Assurance Cases to face the complexity of ML-based systems verification

Vincent Mussot, Eric Jenn, Florent Chenevier, Ramon Conejo Laguna, Yassir Id Messaoud, Jean-Loup Farges, Anthony Fernandes Pires, Florent Latombe, Stephen Creff

Assurance Cases to face the complexity of ML-based systems verification

Vincent Mussot*‡, Eric Jenn‡, Florent Chenevier*§, Ramon Conejo Laguna*‡, Yassir Id Messaoud*, Jean-Loup Farges*†, Anthony Fernandes Pires*†, Stephen Creff*, Florent Latombe¶
*IRT SystemX, †ONERA, ‡IRT Saint Exupéry, §Thales, ¶Obeo

***Abstract*—The verification and validation of AI-based systems raise new issues that are not easily addressed by existing practices and standards. We think that this gap is actually an opportunity to introduce new practices and establish a clearer and more formal link between the engineering activities and artefacts, the expected properties of the system, and the verification and validation evidence.**

Therefore, in this paper, we describe and illustrate an approach integrating (i) the definition and modelling of an AI-based system engineering workflow, (ii) the identification of the trustworthiness properties, and (iii) the argumentation demonstrating the satisfaction of these properties. This approach is centred on the model of Assurance Cases, a semi-formal representation of argumentation which supports the claim of system trustworthiness. In addition, we present supporting tools for this formalism that enable the automatic production of Verification and Validation plans for specific properties of AI-based systems.

***Index Terms*—Assurance Case, Machine Learning, Robustness, V&V**

I. Introduction

There is an obvious and strong willingness to leverage the capabilities of Machine Learning in all domains of industry, including those delivering business- or safety-critical services. However, the adoption and deployment of this technology remain slow, for we fundamentally lack confidence in these methods.

In the Confiance.ai program[1], we address this problem by tackling all stages of the development of systems involving Artificial Intelligence (AI), spanning from defining engineering workflows to implementing and deploying ML-related components on hardware platforms. This process revolves around the notion of trustworthiness, which becomes increasingly crucial as AI plays a larger role in the system. Besides, the program's effort is essentially directed toward identifying and addressing the novel challenges that emerge during the integration of AI into such systems. More specifically, focus is placed on specific attributes of ML-based systems, such as model robustness, explainability, or fairness, with the broader goal of improving our confidence in the final system. Besides, the justification of this confidence forms a necessary condition of the safety case for such system. It requires a global approach to explicit the link between the different parts of this safety case: the Engineering Items produced during development, the Activities that produce them, the expected Properties of these items and the justifications that these properties actually contribute to the confidence on the ML-based system, all these used in a structured argumentation allowing to demonstrate the satisfaction of those properties. This approach is needed to guarantee the completeness, transparency and auditability of the argumentation. It also supports traceability and impact analysis.

The approach is based on Assurance Cases (ACs) [1], a method that associates the property of interest to be demonstrated with the evidence supporting it through convincing and valid reasoning. The Goal Structuring Notation standard (GSN) [2] used in this work is one of several formalisms (e.g. Claims Arguments Evidence (CAE) [3] or Structured Assurance Case Metamodel (SACM) [4]) designed to model an assurance case. It consists of decomposing, according to a specific strategy, a high-level claim representing a property of interest into elementary sub-goals that can be easily proven with evidence. This formalism has already been adopted in several industries, particularly under the specific form of Safety Cases [5] (when the argued property is safety). It is also a practice recommended by several international standards such as IEEE and ISO [6], [7]. By providing clear and explicit reasoning to demonstrate the property of interest, assurance cases simplify the tasks of reviewing the argument, as well as correcting it or completing it if necessary.

We argue that this approach is particularly suited for ensuring properties about an ML-based system for which the guarantees provided by conventional engineering practices are sometimes insufficient and often simply not applicable [8]. The approach is not specific to any particular industrial domain, but its application

[1] https://www.confiance.ai/en/

clearly makes more sense in domains where significant stakes are involved. These stakes may relate to the impact on individuals (such as aeronautics, industry, or automotive), costs, or overall image (across all domains). **Our contributions are:** • A process and the associated tool that enable (i) building a generic argument to demonstrate that the system actually complies with some expected overarching properties thanks to appropriate development practices and/or V&V activities, (ii) formalising the relationship between this argument and the development artefacts of the system (engineering activities, engineering items and their properties). • A set of argumentation templates covering some major properties expected for a system involving AI. • A means to (i) derive a generic argument based on the contribution of each of its elements and (ii) produce the complete workflow including both the development and the V&V activities determined by the argument.

All these contributions are supported by a dedicated *tooling support* section which details how the method used is effectively implemented in our framework and how the tool can be used to reproduce each step or activity.

The remainder of this paper is structured as follows. Section II gives an overview of our approach which is then further decomposed into the 5 sections: Section III presents an extract of a development workflow and the associated engineering items of interest and Section IV identifies a set of properties for which guarantees are expected. Section V provides several generic assurance cases to ensure these properties and Section VI details how these assurance cases can be instantiated. The choices made during this phase lead to the selection of Development and Verification and V&V activities that must be integrated into the workflow and compiled into a V&V plan as presented in Section VII. Finally, Section VIII provides an overview of the related works, and Section IX concludes the document and opens some questions and future work.

II. Approach overview

Our main objective is to exhibit the elements to be provided to demonstrate the satisfaction of a safety-related property of the system, considering the activities conducted to build it, and the artefacts produced in the process. The demonstration in this case is neither formal nor mathematical, but it shall nevertheless provide the argumentation and evidences necessary to convince – for instance, a regulation authority – that the property actually holds.

One necessary condition to support this objective is to define precisely the concepts involved in the development, verification and validation of a system, and their relationships. Towards that goal, we rely on the Model-Based Systems Engineering (MBSE) approach where activities and Engineering Items are well defined (*i.e.*, they comply with a metamodel) and are associated with modelling artefacts.

Figure 1 presents the different steps that compose our approach to verifying the safety-related properties of the system. The numbering corresponds to the one used in the sections of this paper.

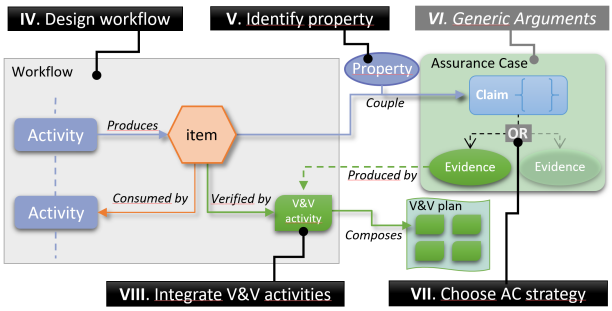

Fig. 1. General approach to demonstrate the satisfaction of safety-related properties

The starting point of this approach is the engineering workflow, from which we will extract the engineering items of interest. On each of these engineering items, we will then identify properties related to requirements on the system. The couple ⟨item, property⟩ will therefore constitute the root of an Assurance Case, claiming that the property is satisfied on this particular item. We will provide in Section V a set of generic Assurance Case patterns for specific ML-related properties. However, these generic patterns cannot be used as is. They may contain contexts and elements that need to be instantiated, typically with engineering items from the current system of interest. Besides, these argumentation trees may contain alternative branches in their decomposition, sometimes mutually exclusive, which implies that a choice has to be made in the argument by the user. This step is crucial as the resulting argument will dictate which Verification and Validation activities must be performed and integrated in the workflow. These new activities will finally be extracted in the form of a Verification and Validation plan which can be followed to ensure the expected property.

Tooling support details

Within the Confiance.ai program, the MBSE approach is implemented in the Capella[2] solution: an open source

[2]https://mbse-capella.org/

extensible Eclipse[3] application dedicated to Systems Architecture modelling. The solution implements the Arcadia method [9] that promotes the use of dedicated modelling perspectives: "Operational Analysis", "System Analysis", "Logical Architecture", "Physical Architecture", "EPBS architecture". The Confiance.ai program approach extends the "Operational Analysis" (OA) one, to provide a methodological end-to-end engineering approach to support the particularities of engineering activities related to critical AI-based systems [10]. It defines an *"Engineering Activities for trustable AI"* Capella Viewpoint, in which: Engineering Process, Process Sequence Flow, Process Activity, Process Item Flow, Engineering Role, Engineering Activity, and Engineering Exchange concepts, are mapped to Capella objects.

III. Workflow design

The development *workflow*, which models the development activities and the corresponding Engineering Items (more specifically *Exchanged items* in Capella) they produce, is the starting point of our approach. Figure 2 is an excerpt of the comprehensive workflow for developing ML-based systems produced in the Confiance.AI program [11]. It presents a view of the activities and sub-activities commonly considered during the model engineering phase [12]. The ML model development is typically absent from conventional software workflow, as it is an ML-related activity. There are 3 possibilities to ensure a specific property on this activity:

1) Ensure that property holds for the item resulting from this activity (here the `trained ML model`),
2) Rely on a development activity which guarantees the property *by design* (or *by construction*),
3) Verify that the property holds on the item *before* the activity, and rely on guarantees that the activity preserves the property.

For this part of the workflow, we focus on the first two solutions, as the last solution implies to preserve a property during the training phase. This can be particularly challenging due to the absence of control over the optimisation process used during ML model training. Moreover, some of the properties we are interested in, such as robustness, are typically stemming from the training process itself, making it impossible to ensure any property preservation from a prior activity in this case.

[3]https://www.eclipse.org/

On the opposite, the first solution is typically based on additional verification and validation activities. Providing guarantees by design can also be effective for specific properties but is often insufficient and requires additional verification activities to obtain the appropriate level of guarantees, especially for critical systems.

In the following, we will therefore focus on the `Trained ML Model`, our engineering item of interest, and on the `Train ML Model` activity that produces it.

IV. Properties identification

In a safety-critical context, one major concern is the system's trustworthiness. Therefore, this paper focuses on trust-related properties that are specific to Machine Learning ([13]) and particularly difficult to verify. Historically, these properties are determined by the potential threats to which each activity is exposed. However, in the case of ML-based systems, the list of new threats is considerable [14], and highly dependent on the system of interest and the different activities involved in the development workflow. Therefore, in practice, we rely on specific regulation [15] or general guidance [16] to express these properties. Among the key aspects often considered in these documents, we selected *Robustness*, *Explainability* and *Fairness* as the three main properties to ensure on the `Trained ML Model` engineering item, as illustrated in Figure 3.

Furthermore, for these properties to be meaningful and carefully considered during the design of a system, they must be refined into low-level requirements that directly relate to the item of interest (the `Trained ML Model` in our case). For now, it is up to each industrial to interpret these guidelines and produce his own refinement of these properties in coherence with his use case. This may imply to consider the Operational Design Domain (ODD), which is a set of conditions in which the system is designed to operate [17], [18]. Indeed, some of these conditions can be propagated through the system down to the ML component and expressed as input constraints. Depending on the condition considered, this propagation may establish a link with a specific property like *robustness* or *fairness*. A typical example of operational conditions impacting the robustness of a trained ML model could be scenarios where a camera system is subject to vibrations, resulting in blurred images, or foggy environments leading to noisy images. These effects could then be translated into quantifiable metrics in the image space, for instance in maximum acceptable perturbations expressed with regard to the L_∞-norm.

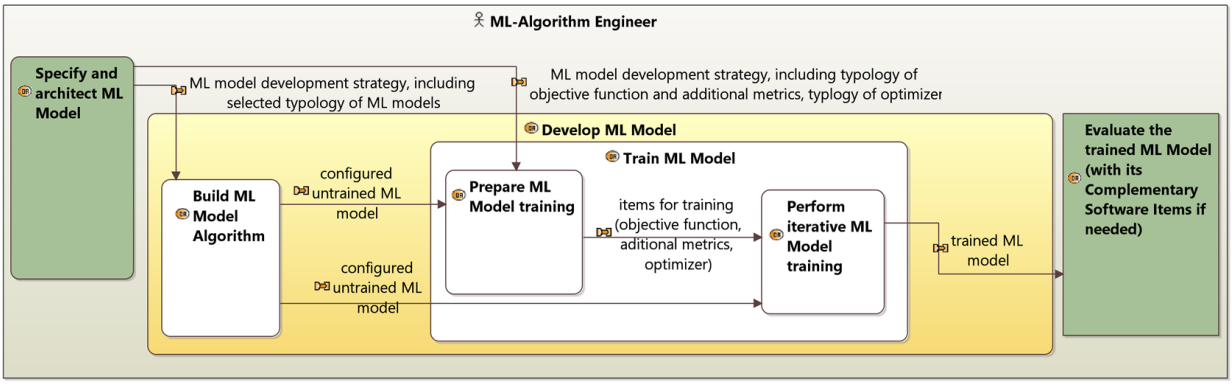

Fig. 2. Generic ML workflow focused on Model Engineering

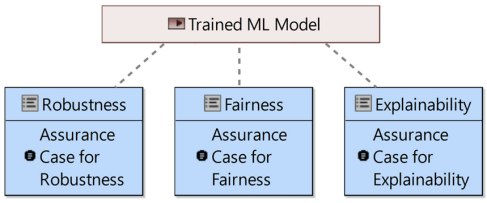

Fig. 3. Focus on the `Trained ML Model` Exchange Item in an extended Capella Class Diagram [CDB] with *Engineering Properties*

Although these analyses are typically application-dependant, we argue that, at least for the *robustness* property, a first level of refinement can be made generically, aligned with the capabilities of existing tools for verifying these properties or ensuring them by design.

Low-level refinements

For the *Robustness* property, we consider the following definition of *local* robustness: *"A Trained ML Model is locally robust for a single input x to a perturbation radius λ if it produces the same output for any perturbation x' with* $\texttt{distance}(x, x') \leq \lambda$*"*, where `distance` can be the l_2-norm or the l_∞-norm commonly used in that context.

Leveraging this definition which focuses on a single input, we can express global robustness criteria using three possible metrics:

- **Percent robust:** The percentage of samples that are locally robust for a fixed λ
- **Max robust:** The maximal λ for which all samples are locally robust
- **Mean robust:** The mean of the maximal λ for which each sample is locally robust

These three possible criteria expressed at the level of the ML component, coupled with our engineering item of interest –the `Trained ML Model`– form the root goals of three different argumentation trees. This helps separate methods depending on whether they support the corresponding norms and metrics, although these root goals must still be instantiated with the appropriate λ and l-norm.

The *Robustness* property can be formalised using mathematically grounded concepts and formulas, which make it suitable for the refinement presented above. However, it is less direct for more softer properties. For instance, considering the main usages of *Explainability* in our case, we refined this property in two main requirements: providing explanations for successful model decisions or ensuring the absence of bias in the model's decision-making process. These two aspects reflect the separation between local and global explanations, which consist of either explaining a single decision or explaining a set of decisions. Moreover, we further divided the local explanations into success-case and failure-case explanations. However, the primary usage of failure-case explanations is for the ML-Algorithm Engineer (see Figure 2) to find, during training, the reasons for a model failure, and to use these insights to retrain it and correct it. Conversely, the verification and validation of the `Trained ML Model` resulting from the application of the assurance case consider the model to be in a final, stable version. In that state, there should be no more failure cases to explain, but only success cases, which would still need those local explanations for increased trustworthiness. Therefore our main refinement for local explanations is expressed as *"The correct decisions of the model are explained"*.

Global explanations, on the other hand, serve to detect general biases in the model, which might reflect a problem during training or even an issue with the training dataset. Therefore we proposed *"The model is unbiased"* as a second refinement of the *Explainability*

in a separate Assurance Case.

Finally, the expression of the *Fairness* property was made more explicit with the following refinement: *"The ML model does not contribute to any undesired discrimination"*.

Tooling support details

In our approach, this decomposition of properties into low-level requirements is supported by a dedicated tool named *pure::variants*[4] entirely integrated into the Capella environment (more details are provided in the following sections). This tool handles both the workflow model and the assurance case trees and will be used during several more or less complex steps of our approach. Notably, it offers a filtering functionality (configuration process of *pure::variants*) that extends up to the selection of the appropriate low-level requirements, as illustrated in Figure 4.

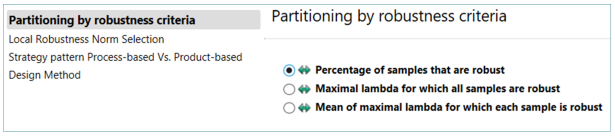

Fig. 4. Illustration of the filtering capability of *pure::variants*.

V. Generic Assurance Cases

We provide in a git repository[5] a set of generic assurance case patterns for the identified properties, namely *Robustness*, *Explainability* and *Fairness*, modelled in GSN. As explained in previous section, we refined these properties into more specific requirements, each resulting in a new Assurance Case. We present two extracts of these argumentation trees in Figures 5 and 6.

These argumentation trees are modelled in Capella using a dedicated *Assurance Case Viewpoint* supporting an enriched version of the GSN meta-model, and integrated in the Arcadia method.

The first extract in Figure 5 shows the decomposition of a robustness criterion according to several strategies, including the type of method used (*by design* or *by evaluation*) and the choice between these methods which is materialised in the tool by a diamond-shaped node. In these assurance cases, no choice has been made yet between the multiple branches, as these trees remain generic and thus can be considered and adapted to a variety of use cases.

[4]https://www.pure-systems.com/purevariants
[5]https://github.com/ TBCompleted!

The second extract, presented in Figure 6 shows a generic argument for the local explainability of a *Trained ML model*. It focuses on the use of attribution methods for computer-vision tasks, ensuring that, for each decision, the part of the images used in each image is relevant. The main branch of this argument relies on the validation of the method results by experts whose legitimacy is verified in a specific subgoal. However, the AC also contains a specific branch dedicated to ensuring that the explainability method used is trustworthy, which will require additional V&V activities.

Using these generic assurance cases in our approach consists of instantiating the generic contexts and resolving the pending choices.

Tooling support details

As introduced before, the Capella environment is enhanced with an *Assurance Case Viewpoint*. It provides new Capella modeling elements, and new and enhanced diagrams at operational analysis level. The modeling elements added to support this Viewpoint are twofold:

- The concepts from the GSN standard in the version 3, cf. [19]. These are added to the "Operational Analysis" metamodel part and gathered into "Assurance case" elements in "Assurance Cases Pkg" ones. A specific GSN diagram implements the diagrammatic concrete syntax of the assurance case, composed of goals, strategies, solutions, etc. The relation with engineering operational activities is made via "Engineering item" and "Engineering item elements" referenced from GSN Solution elements.
- Some Glossary entries are also added to the "Operational Analysis" metamodel part and grouped into a "Glossaries pkg" element. Indeed, each GSN element is described (in rich text format) with hyperlinks to definitions in the glossary, or to external referential.

VI. AC Instantiation and Choices

Two tasks must be completed to instantiate an AC: The choices between branches must be made and the generic contexts must be instantiated. These tasks cannot be performed independently since, the selection of one branch might depend on the availability of the corresponding contexts, and on the opposite, only the instantiation of contexts of selected branches is mandatory to obtain a complete, *instantiated* AC.

An example of choice is provided on the extract of the *Robustness* AC in Figure 5. The cardinality of choice (here `[1..4]`) indicates that each method provided below in isolation is able to provide a certain degree

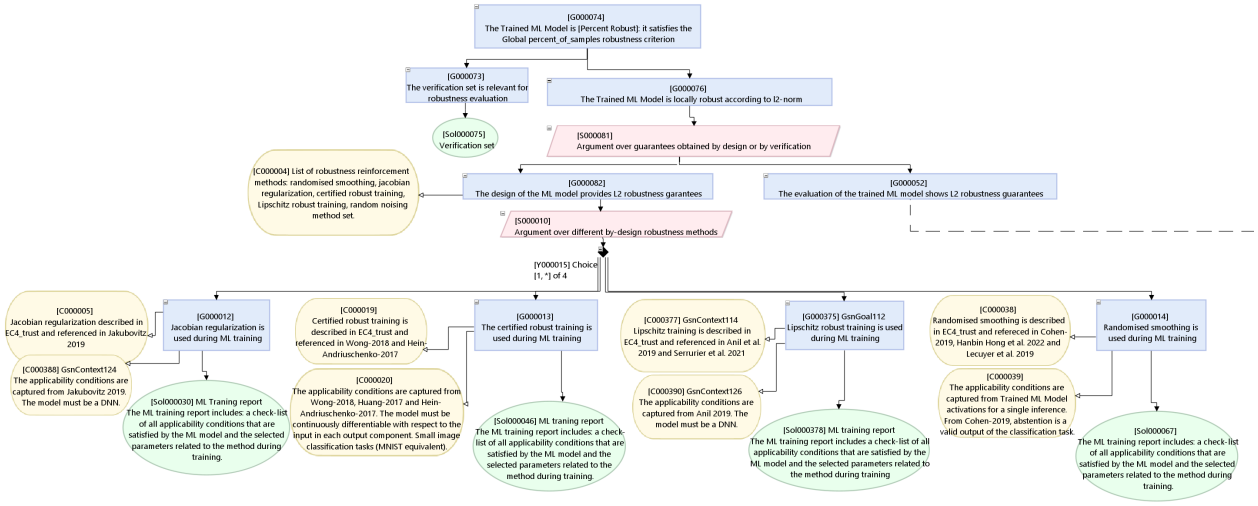

Fig. 5. Extract of the Assurance Case for the robustness of the `Trained ML Model`, considering a *Percent robust* metric with a l_2-norm, with a focus on the methods providing guarantees *by-design*. The whole assurance case is provided on the following git repository

of guarantee for the l_2-norm robustness. This choice can be based on several criteria, such as the criticality of the property of interest, the cost resulting from the deployment of a certain strategy/approach (i.e., time, effort, computational power, etc.) or other considerations related to the profile/experience of the engineer or its company. For instance, a critical property in a critical system will require the approach, or the combination of approaches, that brings the most guarantees to the satisfaction of the property of interest. On the other hand, numerous approaches [20], [21], [22] of confidence/uncertainty assessment of assurance cases, based on experts' judgements, can be used as a selection criterion. Indeed, a user may choose the strategy or the approach which provides the most (resp. the least) confidence (resp. uncertainty) to the satisfaction of its property.

In the generic arguments provided, nothing prevents the user from selecting several, if not all methods, regardless of their compatibility. Despite this issue, which needs to be verified on a case-by-case basis, it is even often recommended to use all possible methods available for a given choice to increase the overall confidence in the top-level argument. However, this will naturally come with increased costs, appearing when producing the corresponding V&V plan.

On the right of the *Explainability* AC extract in Figure 6, the context `C000139` (in yellow) is an example of node requiring instantiation: The list of attribution methods to verify is not known in advance as it will be highly use-case dependant and may change over time, which is why only a few suggestions can be provided. The context node `C000176` also requires instantiation, as it is used in the attached goal as the set of "selected metrics" which must be evaluated for each explanation method considered.

These choices can be made in the *assurance case viewpoint* in Capella, where the user can either directly select the appropriate method for his use case , or use the tool *pure:variants* presented earlier, as illustrated in Figure 7.

The Assurance Case is considered *instantiated* when these two tasks are completed. However, this instantiation (especially the choices) structurally removes branches from the AC, leaving only the subset of V&V activities required to produce the expected pieces of evidence. Hence, the instantiation process impacts both the workflow and the AC. Depending on the structure of the workflow and the argumentation, and the potentially dual role of engineering items – i.e., playing a role in the argumentation and in the development workflow – this chain of effects can propagate back and forth between the workflow model and the argumentation model.

Tooling support details

In summary, GSN and Capella extended OA models are defined exhaustively, i.e. considering every possible goal, strategy and solution for any ML-based component and any property. However, an argumentation, in the end, has to be fit for a specific purpose. This means methodically removing specific model elements (On

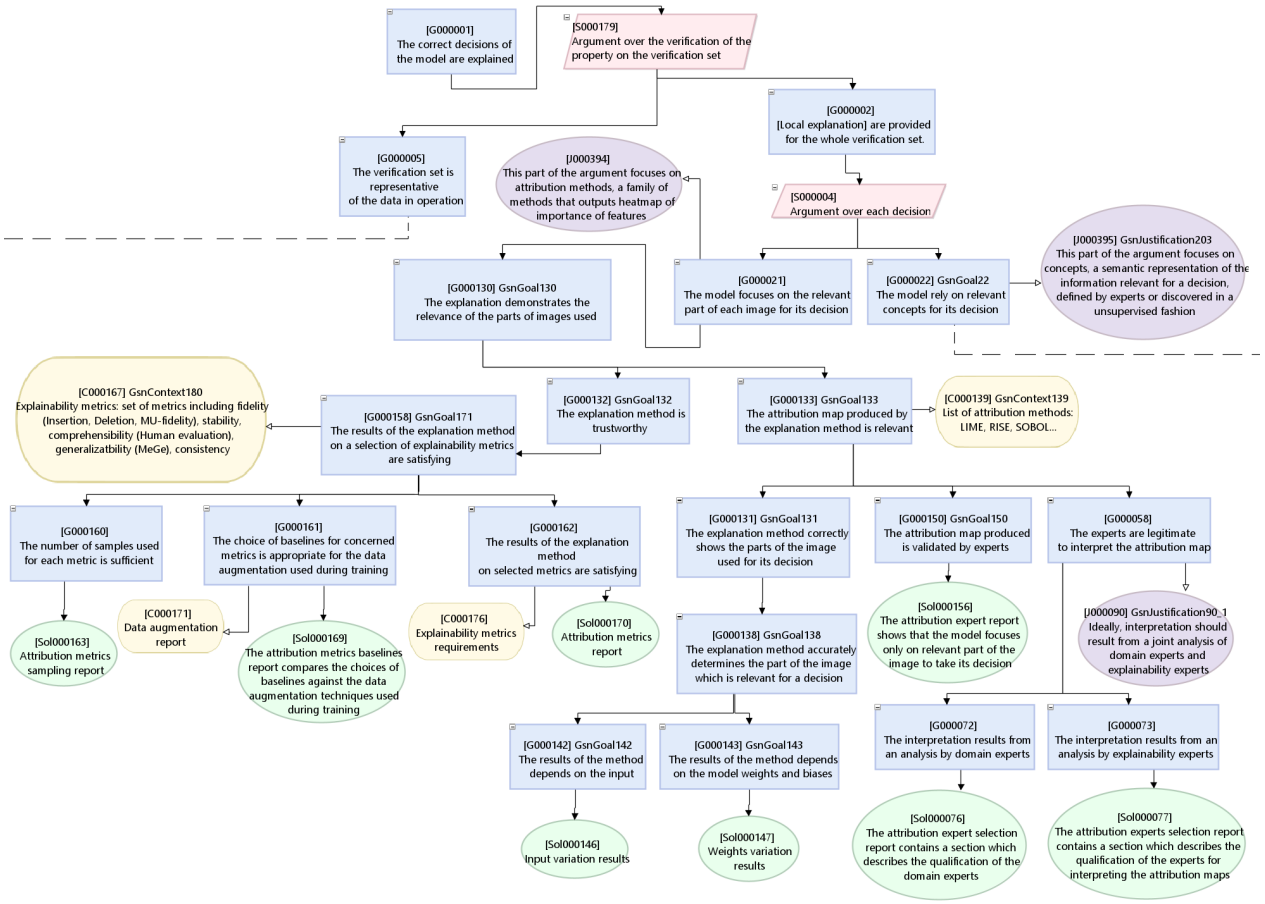

Fig. 6. Extract of the *Local Explainability* Assurance Case with a focus on the attribution branch. The whole assurance case is provided on the following git repository

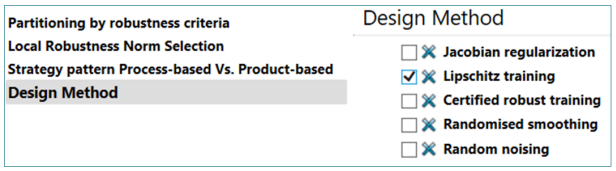

Fig. 7. Illustration of the functionalities of *pure::variants* for the selection of AC choices.

both Assurance case and Capella engineering parts) from the initial exhaustive models.

One community has already addressed these implementation problems for another intent: reuse and rationalisation of product families. Indeed, Software product line (SPL) [23] techniques aim at deriving a tailored product from a set of features [24]. The variability between these features is consistently managed, defining options, alternatives (`AND`, `OR`, `XOR`), and mandatory and exclusive features. The implementation follows a so-called negative variability [25] (or annotative [26]) approach, which uses some form of explicit or implicit annotations in the models. This approach is here, as previously introduced, implemented with the use of a commercial SPL tool named *pure::variants*.

In our case, the exhaustive assurance case modelling represents a so-called 150% modeling [27] that acts as a base model, including all supported variability. Options or alternatives that are not selected for a specific argumentation during the configuration phase are then removed from the base models in the AC *instantiation* phase.

The implementation is done as depicted in Figure 8:

- (A) In the feature models (.xfm files) are encoded the valid possible configurations of the tactics that are applicable to a given property. An example for the robustness AC is given in Fig. 9. The feature model defines first-order logical constraints

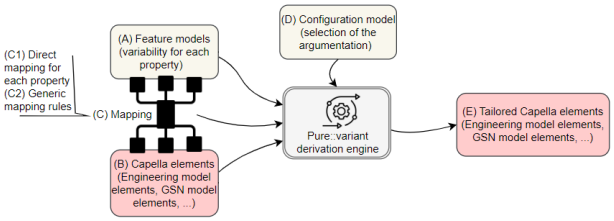

Fig. 8. Pure::variant implementation and workflow overview

between features with operators (e.g. the `AND` operator between "Local Robustness" and applying a "strategy pattern" as represented in the Figure), and direct constraints (e.g. selecting "Property satisfied by design" requires to apply a "Design Method"). It gathered all possible choices available in the configuration, from the AC selection property to consider (Non-exclusive `OR` between "Robustness", "Fairness", "Explainability"), to the partitioning choices (cf. 4), to the different strategy choices into the GSN model (i.e. the cardinality of choice in the GSN model, e.g. `[1..4]` in Figure 5 has to be consistent with the one in the feature model – Non-exclusive `OR` with cardinality constraints).

- (B) In the Capella model is represented exhaustively (150%) the modelling element (engineering and assurance cases in GSN).
- (C) The mapping between the external feature model is performed at two levels: (C1) a specific mapping between one feature model and the Capella model (.ccfm file); (C2) generic deletion (propagation) rules. Indeed, in the extended Capella, different element types are semantically related to each other. Propagation rules utilise these semantic relations to simplify the mapping of Capella elements to conditions. Basically, a propagation rule ensures that if Element A is removed during transformation, also Elements B, C, and D are removed. We defined 16 customised rules.
- (D) The configuration model drives the possible choices according to the feature model logic (.vdm file). It also stores the final configuration. This configuration is graphically customised via the use of a configuration wizard model (.vcwm file) to provide a graphical interface as illustrated in Fig. 4 or Fig. 7.

With the information given in all these files, the *pure::variants* derivation engine produces a new Capella project containing only the instantiated AC, as represented in Figure 8.

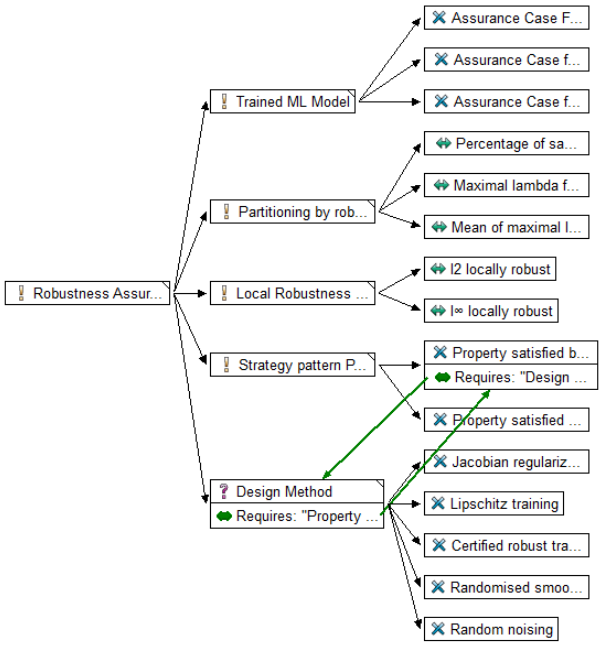

Fig. 9. Extract from the *Pure::variants* feature model on the robustness AC

VII. V&V ACTIVITIES INTEGRATION

The instantiated AC provides a list of evidences that must be produced through specific V&V activities which must therefore be added to the development workflow[6]. These activities produce the required evidence as V&V Engineering Items, and they should take as input either development Engineering Items or other V&V Engineering Items. At this stage, the **Workflow** contains both the development activities and the V&V activities, as illustrated Figure 10.

The AC serves as an intermediate yet important model for building a comprehensive and convincing V&V plan, by combining the new V&V activities needed to produce the evidences required. This plan can thus be automatically generated from information related to the engineering item, the property to be verified and all other elements carried by the argument. It also includes other information, such as the glossary, which defines all key terms used in the argumentation (e.g., local robustness, l_2 local robustness, etc.). Only the part of the V&V plan dealing with specific solutions which require

[6]In practice, the V&V activities that produce the evidences required by all the branches of the AC are added a priori to the workflow and simply selected depending on the choice made in the AC

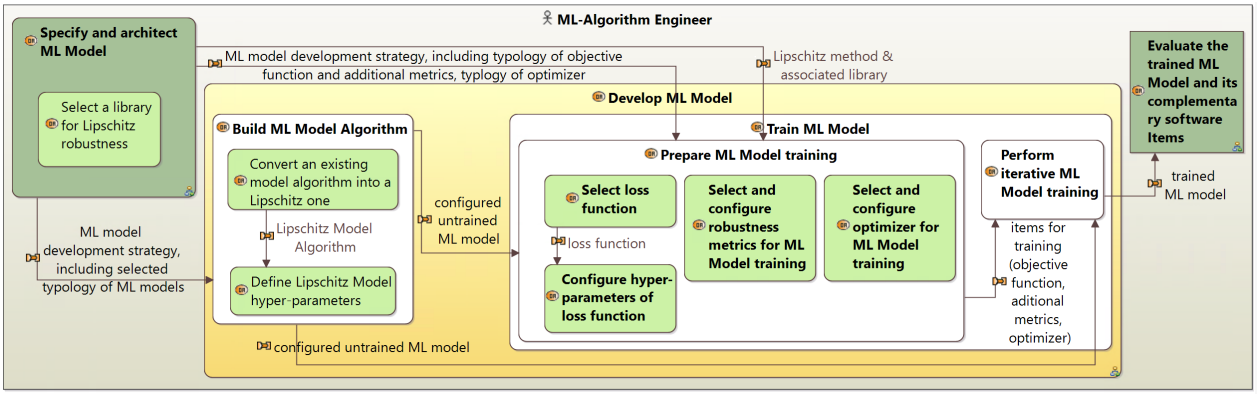

Fig. 10. Example of enrichment of the generic workflow of Figure 2 with the new activities (in light green) required by a specific branch of the robustness assurance case (robustness *by design* using Lipschitz training).

precise knowledge of the methods to be used and how to implement them needs to be manually produced by the experts of the method. Figure 11 shows an extract from such a plan, related to the robustness property, for the implementation of a Lipschitz network.

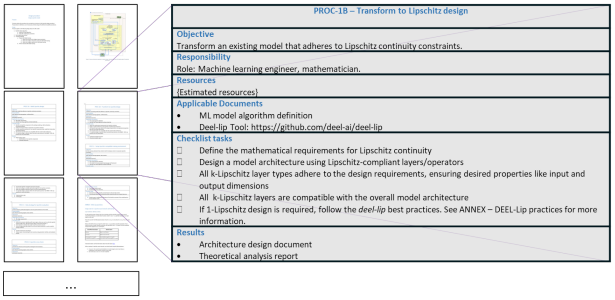

Fig. 11. Extract from the V&V plan related to robustness property

VIII. Related works

This section focuses on existing works that applied ACs to ML-based systems. Indeed, as mentioned previously, Assurance Cases appear as relevant tools for addressing certification-related issues and ensuring systems safety [28]. Applying this approach to ML-based systems seems like an appropriate solution, as no consensual method exists in this domain and standards are not yet established [29].

A first step in this direction was made by Hawkins *et al.* with AMLAS [30], a general guidance on the use of ACs for ML-based system. Their analysis covers a generic ML development workflow, which they enriched with high-level ACs at each main step of the development.

While this paper is of major interest for the application of ACs to an ML-based workflow, the arguments provided rely heavily on requirements *to be defined* and contain numerous elements (variables) that need *to be instantiated*. This approach is highly generic and handles any type of workflow, as the AC's solutions are mainly assessments of the satisfaction of the requirements. Yet, for this reason, the decomposition of main goals remains limited to a few steps, since refining argumentation until concrete evidence needs requirements to be defined and variables to be instanciated. Nevertheless, this paper provides a solid foundation for developing ACs further. In particular, this approach is complementary to ours: it could be used as a common high-level argument for linking all the main properties of interest of ML-based systems such as robustness, fairness or explainability presented in our work. This would provide additional steps of decomposition in the assurance case before reaching the step where the requirements need to be instantiated.

In addition to this general guidance, a few recent studies provide argumentation patterns applicable to ML-based systems. These generic solutions to common problems are intended to be reused and instantiated in various contexts for similar problems. However, they often highlight the fact that a concrete operational context or risk analysis is needed in order to go down in the argumentation until reaching final evidence.

For instance, the work of Picardi *et al.* [31] defines ACs for the deployment of ML-based systems in a medical context. Their approach focuses on the interpretability of the system's outputs, going beyond conventional performance measurements. They propose an argumentation pattern that encompasses the entire context of the ML component, from datasets to model

architecture. This pattern also covers domain-specific contexts, such as information about experts involved and the technical tasks they realise. In [32], they refine their generic pattern with a more precise taxonomy and focus on providing additional guarantees on the confidence in the Trained ML Model (called "Machine Learning Learnt" in their paper) and on the confidence in the data. However, these new arguments are composed of a single decomposition step and thus remain particularly generic and high-level. Finally, the authors extend their work in [33] to reflect better the relationship between the ML models and the safety of the system. In the process, they also develop an assurance process for the engineering of ML components built upon existing best practices. This process facilitates the instantiation of the confidence argument patterns through consideration of the required activities to be undertaken and the artefacts to be generated at each stage in the ML lifecycle. They also propose a generic way of decomposing ML requirements ("desideratas") into property-specific arguments such as model performance, model robustness and model interpretability. However, these properties are only decomposed one step further, and their subgoals remain "to be developed", which contrasts with our approach that proposes a multi-step decomposition down to concrete evidence.

Although previous literature provides a significant foundation in the development of assurance cases for ML-based systems, its traceability with respect to the ML development workflow is still exploratory, and none of them provide the associated tooling support.

In the automotive domain, Bloomfield *et al.* [34] propose an AC template for an experimental autonomous vehicle and its social context. The decomposition of their argumentation follows a top-down approach, from the system to its components, including the ML model. To ensure the trustworthiness of their system, their systematic approach explicitly considers sources of doubt and vulnerabilities in the system's behaviour. To this aim, their pattern is designed to identify gaps and challenges during the justification of system behaviours, as well as gaps within the assurance framework itself. This approach is complementary to ours and could be used to identify new gaps and properties on specific engineering items of the complete workflow of an ML-based system, providing a starting point for developing new arguments.

Among the most recent studies, Dong *et al.* [35] present a specific 'end-to-end' AC applied to an ML model. First, they introduce a framework called the Reliability Assessment Model (RAM) that assesses the reliability of a classification model, covering both its robustness and its operational profile. Then, all evidence produced by the RAM is represented with an AC that tackles the argumentation from a probabilistic point of view and ends with quantitative evidence. This rigorous approach provides a complete, vertical argumentation for an ML property. While we share the authors' argumentation approach, our work differs in two key aspects. First, we do not focus on a specific property including all the mathematical formalism. Our focus is set on an end-to-end assurance case with a model-based formalism. In other words, an assurance case whose evidence and contextual information are mapped onto the engineering workflow. Each necessary element of the assurance case considers a pre-established workflow of activities alongside the injection of V&V activities to be performed. Second, our proposal develops assurance cases horizontally, while they include the selection of multiple sub-arguments for the user, which allows them to select different demonstration approaches.

IX. Conclusion

Throughout this paper, we presented a tool-supported process that enables the systematic derivation of V&V plans, specifically tailored to address the unique challenges posed by the introduction of ML-based systems in critical domains. We addressed these challenges across the entire lifecycle of AI-based systems, integrating our approach into an engineering workflow specific to such systems. This workflow constitutes a pivotal resource for identifying relevant properties and integrating the V&V activities required by our assurance case patterns. Indeed, our contributions extend beyond the mere formulation of a process and offer a set of argumentation templates, focusing on key properties of these systems, such as *robustness*, *explainability* and *fairness*. Nevertheless, combining these arguments under the same overarching property is an open problem. A first step in this direction is provided with the AMLAS [30], which can be used as a common high-level argument, but conflicts between branches may still arise. However, while additional research is required to explore this subject, our tooling and the choice mechanic offer a partial solution to this problem since the branches are structurally removed from the argument, and conflicts will thus appear in the form of non-supported goals during the instantiation of the assurance case.

Acknowledgments

This work has been supported by the French government under the "France 2030" program, as part of the SystemX Technological Research Institute.

REFERENCES

[1] National Research Council et al. *Software for dependable systems: Sufficient evidence?* National Academies Press, 2007.

[2] Tim Kelly and Rob Weaver. The goal structuring notation–a safety argument notation. In *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*, volume 6. Citeseer Princeton, NJ, 2004.

[3] S Gan and JA Ryan. Claims arguments evidence. www.adelard.com/asce/, 2019.

[4] Object Management Group (OMG). Structured assurance case metamodel (sacm). www.omg.org/spec/SACM/, 2013.

[5] George M Cleland, Mark-Alexander Sujan, Ibrahim Habli, and John Medhurst. *Evidence: using safety cases in industry and healthcare*. The Health Foundation, 2012.

[6] Systems and software engineering - Systems and software assurance - Part 2: Assurance case. Standard, International Organization for Standardization, Geneva, CH, 2022.

[7] R. Palin, D. Ward, I. Habli, and R. Rivett. ISO 26262 safety cases: compliance and assurance. In *6th IET International Conference on System Safety 2011*, pages B12–B12, Birmingham, UK, 2011. IET.

[8] Rasmus Adler and Michael Klaes. Assurance cases as foundation stone for auditing ai-enabled and autonomous systems: Workshop results and political recommendations for action from the examai project, 2022.

[9] Pascal Roques. MBSE with the ARCADIA Method and the Capella Tool. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, January 2016.

[10] Afef Awadid., Boris Robert., and Benoît Langlois. Mbse to support engineering of trustworthy ai-based critical systems. In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering - MODELSWARD*, pages 280–287. INSTICC, SciTePress, 2024.

[11] Juliette Mattioli, Agnes Delaborde, Souhaiel Khalfaoui, Freddy Lecue, Henri Sohier, and Frédéric Jurie. Empowering the trustworthiness of ml-based critical systems through engineering activities. *arXiv preprint arXiv:2209.15438*, 2022.

[12] Rob Ashmore, Radu Calinescu, and Colin Paterson. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys (CSUR)*, 54(5):1–39, 2021.

[13] Wulf Loh, Andreas Hauschke, Michael Puntschuh, and Sebastian Hallensleben. Vde spec 90012 v1.0 - vcio based description of systems for ai trustworthiness characterisation. 04 2022.

[14] B Caroline, B Christian, B Stephan, B Luis, D Giuseppe, E Damiani, H Sven, L Caroline, M Jochen, Duy Cu Nguyen, et al. Securing machine learning algorithms. 2021.

[15] European Commission. Laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts, 2021.

[16] EAT Force and A Daedalean. Concepts of design assurance for neural networks (codann) ii. In *Concepts of Design Assurance for Neural Networks (CoDANN)*. EASA, 2021.

[17] Philip Koopman and Frank Fratrik. How many operational design domains, objects, and events? *Safeai@ aaai*, 4, 2019.

[18] Krzysztof Czarnecki. Operational design domain for automated driving systems. *Taxonomy of Basic Terms ", Waterloo Intelligent Systems Engineering (WISE) Lab, University of Waterloo, Canada*, 2018.

[19] The Assurance Case Working Group (ACWG). Gsn community standard v3, May 2021. Accessed on March. 21, 2024.

[20] Ewen Denney, Ganesh Pai, and Ibrahim Habli. Towards measurement of confidence in safety cases. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 380–383. IEEE, 2011.

[21] Rui Wang, Jérémie Guiochet, Gilles Motet, and Walter Schön. Safety case confidence propagation based on dempster–shafer theory. *International Journal of Approximate Reasoning*, 107:46–64, 2019.

[22] Yassir Idmessaoud, Didier Dubois, and Jérémie Guiochet. Confidence assessment in safety argument structure-quantitative vs. qualitative approaches. *International Journal of Approximate Reasoning*, 165:109100, 2024.

[23] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 1 edition, 2005.

[24] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. 01 1990.

[25] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. In *11th International Software Product Line Conference (SPLC 2007)*, pages 233–242, 2007.

[26] Christian Kästner, Sven Apel, and Martin Kuhlemann. Granularity in software product lines. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, page 311–320, New York, NY, USA, 2008. Association for Computing Machinery.

[27] Krzysztof Czarnecki and Michał Antkiewicz. Mapping features to models: a template approach based on superimposed variants. In *Proceedings of the 4th International Conference on Generative Programming and Component Engineering*, GPCE'05, page 422–437, Berlin, Heidelberg, 2005. Springer-Verlag.

[28] John Rushby. The interpretation and evaluation of assurance cases. *Comp. Science Laboratory, SRI International, Tech. Rep. SRI-CSL-15-01*, 2015.

[29] Fateh et al. Kaakai. Toward a machine learning development lifecycle for product certification and approval in aviation. *SAE International Journal of Aerospace*, 15(01-15-02-0009), 2022.

[30] Richard Hawkins, Colin Paterson, Chiara Picardi, Yan Jia, Radu Calinescu, and Ibrahim Habli. Guidance on the assurance of machine learning in autonomous systems (AMLAS). *CoRR*, abs/2102.01564, 2021.

[31] Chiara Picardi and Ibrahim Habli. Perspectives on Assurance Case Development for Retinal Disease Diagnosis Using Deep Learning. In David Riaño, Szymon Wilk, and Annette ten Teije, editors, *Artificial Intelligence in Medicine*, pages 365–370, Cham, 2019. Springer International Publishing.

[32] Chiara Picardi, Richard David Hawkins, Colin Paterson, and Ibrahim Habli. A pattern for arguing the assurance of machine learning in medical diagnosis systems. In *38th International Conference on Computer Safety, Reliability and Security – SafeComp 2019*, April 2019.

[33] Chiara Picardi, Colin Paterson, Richard Hawkins, Radu Calinescu, and Ibrahim Habli. Assurance argument patterns and processes for machine learning in safety-related systems. In *Proceedings of the Workshop on Artificial Intelligence Safety, NY, USA, 2020*, volume 2560 of *CEUR Workshop Proceedings*, pages 23–30. CEUR-WS.org, 2020.

[34] R. Bloomfield, H. Khlaaf, P. Ryan Conmy, and G. Fletcher. Disruptive innovations and disruptive assurance: Assuring machine learning and autonomy. *Computer*, 52(9):82–89, 2019.

[35] Yi Dong, Wei Huang, Vibhav Bharti, Victoria Cox, Alec Banks, Sen Wang, Xingyu Zhao, Sven Schewe, and Xiaowei Huang. Reliability Assessment and Safety Arguments for Machine Learning Components in System Assurance. *ACM Transactions on Embedded Computing Systems*, 22(3), May 2023. arXiv:2112.00646 [cs].