outside the scope of the current document; the latter is a broad category that spans much of the content of the current document.

C.2.6 Relationship Summary

For ease of reference, the relationships outlined above are summarised in Table 16. Note that this presentation is deliberately simple and top-level.

Table 16: Relationships between Platform-Level Frameworks Considered

Framework	Behavioural Specification	Interacting Items	People	Environment	*Not Addressed*
Waymo's System Safety Report	Behavioural Safety, Functional Safety, Crash Safety	-	Operational Safety, Non-Collision Safety	-	-

Framework	Behavioural Specification	Interacting Items	People	Environment	*Not Addressed*
NHTSA Safety Elements	Object and Event Detection and Response, Fallback, Crashworthiness, Post Crash Automated Driving System Behaviour, Federal State and Local Laws	-	Human Machine Interface, Vehicle Cybersecurity, and Consumer Education and Training	Operational Design Domain	*System Safety, (Validation Methods, Data Recording are computation-level)*
HORIBA MIRA	Intended Behaviour, Malfunctioning Behaviour	-	Malicious Intent	-	-
Uber Advanced Technologies Group	Proficient, Fail-Safe, Continuously Improving, Resilient	-	-	-	*Trustworthy*
AAIP BOK	Defining Required Behaviour, Implementing Required Behaviour, Understanding and Controlling Deviations	-	-	-	*Gaining Approval*
AI Safety Landscape Categories	Specification and Modelling, Verification and Validation	-	Safety-related Ethics Security and Privacy, Human-Machine Interface.	-	*Process Assurance and Certification, AI Safety Foundations*

Overall, the preceding analysis indicates that, based on the selected comparator frameworks, there are no significant omissions from the chosen platform-level framework.

This page is intentionally blank

Appendix D Comparison with AAIP Body of Knowledge

This appendix provides a high-level comparison between the objectives established in this document and the Assuring Autonomy International Programme (AAIP) Body Of Knowledge (BOK) [41] structure.

To simplify presentation, each main section of the BOK argument structure is considered separately in the following subsections, with relevant objectives being highlighted. Brief explanations are provided for cases where there are no related objectives, for example, because of a difference in scope between the BOK and this document. For example, the BOK is concerned with an argument that covers the entire system (or platform); conversely, this document is intentionally focussed on aspects related to autonomy.

Note that the comparison reported in this appendix is deliberately high-level, with the aim of identifying whether there are any notable omissions from the collection of objectives discussed in this document. In particular, matching an objective to a BOK element does not necessarily mean that satisfying the objective will provide sufficient evidence to fully address the BOK element.

D.1 Defining Required Behaviour

Table 17 shows relevant objectives for BOK elements associated with the "defining required behaviour" section of the BOK argument structure.

Table 17: Objectives Comparison: Defining Required Behaviour

BOK Element	Relevant Objective
1.1 Identifying hazards	PLT1-2, PLT1-4, PLT2-2
1.1.1 Defining system scope	PLT1-1, PLT1-2
1.1.2 Defining the operating environment	PLT1-2, PLT2-1, PLT3-2, PLT4-1
1.1.3 Defining operating scenarios	PLT1-2, PLT2-1, PLT3-2, PLT4-1
1.2 Identifying hazardous system behaviour	PLT1-6, PLT2-3, PLT4-2
1.2.1 Considering human/ machine interactions	PLT3-1, PLT3-2, PLT3-3
1.3 Defining safety requirements	PLT1-2, PLT1-3, PLT2-2
1.2.1 Validation of safety requirements	PLT1-5
1.4 Impact of security on safety	PLT1-4, PLT1-6, PLT3-4

This table prompts several observations. Firstly, the related objectives all come from the platform-level framework: since this framework is primarily concerned with requirements, this is reassuring. Secondly, most of the BOK elements have multiple related objectives: this is a consequence of the different structural approaches that have been used; this also emphasises the point that, whilst they were a useful aid when deriving objectives, the projections (and frameworks) need not be slavishly followed. Thirdly, all of the platform-level objectives appear at least once in the table. Fourthly, considering a more detailed point, the BOK differentiates between the operating environment (1.1.2) and operating scenarios (1.1.3); conversely, this document distinguishes between the platform (i.e., behavioural specification), interacting items and the (wider) environment.

More generally, this discussion indicates that the objectives listed in this document provide an appropriate level of coverage of the "defining required behaviour" section of the BOK argument structure. This

observation provides some confidence in these objectives.

D.2 Implementation to Provide the Required Behaviour

Table 18 shows relevant objectives for BOK elements associated with the "implementation to provide the required behaviour" section of the BOK argument structure.

Table 18: Objectives Comparison: Implementation to Provide the Required Behaviour

BOK Element	Relevant Objective
2.1 System-level verification	PLT1-5
2.2 Implementation of SUDA elements	*All computation-level and autonomy architecture-level objectives*
2.2.1 Defining requirements for SUDA elements	COM1-3, COM2-1, COM2-2
2.2.1.1 Validation of requirements for SUDA elements	COM2-1, COM2-2
2.2.2 Defining requirements on components	COM1-3, COM2-1, COM2-2
2.2.2.1 Validation of requirements on components	COM2-1, COM2-2
2.2.3 Controlling interactions between components	PLT1-3, ARC1-1, ARC1-5
2.2.4 Verification of requirements for SUDA elements	PLT1-5, ARC1-5
2.3 Implementing requirements using ML	*All computation-level objectives*
2.3.1 Sufficiency of training data	COM1-1, COM1-2, COM1-3
2.3.2 Effective learning	COM2-3, COM2-4, COM2-5, COM3-1, COM3-2
2.3.3 Verification of the learned model	COM2-5, COM2-6, COM2-7, COM3-3
2.4 Controlling interactions with other systems	PLT2-1, PLT2-2, PLT2-3, PLT4-1
2.5 Controlling interactions at the system-level	PLT1-1, PLT1-3
2.6 Handling change during operation	ARC2-2, ARC3-1, ARC3-2, PLT2-3
2.6.1 Monitoring RAS operation	PLT1-6, ARC1-1, ARC1-2, ARC1-3, ARC1-4
2.7 Using simulation	COM2-6, ARC3-1, PLT1-5
2.8 Explainability	COM3-3, ARC2-1, PLT3-2

Consideration of this table highlights a number of points. For example, the BOK breaks the system down into SUDA elements and further down into components, whereas this document focuses on autonomy-enabling technologies that may be used within, or to deliver, elements and components. This means the AAIP provides a more balanced, system-wide perspective; conversely, by design, this document focuses on aspects related to autonomy.

It is also apparent that the BOK explicitly highlights simulation (BOK Element 2.7). Given the importance of this topic, this explicit highlighting is advantageous. Within the current document, simulation is considered at each framework level. Whilst this potentially dilutes the importance of the topic, it does allow specific aspects to be addressed in greater detail: for example, considerations associated with platform-level

simulation are somewhat different to those associated with computation-level activities.

Although mappings can be, and have been, made, this document's consideration of issues related *fleets* of autonomous systems (e.g., Objective PLT2-3) is not readily apparent in the BOK.

Overall, the preceding discussions do not suggest there are any significant omissions in the objectives listed in this document from the perspective of the "implementation to provide the required behaviour" section of the BOK.

D.3 Understanding and Controlling Deviations from Required Behaviour

Table 19 shows relevant objectives for BOK elements associated with the "understanding and controlling deviations from required behaviour" section of the BOK argument structure.

Table 19: Objectives Comparison: Understanding and Controlling Deviations from Required Behaviour

BOK Element	Relevant Objective
3.1 Identification of potential deviation from required behaviour	ARC1-1, ARC1-2, ARC1-3, ARC1-4, ARC1-5, PLT1-4, PLT3-2, PLT3-4
3.1.1 Identifying 'Sensing' deviations	
3.1.2 Identifying 'Understanding' deviations	
3.1.3 Identifying 'Deciding' deviations	
3.1.4 Identifying 'Acting' deviations	
3.1.5 Identifying infrastructure deviations	
3.1.6 Identifying ML deviations	
3.1.7 Identifying interaction deviations	
3.1.8 Identifying human / machine interaction deviations	
3.2 Mitigating potential deviations	PLT1-4
3.2.1 Failure mitigation	ARC1-3
3.2.2 Managing assurance deficits	*Out of scope*

This table clearly illustrates the different philosophies adopted by the BOK and this document. As noted earlier, the former adopts a balanced, system-wide approach that addresses all aspects of safety; it also separately highlights each of the SUDA elements. Conversely, this document is deliberately focused on autonomy-related items and is largely agnostic of where these are used within a system (or platform) architecture.

The table also shows the BOK's explicit focus on assurance, something that is more implicit within the current document. In particular, the lack of objectives that directly relate to the management of assurance deficits (BOK Element 3.2.2) is not considered to be a significant omission. This should occur naturally through appropriate consideration of the various objectives in this document.

More generally, there do not appear to be any significant omissions in the objectives listed in this document from the perspective of the "understanding and controlling deviations from required behaviour" section of

the BOK.

D.4 Gaining Approval for Operation

Table 20 shows relevant objectives for BOK elements associated with the "gaining approval for operation" section of the BOK argument structure.

Table 20: Objectives Comparison: Gaining Approval for Operation

BOK Element	Relevant Objective
4.1 Conforming to rules and regulations	*Out of scope*
4.1.1 Identifying applicable rules and regulations	
4.1.2 Understanding the requirements rules and regulations	
4.2 Risk acceptance	*Out of scope*
4.2.1 Evaluating risks and benefits of RAS operation	
4.2.2 Consideration of ethical issues	
4.3 Provision of sufficient confidence in the required behaviour	COM2-5, PLT1-2, PLT1-5
4.4 Provision for investigation of incidents and accidents	COM3-4, ARC2-3

This table illustrates the different scopes of the BOK and the current document. In particular, the BOK includes laws and regulations, which are intentionally out of scope for this document, as they are expected to be addressed by standard system engineering processes. The BOK also explicitly considers ethics and risks of deployment. The former, whilst important, is out of scope for this document. The latter is not identified as a separate item in this document, but satisfying the associated objectives should provide a considerable body of evidence to inform risk evaluations.

Given these considerations, and taking into account the intended scope of this document, this table does not suggest any significant omissions in this document's objectives from the perspective of the "gaining approval for operation" section of the BOK.

D.5 Non-Related Objectives

Collectively, the preceding four tables include all but four of the objectives listed in this document. Collectively, these four objectives make up the software and hardware projections, both of which relate to the computation-level. These considerations were motivated by the autonomy-focused, projection-based way that objectives were derived. In contrast, this low-level, cross-cutting concern does not readily appear from the approach adopted within the BOK. Despite this, the objectives remain important.

The SASWG view the different approaches adopted by the BOK and this document as strengths rather than weaknesses. Taking different approaches to largely the same question (accepting there is some difference in the respective scopes) helps ensure nothing is overlooked. To that end, the top-level mappings established in this appendix provide some confidence that the collection of objectives listed in this document are appropriate to their intended use.

Appendix E Comparison with UL4600

This appendix provides a high-level comparison between the objectives established in this document and a draft version (dated 2 October 2019) of UL4600 [82] which has been developed by Underwriters Laboratories (UL) and Edge Case Research (ECR).

The relative publication timings of UL 4600 and this document, together with the draft nature of UL4600, meant that an objective-by-objective comparison was neither feasible nor sensible. Consequently, the comparison has focused on identifying objectives or projections from the current document that map to the various section headings in UL4600. This provides some confidence that relevant topics have been addressed: it does not indicate that compliance with this document guarantees compliance with UL4600, or vice versa.

Each section of UL4600 that contains objectives is considered in turn below. This is followed by a very brief summary of the conclusions from this comparison exercise.

E.1 UL4600 Sections

E.1.1 Safety Case and Arguments

This section of UL4600 is mainly concerned with the structure, content and presentation of the platform-level safety case. These considerations are outside the scope of this document. However, it is noted that a platform-level safety argument would be expected to be supported by the sort of evidence produced via compliance with the objectives in this document.

E.1.2 Risk Assessment

This section of UL4600 considers fault and hazard identification, risk evaluation and risk mitigation. These topics are considered in various places in this document. Examples include Objectives PLT1-2, PLT1-3 and PLT1-4, which relate to the behavioural specification projection within the platform-level framework. Other projections in that framework are also relevant, including: Objective PLT2-3, which considers interacting items; Objective PLT3-1, which considers people; and Objective PLT4-1, which considers the environment.

E.1.3 Interaction with Humans and Road Users

The title of this UL4600 section illustrates how its scope (or at least its genesis) differs from this document. In particular, UL4600 has an implicit focus on autonomous road vehicles, whereas this document is intended to cover a much wider variety of autonomous systems including, for example, medical diagnosis systems.

The general contents of this UL4600 section are addressed by projections in the platform-level framework, for example: Objective PLT2-2 in the interacting items projection; Objective PLT3-1 in the people projection; and Objective PLT4-1 in the environment projection. There are, inevitably, some differences in the detail: for example, UL4600 explicitly identifies animals, whereas this document is less prescriptive in terms of possible elements of the environment.

E.1.4 Autonomy Functions and Support

This section of UL4600 considers (using the terminology of this document) autonomy-enabling techniques. It also considers definition of the operational design domain and specific platform-level functions that may be autonomy-related (e.g., sensing, perception, planning, prediction). Considerations related to timing are also included.

At the platform-level, use of autonomy-enabling techniques is addressed by the by the behavioural specification projection, for example, Objective PLT1-1. With regards to platform-level functions, this document is less descriptive than UL4600. However, similar notions are represented, for example: sensing and perception are related to Objective PLT4-2 in the platform-level framework; understanding the performance of ML algorithms is addressed by Objectives COM2-3 and COM2-4, from the task projection in the computation-level framework; likewise, timing requirements are addressed by Objective COM2-2.

E.1.5 Software and System Engineering Processes

The contents of this section of UL4600 are clearly described by its title. From the perspective of this document, software development processes are covered by Objective COM4-1 from the software projection in the computation-level framework; likewise, (computational) hardware development processes are covered by Objective COM5-1 from the hardware projection in the computation-level framework. Software and hardware processes are explicitly included as there are autonomy-specific considerations relevant to these areas. Conversely, within the current document there is no corresponding objective for system-level engineering processes. This reflects a deliberate focus on aspects directly related to autonomy and the associated desire to avoid duplicating existing guidance on general topics. More specifically, the autonomy architecture-level framework should allow autonomy-enabling technologies to be incorporated within standard system engineering processes.

E.1.6 Dependability

This section of UL4600 considers maintaining safety in the presence of faults, including fault detection and recovery. The use of redundancy and isolation are also covered, as are incident response and cyber security.

Within the current document, dependability (including fault prevention and fault tolerance) is covered at all three framework levels. Relevant examples include: Objectives COM4-2 and COM5-2 at the computation-level; all of the objectives related to the tolerance projection in the autonomy architecture-level; and Objectives PLT1-4 and PLT1-6 at the platform-level. Likewise, incident response is covered across a number of levels, for example, via Objectives COM3-4, ARC2-3 and PLT3-2. The same is also true for cyber security; relevant items include Objectives ARC1-4, ARC2-4, ARC3-1, PLT1-6 and PLT3-4.

E.1.7 Data and Networking

This section of UL4600 considers data communications and networks (essentially, data in motion), data storage (essentially, data at rest) and associated infrastructure. The section appears to focus on data communications to, from and within a platform, that is, "platform data": data associated with training algorithms using ML is covered in the "Autonomy Functions and Support" section. The concept of "platform data" is less explicit in this document than in UL4600. Nevertheless, relevant concepts are covered. For

example, data-related faults and failures should be managed by Objectives ARC1-1 and ARC1-3, regardless of whether the data is in motion or at rest.

E.1.8 Verification, Validation and Test

In addition to the three topics listed in the section title, this section of UL4600 also includes run-time monitoring and updates to the safety case. Within this document, the concepts of verification, validation and test are considered at both the computation-level (e.g., Objectives COM2-3 and COM2-5) and the platform-level (e.g., Objective PLT1-5). Run-time monitoring is addressed through Objective PLT1-6. The specific nature and construct of a safety case are outside the scope of this document; likewise, updates to the safety case are also out of scope.

E.1.9 Tool Qualification, COTS and Legacy Components

The contents of this UL4600 section are as indicated by its title. From the perspective of this document, qualification of software and hardware engineering tools is covered by Objectives COM4-1 and COM5-1, respectively. Commercial Off-The-Shelf (COTS) items are not explicitly addressed by this document, mainly because some readers can interpret the term too narrowly, for example, excluding open source frameworks and pre-trained ML models. However, some relevant concepts are covered, for example, by Objectives ARC1-4 and PLT3-4.

E.1.10 Lifecycle Concerns

This section of UL4600 steps through typical lifecycle phases, including requirements, design, manufacturing, operation and disposal. It also includes field modifications and updates.

This document does not include such an explicit listing of lifecycle phases. However, aspects of these are addressed, for example, in Objectives ARC2-2, PLT1-6 and PLT3-3. Updates are covered by the adaptation projection at the autonomy architecture-level. Allowable field modifications are considered in the same way; unauthorised field modifications are covered, for example, by Objectives ARC1-4, PLT1-4 and PLT3-4.

E.1.11 Maintenance

This section of UL4600 includes maintenance and other aspects of non-operational safety. In this document, maintenance is addressed via Objectives PLT3-2 and PLT3-3. Other aspects of non-operational safety are less explicit in this document than in UL4600: they are at least partially addressed by, for example, Objectives PLT1-2, PLT1-4, PLT2-2 and PLT4-1.

E.1.12 Metrics and Safety Performance Indicators

This section of UL4600 is mainly concerned with creating and monitoring platform-level Safety Performance Indicators (SPIs). The need to continually-demonstrate safety is not as explicit in this document as it is in UL4600. However, the same notion is considered by Objective PLT1-6, in the platform-level framework. This is supported by Objective ARC2-2, in the autonomy architecture-level framework, and Objectives

COM2-3 and COM2-4, in the computation-level framework.

E.1.13 Assessment

This section of UL4600 is concerned with assessing conformance to UL4600, including the use of independence and monitoring. This type of conformance is outside the scope of this document.

E.2 Summary

The preceding discussions have provided a high-level comparison between a draft version of UL4600 (dated 2 October 2019) [82] and this document. Whilst there are some intentional differences in scope, this UL4600-based analysis has not identified any significant omissions from this document.

Appendix F Comparison with OECD Principles on AI

In May 2019, member countries of the OECD adopted a number of principles[9] on AI. This appendix provides a high-level indication of how the contents of this document may support these principles.

In general, AI only has an effect when it is embodied within a wider system (or platform). Consequently, all three frameworks used in this document are potentially relevant to the OECD principles. The following paragraphs indicate how the projections and, where relevant, objectives associated with these frameworks relate to each of the principles.

F.1 Principles

F.1.1 AI should benefit people and the planet by driving inclusive growth, sustainable development and well-being.

This principle is focused on the effects of the AI. This relates most directly to the requirements that are placed on the behaviour of the associated platform: this is addressed in the behavioural specification projection within the platform-level framework.

F.1.2 AI systems should be designed in a way that respects the rule of law, human rights, democratic values and diversity, and they should include appropriate safeguards – for example, enabling human intervention where necessary – to ensure a fair and just society.

From the perspective of this guidance document, the "rule of law" part of this principle is expected to be covered by standard systems engineering process. Hence, it is not directly related to any projection (or objective).

The "human rights" and "democratic values" pieces are, arguably, about platform-level requirements: these are covered by the behavioural specification projection within the platform-level framework. Including appropriate interfaces (e.g., to support explanation of an AI-based decision) may also be relevant; this relates to Objective PLT3-2.

In order for an AI to respect diversity, this must be included in the data used to support the AI's development: considerations associated with the experience projection, in the computation-level framework, are relevant here.

The current document's focus on safety assurance means that safeguards are considered from multiple viewpoints. For example: Objectives COM4-2 and COM5-2 provide safeguards against software and hardware misbehaviour; the tolerance projection, in the autonomy architecture-level framework, provides safeguards against faults, failures and adversarial attempts to disrupt a computation; Objective ARC3-2 provides safeguards against inappropriate adaptation; and Objective PLT1-4 provides safeguards against foreseeable misuse and abuse.

Finally, the people projection, within the platform-level framework, supports the need for human intervention, where necessary.

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

F.1.3 There should be transparency and responsible disclosure around AI systems to ensure that people understand AI-based outcomes and can challenge them.

In general, publishing information against the objectives listed in this document will support transparency and responsible disclosure. This information should also allow for reasonable challenge. This could arise, for example, as part of a formal certification process; alternatively, it could come from less formal interactions with the general public.

In addition, the objectives associated with the information provision projection, within the autonomy architecture-level framework, should ensure that people are provided with accurate information. Furthermore, the objectives associated with the people projection, in the platform-level framework, should ensure that appropriate information is provided in an intelligible manner.

F.1.4 AI systems must function in a robust, secure and safe way throughout their life cycles and potential risks should be continually assessed and managed.

Arguably, all of the objectives in this document are relevant to this principle. Picking out some specific examples: Objective PLT1-2 leads to a definition of "safe operation"; Objective PLT1-4 maintains safety in the presence of faults and failures, as well as foreseeable misuse and abuse; Objective PLT1-6 provides monitoring during operational use (e.g., to identify new hazards, as part of continual assessment and management); Objective COM1-4 protects against distribution shift; Objective ARC1-2 ensures the platform is tolerant to "out of support" operational inputs; Objective ARC1-4 protects against adversarial attempts to disrupt a computation; Objective ARC3-1 protects against inappropriate or unauthorised adaptations; and the people projection, within the platform-level framework, explicitly covers the whole system lifecycle.

F.1.5 Organisations and individuals developing, deploying or operating AI systems should be held accountable for their proper functioning in line with the above principles.

This principle is mainly concerned with the legal and regulatory environment within which AI systems are used. These considerations are deliberately outside the scope of the current document. Nevertheless, requiring compliance with the objectives in this document may be one way of holding to account those responsible for developing, deploying or operating such systems.

Appendix G 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.

- The objectives in this document have been developed largely from a theoretical basis. Efforts have been made to provide confidence in the objectives, for example, by comparing them with a number of related documents. However, to date, the objectives have not been "proven through use". In particular, there would be significant value in completing several worked examples.
- At the moment, the document makes no distinction between different criticality levels, for example, DALs or SILs. As exemplified by the last of the "four plus one" software safety assurance principles [42], it is beneficial to target effort towards more critical system elements. In order to achieve this, some form of graduation is necessary. This could be achieved by requiring certain objectives only to be completed at higher criticality levels. Alternatively, higher criticality levels may require some objectives to be completed with independence. Another approach involves addressing an objective more thoroughly as the criticality level increases. It is unclear which combination of these options will be most appropriate for the SASWG's work but, at the current time, it is considered likely that most objectives will need to be satisfied, regardless of criticality level.
- At the moment, the document makes little distinction between different types of ML approach. In some cases this may be captured in the detailed response to an associated with an objective: for example, Objective COM3-2 relates to protecting against typical errors, which will differ between different types of ML approach. However, there are several places where a more nuanced approach may be more valuable. One such area is the relationship between the adaptation projection (discussed in subsection 5.3) and ML approaches that continue to learn during operational use.
- Objective COM2-7 notes the difference between algorithm instances, which may produce different results, and algorithm variants, which are expected to produce different results in some circumstances. It is not clear at which point an algorithm ceases to be a variant and becomes something that should be considered as an item in its own right. A key consideration is how easily safety assurance evidence can be transferred between items. More generally, this topic is related to reuse and software product line engineering.
- Currently, the document does not provide much information on integration of Software (SW) and Hardware (HW); neither SW-SW integration nor SW-HW integration is considered in any detail. Although the objectives associated with the autonomy architecture-level framework are helpful in this regard, this is an area that may be addressed in more detail in future editions.
- 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.

This page is intentionally blank

Appendix H Abbreviations

AAIP	Assuring Autonomy International Programme
AD	Autonomous Driver
ADS	Automated Driving System
AI	Artificial Intelligence
ANN	Artificial Neural Network
AS	Autonomous Systems
BIT	Built-In Test
BOK	Body Of Knowledge
CM	Configuration Management
CNN	Convolutional Neural Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DAL	Development Assurance Level
DfT	Department for Transport
DLOD	Defence Line of Development
DNN	Deep Neural Network
ECR	Edge Case Research
FMEA	Failure Mode and Effects Analysis
FPGA	Field Programmable Gate Array
FRAM	Functional Resonance Analysis Method
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit
GSN	Goal Structuring Notation
HAZOP	Hazard and Operability Study
HMI	Human-Machine Interface
HW	Hardware
ISO	International Organization for Standardization
ISSRE	International Symposium on Software Reliability Engineering
MAPE	Monitor, Analyse, Plan, Execute
MC/DC	Modified Condition / Decision Coverage
MEL	Minimum Equipment List
ML	Machine Learning
MOD	Ministry Of Defence

NaN	Not a Number
NCSC	National Cyber Security Centre
NHTSA	National Highway Traffic Safety Administration
NN	Neural Network
OECD	Organisation for Economic Co-operation and Development
PCA	Principal Component Analysis
PDI	Parameter Data Item
PR	Process Requirement
PSH	Product Service History
RAM	Random Access Memory
RAS	Robotics and Autonomous Systems
REQ	requirement
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SASWG	Safety of Autonomous Systems Working Group
SCANN	Safety-Critical Artificial Neural Network
SCSC	Safety Critical Systems Club
SEU	Single Event Upset
SIL	Safety Integrity Level
SMS	Safety Management System
SoC	System-on-Chip
SOTIF	Safety Of The Intended Function
SOUP	Software of Uncertain Pedigree
SPI	Safety Performance Indicator
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
UL	Underwriters Laboratories
V&V	Verification and Validation
WCET	Worst Case Execution Time
WoSoCer	Workshop on Software Certification

This page is intentionally blank

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

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

[7] 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.

[8] 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.

[9] 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.

[10] D. Bedford, G. Morgan, and J. Austin, *Requirements for a standard certifying the use of artificial neural networks in safety critical applications*, in Proceedings of the international conference on artificial neural networks, 1996.

[11] 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).

[12] J. Birch, *Safety argument framework and considerations for highly automated vehicles*. Personal Communication, September 2017.

[13] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, *What's your ML test score? A rubric for ML production systems*, in NIPS Workshop on Reliable Machine Learning in the Wild, 2016.

[14] S. Burton, L. Gauerhof, and C. Heinzemann, *Making the case for safety of machine learning in highly automated driving*, in International Conference on Computer Safety, Reliability, and Security, Springer, 2017, pp. 5–16.

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

[16] 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.

[17] 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).

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

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

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

[21] 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.

[22] 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.

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

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

[25] 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.

[26] M. Douthwaite and T. Kelly, *Safety-critical software and safety-critical artificial intelligence: Integrating new practices and new safety concerns for AI systems*, in Evolution of System Safety, Proceedings of the Twenty-sixth Safety-Critical Systems Symposium, Safety-Critical Systems Club, 2018. ISBN 978-1979733618.

[27] 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.

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

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

[30] 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.

[31] 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.

[32] 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.

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

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

[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] R. Hawkins, *Body of knowledge - structure and scope*, tech. rep., Assuring Autonomy International Programme, April 2019.

[42] R. Hawkins, I. Habli, and T. Kelly, *The principles of software safety assurance*, 31st International System Safety Conference, Boston, Massachusetts USA, (2013).

[43] 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.

[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] 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.

[47] ISO, *Road vehicles - functional safety*, Tech. Rep. ISO 26262, ISO, 2011.

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

[49] 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.

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

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

[52] 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.

[53] 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.

[54] 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).

[55] 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.

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

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

[58] C. Menon and R. Alexander, *A safety-case approach to ethical considerations for autonomous vehicles*, in Proceedings of the 12 IET International Conference on System Safety and Cyber Security, IET, 2017.

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

[60] 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).

[61] 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.

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

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

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

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

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

[67] 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.

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

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

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

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

[72] 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).

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

[74] 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.

[75] 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.

[76] 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.

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

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

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

[80] 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. O.

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

[82] Underwriters Laboratories, *The standard for safety for the evaluation of autonomous products*, tech. rep., Underwriters Laboratories, Edge Case Research, October 2019.

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

[84] K. Wagstaff, *Machine learning that matters*, in Proceedings of the 29th International Conference on Machine Learning, 2012, 2012, pp. 529–536.

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

[86] 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).

[87] Y. Yang, M. He, M. Li, Q. Wang, and B. Boehm, *Phase distribution of software development effort*, in Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '08, New York, NY, USA, 2008, ACM, pp. 61–69.

[88] 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.

This page is intentionally blank

Appendix J 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:

- Rob Alexander, University of York
- Hamid Asgari, Thales UK
- Rob Ashmore, Dstl
- Andrew Banks, LDRA
- Rajiv Bongirwar, Hemraj Consultants
- Ben Bradshaw, ZF
- John Bragg, MBDA UK Ltd.
- John Clegg, Independent (ex-QinetiQ)
- Jane Fenn, BAE Systems
- Chris Harper, University of the West of England
- David Harvey, Thales UK
- Nikita Johnson, University of York
- Catherine Menon, University of Hertfordshire
- Roger Rivett, Independent (ex-Jaguar Land Rover)
- Philippa Ryan, Adelard LLP
- Mark Sujan, Human Reliability
- Nick Tudor, D-RisQ
- Stuart Tushingham, Altran

Contributors to the previous versions include:

- Rob Alexander, University of York
- Rob Ashmore, Dstl
- Andrew Banks, LDRA
- John Birch, Horiba-MIRA
- Ben Bradshaw, ZF
- John Bragg, MBDA UK Ltd.

- Lavinia Burski, AECOM
- John Clegg, Independent (ex-QinetiQ)
- Timothy Coley, XPI Simulation
- Chris Harper, Atkins
- Neil Lewis, Dyson
- Catherine Menon, University of Hertfordshire
- Ken Neal, Ebeni
- Ashley Price, Raytheon
- Stuart Reid, STA Consulting
- Roger Rivett, Jaguar Land Rover
- Philippa Ryan, Adelard LLP
- Alan Simpson, Ebeni
- Rod Steel, Thales
- Nick Tudor, D-RisQ
- Stuart Tushingham, Altran

Authored by the Safety of Autonomous Systems Working Group (SASWG), and building on a previous edition, this document now 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 an emerging consensus of what is required to support autonomous system safety arguments.

Nevertheless, the current edition represents a step on a journey. As further work is conducted, and experience is gained, revisions are expected. To support this, comments on this document are actively encouraged.

In the previous edition, three frameworks were identified and of these the computational-level framework with projections and objectives was described. In this edition, the other two frameworks have been renamed and fully developed.

SCSC
FOR EVERYONE WORKING IN SYSTEM SAFETY

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.

ISBN 9781654029050
90000
9 781654 029050