Evaluation of New Assurance Tools for Airborne Machine Learning-Based Functions

Cong Liu, Heber Herencia-Zapana, Saqib Hasan, Amer Tahat, Isaac Amundson, Darren Cofer
Collins Aerospace
{first.last}@collins.com

Abstract—**As part of the DARPA Assured Autonomy program, our team has developed or evaluated a number of technologies to address gaps in traditional hardware and software assurance processes that make it difficult or impossible to demonstrate the correctness and safety of machine learning (ML) components. These include new approaches for testing and completeness metrics, formal analysis of neural networks, input domain shift assessment, and run-time monitoring and enforcement architectures. Although many of these tools and methods were successfully applied to demonstration platforms, most have not been evaluated on real-world product development efforts in a certification context. In this paper, we describe our evaluation of these new assurance methods and tools applied to ML-based systems that will soon be undergoing certification.**

Index Terms—**Assured Autonomy, neural network verification**

I. Introduction

Use of machine learning (ML) technology is expanding to support autonomy and other advanced functionality in safety/security-critical applications. Unfortunately, when it comes to the regulatory approval of these systems, the certification guidance has not kept pace with the technology.

DO-178C, for example, provides guidance regarding software aspects of certification and is used by the commercial aviation industry and regulators as a means of compliance with airworthiness regulations. DO-178C fundamentally relies on requirements-based testing and structural coverage metrics for confidence that a software development process correctly implements a set of requirements. When requirements-based tests fail to exercise part of the software logic (as revealed by structural coverage metrics), it is reasonable to conclude that either a requirement is missing, or the implementation includes unintended behavior. Since neural networks do not explicitly implement logical decisions, structural coverage can usually be achieved with a single test case and is therefore not helpful in identifying and eliminating unintended behaviors. Because it is difficult to demonstrate assurance by examining the neural network design, other approaches are needed if we wish to take advantage of ML in our high-assurance applications.

As part of the DARPA Assured Autonomy (AA) program, our team has developed or evaluated a number of technologies to address gaps in traditional hardware and software assurance processes that make it difficult or impossible to demonstrate the correctness and safety of ML components. These include new approaches for testing and completeness metrics, formal analysis of neural networks, input domain shift assessment, and run-time monitoring and enforcement architectures. Although many of these tools and methods were successfully applied to demonstration platforms, most have not been evaluated on real-world product development efforts in a certification context.

In this paper, we describe our evaluation of new assurance methods and tools applied to ML-based systems that will need to be approved by certification authorities before they can be deployed on aircraft.

Distribution statement "A" (approved for public release, distribution unlimited). This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA), contract FA8750-18-C-0099. The views, opinions, or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

II. Applications

We have evaluated the tools on a variety of ML-based systems. In some cases, the system used ML to implement new aircraft functionality. In other cases, the goal is to use a neural network (NN) to create a more time- or memory-efficient implementation of an existing function. ML-based functions used for evaluation include:

Remaining Useful Life (RUL) [1]: A convolutional neural network (CNN) uses vibration measurements from rotating equipment to estimate time until maintenance or replacement. The RUL NN is reasonably large, with 1600 inputs (a sequence of snapshots of condition indicators from vibration sensors and other metrics) and 94,500 learnable parameters arranged in 12 layers. The output is the predicted remaining useful life of the equipment. A public version with training data and requirements to be verified was made available as a benchmark for the 2022 International Verification of Neural Networks Competition (VNN-COMP).

Recommended Cruise Level (RCL): Computes time and fuel optimal altitude as a recommendation to the pilot, replacing a complex optimization calculation and saving CPU time. The NN is a fully-connected, feed-forward NN with rectified linear unit (ReLU) activation functions. It has 5 inputs (aircraft and environment conditions) and 2 outputs (time and fuel costs), and 5 hidden layers, each with 10 neurons.

Fuel Quantity Measurement (FQM): Computes fuel mass based on sensor measurements, replacing less-accurate table-based implementations. A NN is trained to invert a function that computes sensor measurements from fuel mass, fuel tank geometry, and aircraft orientation. The NN is a fully-connected, feed-forward network with 6 sensor inputs and one output (fuel mass). The NN is very simple, having a single hidden layer consisting of 50 neurons and uses the *tanh* activation function.

Runway Overrun Protection (ROP): Estimates aircraft landing distance based on weight, speed, weather conditions, runway slope, and other parameters according to the requirements of ED-250, Runway Overrun Awareness and Alerting System (ROAAS). We evaluated one of the NNs from the system which was a fully-connected, feed-forward network with 10 inputs and 3 outputs. The outputs correspond to stopping distances with different brake settings. The NN has 2 hidden layers with 40 neurons each. It uses the *tanh* activation function.

Flight Trajectory Optimization (FTO): A neural network-based implementation of an optimized trajectory function, such as the A* algorithm and derivatives, to reduce computation time in a set of complex flight and weather conditions. The NN is a fully-connected, feed-forward network with ReLU activation functions. It has 7 inputs, which model aircraft position relative to an assigned flight altitude limit, distance to weather/threat, and relative velocity. It has 3 hidden layers with 35, 70, and 70 neurons, respectively. It has 5 outputs representing the next flight direction: up, down, right, left, and straight.

III. Formal Verification

The main purpose of using formal verification in NN certification is to provide evidence that the NN not only performs its intended function, but also exhibits no unintended behavior. We evaluated three state-of-the-art (SOTA) NN verifiers: α-β-CROWN, Marabou, and Venus2. We applied the tools to formally verify the properties of four Collins NN applications. A *property* is an assertion about the mathematical characteristics of a neural network, often in the form of input and output relationships.

Verification Tools: α-β-CROWN [2] reduces the property verification to prove or disprove whether the outputs of a NN are always non-negative, given the input bounds. The NN is created by adding a linear output layer to the original NN, which models the property. It uses abstract interpretation techniques to efficiently propagate the linear bound of neuron values through a NN. Its branch-and-bound approach enables parallelization and it utilizes GPUs to accelerate its performance. It has been ranked as the top NN verifier in recent years of VNN-COMP. It supports many forms of activation functions (e.g., *ReLU*, *tanh*, *sigmoid*).

Marabou [3] [4] is based on satisfiability modulo theories (SMT). It transforms a NN property verification problem to a satisfiability (SAT) instance, which consists of a set of linear and non-linear constraints (due to the activation functions) on the neuron values. The property is falsified if and only if there exists an assignment of neuron values that satisfies all the constraints. Marabou accepts properties that are encoded as general linear constraints on both inputs and outputs. We used Marabou2 (dated March 2024) for our evaluation.

Venus [5] formulates a NN verification problem as a mixed-integer-linear-program (MILP). The property is falsified if and only if the corresponding program is feasible. The MILP can be transformed to a linear program that can be solved in polynomial-time. Venus2 leverages the efficient, commercial-off-the-shelf (COTS) linear program solver, Gurobi. It verifies NNs that only use piecewise linear activation functions. We used Venus2 (dated March 2024) for our evaluation.

Verification Process: Our verification process often consisted of three main steps. First, we formulated a number of formal properties, which were derived from performance or safety requirements. Second, we manipulated the NN structure so that we can express the properties in a format that can be verified by the NN verifiers. For example, α-β-CROWN does not support general linear constraints on NN inputs. We added an extra output layer to connect the NN inputs and outputs in parallel, using ONNX *Concat* operator. Third, we tuned the tool executable parameters. Each tool has its own parameters, which could have a major impact on the verification performance for a specific instance. Tuning the parameters is not a straightforward process; it often requires knowledge of the underlying verification algorithm.

TABLE I: Verification Results

Property	ABCROWN	Marabou2	Venus2
ftoMaxfl	UNSAT (7.3)	UNSAT (493)	UNSAT (69.2)
ftoMaxflsat	SAT (3.0)	SAT (20)	*Abort*
ftoMinfl	UNSAT (10.5)	UNSAT (1808)	UNSAT (30.3)
ftoMinflsat	SAT (3.1)	SAT (1448)	*UNSAT (28.5)*
ftoaboveStr	UNSAT (12.8)	*SAT (795)*	UNSAT (39.5)
ftoDown	UNSAT (6.5)	UNSAT (67)	UNSAT (25.6)
ftoStr	UNSAT (6.4)	UNSAT (0)	UNSAT (17.9)
ftoUp	UNSAT (6.8)	UNSAT (357)	UNSAT (52.4)
ftobelowStr	UNSAT (24.2)	*SAT (223)*	*Abort*
rclOut	UNSAT (6.5)	UNSAT (0)	UNSAT (24.6)
rclMaxFuel	SAT (1.8)	SAT (0)	SAT (1.0)
ropOrder	UNSAT (7.0)	*Timeout*	-
ropRange	SAT (2.4)	*Timeout*	-
ropMono	SAT (0.5)	-	-
fqmAcc1	UNSAT (4)	-	-
fqmAcc2	UNSAT (12)	-	-

Verification Results: The verification results are summarized in Table I. The first three letters of the property name indicate the application (FTO, RCL, ROP, or FQM). A result of UNSAT means the property is proved. And a result of SAT means a counterexample was found. The runtime is in seconds. The timeout is set to 2000 seconds. The verification was executed on a Linux server with 128 AMD EPYC 7601 32-Core Processors and 503GB memory. We ran Marabou2 with Split and Conquer (SNC) mode and four parallel threads. Since Venus2 does not support the *tanh* activation function and Marabou2 does not support the ONNX operator *concat*, they could not verify some instances, which are indicated by the dashes in the table.

Overall, α-β-CROWN was the most efficient and produced the correct results on all instances. Marabou2 generated incorrect results for two instances. And Venus2 generated incorrect results for one instance and aborted for two instances. Our preliminary investigation in the incorrect or aborting cases indicates tool implementation issues.

Note that the VNNLIB parsers used by the tools do not support general linear constraints (i.e., $AX \leq B$). They support a limited subset of constraints defined by the VNNCOMP benchmarks (e.g., variable compared to constant, comparing two variables). We modified the VNNLIB parser of α-β-CROWN to verify properties that are encoded as general linear constraints.

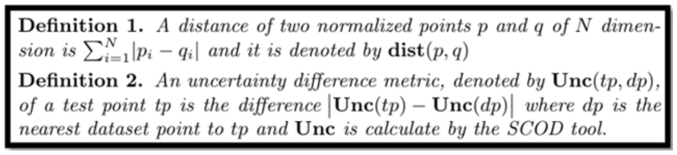
IV. Out-Of-Distribution Monitoring

Deep Neural Networks (DNNs) have been instrumental in driving significant progress in deriving actionable insights from complex input data, such as videos or images. The primary function of a DNN is to process input sets, execute iterative computations, and generate outputs aimed at addressing real-world challenges, including classification, prediction, and recommendation. However, a notable drawback of these models is their susceptibility to unpredictable behavior when presented with inputs markedly different from those encountered during training. Out-of-distribution (OOD) input for a DNN comprises data that diverges from the dataset used in model training, stemming from varying temporal, environmental, or conditional factors compared to the in-distribution data.

Detecting OOD instances involves determining whether a new sample aligns with the distribution of known data. There are numerous methods and tools available for detecting OOD inputs [6]–[9]. One such tool discussed and evaluated in this paper is the Sketching Curvature for Out-of-Distribution Detection (SCOD) tool [10]. This tool, jointly developed by Stanford and MIT, computes an uncertainty metric, $\mathbf{Unc}(\mathbf{x})$, which is designed to be low when queried on inputs drawn using the probability distribution of the training data, but high for inputs far from this data manifold. A critical component for the OOD analysis using the SCOD tool is establishing a *threshold* value for classifying inputs as OOD, which necessitates knowledge of the probability distribution of the training data or labeled OOD data.

Determining the threshold value for classifying inputs as OOD can be accomplished through diverse methodologies. One approach entails leveraging the probability density function (pdf) of the training data to generate inputs and compute the uncertainty metric using the SCOD tool. However, in practical scenarios, the pdf of the training data is frequently unknown, posing a significant challenge. Another approach entails acquiring labeled inputs from both within and outside the distribution, then using receiver operating characteristic (ROC) analysis to determine the threshold. However, obtaining such labeled data can be challenging in certain scenarios. For example, for the Collins DNN, obtaining these labeled inputs would be impractical and costly. Therefore, to address this challenge, we developed the following methodology. First, we define the notion of data similarity and quantify it using generalization performance parameters (δ, ϵ). Second, we compute the performance parameters for data points derived from the training set along with the respective lines connecting these data points. Third, we refine the performance parameter values by utilizing the convex hull of a DNN training dataset. This methodology is illustrated using the ROAAS DNN, and a review of the SCOD tool is provided.

First, the OOD concept captures the notion that a DNN exhibits high performance when presented with input data similar to the training dataset. However, it may struggle to make accurate decisions when encountering inputs significantly divergent from the training data. Notably, the notion of similarity and dissimilarity is not solely dependent on the dataset but also on the model architecture. For instance, a data point may be considered OOD for one model but within the distribution for another model, even if trained on the same dataset. Hence, it is crucial to formalize not only the notion of proximity in terms of distance but also to capture the sensitivity of the DNN model's weight, as indicated by the SCOD uncertainty metric $\mathbf{Unc}(\mathbf{x})$. Fig 1 shows two definitions that capture these concepts. With these two definitions, it is feasible


Definition 1. *A distance of two normalized points p and q of N dimension is $\sum_{i=1}^{N} |p_i - q_i|$ and it is denoted by $\mathbf{dist}(p, q)$*

Definition 2. *An uncertainty difference metric, denoted by $\mathbf{Unc}(tp, dp)$, of a test point tp is the difference $|\mathbf{Unc}(tp) - \mathbf{Unc}(dp)|$ where dp is the nearest dataset point to tp and $\mathbf{Unc}$ is calculate by the SCOD tool.*

Fig. 1: Basic definitions.

to mathematically represent that a test point, tp, is similar to a training data if there exists a data point, dp, from the training data such that $\mathbf{dist}(\mathbf{tp}, \mathbf{dp}) \leq \delta$ and $\mathbf{dist}(\mathbf{tp}, \mathbf{dp}) \leq \epsilon$. The imposition of these two inequalities is essential to prevent scenarios where a test point with a low uncertainty metric is erroneously classified as within the distribution, despite being significantly distant from the training dataset. This rationale motivates the following definition, shown in Fig 2. These definitions facilitate reframing the task of establishing a threshold value for OOD into the task of determining the generalization performance parameters, (δ, ϵ), for managing OOD instances. The computation of these generalization performance parameters is conducted leveraging a DNN model, its associated training dataset, and the SCOD uncertainty metric.

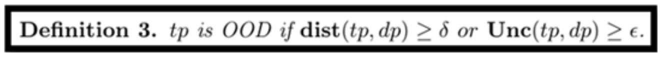
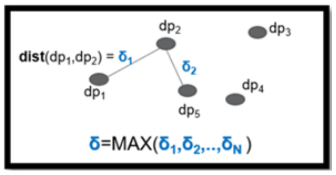
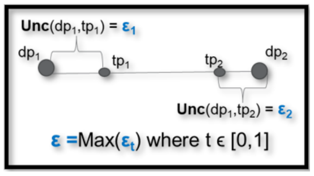

Definition 3. *tp is OOD if $\mathbf{dist}(tp, dp) \geq \delta$ or $\mathbf{Unc}(tp, dp) \geq \epsilon$.*

Fig. 2: OOD definition.

Second, we calculate (δ, ϵ) for data points from the training set and their respective lines between training data points. The computation of the performance parameters occurs through a two-step process, involving the calculation of δ followed by the computation of ϵ. Initially, δ is determined as follows: First, for each data point within the training dataset, the closest distance to another data point is computed. Next, δ is derived

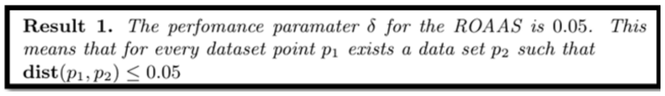


(a) δ calculation



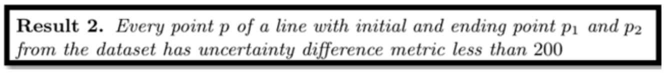
(b) ϵ calculation

Fig. 3: (δ, ϵ) calculation



Result 1. *The perfomance paramater δ for the ROAAS is* 0.05. *This means that for every dataset point p_1 exists a data set p_2 such that* $\mathbf{dist}(p_1, p_2) \leq 0.05$

(a) δ for ROAAS



Result 2. *Every point p of a line with initial and ending point p_1 and p_2 from the dataset has uncertainty difference metric less than* 200

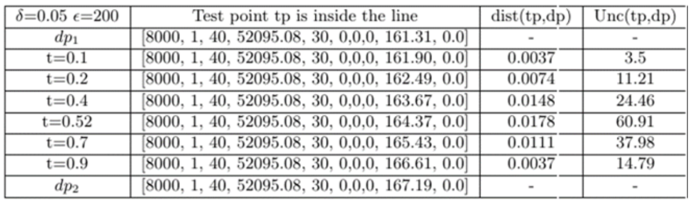
(b) ϵ for ROAAS

Fig. 4: (δ, ϵ) for ROAAS

as the maximum value among all such closest distances, as illustrated in Fig 3a. Next, ϵ is computed as follows: For each data point within the training dataset, a line is generated connecting it to its closest point. Then, for each generated line, the maximum uncertainty difference metric among its constituent points is calculated using the SCOD tool. This process is depicted in Fig 3b. Ultimately, ϵ is determined as the maximum uncertainty metric among all generated lines. These generalization performance parameters capture the notion that employing the DNN is deemed safe when a test point either is a training data point (trivial case) or lies within a line (reflecting some degree of generalization). However, it is worth noting that the baseline performance parameters may err on the side of caution.

Third, to further generalize the performance parameters utilizing the δ parameter, it is necessary to employ a procedure that systematically assesses a set of inputs to identify a subset conducive to generalization. The concept entails leveraging the Operational Design Domain (ODD) [11]–[13] as the overarching set, within which a subset conducive to generalization can be identified. The ODD within the DNN framework draws inspiration from autonomous vehicles and robotics. It delineates the specific conditions within which a DNN is engineered and intended to function safely and efficiently. The ODD factors in various considerations, encompassing environmental conditions, temporal aspects, safety constraints, and characteristics of input data. In our pursuit of enhancing the generalization of the performance parameter δ, we will particularly concentrate on the input data characteristics, which entail the distribution and range of features. One methodological approach involves constructing a convex hull within the feature space, represented by the inequality $A.x + b \leq 0$. The convex hull constitutes the smallest convex set that encompasses all the given points utilized for training. This attribute is pivotal in defining a concise and fundamental set wherein δ is augmented while maintaining ϵ. In practical terms, we will verify whether the points within the convex hull exhibit a performance parameter ϵ equivalent to the points along a line, while possessing a larger δ.

We illustrate this methodology with the analysis of OOD for the Runway Overrun Awareness and Alerting System (ROAAS) specified in ED-250. Here, our primary objective is to obtain the generalization performance parameters. ROAAS is a flight deck alerting system providing crews with situational awareness about the possibility of exceeding the end of the runway during aircraft approach and landing. The ROAAS DNN model comprises ten inputs and three outputs. The objective is to determine the performance parameters (δ, ϵ) to aid in discerning whether a point is OOD. The computation of these performance parameters unfolds in two steps: the first step involves utilizing a line of the dataset, while the second step entails employing the convex hull of datasets. This phase involves the computation of δ defined as the maximum of all minimum distances. Fig 4a posits the value of δ for ROAAS training data. To exemplify this result, consider the data points: $p_1 = [8000, 1, 40, 52095, 30, 0, 0, 0, 161, 0]$ and $p_2 = [8000, 1, 40, 52095, 30, 0, 0, 0, 167, 0]$, where the distance between them, denoted by $dist(p_1, p_2)$, equals 0.037. Subsequently, the calculation of the maximum uncertainty difference metric is executed utilizing the SCOD tool. Fig 4b posits the value of ϵ. To illustrate this result, contemplate the line connecting the data points dp_1 and dp_2. Each point lying within this line exhibits a performance parameter ϵ less than 200, as indicated in Fig 5. The baseline values $\delta = 0.05$



δ=0.05 ϵ=200	Test point tp is inside the line	dist(tp,dp)	Unc(tp,dp)
dp_1	[8000, 1, 40, 52095.08, 30, 0,0,0, 161.31, 0.0]	-	-
t=0.1	[8000, 1, 40, 52095.08, 30, 0,0,0, 161.90, 0.0]	0.0037	3.5
t=0.2	[8000, 1, 40, 52095.08, 30, 0,0,0, 162.49, 0.0]	0.0074	11.21
t=0.4	[8000, 1, 40, 52095.08, 30, 0,0,0, 163.67, 0.0]	0.0148	24.46
t=0.52	[8000, 1, 40, 52095.08, 30, 0,0,0, 164.37, 0.0]	0.0178	60.91
t=0.7	[8000, 1, 40, 52095.08, 30, 0,0,0, 165.43, 0.0]	0.0111	37.98
t=0.9	[8000, 1, 40, 52095.08, 30, 0,0,0, 166.61, 0.0]	0.0037	14.79
dp_2	[8000, 1, 40, 52095.08, 30, 0,0,0, 167.19, 0.0]	-	-

Fig. 5: ϵ for points in a line

and $\epsilon = 200$ reflect the concept that the model is deemed reliable when the test point either aligns with a training data point (trivial case) or falls within the line (indicating some degree of generalization). A point is classified as OOD if either its ϵ exceeds 200 or its δ surpasses 0.05. However, this approach may err on the side of caution. To determine the extent to which we can increase δ, we need a benchmark, and the convex hull serves as our reference point. Our goal is to increase δ based on the convex hull while ensuring $\epsilon \leq 200$. The convex hull is defined by the equation $A.x + b \leq 0$, where A and b were computed using the ROAAS training dataset. Through random testing, we have formulated a value for δ as shown in Fig 6. Based on this analysis, it is determined that for the ROAAS DNN, a point is considered OOD if either its ϵ exceeds 200 or its δ surpasses 0.3.

Through the utilization of the SCOD tool for this analysis, the observations suggest that it exhibits a notable degree of

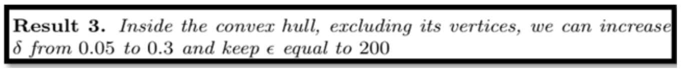

Result 3. *Inside the convex hull, excluding its vertices, we can increase δ from* 0.05 *to* 0.3 *and keep ϵ equal to* 200

Fig. 6: Convex hull

adaptability and customization, thereby making it well-suited for a wide spectrum of DNN models. More specifically, it was utilized to compute the **Unc** metric for three Collins models. Its configurability facilitated the uploading of datasets and DNN models, requiring only minor adjustments such as file format conversions. For instance, public Python libraries were leveraged to convert ONNX models to PyTorch. Although employing ROC analysis to calculate the OOD threshold proved impractical for Collins DNN models, an alternative methodology was devised and implemented as outlined in this paper. Thanks to SCOD's flexibility, integrating the SCOD tool with the Python implementation of this methodology was straightforward, enabling the calculation of generalization performance parameters.

V. Manifold-Based Test Generation

Neural networks created and deployed in mission critical domains must demonstrate high confidence in their predictions. However, without appropriate training data, it is impossible to evaluate the accuracy of the model. In addition, without considering edge cases and other special scenarios which may or may not be infrequent, the neural networks cannot be trained to handle such cases and make accurate predictions when encountering such scenarios. In high-dimensional input data applications such as image classification, required patterns cannot be easily captured by the available training dataset, which could result in these situations quite easily.

Manifold-based test generation [14]–[16] is a technique that provides a means for capturing the necessary patterns such that the NN can learn these patterns in a low-dimensional manifold space. This is achieved by projecting the data points from a high-dimensional input space to a low-dimensional space. The approach uses a Conditional Variational Autoencoder (CVAE) to capture the manifold space, which is then utilized to generate novel fault-revealing test cases. Note that a unique feature of this approach is that these test cases are generated along with the labels. The resulting fault-revealing test cases can be utilized in two ways: 1) to create a test suite that can evaluate the performance of the neural network, 2) to include the fault revealing test cases as part of the training set such that the neural network performance and accuracy can be improved.

Fig 7 shows the workflow of the manifold based test generation tool. First, the NN is trained using the dataset as shown in Fig 7 (step 1). The trained NN can be evaluated for its performance using a separate testing dataset, which is not shown in this workflow. Next, a trained VAE is generated by using the same training dataset used for the NN (step 2). Once the trained VAE model is developed, it can then be used to train the Latent Space Classifier (LSC), as shown in Fig 7

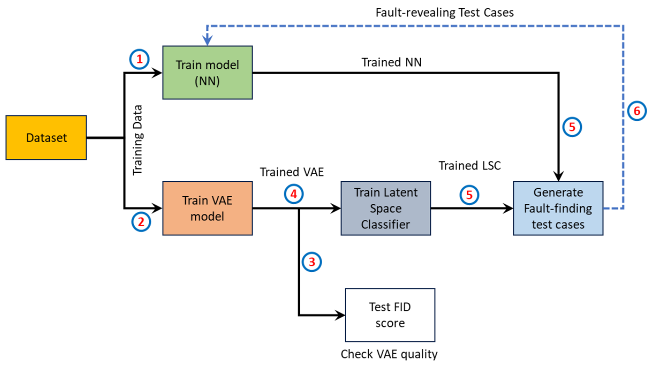


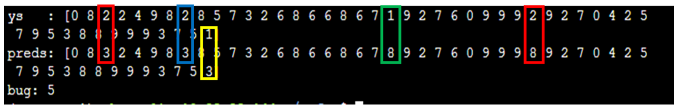
Fig. 7: Manifold-based test generation workflow.

(step 4). Note that if needed, the the quality of the trained VAE can be evaluated by measuring its Frechet Inception Distance (FID) score (step 3). The closer the FID score to zero, the better the trained VAE is assumed to be. Note that this step can also be performed before training the LSC. The LSC is responsible for learning the manifold space. Further, in order to generate fault-revealing test cases from the manifold space, the trained NN and the LSC are passed as an input to the test generation algorithm (step 5). The algorithm then selects the test cases from the manifold space, which includes the label and evaluates it using the NN. If the NN mis-predicts the output, the test case is regarded as a fault-revealing test. The algorithm is able to generate the desired number of faulty test cases, which can be utilized as part of the training data set to re-train the original NN (step 6). Note that in the figure this step is represented using a dashed line because this feature is currently not provided by the tool.

A. Tool evaluation:

We evaluated the tool on the MNIST dataset and followed the workflow as describe in Fig 7. Fault revealing test cases can be generated by following various algorithms, but the two utilized in our evaluation are the random test generation and the search-based test generation algorithms. We show the results for both of them below.

- *Random Test Case Generation:* Using this test generation algorithm, the desired number of test cases are generated. However, not all test cases may be fault-revealing. Fig 8 represents the generation of 50 random test cases. Out of the 50 test cases, only five of these were identified as fault-revealing. The test cases are shown with different color boxes. From Fig 8, it is impossible to understand the reasoning behind the selection of the fault-revealing test cases, meaning one cannot tell whether the tool identified the test cases correctly or not. Hence, a visual representation can provide better insight. Therefore, we have generated visual representations for each of the test cases in Fig 8 as shown in Fig 9.
- *Search-Based Test Case Generation:* Using this test generation algorithm, all the generated test cases are fault-revealing. Fig 10 represents generation of 15 search-based



```
ys   : [0 8 2 2 4 9 8 2 8 5 7 3 2 6 8 6 6 8 6 7 1 9 2 7 6 0 9 9 9 2 9 2 7 0 4 2 5
 7 9 5 3 8 8 9 9 9 3 7 5 1
preds: [0 8 3 2 4 9 8 3 8 5 7 3 2 6 8 6 6 8 6 7 8 9 2 7 6 0 9 9 9 8 9 2 7 0 4 2 5
 7 9 5 3 8 8 9 9 9 3 7 5 3
bug: 5
```

Fig. 8: Randomly generated test cases.

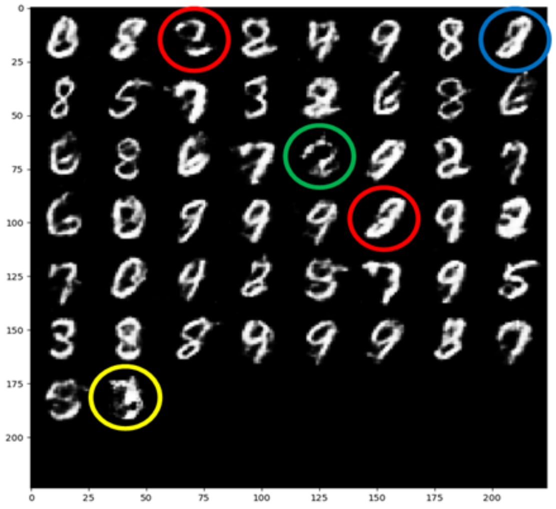
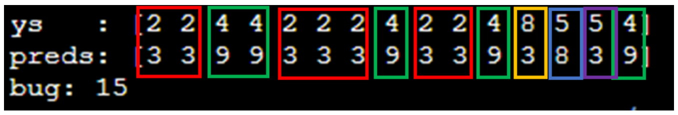


Fig. 9: Visual representation of randomly generated test cases.

test cases. As shown in the figure, all of the 15 test cases were fault-revealing and they are shown in different color boxes for clarity. Fig 11 shows the visual representation of each of these fault-revealing test cases.



```
ys    : [2 2 4 4 2 2 2 4 2 2 4 8 5 5 4]
preds: [3 3 9 9 3 3 3 9 3 3 9 3 8 3 9]
bug: 15
```

Fig. 10: Search-based generated test cases.

B. Tool Limitations:

Based on our evaluation, the tool provides a unique mechanism for generating fault-revealing test cases for high-dimensional datasets. However, the tool still has the following limitations:

- The generated fault-revealing test cases require manual inspection to be considered as part of the training test suite to improve prediction accuracy. As seen from Figures 9 and 11), it is obvious that some of the predictions and their visual representations do not match and hence require a manual step to analyze the generated test cases. This could become a significant bottleneck when generating a large number of fault-revealing test cases. An automated approach to address this issue would make the tool significantly more useful.
- The tool was developed as a prototype to demonstrate the proof of concept and still requires several updates to completely become usable in this space. For instance, the algorithm for search-based test generation was modified slightly to capture only fault-revealing test cases.

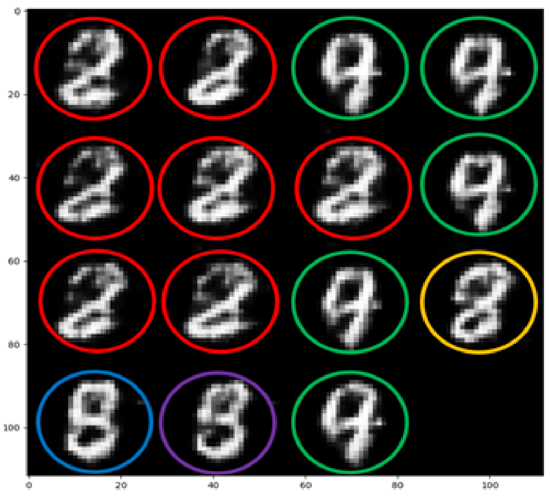


Fig. 11: Visual representation of search-based generated test cases.

- The tool is currently capable of working with MNIST, CIFAR, FASHION, and EMNIST datasets. To use other datasets as inputs, minor tool modifications are required.
- The approach works on high-dimensional datasets; however, for low-dimensional classification problems the tool is unable to handle the datasets appropriately. We believe the approach is sound and should be easily applicable to low-dimensional classification problems. This could easily enable even the neural networks in other classification problem domains to acquire better accuracy. However, the tool is currently unable to handle low-dimensional training datasets and therefore requires modification. The extent of needed modifications was not evaluated as part of our process.

VI. Property Inference

The Prophecy tool developed by NASA [17], [18] derives invariant properties of feed-forward neural networks, showcasing the networks' capacity to learn decision logic from neuron activation patterns. The method involves identifying decision patterns as network invariants connected to specific outputs, achieved through extracting input invariants and layer invariants. Formal verification utilizing decision procedures like Reluplex [19], a predecessor of Marabou, ensures that given invariants lead to desired outcomes, enhancing explainability, robustness, and aiding in network simplification and distillation. However, scalability issues, particularly with formal verification, pose significant challenges, such as timeouts when dealing with large-scale networks.

In evaluating the Prophecy formal verification approach, we applied it to the Flight Trajectory Optimization (FTO) application. FTO can be used to help aircraft navigate around hazardous weather conditions. Challenges like scalability, lack of modularity, and outdated documentation hindered progress.

An updated version of Prophecy [20] integrated Marabou for the formal verification step, and a parallel execution algorithm to enhance scalability, addressing some of the challenges encountered. Initial tests on ACAS-Xu [21] showed promising results, with Marabou successfully verifying some properties in a significantly reduced timeframe compared to applying Reluplex directly.

For example, Reluplex may need 12 hours while trying to verify the Clear of Conflict (CoC) property over certain domain invariants of large size. However, the tool successfully verified the same ACAS-Xu CoC properties in approximately 3.5 hours using Marabou over similar invariants.

The most resource-intensive step in the formal verification process involved refining initial convex hull guesses, aiming to eliminate adversarial examples and ensure robust invariants.

We found that optimizing Prophecy's performance is crucial before deploying the algorithm for applications where the complexity of the formal verification step increases substantially compared to ACAS-Xu.

Techniques like using α-β-CROWN could enhance the refinement step, albeit with limitations. Additionally, more efforts are needed to resolve issues with corrupted invariants.

Future directions involve exploring transformer architecture, such as GPT embedding models [22] and classifiers, to improve the quality of the initial convex hull approximations and reduce the need for redundant formal checks. If successful, this approach could offer scalable solutions for larger datasets and dimensions, enhancing the generalizability of formal verification methods in NN analysis.

VII. Conclusion

Collins researchers and product engineers are exploring many applications of machine learning. In some cases, the goal is to implement new aircraft functionality using the unique capabilities of ML. In other cases, the goal is to use a neural network to create a more time- or memory-efficient implementation of an existing function.

A variety of assurance technologies have been developed and investigated on the DARPA Assured Autonomy program and related efforts. These include new approaches for testing and completeness metrics, formal analysis of neural networks, input domain shift assessment, and run-time monitoring and enforcement architectures. For each selected ML application, we determined one or more AA technologies that can be used to satisfy the relevant certification objectives. We evaluated the effectiveness of the technologies and the evidence produced.

In general, we believe all the approaches covered in this paper can play a role in assuring low-complexity, low-DAL products. The manifold-based test generation and property inference tools have not yet reached the necessary maturity levels to be used on programs of record, but their underlying methodologies show promise. We plan to share these findings with certification authorities to obtain feedback.

References

[1] D. Kirov and S. F. Rollini, "Benchmark: Remaining useful life predictor for aircraft equipment." Berlin, Heidelberg: Springer-Verlag, 2023, p. 299–304. [Online]. Available: https://doi.org/10.1007/978-3-031-46002-9_18

[2] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *Advances in Neural Information Processing Systems*, vol. 31, pp. 4939–4948, 2018. [Online]. Available: https://arxiv.org/pdf/1811.00866.pdf

[3] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, ser. Lecture Notes in Computer Science, I. Dillig and S. Tasiran, Eds., vol. 11561. Springer, 2019, pp. 443–452. [Online]. Available: https://doi.org/10.1007/978-3-030-25540-4_26

[4] H. Wu, O. Isac, A. Zeljić, T. Tagomori, M. Daggitt, W. Kokke, I. Refaeli, G. Amir, K. Julian, S. Bassan, P. Huang, O. Lahav, M. Wu, M. Zhang, E. Komendantskaya, G. Katz, and C. Barrett, "Marabou 2.0: A versatile formal analyzer of neural networks," 2024.

[5] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of relu-based neural networks via dependency analysis," in *AAAI Conference on Artificial Intelligence*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:213299187

[6] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," *arXiv preprint arXiv:1610.02136*, 2016.

[7] H. Ritter, A. Botev, and D. Barber, "A scalable laplace approximation for neural networks," in *6th international conference on learning representations, ICLR 2018-conference track proceedings*, vol. 6. International Conference on Representation Learning, 2018.

[8] V. Abdelzad, K. Czarnecki, R. Salay, T. Denounden, S. Vernekar, and B. Phan, "Detecting out-of-distribution inputs in deep neural networks using an early-layer output," *arXiv preprint arXiv:1910.10307*, 2019.

[9] D. Madras, J. Atwood, and A. D'Amour, "Detecting underspecification with local ensembles," *arXiv preprint arXiv:1910.09573*, 2019.

[10] A. Sharma, N. Azizan, and M. Pavone, "Sketching curvature for efficient out-of-distribution detection for deep neural networks," in *Uncertainty in artificial intelligence*. PMLR, 2021, pp. 1958–1967.

[11] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, SAE on Road Automated Driving Committee SAE J3016.

[12] H. Torfah, C. Xie, S. Junges, M. Vazquez-Chanlatte, and S. A. Seshia, "Learning monitorable operational design domains for assured autonomy," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2022, pp. 3–22.

[13] P. Irvine, X. Zhang, S. Khastgir, E. Schwalb, and P. Jennings, "A two-level abstraction odd definition language: Part i," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2021, pp. 2614–2621.

[14] T. Byun and S. Rayadurgam, "Manifold-based test generation for image classifiers," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 221–221.

[15] ——, "Manifold for machine learning assurance," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 2020, pp. 97–100.

[16] T. Byun, S. Rayadurgam, and M. P. Heimdahl, "Black-box testing of deep neural networks," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 309–320.

[17] D. Gopinath, A. Taly, H. Converse, and C. S. Păsăreanu, "Finding invariants in deep neural networks," *ArXiv*, vol. abs/1904.13215, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:140218354

[18] D. Gopinath, H. Converse, C. Pasareanu, and A. Taly, "Property inference for deep neural networks," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Los Alamitos, CA, USA: IEEE Computer Society, nov 2019, pp. 797–809. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ASE.2019.00079

[19] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: a calculus for reasoning about deep neural networks," *Form. Methods Syst. Des.*, vol. 60, no. 1, p. 87–116, jul 2021. [Online]. Available: https://doi.org/10.1007/s10703-021-00363-7

[20] Safe Deep Learning for Space Systems Group, NASA Ames Research Center, “ACASX Prophecy parallel experiments.” [Online]. Available: https://github.com/safednn-nasa/Prophecy/blob/master/ACASX_Prophecy_parallel_experiments.ipynb

[21] W. Feng, C.-C. Huang, A. Turrini, and Y. Li, “Modelling and implementation of unmanned aircraft collision avoidance,” in *Dependable Software Engineering. Theories, Tools, and Applications: 6th International Symposium, SETTA 2020, Guangzhou, China, November 24–27, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 52–69. [Online]. Available: https://doi.org/10.1007/978-3-030-62822-2_4

[22] OpenAI, “GPT-4 Technical Report,” 2023. [Online]. Available: https://arxiv.org/pdf/2303.08774.pdf