Author Pre-print Version. 39th IEEE/AIAA Digital Avionics Systems Conference (DASC 2020).

Assured Integration of Machine Learning-based Autonomy on Aviation Platforms

Erfan Asaadi*, Steven Beland†, Alexander Chen†, Ewen Denney*, Dragos Margineantu†,
Matthew Moser†, Ganesh Pai*, James Paunicka‡, Douglas Stuart‡, Huafeng Yu§
*KBR, Inc., NASA Research Park, Moffett Field, CA 94035, USA
{easaadi, edenney, gpai}@sgt-inc.com
†The Boeing Company, Seattle, WA 98108, USA
{steven.c.beland, alexander.z.chen, dragos.d.margineantu, matthew.a.moser}@boeing.com
‡The Boeing Company, St. Louis, MO 63134, USA
{james.l.paunicka, douglas.a.stuart}@boeing.com
§The Boeing Company, Huntsville, AL 35824, USA
huafeng.yu@boeing.com

***Abstract*—Dynamic assurance cases (DACs) are a novel concept for the provision of assurance—both during development and, subsequently, continuously in operation—that can be usefully applied to machine learning (ML)-based autonomous systems. We describe the application of a DAC for dependability assurance of an aviation system that integrates ML-based perception to provide an autonomous taxiing capability. Specifically, we present how we: *i*) formulate and capture risk-based safety and performance objectives, *ii*) model architectural mechanisms for risk reduction, *iii*) record the rationale that justifies relying upon autonomy, itself underpinned by heterogeneous items of verification and validation evidence, and *iv*) develop and integrate a computable notion of confidence that enables a run-time risk assessment and, in turn, dynamic assurance. We also describe our evaluation efforts, currently based on a hardware-in-the-loop simulator surrogate of an airworthy flight platform.**

***Index Terms*—Assurance, Autonomy, Confidence, Machine learning, Quantification**

I. Introduction

Equipping aircraft with autonomous capabilities, especially those implemented using machine learning (ML) technologies, will greatly enhance the operational effectiveness of un-piloted aircraft and may have applications in piloted aircraft to aid pilot decision making and workload management. KBR is supporting Boeing in their exploration of how ML can bring benefits such as improved mission capability, faster-than-human reaction times, decreased operating cost with a reduction in the need for human interaction during mission operations, and increased safety to aircraft—including application to airplanes, cruise missiles, and smart munitions. The ongoing work described here is part of a wider effort providing important theory, technology, and toolsets that aircraft manufacturers can leverage to support airworthiness certification processes for aircraft with ML-based flight software.

Distribution A. Approved for public release: distribution unlimited. This work was supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under the Assured Autonomy Program. The opinions, findings, recommendations or conclusions expressed are those of the author(s) and do not represent the official views or policies of DARPA, AFRL, the Department of Defense, and the United States Government.

A critical step towards operational deployment is providing assurance of dependable autonomy and, in turn, of an airworthy integrated system. In conventional aircraft systems, i.e., those that do not rely on ML for safety-critical functions, designs are carefully and extensively validated and verified before operation, with equipment issues (such as hardware failures, out-of-range conditions, bounds on sensor inputs, etc.) being monitored in operation to take corrective actions (including reconfiguration, degradation, or shut down) in response to detected unacceptable conditions.

ML is to meant to infer input-output relationships from samples of intended behavior, and then generalize the learned function to unknown and unforeseen inputs. Thus, due to their very nature, not only may it not be possible to design and verify ML-based components in the same was as in the prevailing practice [1], but also there is a credible potential for unexpected responses that detrimentally contribute to unanticipated emergent system behavior. This suggests a need to broaden the scope of in-operation monitoring to address conditions that would otherwise have been addressed during development.

Moreover, safety assurance of conventional aviation systems has been aided by performance standards and assurance guidelines that certification authorities recognize as an acceptable means of showing compliance with aviation regulations. However, these are not yet available for systems embedding ML-based autonomy, although domain-specific standardization is emerging[1] along with candidate standards [2]. A growing consensus in these efforts is to use an *assurance case* (AC) as the mechanism by which to convince various stakeholders, especially regulators, that an autonomous system can be relied upon. An AC is a comprehensive, defensible, and valid justifi-

[1]For example, through the efforts of the SAE G-34 Committee for Applied Artificial Intelligence in Aviation who, together with EUROCAE WG-114, are capturing the specific concerns presented by ML components and will be defining assurance guidelines for such systems. If accepted by regulators, it is envisioned those guidelines would be used as an acceptable means of compliance with applicable regulations. See http://profiles.sae.org/teag34/

cation that a system or service will function as intended for a defined application and operating environment [3]. They have already proven to be useful in application to novel technologies requiring operational approval under nascent regulations and emerging standards, e.g., unmanned aircraft systems (UASs) operating beyond visual range [4].

Dynamic Assurance Cases (DACs)—described in more detail in Section II—generalize and extend the AC concept, providing a framework for both development and operational assurance of autonomous aviation systems (AAS). As a proof-of-concept, in this paper we investigate the viability of the DAC framework by applying it to integrate an ML-based perception function on an aircraft platform, to enable autonomous taxiing (Section III).

Specifically, we describe the DAC we are creating to provide justified confidence that: *i*) a deep *convolutional neural network* (CNN) can enable autonomous runway centerline tracking whilst taxiing, maintaining the aircraft within a required maximum lateral offset from the centerline; and that *ii*) it is feasible to preclude potential lateral runway excursion using a run-time assessment of (sufficient) confidence in the ML-based centerline tracking capability. The former constitutes development assurance, communicated through core, interrelated DAC components (Section II-B), while the latter represents an architectural mechanism for the provision of *dynamic assurance* when integrated into the aircraft system (Section II-B2, Section III-C). Our effort to assess and validate our approach (Section IV) leverages a hardware-in-the-loop (HIL) simulator surrogate of the aircraft. In this paper, we describe how we gathered the data required to develop and evaluate dynamic assurance, also discussing the corresponding results (Section V).

II. Background

A. Terminology

Assurance is the provision of justified confidence that a system, item, or service possesses the required *assurance properties*. An *assurance property* is a logical, possibly probabilistic, characteristic associated with *assurance concerns*, i.e., functional capabilities and dependability attributes. Applying one or more assurance properties to an item gives an *assurance claim* for that item. For instance, in the assurance claim "*the aircraft does not depart the runway pavement during taxiing*", the *aircraft* is the system for which the assurance property of interest (associated with the dependability attribute of *safety*) is "*not departing the runway pavement during taxiing*."

We distinguish *development assurance*, which provides confidence during system development to support the decision to deploy a system into operation, from *operational* or *dynamic* assurance, which is a continuous assessment and provision of confidence under changing system and environment state.

B. Dynamic Assurance Cases

DACs offer a multi-viewpoint model-based framework for communicating various aspects of development and operational assurance of AAS embedding ML-based components.

1) Assurance during System Development: A technical basis for sufficient assurance—in terms of the risks to safety (as well as performance and mission) objectives, and the associated risk mitigation requirements—is given by *assurance policies*.We use tabular models to record the above, also capturing the assumptions made, as well as the mappings to system components and functions from, respectively, the safety objectives, the associated hazards, and the required functional and non-functional guarantees.

An *assurance architecture* shows how system functions and components contribute to risk modification. It can be seen as an architecture-level realization of assurance policies, modeled as event-chain based risk scenarios, that capture the impact of both design-time and operational risk mitigations. For this purpose, we have adopted and extended an earlier notion of safety architecture that composes *barrier models* represented using *bow-tie diagrams* (BTDs) [5].

Assurance rationale exposes the reasoning why it should be concluded that (there is sufficient confidence that) the system has met its safety (or performance, or mission) objectives, as embodied by the requirements captured in assurance policies. This reasoning is underpinned by a compendium of heterogeneous verification and validation (V&V) evidence items, e.g., development artifacts such as the results of mathematical analyses, modeling and simulation, formal verification, testing, etc. We present assurance rationale as *structured arguments* in the *goal structuring notation* (GSN) [6]. GSN is a standardized, graphical notation to present the structure and elements of an assurance argument, namely: claims[2], qualifying contexts and assumptions, reasoning steps (strategies) with which claims are further developed, applicable justifications, and references to evidence items (solutions). For an example, see Fig. 4.

Assurance quantification primarily applies to measurable assurance properties, and it helps to establish a quantitative baseline level of confidence in those properties towards supporting the decision to release a system into service. To characterize confidence in assurance properties, we use uncertainty quantification (UQ) techniques on suitably abstract probabilistic models of the system behavior.

Of the above DAC components, the assurance policies, architecture, and rationale components mainly facilitate a qualitative assessment of the sufficiency of the confidence that a system is fit for operational use, while assurance quantification gives a quantitative basis for that decision. Together, they establish a baseline level of assurance for an AAS that ought not to change in operation, and that stakeholders can inspect, negotiate, and agree upon as acceptable; hence, we refer to them as *static* DAC components (Fig. 1). In this paper, we do not discuss how the qualitative and quantitative assessments of confidence are to be reconciled into an internally consistent and valid determination of sufficient assurance; however, we

[2] Practically, an assurance claim is a requirement captured in assurance policies, for which there is supporting evidence. As such, that requirement is effectively an assurance property that has been allocated to a system or item (see Section II-A).

note that both types of assurance are inevitably required in practice.

2) Assurance during System Operation: *Assurance measures* implement the models used for assurance quantification during development as executable components suitable for deployment during AAS operation. Thus, by using operational data as input, we can use assurance measures to evaluate whether or not the quantitative level of confidence in (certain) assurance properties—that was established as an acceptable baseline during development—continues to be maintained during system operation.

Our concept of dynamic assurance integrates assurance measures into an AAS, so that a run-time confidence assessment is feasible. In turn, that can support decision making and continuous risk mitigation during system operation. Conceptually, the integration architecture is closely related to the *simplex* architecture, and its variants [7]: a decision logic that is typically a form of high-assurance *run-time monitoring* or *run-time verification*, taking system and environment state as input to evaluate whether or not system properties are satisfied, and switching to a high-assurance, low-complexity safety controller in the latter case. In contrast, assurance measures provide an additional level of analytic redundancy, taking *both the inputs and outputs of run-time monitoring* to quantify the *uncertainty* in whether or not system properties are satisfied. Thus, assurance measures can be considered as a form of monitoring albeit with a wider scope, fusing information from a range of sources that include other run-time monitors, whilst providing an assurance forecast.

By separating the decision logic from the inputs used to make decisions (here, the confidence in assurance properties, among others), the integration of assurance measures can provide both a passive run-time risk assessment, as well as active mitigation support, as appropriate. Section III-C gives a concrete architecture in the context of our proof-of-concept example, which will further illustrate the above concepts.

C. Methodology

Fig. 1 shows the static and dynamic DAC components from the preceding discussion and a methodology for their use for dependability assurance of AAS. Collectively, the static components constitute an assurance toolkit. As such, we can use some or all of them to provide confidence in the concrete system-specific DAC components themselves, e.g., giving explicit rationale to justify the otherwise implicit assumptions in the particular assurance architecture for a system. Thus, as we will see subsequently, we use the static DAC components to provide confidence in both the quantification models and the assurance measures derived from them.

First, we develop the static DAC components focusing on the system requiring assurance, itself characterized by the physical and logical system descriptions (including the ML components), and the concept of operations. Our methodology is compatible with prevailing standard processes, practices, and techniques in aircraft development [1], and safety assessment [8]. Moreover, to facilitate easier adoption into practice,

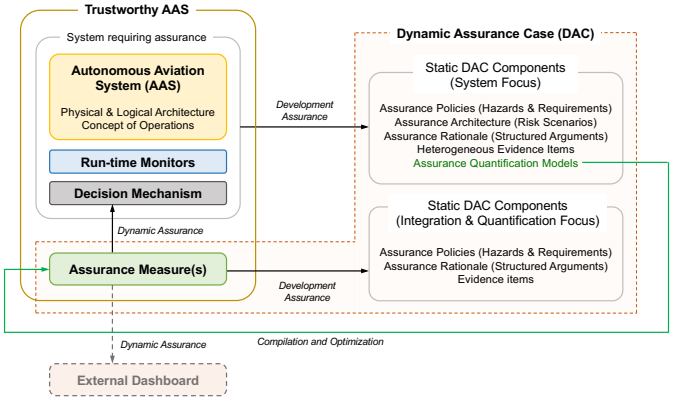

Fig. 1. DAC methodology and concept: development assurance focuses on both the AAS and (the integration and quantification aspects of) assurance measures. Dynamic assurance entails compiling and deploying quantification models for a run-time assessment of confidence in assurance properties, which is then input to a decision mechanism, or optionally displayed externally (as shown by the dashed arrow and box labelled "External Dashboard").

some DAC components intentionally capture information that overlaps with the results of applying those techniques.

Thus, for example, in formulating assurance policies at an aircraft function level, we can apply a functional hazard analysis (FHA), while guide-word based deviation analyses applied systematically to the inputs and outputs of specific ML-based functions—as in methods such as systems theoretic process analysis (STPA) [9]—can provide component-focused assurance policies. Likewise, a fault tree analysis (FTA) can assist in developing an assurance architecture as it elaborates specific failure scenarios, while applying STPA can identify scenarios leading to unsafe control actions. Following the safety assessment steps in [8] aids to iteratively refine the assurance policies and architecture. During these steps, we additionally formulate evidence requirements that specify, for example, the techniques to be used to validate the identified safety requirements and to verify the implementation against those requirements. This is similar, for example, to the type of requirements that result from the interaction between a traditional aircraft development assurance process [1], and its safety assessment process [8].

We then develop structured arguments, e.g., by following the steps in [3], to provide the rationale why the assurance claims are themselves valid, why the results of verification substantiate those claims and, thereby, the requirements stated in assurance policies (See footnote 2).

For assurance quantification during development, in general we use stochastic process models whose underlying random variables (RVs) describe the system state space. The assurance properties of interest are specific realizations of those RVs. We express the uncertainty (conversely, confidence) in those properties as probability distributions over those RVs which we can later update using Bayesian approaches, when observations of the state space are available.

Together with assurance policies, quantification models help to identify and discriminate between properties for which

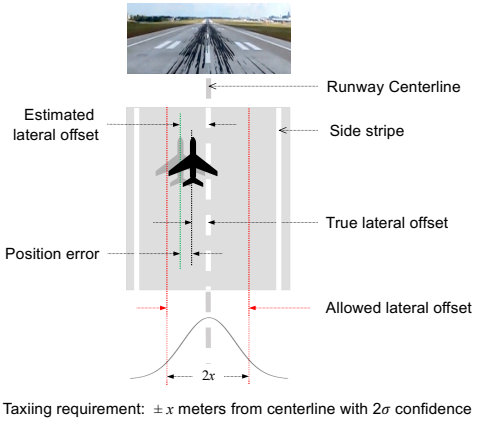

Fig. 2. Concept of operations for autonomous taxiing: the aircraft must track a runway centerline for the duration of taxiing, not exceeding a defined lateral offset of $x = 2\text{m}$ on either side of the centerline.

run-time monitoring may be sufficient, and those for which assurance measures are more appropriate. For instance, monitoring of assumptions made in verifying certain component-level properties may be sufficient, while assurance measures may be better suited to system-level properties affected by emergent behavior. In general, we assume that there exist run-time monitors, some of which are part of the system requiring assurance, while others are tied to the validity of the evidence items used.

Earlier (Section II-B2), we discussed how assurance measures facilitate dynamic assurance. In addition to supporting decision-making on board the flight platform of an AAS, we can also passively provide a real-time confidence assessment via dashboard-based visualizations (see Section IV for an example) for those scenarios where remote intervention, possibly involving humans, may be necessary. Integrating assurance measures necessitates additional confidence in both the measures being fit for purpose and the safety of integration.

As discussed at the beginning of this section, we develop additional static DAC artifacts to address properties such as assurance measure timeliness in the context of decision-making, forecast accuracy, and the absence of unintended functionality and hazardous interactions.

III. Example: Autonomous Taxiing

A. System Description

We consider an un-piloted, single engine, fixed-wing aircraft system implementing an autonomous taxiing capability over taxiways and runways, as an example AAS. Fig. 2 shows one of the scenarios in this operational concept, where the aircraft is to track and follow the runway centerline without exceeding a pre-specified lateral offset on either side. The aircraft is one part of the overall testbed that we use to demonstrate assured ML-based autonomy; the other parts include an *iron bird* facility for real-time HIL testing, and simulation tools. The experimental results reported in this paper are based on Boeing flight software integrated with the assurance measure for the system, executed within the HIL iron bird.

To realize autonomous taxiing, a deep convolutional neural network (CNN) implements a perception function that ingests video images from a wing-mounted camera pointed to the nose of the aircraft. In effect, after being trained under a supervised learning scheme, this ML component performs regression over input images to produce estimates of *cross track error* (CTE) and *heading error* (HE) as output. Here, CTE is the horizontal distance between the aircraft nose wheel and the runway centerline; HE is the angular distance between the *aircraft heading* (the compass direction along the aircraft roll axis) and the *runway heading* (the compass direction of the runway centerline). These are then supplied as inputs to the appropriate controllers that generate the required steering and actuation signals.

An efficient MobileNetV2 deep learning architecture [10] implemented[3] using the state-of-the-art inverted residual blocks forms the base CNN model. For protection from outliers, the output layer is a *soft sign* activation function, selected due to its difficulty to saturate. L1 and L2 weight regularization as well as dropout improve the capability of the model to generalize its results. For training and testing the CNN during development, a classical proportional-integral-derivative (PID) controller abstracts the autonomous executive (AE) and vehicle management system (VMS) of the aircraft platform and its surrogate HIL simulator (see Fig. 7).

B. Development Assurance

Our toolset AdvoCATE [3] provides a model-based implementation of the DAC concept and its static components.[4] We now give illustrative excerpts and fragments of the DAC we are building to provide development assurance for autonomous taxiing.

1) Assurance Policies: Safety during autonomous taxiing can be characterized as avoiding *lateral runway excursions*, i.e., veering off the runway pavement by crossing the side stripes (see Fig. 2). Although avoiding obstacles on the runway is also a safety concern, we do not address it for this paper. A related performance objective is to maintain an acceptable lateral offset (ideally zero) on either side of the runway centerline. Assurance that this performance objective holds also contributes to safety assurance since the closer the aircraft is to the runway centerline during taxiing, the less likely it is to cross the side stripes (equivalently, the more confidence there is that the aircraft is on or near the centerline, the lower the uncertainty that it is located at or past the side stripes).

AdvoCATE provides a tabular assurance policy model, implemented as a collection of hazard and requirements tables. Assurance policies thus include the requirements corresponding to the objectives above. For example, one such requirement is: "*the aircraft CTE shall not exceed a specified offset whilst taxiing*." We model this as the assurance property (see Section II-A) `AssuredTaxi` : $|\text{CTE}_a| < \texttt{offset}$, allocated to the system `Aircraft` that is declared in a physical decomposition

[3] In *Keras* (https://keras.io/) and Tensorflow (https://www.tensorflow.org/).

[4] At present, we develop quantification models using external probabilistic modeling tools. We plan to integrate them into AdvoCATE in the future.

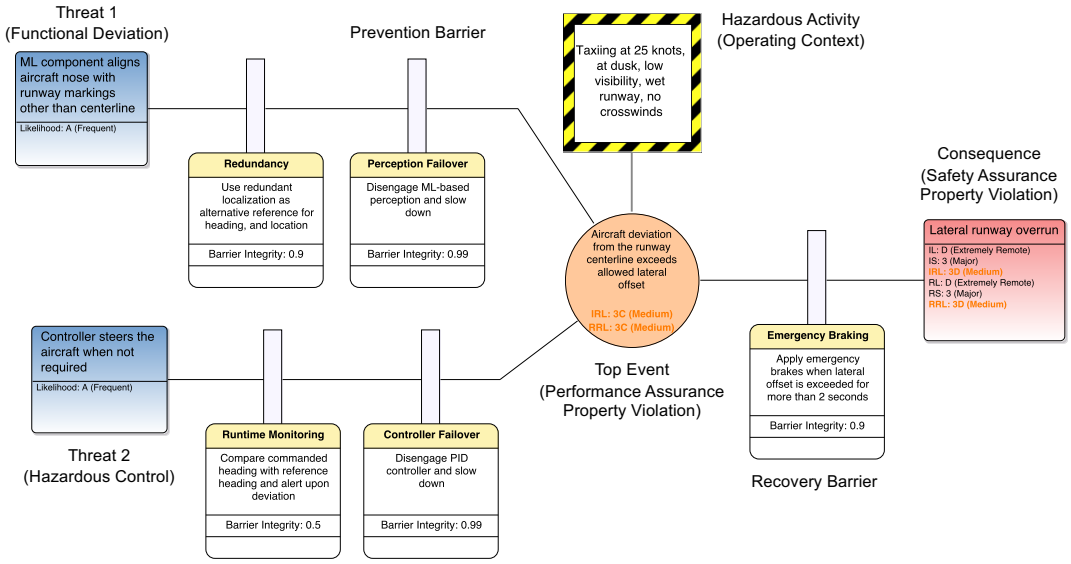

Fig. 3. Annotated BTD fragment, showing one out of a collection of risk scenarios that may be encountered in autonomous taxiing. The composition of different such BTDs gives the assurance architecture DAC component for the system. Annotations highlight BTD notational elements.

model available natively in the tool. Here `offset` is the maximum acceptable lateral offset on either side of the runway centerline, and CTE_a is the true CTE. For this application and aircraft type, $\texttt{offset} = 2\mathrm{m}$, and CTE_a is a signed, real valued scalar whose absolute value gives the magnitude of the offset, and whose sign indicates the location relative to the centerline (i.e., to its left or right).

Assurance policies additionally include hazards that can lead to a violation of the above objectives, their causes, as well as the corresponding assurance requirements, including those for hazard mitigation. For instance, some candidate causes of the hazards to performance and safety include:

- *functional deviations*, e.g., aligning the aircraft nose with a different runway marking instead of the centerline;
- *hazardous control*, e.g., providing a steering actuation in a direction other than as required from the current operating situation and aircraft location; and
- *item malfunctions* or *failure modes*, such as stuck actuators or control surfaces, and sensor failures.

Likewise, precursors of these causes can include *hazard contribution modes* (HCMs) of the ML components [11], e.g., *accurate CTE estimates from runway markings other than the centerline*. To complete the assurance policy specification, we give mitigations to the above conditions, which can include, for example, a combination of *i*) *system-level prevention and recovery devices*, e.g., emergency brakes, or redundant aircraft localization; and *ii*) *design modifications* aimed at improving ML component accuracy, e.g., retraining the ML component using training data augmented with test data from prior or lower-accuracy variants.

2) Assurance Architecture: As previously indicated (Section II-B1), we use bow-tie diagrams (BTDs) to model the scenarios that, when composed, eventually constitute the assurance architecture for the system. Fig. 3 shows a snapshot of a partially developed operational risk scenario, modeled using a BTD. which captures the risk posed to safe autonomous taxiing by a performance property violation. We model the operational context for this scenario as a *hazardous activity*: "taxiing at a speed of 25 knots on a wet runway under low visibility conditions at dusk, when there are no crosswinds."

Here, a violation of the `AssuredTaxi` property (shown by the central, *top* event) can lead to a safety mishap (shown by the terminal, *consequence* event). This scenario shows two potential initiating events (*threats*, in BTD terminology), one of which is a functional deviation (labelled `Threat 1`), while the other is a hazardous control action (labelled `Threat 2`). *Barriers* represent mitigations that modify risk in specific ways on this chain of events: here, a combination of redundancy, failover mechanisms for the perception and control components, runtime monitoring, and emergency braking, invoked to assure that taxiing is both safe and performant. A quantitative *barrier integrity* reflects the extent of risk modification that a barrier provides [5].

This scenario is partially developed because we can further refine it based on the assessment of residual risk levels (RRLs) of the relevant top events and consequences. For instance, if the RRL is at an unacceptable level, additional risk mitigations may be required. Although not shown here, potential causes of the two threats given in Fig. 3 are the HCMs of the deep CNN. For instance, a CTE estimate due to incorrectly perceiving a non-centerline runway marking as the centerline, can potentially produce a steering action that deviates the aircraft away from the centerline when it should otherwise have maintained the prevailing heading, i.e., `Threat 2`. We can develop this model and refine the associated scenario-driven analysis not only at the level of the system (as discussed here), but also at a lower, component-level: e.g., by modeling the *escalations* that can compromise or render ineffective each

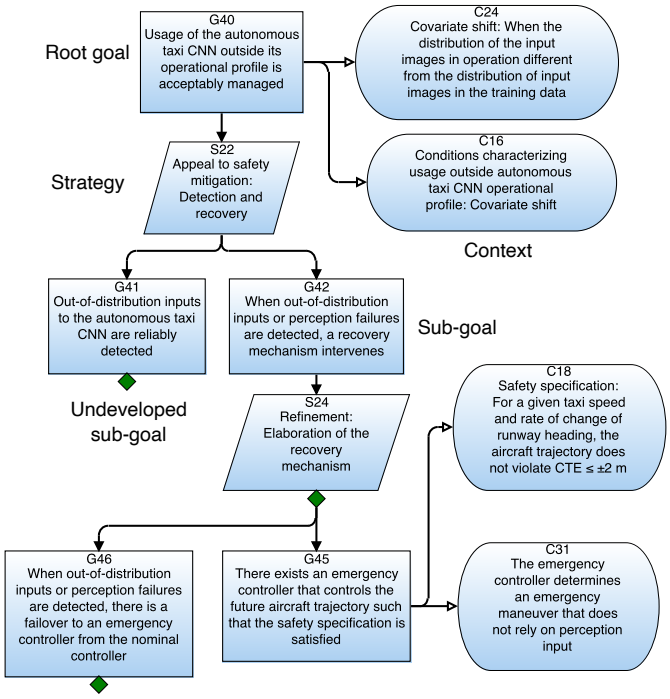

Fig. 4. Fragment of assurance rationale captured as an argument in GSN; Text annotations outside the corresponding node types indicate the notational elements. This argument supports the claim (G40) that operating the autonomous taxi deep CNN outside its operational profile is acceptably managed, based on an emergency maneuver that can be synthesized (G45, further developed in Fig. 5), and whose implementation safely brings the aircraft to a halt (G46).

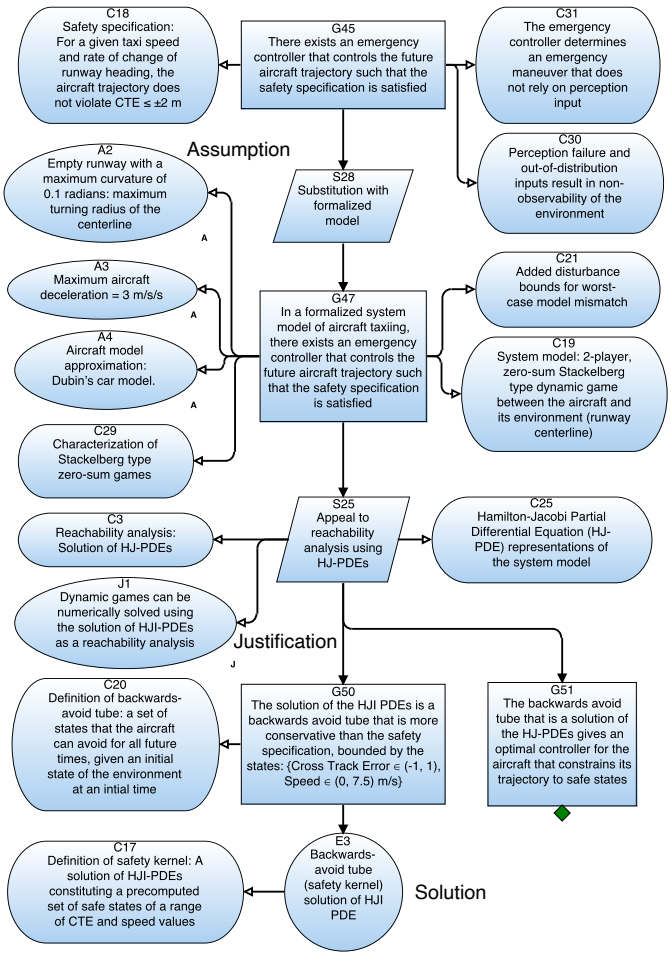

Fig. 5. GSN argument further developing the claim (G45, also the leaf claim in Fig. 4) that there exists an emergency controller that constrains the aircraft trajectory to meet the safety specification, shown by the solution (E3) of a specific form of reachability analysis (S25, G49). Again, text annotations outside the corresponding node types indicate additional notational elements.

of the identified barriers, and identifying the *escalation factor barriers* that mitigate those lower-level threats. In this way, the assurance architecture can relate system-level operational assurance concerns to component-level mechanisms that provide assurance both during development and in operation.

3) Assurance Rationale and Evidence Items: As explained in Section II-B1, we use the GSN for rationale capture. Annotations in Fig. 4 and Fig. 5 show some of the notational elements used, e.g., *goal* nodes (G40, G41, ...) specify the assurance claims, and *strategy* nodes (S22, S24, ...) give the reasoning steps being used. Besides the specific types of nodes shown, the GSN provides two types of relations between nodes: inferential support via *is supported by* links (shown with filled arrowheads), and contextual clarification using *in context of* links (shown with open arrowheads). More details on GSN are in [6].

Fig. 4 presents an argument fragment embodying the rationale to be confident that it may be acceptable for the CNN to be used outside its operational profile during autonomous taxiing. Specifically, the reasoning being conveyed here is that when so-called out-of-distribution (OOD) inputs are supplied to the CNN component, they are reliably detected (goal G41), and that a recovery mechanism intervenes (goal G42). G41 is an *undeveloped goal*, meaning that it is yet to be supported by evidence (indicated by the diamond decoration on the node).

To show G42, the argument proceeds by refinement to claim: *i*) in G45, that an emergency controller exists that constrains the future aircraft trajectory to meet the safety specification (clarified in context node C18); and *ii*) in G46, that the emergency controller to which the claim in G45 alludes is a failover mechanism for a nominal controller. That is, when perception fails or when there are OOD inputs, the system switches from a nominal controller (that is both optimal and uses perception) to a safe emergency controller that is optimal but does not require perception.

The argument supporting G46 is yet to be developed, while the claim in goal G45 (further developed in the argument shown in Fig. 5) is supported by evidence (solution node E3) produced by applying a particular type of formal reachability analysis[5] (strategy node S25) that uses the solution of Hamilton-Jacobi partial differential equations (HJ-PDEs) [12] (goal node G50)—itself a transformation of a game-theoretic model of the system (context node C19)—which facilitate a numerical analysis (justification node J1). This evidence

[5]The authors of [12] performed this analysis.

is a *safety kernel* that is more conservative than the safety specification.

The root assurance claim in Fig. 4 is, in fact, related to component-level causal factors of some of the hazards to assured performance and safety. In particular, the argument documents how assurance is provided against *misperception*, a class of HCMs of the CNN component that can produce inaccurate outputs when processing OOD inputs. The latter is a manifestation of *covariate shift*, the phenomenon where the distribution of operational inputs to an ML component differs from that of its training data.

At a system level, misperception can lead to inadequate controller responses, potentially unsafe control (e.g., as discussed in Section III-B2) and, eventually, to a violation of performance and safety assurance properties. Although not shown here, we capture the rationale that these (and other) risk scenarios have been mitigated through argument structures similar to those shown in Fig. 4 and Fig. 5. Thus, those arguments would show how the root goal in Fig. 4 (G40) emerges from a systematic decomposition and refinement of the system- and component-level assurance claims (themselves, captured in the assurance policies DAC component). Moreover, arguments additionally capture the rationale for the sufficiency and efficacy of *i*) the mitigations represented in the assurance architecture DAC component, *ii*) the assurance architecture itself, *iii*) the quantification model (described next), and *iv*) the corresponding assurance measure (Section III-C). Due to space constraints, we do not give those arguments here.

In general, the evidence for static assurance includes artifacts from system development and verification. For this example, we used both formal and descriptive models of the environment and the aircraft system, scenario descriptions, simulation results, specifications of safety properties and constraints, and formal verification results, e.g., of reachability properties as highlighted in the preceding discussion.

A key observation is that rationale capture, as in Fig. 5, explicitly highlights the various assumptions made (nodes A2, A3, A4) in such kind of formal-verification based evidence. These assumptions are either to be validated as part of development assurance, or monitored for violation at run-time to preclude continued operation if the verification is invalidated.

In a completed argument, undeveloped goals are supported by concrete evidence items that include, among other things, additional assurance claims to support and enforce those goals, validation data showing that those goals were appropriate, and verification data to show that the claims hold.

4) Assurance Quantification Model: We capture system behavior due to autonomous taxiing as a stochastic process model of the time series evolution of true CTE (CTE_a), in particular a *dynamic Bayesian network* (DBN) that takes into account both the temporal evolution of the underlying RVs, and the known and assumed conditional independence relations (Fig. 6). Assurance quantification using this model aims to query the DBN model for the posterior probability that the true aircraft location is within the specified lateral offset bounds, that is the interval $(-2, 2)$, at a given time. If that

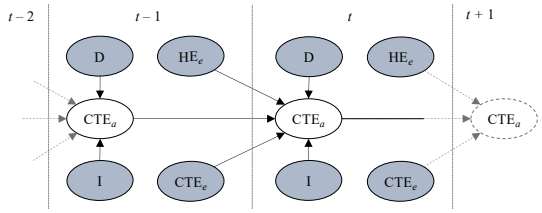

Fig. 6. DBN structure to quantify assurance in the `AssuredTaxi` reliability property [13]. Two adjacent slices are shown at times $t-1$, and t. The shaded nodes represent the observed variables, i.e., CNN estimates of CTE (CTE_e) and HE (HE_e); image inputs (I), a detection of outliers in image inputs (D). The clear nodes are the uncertain, latent variables, i.e., true CTE (CTE_a) and true HE (HE_a). For more details, see [13].

value is, say, 0.98, then $(-2, 2)$ is a 98% *credible interval* for the true aircraft position. In other words, based on the model we can be 98% confident that the aircraft is truly located within the allowed lateral offset. For a more detailed discussion on how we built and validated this model, see [13]. A component-focused model can also be developed in a similar way, e.g., to quantify the confidence in the accuracy of the CNN for estimating CTE and HE. We refer the reader to [14] for more details.

C. Platform Integration of Assurance Measures

The platform system architecture constrains how the assurance measure is implemented and deployed. As indicated in Section III-A, for this paper the integration platform is the iron bird HIL testbed (Fig. 7 shows a block diagram). The software components of this platform leverage the open source robot operating system (ROS)[6] framework for integration and communication, where each component is implemented as a ROS *node*. Asynchronous internode communication is achieved through a *publish-subscribe* pattern. Dedicated communication channels are implemented as so-called ROS *topics* to which ROS nodes subscribe (receive) or publish (send). All software components, except communication with the X-Plane software[7] use ROS topics for communication. Communication with X-Plane leverages the open source XPlaneConnect[8] plugin, which uses user datagram protocol (UDP) sockets.

As shown Fig. 7, assurance measure outputs are actively used for onboard contingency management, and also provided as a passive run-time risk assessment to an external visualization dashboard, shown here as inputs to the *synoptic displays*. We now mainly focus on the role of the assurance measure, the decision logic and the contingency manager components, which are relevant to realize our concept of dynamic assurance. We defer the discussion of the remaining components to Section IV-A.

As clarified in Section II-B2, the assurance measure here is an executable ROS node that implements the quantification model shown in Fig. 6. This receives simulator inputs of images as would be sent from the camera during an actual

[6]Robot operating system (ROS): http://wiki.ros.org/
[7]X-Plane 11 Flight Simulator: https://www.x-plane.com/
[8]NASA XPlaneConnect: https://github.com/nasa/XPlaneConnect

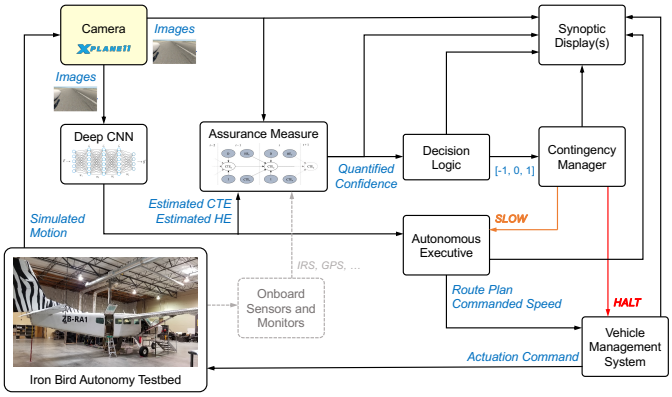

Fig. 7. Assurance measure integration architecture: The assurance measure is an executable component computing a quantitative value of confidence in the `AssuredTaxi` property, from a stream of images and CNN estimates of CTE and HE, which is then input to a decision logic that triggers contingency actions under conditions leading to taxiing hazards.

taxiing mission, along with CNN estimates of CTE and HE. Although it is also possible to receive additional inputs from other run-time monitors and sensors (e.g., position information from onboard localization systems, shown in Fig. 7 as GPS and IRS), for this proof-of-concept the only run-time monitor considered is an OOD detector that is already a part of the quantification model [13] (also see Fig. 6).

The output from the assurance measure is a probability distribution over the states of CTE_a forecasted for a time interval corresponding to the time taken to laterally depart the runway, when there are no mitigations. A decision logic consumes this output to produce a fault code for the contingency manager, selected from its application interface specification: $[-1, 0, 1] \Leftrightarrow [\texttt{HALT}, \texttt{SLOW}, \texttt{TAXI}]$. The code to be selected is based on

- a decision criterion applied to the assurance measure output: when there is $\geq 30\%$ confidence that the true CTE exceeds the allowed lateral offset, the assurance claim does not hold; and
- a simple model of the system-level effect, i.e., the probability of, and time to, a lateral runway overrun given the current system state, the assurance measure output, and assumptions about vehicle dynamics.

The decision logic is simply that when `AssuredTaxi` holds then the default code, 1, is maintained, otherwise the time to the unsafe consequence determines which of the remaining two fault codes, $[-1, 0]$, are sent.

The assurance measure described here itself presents a consequence to be characterized and a risk to be managed. This can employ established safety assessment processes together with our DAC methodology (Fig. 1). Accordingly, the overall DAC for this AAS would additionally include a structured argument with an integration focus that justifies the design decisions made above, through diverse evidence including, for example, Monte Carlo simulations of aircraft braking profiles under various operating conditions, HIL platform based simulations of the integrated system, and flight platform testing.

IV. Evaluation and Discussion

A. Simulation Environment

The simulation environment to assess the efficacy of dynamic assurance consists of the components shown by the system architecture in Fig. 7. Here, the X-Plane software simulates taxiing by streaming runway images to the deep CNN component—as well as to the assurance measure, and to the synoptic displays (described subsequently)—at a predefined frame rate. This is based on the vehicle dynamics, and the steering control and actuation inputs that it receives from the iron bird HIL platform.

Those, in turn, are the responses to the rudder actuation commands received from the VMS, which itself also simulates brake and throttle actuation. The VMS contains the control logic for vehicle steering and it uses the CNN estimates of CTE and HE embedded in the route plan inputs that it receives from the autonomous executive (AE) component, along with speed commands. The AE, for its part, manages the route plan and speed profile of the vehicle, also calculating stopping distance horizons. Both the AE and the VMS receive contingency signals as shown in Fig. 7, based on the fault flags signaled by decision logic based on assurance measure outputs, as discussed in Section III-C.

The synoptic displays (Fig. 8) show, in general, system status and telemetry information. Here, they display the camera images streamed from the X-Plane simulator, the moving map output from the AE, and—the focus here—an assurance dashboard (Fig. 8, bottom right). This dashboard contains visualizations of confidence quantification outputs from the assurance measure, specifically: *i*) the probability of violating the `AssuredTaxi` property for a predefined number of look-ahead time steps and for different values of `offset` ($1.43\mathrm{m}$ and $2\mathrm{m}$), in the `AssuredTaxi` property, and *ii*) a forecast of the uncertainty in the true lateral aircraft position on the runway.

This display implements the visualization specified during the development of the underlying quantification model (Fig. 8, bottom left), showing the true location (CTE_a), the uncertainty in the true location (the shaded blue regions), and the deep CNN output (CTE_e). The horizontal axis discretizes CTE_a, also showing the runway centerline and specified offsets, while the vertical axis discretizes time. Note that to validate the assurance measure, CTE_a must be known a priori; therefore it appears in the visualization display. For more details, see [13]. The assurance dashboard additionally shows the relevant aircraft *modes*—i.e., *initial roll-out* and *taxi*—for this operational concept, and aircraft state in terms of the fault codes selected by the decision logic, visualized as a stoplight: [`HALT`, `SLOW`, `TAXI`]. A batch simulation script is used for gathering data from the simulation runs.

B. Data Collection and Experimental Assessment

The goal in building the data generation scenarios for evaluation was to cover as large a region of the input space to the deep CNN component as reasonably possible. A diverse

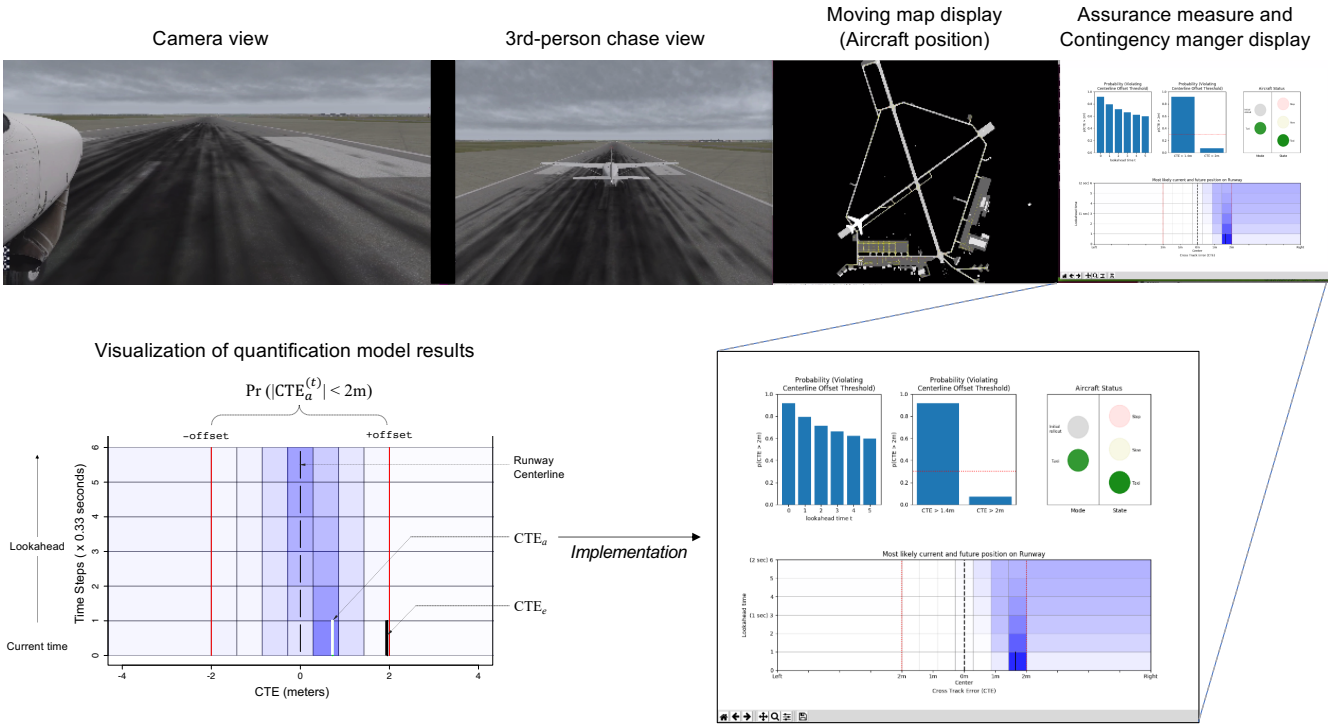

Fig. 8. Synoptic displays for run-time evaluation showing the run-time behavior of the system with the assurance measure integrated into the HIL platform, in terms of simulated images from the wing-mounted camera, a third-person chase plane, and a moving map display. An assurance measure display additionally visualizes run-time assurance quantification and decision logic outputs.

set of trajectories was generated that, in turn, resulted in different positions and headings relative to the runway and its centerline. Image instances were also generated with varying amounts of tyre rubber marks (since they could influence CNN output accuracy), at different times of the day (a relevant factor for the model because of the position of the shadow of the airplane) and under different cloud conditions. For evaluation, three variants of the CNN were used, representing ML models trained under differing cloud conditions and time of day. The test instances were generated based on wider distributions over the environment variables (i.e., position, heading, cloud, time-of-day, tyre marks), such that the robustness to low density regions of the data could be tested.

Boeing developed the operational concept and the trained deep CNN component, also supplying data (drawn from the training and validation environments of the CNN) to support assurance measure development. Boeing also supplied the HIL testbed, and formulated the evaluation test scenarios, whilst providing data drawn from the environments representative of the test scenarios for tuning the assurance measure prior to platform integration. The authors from KBR developed the DAC, its constituent assurance measure, the initial decision logic, and the assurance measure visualization. The latter two were then jointly refined to ensure that component interfaces were consistent and compatible.

Twenty four different scenarios involving different simulated environmental conditions were executed, followed by qualitative and quantitative assessments. The qualitative assessment was performed based on the observed behavior of the simulated aircraft during testing. Fig. 8 (bottom right), shows a snapshot from one test simulation run under overcast conditions. Here, the assurance measure shows a low probability of violating the $2\mathrm{m}$ offset, also indicating that there is more confidence that CTE_a lies in the interval $(1.43\mathrm{m}, 2\mathrm{m})$, than in other intervals for the current and the next three subsequent time steps. Consequently, the aircraft continues to taxi, as shown by the green `TAXI` state.

However, due to the tight bounds on the assurance properties, most evaluation scenarios ended with the assurance measure triggering a contingency management action stopping the aircraft during the execution of its route. In only one of the 24 scenarios, a potential safety violation was observed, in particular at runway intersections, emerging from unanticipated behavior at the interface between the decision logic required to integrate the assurance measure and the aircraft avionics.

This emergent behavior was, itself, the consequence of a decision logic design resulting from the choices made during its development: e.g., the criteria for sufficient assurance and the temporal conditions necessitating contingency actions. We relied upon engineering judgement for design decisions so as to the balance the conservative behavior that safety demands, with the need for a more permissive and performant system, i.e., to avoid frequent, unnecessary stopping.

In general, deciding between a collection of options in the presence of conflicting and uncertain outcomes is a special case of *decision making under uncertainty* [15]. We plan to

investigate such techniques to develop a principled approach to integrate assurance measures. Moreover, our assurance measure did not integrate inputs from on-board localization.

The quantitative analysis uses 49050 messages from 31 ROS bags, collected during the experiments, which contain the outputs of both the deep CNN component, and the assurance measure. From these, 9887 (20.16%) were used to evaluate the effectiveness of the combination of the assurance measure and decision logic. The analysis yielded, respectively, a false alarm rate of 2.1% and a false negative rate of 4.9% for the SLOW contingency action, while for the HALT action, those were 0.8%, and 5.7% respectively.

Here we have presented the results of assessing the integrated system that embeds the assurance measure. The assessment is generally suggestive of the need for further study to better understand the requirements for assurance measure integration, and how to monitor and enforce assurance requirements due to ML components. For details on how we built, calibrated, and empirically assessed the system- and component-level assurance quantification models themselves, see [13] and [14], respectively.

V. Concluding Remarks

The aviation industry is examining the potential roles of ML components to enhance aircraft performance and safety while defining ways to address the uncertainty in their behavior, but recognizes the limitations of established compliance approaches. Hence, a practicable approach to the development and deployment of autonomous aviation system must address both the needs of real-time assurance and the longer term needs of certification. To that end, in this paper, we have described the dynamic assurance case (DAC) concept and an accompanying methodology for assurance during development, and subsequently, continuously in operation, incorporating both qualitative and quantitative aspects.

Our framework addresses assurance concerns of the integrated system (i.e., the baseline system augmented with assurance measures), from both a system focus, and from an integration focus. We have described the main DAC components, and its application to the assurance of an autonomous taxiing capability, integrated on a single engine fixed-wing aircraft system platform, as a proof-of-concept.

To assess our approach, we have relied upon a hardware-in-the-loop (HIL) simulator that is a surrogate for the airworthy flight platform. One of the challenges we faced was obtaining sufficient useful data to build the models underpinning assurance measures. We believe that a more principled approach to specifying training data should be possible and that such specifications could be derived from a high-level *domain-specific language* (DSL) that will let us abstract from the details of the individual probabilistic models, while ensuring consistency with the other DAC components.

A broad set of assurance objectives for autonomous systems, have been formulated in [16], while a category of additional objectives and methods for generating evidence to meet those objectives have been summarized in [17]. In [18], an assurance lifecycle for ML components has been proposed, along with approaches to characterize so-called *generalization bounds* on ML models. Both the safety analysis process and the assurance lifecycle proposed in [18], as well as the assurance objectives and supporting methods from [16], [17] are fully compatible with our methodology, and can be mapped to one or more of the DAC components since the latter are largely agnostic to the specific processes used. Indeed, specific assurance objectives can be mapped to DAC assurance policies, while the artifacts produced constitute DAC evidence items. Moreover, assurance rationale is a core DAC component by which justification can be provided as to why and how the proposed assurance artifacts and methods entail platform or application-specific assurance claims.

References

[1] S-18, Aircraft And System Development And Safety Assessment Committee, *ARP 4754, Guidelines for Development of Civil Aircraft and Systems*, SAE International, Dec. 2010.

[2] Underwriter Laboratories Inc., "Standard for Safety for the Evaluation of Autonomous Products UL 4600," April 2020.

[3] E. Denney and G. Pai, "Tool Support for Assurance Case Development," *Journal of Automated Software Engineering*, vol. 25, no. 3, pp. 435–499, September 2018.

[4] R. Clothier, E. Denney, and G. Pai, "Making a Risk Informed Safety Case for Small Unmanned Aircraft System Operations," in *17th AIAA Aviation Technology, Integration, and Operations Conference (ATIO 2017), AIAA Aviation Forum*, AIAA 2017-3275, June 2017.

[5] E. Denney, G. Pai, and I. Whiteside, "The Role of Safety Architectures in Aviation Safety Cases," *Reliability Engineering & System Safety*, vol. 191, 2019.

[6] The Assurance Case Working Group (ACWG), "Goal Structuring Notation Community Standard Version 2," SCSC-141B, Jan. 2018.

[7] ASTM International, "Standard Practice for Methods to Safely Bound Flight Behavior of Unmanned Aircraft Systems Containing Complex Functions," ASTM F3269-17, 2017.

[8] S-18, Aircraft And System Development And Safety Assessment Committee, *ARP 4761, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, SAE International, Dec. 1996.

[9] N. Leveson and J. Thomas, "STPA Handbook," March 2018.

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobilenetV2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[11] E. Denney, G. Pai and C. Smith, "Generic Safety Argument Patterns for Learning-enabled Components," in *AAAI Workshop on Artificial Intelligence Safety (SafeAI)*, AAAI. CEUR Workshop Proceedings, 2020.

[12] F. Laine, C.-Y. Chiu, and C. Tomlin, "Eyes-Closed Safety Kernels: Safety for Autonomous Systems Under Loss of Observability," arXiv:2005.07144 [eess.SY], May 2020.

[13] E. Asaadi, E. Denney, and G. Pai, "Quantifying Assurance in Learning-enabled Systems," in *39th International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2020)*. Springer, 2020 (to appear).

[14] E. Asaadi, E. Denney, and G. Pai, "Towards Quantification of Assurance for Learning-Enabled Components," in *2019 15th European Dependable Computing Conference (EDCC)*. IEEE, September 2019, pp. 55–62.

[15] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[16] Safety of Autonomous Systems Working Group, "Safety Assurance Objectives for Autonomous Systems," SCSC-153A, February 2020.

[17] R. Ashmore, R. Calinescu, and C. Paterson, "Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges," arXiv:1905.04223v1 [cs.LG], 2019.

[18] J. M. Cluzeau, X. Henriquel, G. Rebender, G. Soudain, L. van Dijk, A. Gronskiy, D. Haber, C. Perret-Gentil, and R. Polak, "Concepts of Design Assurance for Neural Networks (CoDANN)," European Union Aviation Safety Agency (EASA) and Daedalean AG, Public Report Extract Version 1.0, March 2020.