EDITOR **PHIL LAPLANTE**
IEEE Fellow;
plaplante@psu.edu

Assuring Safety-Critical Machine Learning-Enabled Systems: Challenges and Promise

Alwyn E. Goodloe, NASA Langley Research Center

We outline how assurance processes work for conventional systems and identify the primary difficulty in applying them to machine learning-enabled systems. We then outline a path forward, identifying where considerable research remains.

Digital Object Identifier 10.1109/MC.2023.3266860
Date of current version: 23 August 2023

Safety-critical systems, such as automobiles and medical devices, are systems whose failure could result in loss of life, significant property damage, and damage to the environment. The grave consequences of failure have compelled industry and regulatory authorities to adopt conservative design approaches and exhaustive verification and validation (V&V) procedures to prevent mishaps. In addition, strict licensing requirements are often placed on human operators of many safety-critical systems. Ultracritical systems, such as civil transport aircraft and nuclear power plants, are safety-critical systems that society deems should never suffer a catastrophic failure during their operating lifetime and whose development is subject to particularly strict regulatory constraints and rigorous operator training requirements. In practice, the V&V of avionics and other ultracritical software systems

relies heavily on traceability to requirements and system predictability. Technological advances, such as the significant progress in machine learning (ML), are enabling the development of increasingly autonomous cyberphysical systems that modify their behavior in response to the external environment and learn from their experiences. ML is being employed to enable autonomous systems that operate in the real world, but many implementations lack the salient features of traceability and predictability. Currently, in civil aviation, nuclear power, and similar highly regulated areas, there is no regulatory guidance for assuring artificial intelligence (AI) and analogous approaches that do not exhibit predictable behavior.

ML is being employed to enable autonomous systems that operate in the real world, but many implementations lack the salient features of traceability and predictability.

AI-enabled systems pose new dangers to public safety, especially when operating in unexpected environments and when encountering unexpected events. While the challenge of safe AI has been acknowledged,[1] engineers and regulators are not going to abandon established safety-engineering approaches that have been proved to yield safe systems until other equally effective, approaches are developed. In some cases, engineers will likely find it necessary to assemble a considerable body of evidence that an AI-enabled ultracritical system is as safe as conventional systems, including their ability to handle off-nominal situations, hence the need to develop new methodologies for developing and assuring ML safety-critical systems. Efforts are underway to write standards and guidelines to govern the use of this technology, such as ANSI/UL 4600, *Standard for Safety for the Evaluation of Autonomous Products*,[2] and Society of Automotive Engineers (SAE) G34, *Artificial Intelligence in Aviation*, but significant technical barriers must be overcome.

In this article, we provide a brief overview of the processes and practices for assuring conventional software-enabled systems, focusing on the domain of civil aviation, which has an exemplary safety record. We discuss the challenges of assuring ML-enabled systems and discuss some of the tools and techniques that are being proposed for this task and their drawbacks. Although it may seem an impossible task to assure ML-enabled safety-critical systems within current assurance frameworks, we discuss how particular classes of problems are within reach, while others are likely to remain basic research challenges for the foreseeable future.

ASSURING CONVENTIONAL SYSTEMS

Years of experience at building safety-critical software systems, such as aircraft and nuclear power systems, have yielded analysis, design, and development practices that have produced extraordinarily safe systems, such as the current air transportation system. The public now demands that technological advances not lower the level of safety it has come to expect. In this section, we give a high-level overview of design and development processes used for civil transport aircraft.

Like all software systems development, engineering safety-critical systems begins with ascertaining requirements, but in addition, there is a need for one or more safety analyses, such as

- a process used to assess risk, such as a hazard analysis for the identification of different types of hazards
- a process to identify potential failure modes in a system and their causes and effects.

In civil aviation, safety analyses are carried out whenever there are changes to the system. The functional requirements and safety analyses together flow into the system specification, system architecture, and design. A functional specification precisely states what the system is to do, usually in terms of a formal relation of system output given a specified input. Given a functional specification of system requirements ϕ and a system S, if ψ is true and we execute S, then S will terminate in a state where ϕ is true. If it is possible to write a functional specification ϕ precisely stating what the system is to do, it is also possible to deduce what constitutes undesired behavior. When we can refine requirements into such specifications, we call them *actionable specifications*.

The safety analysis will determine what faults the system will be expected to sustain and still operate safely. This is called the *fault model* of the system. Faults and failures are often mitigated at the architectural level by employing sufficient redundancy. Engineers have to demonstrate traceability back to the original requirements and safety analyses at each refinement step of the development process. These practices are often codified in guidelines. Software implementations tend to adhere to very conservative guidelines[3] that constrain nondeterminism and ensure bounds on resource consumption by disallowing dynamic memory allocation and recursion. Although these restrictions may constrain the design space, they make the task of assuring such systems tractable.

The assurance processes for safety-critical systems typically use testing to demonstrate that a system meets the requirements, that there is no unintended behavior, and that the system tolerates specified faults. Coverage metrics are used that measure

how well test inputs exercise the code. In particular, they aim to show the degree to which the test inputs execute all branches of the code. In civil aviation, the DO-178C[3] guidelines require that the most critical software must undergo modified condition/decision coverage testing,[4] where

- Each entry and exit point is invoked.
- Each decision takes every possible outcome.
- Each condition in a decision takes every possible outcome.

In addition to testing, formal methods-based tools are increasingly being used for certification credit.

A 10,000-FT VIEW OF ML

When building conventional systems, one refines requirements into an actionable specification that is then refined into program logic and implemented in a programming language. The resulting program consumes data as input as it executes the program logic, which often makes decisions based on that input. In contrast, ML systems are constructed by providing the system with examples that one can construe as a model of what is to be built.

The most popular application of ML is classification, where an ML-constructed model is used to classify input data. For instance, an ML model could classify a collection of pictures of birds by species. Let us consider the process of building a classifier with supervised learning. The first step is to gather a sufficiently large representative set of examples. These examples need to be labeled with the correct classification so as to serve as "ground truth." We can represent the data as a set of pairs $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i$ is an n-dimensional vector and y_i is the label. Ascertaining that the training dataset is sufficiently large and representative to serve as a "complete" set of examples is a considerable engineering challenge.

A deep neural network (DNN) is composed of an input layer, an output layer, and many hidden layers of neurons. Each layer is connected to the previous layer, where each connection has a parameter that is typically a weight on the connection. At each layer, nodes behave like a linear regression model, computing outputs based on weights and a basis. A layer can be thought of as computing a mathematical function l_i, so the NN can be thought of as being a composition of these layers: $\mathcal{N} = l_k \circ \cdots \circ l_1 \circ l_0$. DNNs are often composed of thousands and even millions of nodes arranged in many layers, making them quite opaque.

To train the NN, the parameters are initialized, and then an iterative process is carried out, where at each pass, training data are applied to the DNN and the parameters are updated so as to minimize the difference between the output of the DNN and the ground truth $||\mathcal{N}(\mathbf{x}_i) - y_i||$. In summary, the model is defined by the model parameters that get updated as the system learns and hyperparameters that influence this process. The optimization process is intended to ensure that the system "generalizes" well, that is, providing the right output to input that was not given in the training dataset. The DNN itself approximates a mathematical function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. In the case of a classifier, f maps n-dimensional input to one of m classes. Note that f is a partial function defined on the distribution of the training data and undefined off the distribution.

ML is often advertised as the approach to use when you do not know how to specify the system you want to build. Industrial use of ML spans almost every domain, from advertising to finance to agriculture. The more cautious safety-critical systems domain has been slower to adapt this technology, especially those areas subject to strict regulatory oversight. As we have seen in the preceding section, safety-critical systems are typically built using a requirements-driven methodology that demands having requirements that can be refined into actionable specifications. Yet, the desire to build autonomous systems that need functionality such as perception, for which we currently do not know how to write actionable specifications, has driven engineers to use ML, as it is currently the option with the best performance. The challenge in adapting the conventional assurance approaches is that, other than large datasets of examples, what constitutes a specification? Consider an ML-based classifier for pictures of birds. There may be many gigabytes of examples, but what exactly would constitute a specification of a hummingbird or a cardinal?

ML is often advertised as the approach to use when you do not know how to specify the system you want to build.

On the other hand, there are use cases where ML is used to replace conventionally built applications because it exhibits superior performance characteristics. An example of such an application is an ML-enabled aircraft fuel management system. Such systems might use significantly fewer computational resources than conventional solutions while better optimizing fuel use. We know how to write specifications for such systems because we already do it.

In addition to functional correctness, we often speak of the software safety properties defined as "something bad does not happen." Typical traditional safety properties are floating-point arithmetic overflows and buffer overflows. ML has its own basket of safety properties. For instance, NN-based perception classifiers have shown themselves to be sensitive to small changes in an image that may not even be recognizable to humans.[5] This

phenomenon is called *adversarial attack*, and systems that exhibit adversarial robustness do not exhibit such sensitive behavior. One of the more popular formulations of adversarial robustness[6] follows. Suppose the DNN $\mathcal{N}$ is a classifier. An NN is said to be δ locally robust at input x_0 if small perturbations do not change the classification:

$$\forall x. ||x - x_0|| \leq \delta \Rightarrow N(x) = N(x_0).$$

Such safety properties are actionable specifications and are attractive because they are amenable to detection by automated tools. One of the reasons adversarial robustness has attracted so much attention is that it is one of the few such properties for which we have actionable specifications.

ASSURANCE APPROACHES

A number of approaches have been proposed for assurance of ML-enabled systems. We briefly survey five of these and assess their strengths and weaknesses.

Testing

Testing is the most well-established approach used in assuring systems, and it definitely plays a role in assuring ML-enabled systems, but there are challenges. Given that the specification for an ML system is captured in the example datasets, it is very difficult to create test oracles.[7] Typically, a set of examples is held back from the training set and later used to test the performance of the ML system. There are numerous efforts to apply techniques that have been successful at testing conventional software, but their efficacy is still being evaluated. Given that the model that is being tested approximates a partial function defined only on the distribution of the training data, rather than random testing, test inputs should be on the distribution.[8] As we have noted, the dataset used to train and test does not really correspond to the traditional notion of an actionable requirement; thus, it is difficult to argue that testing based on such a dataset corresponds to requirements-based testing performed on safety-critical systems.

It is difficult to see how coverage metrics used in conventional software are easily transferred to this setting. A common coverage criterion for NNs is that the test inputs are selected to ensure that all neurons are activated. The branching in NN implementations is not very sophisticated, so achieving such coverage is not difficult but not very meaningful either.[9]

Lacking actionable specifications and good coverage metrics, it is not possible to create tests for assuring that a system performs its intended function and that there is no unintended behavior. Thus, it is not possible to perform the kind of requirements-based test-driven assurance described previously. Discovering a new testing approach that provides the same level of confidence in assuring the system remains the subject of research.

Formal methods

The application of techniques from the formal methods community to the verification of ML-enabled systems is an active area of research. Consider an NN that implements a function $y = f(x)$ for a bounded input domain $\mathcal{D}$. Given a correctness property $\phi(x, y)$, the goal is to show

$$\forall x \in \mathcal{D}. y = f(x) \Rightarrow \phi(x, y).$$

Rather than a direct proof, the problem can be reformulated as a constraint problem. One approach is to recast the problem to be resolved by a satisfiability modulo theories solver. The Marabou[10] tool has pioneered this means of verification.

Abstract interpretation is a static analysis technique that computes a sound and conservative overapproximation of a program by relating the concrete states of a program to a more tractable abstract set of states and then automatically proving that the abstract program satisfies a given safety property. Researchers at ETH Zurich have been investigating how abstract interpretation can be applied to verifying NNs.[11] In this work, NNs are represented as affine transformations guarded by logical constraints. Abstract interpretation tools have been applied to verify adversarial robustness.

These techniques work very well on small examples, but getting any of these approaches to scale remains a difficult problem. As with testing, the biggest challenge is the need for actionable specifications. A huge dataset does not in and of itself constitute a property that we can prove.

Runtime verification

Runtime verification (RV) is a verification technique that has the potential to enable the safe operation of safety-critical systems that are too complex to formally verify and fully test. In RV, the system is monitored during execution to detect and respond to property violations that take place during the actual mission. The Copilot RV framework,[12] developed by the author and his colleagues at NASA, targets the RV of safety-critical systems, with a strong emphasis on certification.[13] Due to the probabilistic behavior of ML, RV can help ensure that input that has never been seen does not result in unsafe behavior.

Just as with testing and formal methods, there must be actionable specifications to check. For instance, an ML-enabled autopilot may have a safety property saying it must stay within a well-defined geofence, and this can be checked at runtime, but it is not possible to check that a classifier has properly detected a Persian cat or a bluebird.

Explainability

Explainability of ML is often touted as the missing piece complementing other assurance practices by providing confidence that the system is operating as intended or at least in a safe fashion. Complicating this argument is the fact that explainability means different things to different people, meaning you always have to ask, Explain what and to whom?

The "black box," common in aviation, is a well-established engineering artifact allowing experts to determine the cause of accidents and incidents after the fact. Given that ML may not always react to new situations in predictable ways and, in the worst case, can endanger the public safety, a similar recording device that provides engineers with enough visibility to ascertain why, given certain inputs, the ML system behaved the way it did, can provide valuable forensic evidence when an accident or incident occurs.[14] Although deploying such a black box is sound engineering practice and the data could be used to improve the system performance, this notion of explainability does not really improve the assurance processes.

A second notion of explainability exposes technical details to developers for the purpose of debugging. The internal operation as well as details about input data are presented to developers with expert knowledge during testing to help them understand why the system may not be performing as expected. While this helps the developers improve the quality of the product, it is not clear how it impacts assurance.

Another concept of explainability is targeted at users. This notion of explainability aims to tell users what is going on, using language they can understand. The idea is to improve trust in ML-enabled systems, but it is well known that trust is often misplaced. For safety-critical systems, it is more important for the system to be trustworthy. Work in this area remains in the realm of basic research.

Licensing

In conventional safety-critical systems, there is usually a clear dividing line between automation and human operator. The computing hardware and software undergo certification, while the human operator is required to be licensed. ML is often employed in autonomous systems to replace functions that have traditionally been carried out by humans. There have been a number of proposals to license the ML-enabled components of a system that are replacing a human in an autonomous system. The idea seems reasonable, as there is often a set of well-documented skills and procedures that are expected to be mastered and demonstrated in the licensing exam. Yet, in addition, there are often minimum age requirements intended to ensure a level of maturity. More research is needed to understand what life experiences contribute to being "mature enough" and how they factor into handling off-nominal situations. There are often mandatory apprenticeship and mentoring requirements that can sometimes last years, with candidates who cannot demonstrate an ability to handle themselves in off-nominal situations washing out of the program. We do not have a good understanding of the role that apprenticeship, mentor evaluation, and maturity often play in the licensing process. Although a very valuable subject for research, a number of complex questions need to be sufficiently addressed before licensing AI safety could play the same role it does in licensing humans.

A PATH FORWARD

We have established that the key feature enabling the assurance of safety-critical systems is possessing actionable specifications. It is critical not to abandon this pillar of assuring safe systems for the sake of expediency. Instead, we should focus on building those systems for which we possess actionable specifications.

Within the domains of cyberphysical systems and aerospace, in particular, there is a significant number of problems where ML can be applied to applications possessing actionable specifications.[15] One instance is a battery management system. We know how to construct battery management systems via conventional means and hence can construct actionable specifications that can be used to construct test oracles and properties to check at runtime. In some cases, we may not know how to obtain an actionable specification of an ML component itself, but we can formulate an actionable specification to ensure safe operation based on other components of the system.[15]

In autonomous systems, the most compelling use cases for ML are areas such as perception, where there are no effective conventional solutions and where the only form of specification is a large dataset. Although we currently do not know how to extract an actionable specification from such a dataset, there is promising research. Mathematicians working in the area of topological data analysis (TDA)[16,17,18] are bringing to bear powerful techniques from algebraic and differential topology as well as differential and algebraic geometry that allow us to compute geometric and topological invariants on the data. At a basic level, the mathematical field of topology is the study of properties of a geometric object that are invariant under continuous deformation. TDA typically focuses on the properties of the manifold where data resides. A large dataset may have holes and loops that can be classified using computational tools developed in an area of topology called *homology*. Persistent homology and distributed persistent homology[19] have emerged as techniques that help distinguish between actual features of high-dimensional datasets from noise in the data. Although the field is relatively new, the techniques have provided valuable insight into the differences between deep and shallow NNs.[20] Given that in ML, the data are the algorithm, it would seem that data invariants should be respected by the implementation and thus constitute actionable specifications that can be checked by testing, formal methods, and at runtime. TDA researchers are developing novel techniques for proving that a learned model does indeed satisfy these topological invariants. TDA is promising and exciting basic research being supported by NASA and other organizations, and it will require some time to achieve a technical readiness level to transfer to industrial practice, but at present, it appears to be our best hope for obtaining actionable specifications from large datasets.

We have discussed how safety-critical systems are designed and assured to ensure the protection of the public and seen how ML poses a challenge to traditional design methodology. We have shown that when there are actionable specifications, it is possible to adapt established approaches to assure these systems. On the other hand, domains for which the only specification is a large dataset remain a challenge. While TDA shows promise to produce actionable specifications from large datasets, much basic research remains to be done. While there is always a temptation on the part of leaders to demand breakthroughs based on a schedule, this is one of those cases where the processes must run their course.

ACKNOWLEDGMENT

The author benefited from valued insights of many researchers, but Darren Cofer, Philip Koopman, Taylor Johnson, Ufuk Topcu, Matthew B. Dwyer, Paul Bendich, Abraham Smith, Carl A. Gunter, members of SAE Working Group 34, and fellow researchers in NASA Langley's Safety-Critical Avionics Systems Branch were all specifically helpful in developing the ideas in this article. The work was supported by NASA's System-Wide Safety project in NASA's Airspace Operations and Safety Program.

REFERENCES

1. J. M. Wing, "Trustworthy AI," *Commun. ACM*, vol. 64, no. 10, pp. 64–71, Oct. 2021, doi: 10.1145/3448248.
2. *Standard for Safety for the Evaluation of Autonomous Products*, UL Standard ANSI/UL 4600. [Online]. Available: https://ulse.org/UL4600
3. *Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Inc., Washington, DC, USA, RCTA/DO-178C, 2011.
4. K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson, "A practical tutorial on modified condition/decision coverage," Nat. Aeronaut. Space Admin., Washington, DC, USA, Tech. Rep. NASA/TM-2001-210876, 2001.
5. I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015, *arXiv:1412.657*.
6. G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Towards proving the adversarial robustness of deep neural networks," in *Proc. 1st Workshop Formal Verification Auton. Veh.*, L. Bulwahn, M. Kamali, and S. Linker, Eds. Turin, Italy: Open Publishing Association, 2017, vol. 257, pp. 19–26, doi: 10.4204/EPTCS.257.3.
7. D. Marijan, A. Gotlieb, and K. A. Mohit, "Challenges of testing machine learning based systems," in *Proc. 1st IEEE Artif. Intell. Testing Conf. (AI Test)*, San Francisco, CA, USA: IEEE, 2019, pp. 101–102, doi: 10.1109/AITest.2019.00010.
8. S. Dola, M. B. Dwyer, and M. L. Soffa, "Distribution-aware testing of neural networks using generative models," in *Proc. 43rd IEEE/ACM Int. Conf. Softw. Eng., Companion (ICSE)*, Madrid, Spain, May 2021, pp. 226–237, doi: 10.1109/ICSE43902.2021.00032.
9. F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, "Is neuron coverage a meaningful measure for testing deep neural networks?" in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 851–862, doi: 10.1145/3368089.3409754.
10. G. Katz et al., "The marabou framework for verification and analysis of deep neural networks," in *Proc. 31st Int. Conf. Comput. Aided Verification*, I. Dillig and S. Tasiran, Eds. New York, NY, USA: Springer International Publishing, 2019, vol. 11561, pp. 443–452.
11. T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI2: Safety and robustness certification of neural networks with abstract interpretation," in *Proc. IEEE Symp. Security Privacy*, 2018, pp. 3–18, doi: 10.1109/SP.2018.00058.
12. I. Perez, F. Dedden, and A. Goodloe, "Copilot 3," NASA Langley Research Center, Nat. Aeronaut. Space Admin., Washington, DC, USA, Tech. Rep. NASA/TM–2020–220587, Apr. 2020.
13. A. Goodloe, "Challenges in high-assurance runtime verification," in *Proc. 7th Int. Symp., Leveraging Appl. Formal Methods*, Corfu, Greece, Oct. 2016, pp. 446–460.
14. G. Falco et al., "Governing AI safety through independent audits," *Nature Mach. Intell.*, vol. 3, no. 7, pp. 566–571, Jul. 2021, doi: 10.1038/s42256-021-00370-7.
15. D. Cofer et al., "Run-time assurance for learning-enabled systems," in *Proc. 12th Int. Symp. NASA Formal Methods*, Berlin, Heidelberg: Springer-Verlag, May 2020, pp. 361–368, doi: 10.1007/978-3-030-55754-6_21.
16. H. Edelsbrunner and J. Harr, *Computational Topology: An Introduction*. Providence, RI, USA: AMS Press, 2010.
17. G. Carlsson, "Topology and data," *Bull. Amer. Math. Soc.*, vol. 46, no. 2, pp. 255–308, Apr. 2009, doi: 10.1090/S0273-0979-09-01249-X.
18. A. Zia, A. Khamis, J. Nichols, Z. Hayder, V. Rolland, and L. Petersson, "Topological deep learning: A review of an emerging paradigm," 2023. [Online]. Available: https://arxiv.org/abs/2302.03836
19. E. Solomon, A. Wagner, and P. Bendich, "From geometry to topology: Inverse theorems for distributed persistence," in *Proc. 38th Int. Symp. Comput. Geometry*, X. Goaoc and M. Kerber, Eds. Wadern, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, vol. 224, pp. 1–16.
20. G. Naitzat, A. Zhitnikov, and L.-H. Lim, "Topology of deep neural networks," 2020. [Online]. Available: https://arxiv.org/abs/2004.06093

ALWYN E. GOODLOE is a research computer engineer at the NASA Langley Research Center, Hampton, VA 23666 USA. Contact him at a.goodloe@nasa.gov.