NPS-CS-23-001

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

UPDATE ON MACHINE-LEARNED CORRECTNESS PROPERTIES

by

James Michael, Doron Drusinsky, and Matthew Litton

January 2023

Prepared for: Naval Information Warfare Systems Command (NAVWAR)

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
January 2023	Technical Report	01-06-2022	01-31-2023

3. TITLE AND SUBTITLE

UPDATE ON MACHINE-LEARNED CORRECTNESS PROPERTIES

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
	N0003922WX01748	
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)

James Michael, Doron Drusinsky, Matthew Litton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Naval Postgraduate School Monterey, CA 93943	NPS-CS-23-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Naval Information Warfare Systems Command 4301 Pacific Hwy., Bldg. OT7 San Diego, CA 92110-3127	NAVWAR	

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Statement A: Approved for public release. Distribution is unlimited.

13. SUPPLEMENTARY NOTES

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

14. ABSTRACT

This report details a novel method which has the potential for improving the U.S. Navy's ability to perform continuous assurance on autonomous and other cyberphysical systems. Specifically, this report presents a novel technique for simulation-driven data generation of explainable machine-learned correctness properties, called ML-assertions, for the purpose of subsequent runtime verification. The method brings the task of providing formal guarantees about the dependability of autonomous systems from the realm of doctoral-level experts into the domain of system developers and engineers. Preliminary experimentation demonstrates that ML-assertions can be utilized for behavior prediction in complex multi-agent systems, serving as a state-of-the-art method for conducting verification and validation on autonomous cyberphysical systems.

15. SUBJECT TERMS

assurance, machine learning, runtime verification, formal methods, cross entropy, autonomous systems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	UU	77

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
James Michael	831-656-2634

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Ann E. Rondeau
President

Scott Gartner
Provost

The report entitled "Update on Machine-Learned Correctness Properties" was prepared for and funded by Naval Information Warfare Systems Command (NAVWAR).

Distribution Statement A: Approved for public release. Distribution is unlimited.

This report was prepared by:

James Michael
Professor

Doron Drusinsky
Professor

LCDR Matthew Litton
United States Navy

Reviewed by:

Gurminder Singh, Chairman
Department of Computer Science

Released by:

Kevin B. Smith
Vice Provost for Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This report details a novel method which has the potential for improving the U.S. Navy's ability to perform continuous assurance on autonomous and other cyberphysical systems. Specifically, this report presents a novel technique for simulation-driven data generation of explainable machine-learned correctness properties, called ML-assertions, for the purpose of subsequent runtime verification. The method brings the task of providing formal guarantees about the dependability of autonomous systems from the realm of doctoral-level experts into the domain of system developers and engineers. Preliminary experimentation demonstrates that ML-assertions can be utilized for behavior prediction in complex multi-agent systems, serving as a state-of-the-art method for conducting verification and validation on autonomous cyberphysical systems.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

ACC	adaptive cruise control
ADAS	advanced driver assistance system
ADN	advance notice system
AFRL	Air Force Research Labs
AGCAS	Automatic Ground Collision Avoidance System
AI	artificial intelligence
AV	autonomous vehicle
C2	command and control
CA	California
CA DMV	California (CA) Department of Motor Vehicles
CE	cross-entropy
CFIT	controlled flight into terrain
CONOP	Concept of Operation
DL	deep learning
DAD	dimension of autonomous decision-making
DON	Department of the Navy
DNN	deep neural network
FM	formal methods
HOL	higher-order logic
JCS	Joint Chiefs of Staff
LKA	lane keeping assist
LSTM	Long Short Term Memory
ML	machine learning

MUM-T	manned-unmanned team
NASA	National Aeronautics and Space Administration
NAVCENT	U.S. Naval Forces Central Command
PDF	probability distribution function
PED	processing, exploitation, and dissemination
PRC	People's Republic of China
RDF	random decision forest
RM	runtime monitoring
RNN	Recurrent Neural Network
ROE	rule of engagement
SAE	Society of Automotive Engineers
SIGINT	Signals Intelligence
SUT	system under test
SVM	Support Vector Machine
TL	temporal logic
UAS	unmanned aerial system
V&V	verification and validation
USAF	U.S. Air Force
USN	U.S. Navy

Publications

This Technical Report is supported by the following publications which are included in the appendices to this report:

- D. Drusinsky, M. Litton, and J. B. Michael, "Lightweight verification and validation of cyberphysical systems using machine-learned correctness properties," *Computer*, vol. 55, no. 2, Feb. 2022. Available: https://doi.org/10.1109/MC.2022.3142829

  See Appendix B for full article.

- D. Drusinsky, J. B. Michael, and M. Litton, "Machine-learned specifications for the verification and validation of autonomous cyberphysical systems," in *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2022 [Online]., pp. 333–341. Available: https://doi.org/10.1109/ISSREW55968.2022.00089

  See Appendix C for full article.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Autonomous systems are poised to provide transformative benefits to society. Autonomous vehicles (AVs) have the potential to reduce the frequency and severity of collisions, enhance mobility for blind, disabled, and underage drivers, lower energy consumption and environmentally harmful emissions, and reduce population density in metropolitan regions [3]. In civilian aviation, increasingly autonomous systems could mitigate two of the most costly features of human pilots: the cost associated with training and paying highly skilled operators, and the reduced efficiency incurred by flight time limitations and crew rest requirements [4]. Additionally, autonomous air traffic management systems could reduce the cognitive burden on air traffic controllers by automating the monitoring and analysis of high volumes of data, alerting a human operator only when complex decisions must be made to mitigate risk [4]. Within the power distribution industry, innovations in "micro-grid" technology can allow better utilization of alternate energy sources while decreasing vulnerability to failure compared to current centralized power distribution, but such decentralization necessitates highly adaptive autonomous systems to carefully synchronize energy production and consumption [5]. Medical devices are currently designed to function for a large group of patients with similar conditions, but adaptive patient-specific algorithms could respond more effectively to individual patient needs, increasing lifespan and quality of life. In military aviation, Automatic Ground Collision Avoidance System (AGCAS) has saved the lives of eleven pilots by fusing terrain mapping and sensor data to autonomously maneuver to avoid ground contact if necessary [6]. Elsewhere in the military, autonomous weapons systems are increasingly critical to enhancing warfighting capability. In today's warfighting environment, autonomous vehicles, unlike unmanned vehicles which require uninterrupted interactive remote control, are becoming increasingly necessary for operation in denied, degraded, or otherwise unsafe environments. Also, the vast number of sensors streaming data from these systems is overwhelming the ability of human analysts to conduct processing, exploitation, and dissemination (PED) for intelligence purposes. Algorithms supporting autonomous operation, combined with advances in edge computing, can be used to lower the cognitive burden on human analysts by performing autonomous data triage for alerting

humans to data patterns of interest.

Despite the potential advantages to be derived from employing autonomous systems, evidence-based assurance of dependability is a key to obtaining users' trust in these systems. Such assurance of dependability is often lacking, leading to a lack of trust. When we look at the optimistic statements made within the last five years forecasting the rapid proliferation of AVs, it is clear that such optimism did not survive contact with reality. For these reasons, AVs have become emblematic of the issues associated with trust in autonomy, and therefore, we use them as the system of interest for our research and experimentation. For AVs, the primary barrier to garnering trust in their dependability is answering the question "Are they safe enough?" Answering this question poses two difficulties:

1. What is the standard for 'safe-enough' or otherwise 'dependable enough'?
2. How do we establish that AVs meet the stakeholders' standards or expectations?

Answering these questions is still a work-in-progress. According to a survey sponsored by Partners for Automated Vehicle Education (PAVE) in 2020, three in four Americans believe that AV technology is "not ready for primetime," and 20% of Americans surveyed believe they will never be safe [7].

For domains outside of the automotive industry such as space exploration and defense, the question to be answered is "Will these systems perform the mission well enough?" On the battlefield, warfighters must often prioritize the mission over and above their own safety, and they must have assurance that the humans and autonomous systems supporting them will behave in the same manner. Similarly, in order for commanders to deploy autonomous systems in war zones at scale, they must be able to trust that such systems will accomplish their intended mission alongside their human counterparts, while operating within legal and ethical bounds. Nevertheless, the lack of trust present in civilian society is equally present in the military community. Soldiers were surveyed concerning their willingness to deploy alongside autonomous systems in a manned-unmanned team (MUM-T). Many expressed skepticism of the perceived benefits of deploying in a MUM-T citing reasons such as potential deterioration of command authority, impact on unit cohesion, and the potential for increased emotional stress and burnout (an issue already identified as prevalent among drone pilots) [8]. Such perspectives highlight the difficulties in not only trusting autonomous

systems themselves, but also trusting the ways in which they interact with humans through human-machine teaming.

For our research project, we asked the question: **"How can we provide evidence for the dependability of autonomous systems, and best quantify the risks of deploying them?"**. In order to answer this question, we need to define what an autonomous system *should* do, and then determine the probability that it actually *does* do what it should. In order to monitor the correct behavior of an autonomous system, you must first define the behavior in a way that is monitorable. We made the focus of the research reported here on developing a new approach to defining what correct behavior looks like (the *should*), so that we can better understand a system's dependability (how often the system does what it should) and consider the risks associated with its failures (what are the consequences if the system *doesn't* exhibit correct behavior, and what caused the failure?).

Of course, simply identifying incorrect behavior is not the end-state; such identification should be used for correction of faults in order to increase aspects of dependability, such as reliability and safety. Such actions garner trust in autonomous systems by ensuring that they behave in a *bounded* way, even while the autonomous systems learn in their operational environment in an *unbounded* way. In the late 1990s, the U.S. Air Force (USAF) identified that controlled flight into terrain (CFIT), caused by pilot distraction, task-saturation, incapacitation, or G-Force-induced loss of consciousness, accounted for 26 percent of aircraft losses and 75 percent of all F-16 pilot fatalities [9]. Lockheed Martin, the National Aeronautics and Space Administration (NASA), and the Air Force Research Labs (AFRL) developed an autonomous system that continuously monitors aircraft navigation, telemetry, and terrain elevation data to determine whether ground collision is imminent. If the system detects such a condition, it autonomously executes an avoidance maneuver to prevent ground impact [9].

AV regulators may one day require that all autonomous vehicles be equipped with standardized supervisory systems (analogous to AGCAS) that attempt to guard against the most severe failures (e.g., collisions with pedestrians, high-speed collisions between other vehicles), even if the underlying autonomy becomes task-saturated, malfunctions, or stops functioning. The DoD could similarly require that lethal autonomous systems be equipped with a standardized automated supervisor to ensure that the underlying autonomous system cannot violate the rules of engagement (ROEs). Even if such a system does not override the

underlying human or autonomous control in real-time, recording incorrect behavior supports retrospective analysis along with more explainable and timely data to investigators and regulatory bodies than today's black box recorders. The European Union has taken a step in this direction, requiring the automotive industry to equip all new vehicles with Intelligent Speed Assist technology that can use passive or active methods to ensure that drivers obey speed limits [10]. But such rule-based systems do not account for the subtlety of defining correctness in the face of the myriad of exceptions to correct behavior that human drivers encounter. While it is generally correct not to travel faster than the speed limit, it may be *more correct* to accelerate past the speed limit to avoid a collision with a vehicle running a red light. In the absence of a human backup driver to overrule a system designed to enforce speed limits, a computer-controlled supervisor must also account for such complex and unpredictable interactions in its definitions of correct and incorrect behavior. Additionally, such a supervisory system must balance enforcing safety with non-interference with the underlying human or autonomous control. If such a system interferes too often, vehicles spend all their time safely on the side of the road, or humans users will find ways to disable the system. If it does not interfere when it should, it allows an autonomous system to harm itself and others. In either case, the measure of effectiveness of the supervisory system goes to zero.

The focus of this research is on developing a new approach to machine capture of correct and incorrect behavior of autonomous systems, in order to monitor their compliance with such specifications throughout development, testing, and during runtime. Our expectation is that this approach will aid in the verification and validation (V&V) of autonomous systems, increase trust in their dependability, and provide policy-makers, regulatory bodies, warfighters, and users of autonomous systems a better understanding of the risks and benefits of deploying them.

1.1 U.S. Navy Use of Autonomous Systems

Autonomous systems play an increasingly vital role in warfighting and operations other than war. They afford military forces the ability to speed up their decision cycles, as well as reduce risk-to-mission and risk-to-force, which in turn can provide a decisive advantage over adversaries. The U.S. Navy (USN) has recognized this need and responded with both

written strategy and operational employment. In July 2021, the USN released its "Strategy for Intelligent Autonomous Systems" [11], and in September 2021, U.S. Naval Forces Central Command (NAVCENT) established Task Force 59 with the mission of integrating unmanned systems and artifical intelligence with maritime operations [12].

Following the National Security Commission on Artificial Intelligence's March 2021 finding that, "America is not prepared to defend or compete in the AI era," Navy leadership has voiced the need to improve its acquisition and operationalization of autonomy [11]. However, the speed at which the U.S. DoD has historically conducted acquisition and sustainment is orthogonal to the speed at which autonomous solutions ripe for military use are being developed in the commercial sector. Smaller forces that have rapidly deployed intelligent, attritable machines, have been successful in recent conflicts against larger forces previously thought to be superior. In order to remain competitive, the DoD acquisition enterprise requires new approaches which can permit rapid acquisition and deployment of technology which learns and adapts in the unbounded environment of war, while still ensuring that it behaves within the bounds of military directives, national and international law, and public policy.

1.1.1 U.S. Navy Use Case: Maritime Intelligence Collection

Consider the scenario in which the U.S. military receives intelligence reports indicating that the People's Republic of China (PRC) is potentially mobilizing for an imminent invasion of Taiwan (Republic of China). The Joint Chiefs of Staff (JCS) require close-proximity confirmation of such mobilization in a matter of hours in order to plan a response. An increased threat to air operations prevents manned overflight, and widespread jamming makes it impossible to directly control unmanned systems due to the inevitable loss of control link. The deployment of a small swarm of undersea and surface autonomous systems would be well-suited to provide decision superiority through a multi-domain situational assessment. Due to storage limitations and the inability to exfil sensor data, the autonomous swarm will process data at the edge, returning an assessment of the PRC's actions and intentions to theater commanders and the JCS. How can key DoD decision makers trust that the swarm navigated to the correct locations to make relevant assessments? How can the commanders be sure that the sensors collected an adequate amount and diversity of data to make an accurate enough estimate? Did the systems collect signals from vessels used

for military mobilization, including both military vessels and merchant vessels which had been co-opted to transport troops and materiel? A commander cannot simply have human analysts re-do all of the analysis performed at machine speed by the autonomous system, so what will it take to trust the autonomous systems' output, which will be used as an input to a significant strategic decision?

The answer to these questions is *specifications*. Specifications, also known as *correctness properties* or *assertions*, describe in a machine-readable and unambiguous way when a system is behaving correctly, and identify if a machine is behaving incorrectly. In the example above, there might be specifications about what the maritime swarm *should* do in accomplishing its mission (these are also known as *liveness properties*) such as:

1. Collect at least 30GB of data on specific communications and radar emitters associated with enemy mobilization activity
2. Within two hours from launch, navigate to within 20NM of three key areas assessed to be associated with enemy mobilization activity

There would likely also be specifications about what the maritime swarm *should not* do in accomplishing its mission (these are also known as *safety properties*) such as:

1. Do not navigate within 3NM of other surface or subsurface vessels
2. Maintain stealthy electromagnetic signature by not radiating the automated identification system (AIS)

Because each specification acts as a representative for some measure of correctness, evaluating an autonomous system's behavior against the pre-determined specifications can provide evidence for their dependability and help humans know what level of trust to place in them. For example, a violation of liveness property #1 above which is concerned with amount of sensor data collected would indicate that a the system's estimate was based on an insufficient amount of Signals Intelligence (SIGINT), and a theater commander would have a reason to place less trust in the maritime swarm's assessment.

1.2 Traditional Assertions

Historically, specifications have been constructed by using formal, formalisms such as temporal logic (TL). TL and other higher-order logics (HOLs) have many advantages including the fact that they have well-defined syntax and semantics and they are amenable to automation such as computer-based runtime monitoring (RM) and search-based techniques such as falsification. Proponents of TL would also say that such logics are explainable because they can be translated to their natural language equivalent by experts in TL. However, to those without education, training, or experience in applying formal methods (FM), reading formal specifications is challenging, and even for experts, ensuring that the formal specification functions as intended on the system under test (SUT) is also a complex task. The only way to ensure that an assertion behaves as intended is to construct an accompanying test-suite, demonstrating how it is expected to behave over some range of conditions. Ultimately, writing formal specifications that accurately and fully capture requirements using TL is still a manual, time-intensive, and error-prone process even for FM experts.

In addition to the difficulty in implementing specifications using TL and other widely-used formalisms, the fact remains that TL is still an expressively weak language. Formally, TL is sub-regular in its expressive power; it belongs to a subset of regular languages known as *star-free regular requirements* [13]. Informally, TL specifications typically reason about the behavior of a system using a small number of observables. For example, a specification for a vehicle may say that it should never come within 10 meters of a pedestrian and would be written as:

$$\square \left(|location_{vehicle} - location_{pedestrian}| > 10 \right) \tag{1.1}$$

The $\square$ is translated as 'always', indicating the condition should hold across all time. This relatively simple assertion is easily translatable into TL, but what happens when additional real-world factors are considered? What happens when the pedestrian chooses to cross the street behind the vehicle which is stopped in traffic - a common circumstance in populated urban areas that is unlikely to result in a collision? What if another vehicle runs a red light, forcing the vehicle to maneuver closer than 10 meters to a pedestrian to avoid a collision? The TL formula will become much more complicated with additional nested clauses; not only will considering additional real-world factors make the formula more difficult to read, but it will become even more difficult to ensure that the specification captures the intended

behavior. Ultimately, the expressive power of TL is quickly exhausted. There is also the problem of not being able to manually or automatically generate specifications for all possible circumstances.

1.3 Machine-Learned Assertions

Traditional specifications are constructed by analyzing a system to determine what conditions make it behave correctly and incorrectly, and then writing a logic formula to represent and detect those conditions. We propose to use actual system behavior (labeled as correct and incorrect) as inputs to a machine learning (ML) classifier which can automatically extract the transformation from observed system behavior to a determination of correctness. Therefore, an ML-assertion is a ML classifier which is trained on data generated by the SUT representing correct and incorrect behavior for some behavior of interest.

Machine-learned assertions address three of the primary issues with the current approach to specifications:

1. Explainability
2. Expressiveness
3. Manual translation (i.e., from natural language to specification)

1.3.1 Explainability

While TL are technically explainable since they are based on a formal grammar, it is not always obvious, even to subject-matter experts familiar with the SUT, what behavior the formal specification is meant to capture. However, using ML-assertions allows one to use actual system behavior as the train/test set for the ML model which enhances the explainability of the assertions. The understanding of "What type of behavior will satisfy this assertion?" and "What behaviors will cause the assertion to flag?" is embedded directly into the ML data which represents correct behaviors and incorrect behaviors in the context of a specific assertion. To answer those questions, one has to simply observe the incorrect behaviors used for training/testing the ML model to know what behaviors will cause the assertion to flag. Likewise, observing the correct behaviors used for training will demonstrate, in a human-understandable way, the types of behaviors that will satisfy the assertion.

1.3.2 Expressiveness

TL is an expressively weak formalism, typically used to reason about simple systems or relatively simple subcomponents. It is not easily extended to capture the behavior of complex behaviors in equally complex environments such as the domain of AVs. However, the expressive power of ML-assertions extends to the entirety of the operational domain of the SUT. Because ML-assertions use actual system behavior to learn a determination of correctness or incorrectness, any behavior or interaction that the system can experience in its operational environment can be used to train the model.

Manual Translation

Current approaches use two levels of manual translation: natural language specification to TL, and TL to executable specification. Both of these processes are almost entirely manual, require high levels of expertise to perform, and are error prone. Even when part of either of these processes is automated, the toolchains used for automation themselves require verification which is also a non-trivial process.

1.3.3 Obtaining Data for Machine-Learned Assertions

A critical aspect of training ML-assertions is obtaining the proper amount and diversity of data to ensure accurate training and real-world generalization. For commercial autonomous systems such as AVs, this data contains information about the performance and behavior of the AV, other vehicles, and the surrounding environment in critical instances (e.g., crashes, near-misses, and disengagements). This data is the type necessary for building assertions, but is rarely released to the public by the manufacturers of AV systems due to litigation and intellectual property concerns. Likewise for military systems, necessary data would contain details about the behavior of autonomous systems when interacting with specific adversaries in wartime environments. However, such data is typically not stored onboard (due to storage constraints) or transmitted back to a central storage site (due to inadequate network bandwidth). When the data is recorded, it is often classified or proprietary and thus unavailable for research purposes.

While using real-world data is desirable, there are two advantages to the use of simulation generated data. The first advantage is that with simulation-based data it is possible to

generate data that can predict undesirable behavior before it happens, such as predicting a driverless AV is about to collide and alerting the human in the car. The second advantage is that certain types of rare undesirable AV behavior appear in only a few instances of street-data, such as a vehicle being squeezed into a collision by two other cars. Our suggested simulation based search methods is capable of generating a plurality of such scenarios.

The focus of this report is on presenting an approach for building explainable (i.e., data which makes the cause-and-effect relationship transparent between real-world behavior in the system's operational environment and whether such behavior is correct or incorrect), high-variance (i.e., data which captures the range of possible real-world behaviors) datasets through simulation which are suitable for constructing ML-assertions. The report describes our proposed approach to data generation and ML-assertion training along with the results we have so far obtained from experimenting with the approach.

CHAPTER 2:
Method

We developed a method for implementing our approach in which we simulate the behavior of an *ego* vehicle (the AV) in the presence of one or more *adversary* vehicles. Our goal is to identify both correct behavior (i.e., behavior that satisfies some specification) and incorrect behavior (i.e., behavior that violates some specification) for the training of ML-assertions. Rather than identifying these events through random sampling techniques such as Monte Carlo methods, we search for these events using the cross-entropy (CE) method.

There are five novel parts to our method of CE-based data generation for ML-assertions.

1. Search for *explainable* paths
2. Vanilla/Perturbation path pairs
3. CE of hybrid probability distributions
4. Integration of data-set variance into the CE cost function
5. Use of a weighted cost function

2.1 Search for Explainable Paths

An important factor to establishing and maintaining trust in increasingly autonomous systems is probing the *explainability* of the information and actions that they produce. While humans are able to articulate a logical sequence of events that causes them to make decisions based on various observables, the ability of autonomous systems to imitate cognitive reasoning and decision-making abilities of humans is based on trained models that are often less transparent and explainable. In this instance, where the goal is to train a model to recognize correct and incorrect behaviors involving AVs, there is a need to establish a foundation of explainability for the types of interactions between AVs and other actors that we study, because these interactions will ultimately produce the training and testing data for the ML models. In order to ensure our models are trained on data that is explainable (and translatable) in real-world interactions, we use data from the CA Department of Motor Vehicles (CA DMV).

In order to enable the safe testing and deployment of AVs in California, in 2014 the CA DMV began requiring companies testing AVs on public roads to submit disengagement reports. These reports record instances where the vehicle's autonomous control system is disengaged, either by the vehicle or by the human backup driver, and the CA DMV publishes the disengagement reports (as well as crash reports) from all companies testing AVs every year. Disengagement reports are of particular interest for a variety of reasons:

- Because most AV technology being tested on public roads is relatively mature, and hence unlikely to make lots of mistakes under normal driving conditions, disengagements usually indicate the occurrence of critical/unprecedented scenarios.
- Disengagements represent a failure of human-machine teaming, a critical component in developing trust in application of autonomous technologies. If the disengagement is initiated by the AV, it indicates an error in the underlying autonomy which reduces the human's trust in the system. If the disengagement is driver-initiated, it usually represents a preemptive takeover to prevent a potential perceived incorrect action by the autonomous system, also indicating lack of trust.

Therefore, when we look at what type of interactions for which we would want to train ML-assertions, we use natural language descriptions from CA DMV reports as a starting point to search for paths of interest. For example, on 9/24/2019, the AV company Zoox reported a disengagement by a backup driver due to "Prediction discrepancy; incorrect yield estimation for a vehicle cutting into the lane of traffic" [14]. Therefore, developing a dataset for the behavior of an AV interacting with nearby vehicles performing lane change maneuvers would contribute to an explainable ML-assertion. This is the example we take for our experimentation.

2.2 Vanilla and Perturbation Path Pairs

In the simulation-based search for correct and incorrect behavior, we attempt to rule out the types of behavior that would be obviously correct or incorrect, but are not likely to be seen in real traffic scenarios or are not useful for generating runtime assertions. For example, simply making a vehicle perform a high-speed collision with a AV travelling predictably on a highway is clearly an undesirable scenario, but is not an event of interest for ML assertions because there is little the AV can do about it. Similarly, saying that two vehicles changing

lanes on different streets is not helpful for studying their interactions in close proximity. Therefore, we take the philosophical approach that most collisions (or near-misses) result from only slight variations from otherwise acceptable behavior. The "normal" behavior that does not result in a collision (or violation of another condition of interest) we call a "vanilla" path. The behavior that is similar to the correct path, but does result in a violation, we call the "perturbation" path. The concept of vanilla/perturbation paths (i.e., showing incorrect behavior that is subtly different from correct behavior) helps to support both the realism and explainability (Section 2.1) of the paths ultimately used to train the ML-assertions.

2.3 Cross Entropy Search Using Hybrid Probability Distributions

For an overview of the Cross Entropy Method see Appendix A and associated references.

Typically, the CE method focuses on drawing samples from a single underlying probability distribution function (PDF) for the simulation of rare events or solving a combinatorial optimization problem [15]. In our case, we take the novel approach of optimizing a *hybrid* probability distribution consisting of the following underlying distributions:

- Multinomial (Categorical) Distribution: This distribution is used to represent the adversary's position
- Gaussian (Normal) Distribution: This distribution is used to represent the adversary's speed/acceleration

By using a hybrid distribution, CE can search through a wide range of paths that are variable in both position and speed, allowing the resulting paths to encompass the full range of driving behavior in order to find rare paths that may lead to collisions or near misses.

The suggested approach is not the same as searching for solutions that optimize one distribution and then searching, with that set, for solutions that optimize the other distribution. Rather, CE samples are drawn from the hybrid of both distributions, and the CE cost function is a hybrid cost function, inducing a hybrid score used for creating a new set of hybrid distributions.

2.4 Integrating Data-Set Variance into the Cross Entropy Cost Function

In Algorithm 2 (Appendix A), Step 4 states that each scenario is scored according to a cost function. In CE, the cost function is the method that determines which scenarios are used to update the probability distributions. In our implementation of the algorithm, each batch of CE (i.e., a certain number of CE iterations that occur prior to the gamma stabilizing) results in a single pair of paths - a perturbed path (which will serve as a 1-labeled path for the ML-assertion) and a vanilla path (which will serve as a 0-labeled path for the ML-assertion). Because we have a goal of ensuring the ML-assertion is trained on a wide variety of data to maximize generalizability of the assertion, we integrate the variance between perturbed paths into the cost function when generating a data set. Likewise, we also integrate the variance between vanilla paths into the cost function. In other words, because collisions can happen in a variety of different ways, we incentivize CE to discover collisions that differ from previous collisions to ensure that a variety of perturbed paths are represented in the training data.

2.5 Weighted Cost Function

Including accounting for variance within the dataset as described in Section 2.4 above, the cost function accounts for five total factors:

1. Categorical cost: prioritize shorter paths
2. Normal cost: ensure the normally distributed speed values are positive
3. Distance from Vanilla to Perturbation Path: ensure that the vanilla and perturbation paths are similar according to the philosophy discussed in Section 2.2
4. Variance: ensure that the next path added to the dataset maximizes the distance from the previously chosen paths (as discussed in Section 2.4)
5. Rigid Constraints: the path must obey certain rigid constraints (e.g., for a vanilla path, this would include not having collisions, for all paths it would include avoiding obstacles)

The first four factors above (categorical cost, normal cost, vanilla/perturbed distance, and variance) can each be assigned different weights according to the prioritization placed on

each component of the score. For example, placing a higher weight on the categorical cost indicates a preference for shorter paths over and above the other factors. Similarly, a lower weight for the vanilla/perturbed distance (which is a smaller negative number the closer they are to each other) would indicate a lower emphasis on the similarity between the vanilla (non-collision) and perturbation (collision) paths. The final factor (rigid constraints) is not weighted because it is either zero (all rigid constraints satisfied) or it is an extremely high value that dwarfs all the other factors, indicating that rigid constraints were violated.

2.6 Implementation

The technique and novel components described above are implemented in 8,171 lines of code (in the latest version) which consists of the following components:

1. A CE generator that implements the CE method in Java and produces paths to be executed by the simulator.
2. A scenario executor which takes the paths generated by CE and executes/analyzes them in CARLA. The scenario executor is written in Python and utilizes multiple Python libraries as well as CARLA's Python API.

The code base is continually evolving as the research progresses.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Results and Current Findings

This chapter discusses the results and current findings from completed experimentation with CE-guided simulation in CARLA. The results are continually evolving as new experiments are conducted.

3.1 Data Generation

The approach successfully generated pairs of paths ($< vanilla, perturbation >$) which were similar to each other, and where one path (perturbation) resulted in a collision, and the other (vanilla) did not.

One such set of paths can be seen in Fig. 3.1. This visualization clearly depicts the similarity in position (the Multinomial part of the hybrid distribution discussed in Section 2.3) between the vanilla and perturbed paths prior to the collision. The perturbation path collides with the ego vehicle, hence its path is truncated at the point of collision (the trajectories do not intersect because the trajectory shows the position of the center of the vehicle only). The vanilla path does not collide with the ego vehicle. Note how close the vanilla and perturbation paths are, a direct result of the Hybrid-Pair search approach discussed earlier.

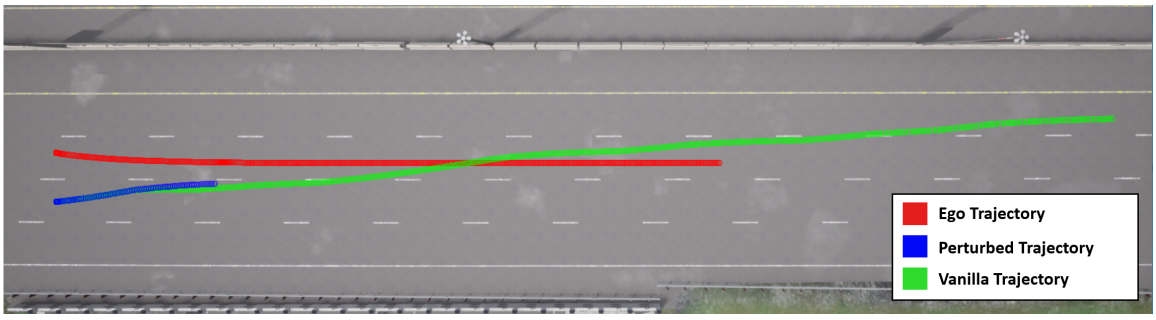


Figure 3.1. A visual depiction of a vanilla and perturbation path with the ego vehicle's trajectory. Note the similarity of the position of the vehicle in the vanilla and perturbed path prior to the collision.

Likewise, Fig. 3.2 displays timestamps overlaying the vehicles' positions every two seconds to aid in visualizing their relative speeds. This visualization clearly depicts the similarity in speed (the Gaussian part of the hybrid distribution discussed in Section 2.3) between the vanilla and perturbed paths prior to the collision. As before, the fact that the speed along the vanilla and perturbation paths are similar, is a direct result of the Hybrid-Pair search approach.

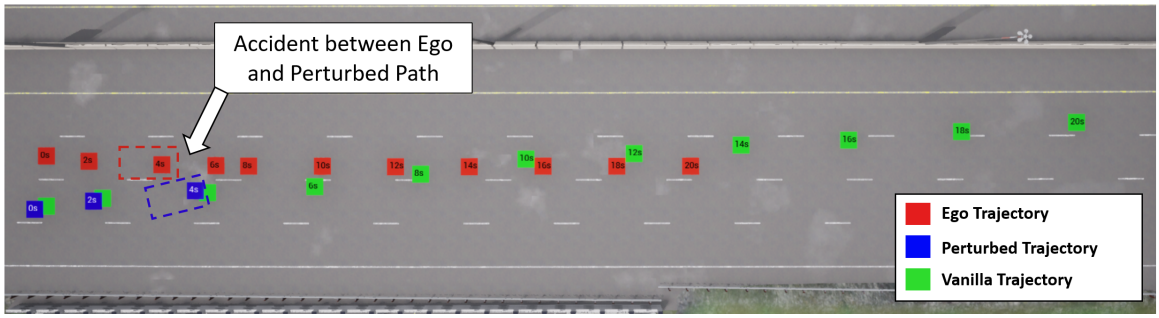


Figure 3.2. A visual depiction of a vanilla and perturbation path with the ego vehicle's trajectory. Note the similarity of the position between the vanilla and perturbed paths at 0s, 2s, and 4s which indicates similarity between their speeds. The slightly higher speed of the vanilla path allows the vehicle to change lanes safely in front of the ego while avoiding a collision.

Finally, Figures 3.3 to 3.6 depict a photo-realistic sequence from CARLA displaying the first four seconds of the simulation, until the accident at $t = 4s$. The similarity between the paths is evident; they diverge at Fig. 3.6 where the perturbed path takes the adversary vehicle far enough into the ego's lane to cause a collision, while the vanilla path doesn't encroach far enough to cause an collision.

(a) Vanilla

(b) Perturbed

Figure 3.3. $t = 1.0s$

(a) Vanilla

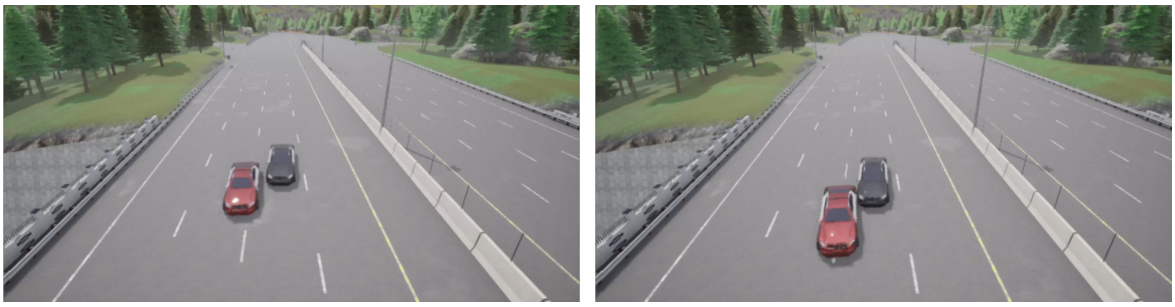
(b) Perturbed

Figure 3.4. $t = 2.0s$

(a) Vanilla

(b) Perturbed

Figure 3.5. $t = 3.0s$



(a) Vanilla (b) Perturbed (Collision)

Figure 3.6. $t = 4.0s$

Data Variance

As discussed in Section 2.4, obtaining a high variance dataset is the key to training ML-assertions capable of effectively generalizing to data not seen in the training set. For this dataset, we generated 94 ($< vanilla, perturbation >$) pairs (i.e., 94 0-labeled sequences, and 94 1-labeled sequences). For variance, we focus on the variance of the perturbed paths prior to the collision.

Fig. 3.7 depicts a subset of the 94 perturbation paths to show the variance between their positions prior to their collisions.

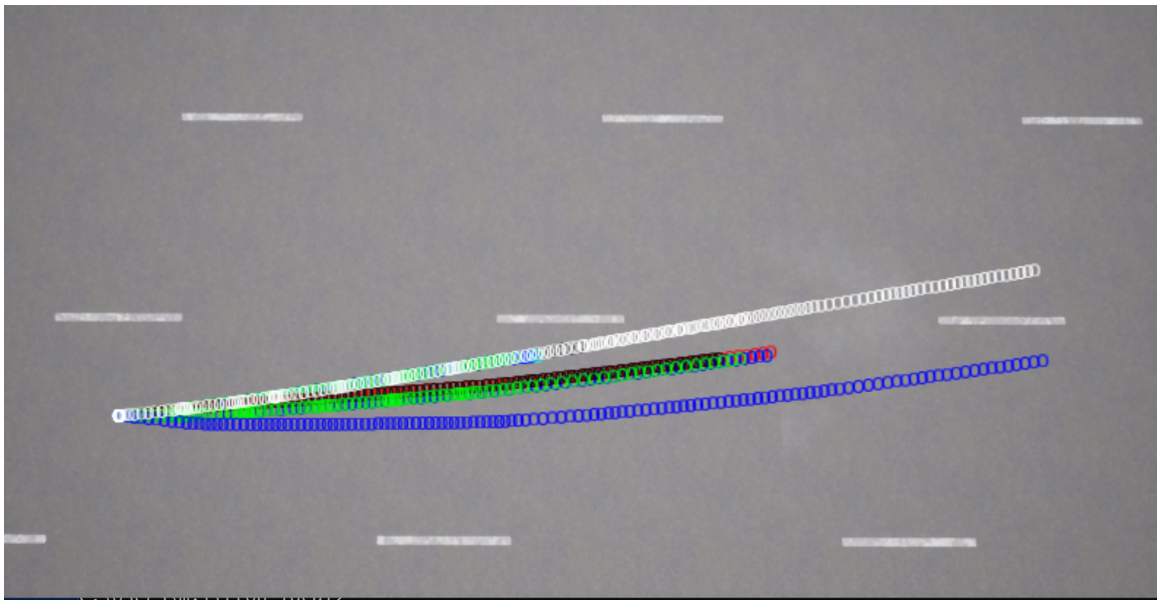


Figure 3.7. A depiction of the positions of 10 different perturbation paths prior to their accident. Note that this shows only a portion of the variance (position) - it does not depict variance in speed.

3.2 Machine Learning

In order to test the ability to train ML models from the data gathered above, we first pre-process the data by taking the following steps:

- For each ($< vanilla, perturbation >$) pair, identify the time at which the collision occurs in the perturbation path. This time is denoted as $t_{collision}$.
- Starting at $t_{collision}$, go back in time some number of seconds (X) prior to the collision ($t_{collision-X}$).
- Then take a series of time-sequenced features at some sampling rate (R) for some number of seconds (Y) before that ($t_{collision-(X+Y)}$).
- Take features from the corresponding vanilla path at exactly the same intervals and frequency as the perturbed path.

For example, if the accident occurred at $t = 5s$, you may choose to set $X = 1$, $Y = 2$, and

$R = 0.5$. This would mean that the features taken from the perturbed and vanilla paths would be a sequence of features starting at $t = 2s$ and ending at $t = 4s$, sampled every $0.5s$. If 20 features are collected at every timestep (Table 3.2), then there would be 100 total features for each path; the row of 100 features taken from the vanilla path is labeled as a '0', and the row of 100 features taken from the perturbed path is labeled as a '1.' The the ML classifier is trained on this data.

Essentially, the classifier is being trained to answer the question (with the above parameters), "Given data from 3 seconds before the potential collision up to 1 second before the potential collision, can I predict whether or not there will actually be a collision?" Intuitively, you can get a sense for this problem by looking back at the images in Section 3.1. A human performing the same task would be instructed to look at Figures 3.3, 3.4, and 3.5, and asked to predict whether or not there will be a collision in Fig. 3.6.

We experimented with several different ML and deep learning (DL) models on the dataset, and found that random decision forests (RDFs) performed well without significant hyper-parameter tuning and were easy to train. For comparison, we also tested the performance of a Support Vector Machine (SVM) on the same dataset. The SVM was tested because it often performs well on datasets where the number of features is high relative to the number of samples as is the case here. Nevertheless, it did not perform as well as the RDF. Finally, we compared the performance of the ML models to a DL solution. Though DL models are often more time-consuming to train, we wanted to weigh the tradeoff between training time and accuracy. For this experiment, we chose a Recurrent Neural Network (RNN), which is a type of deep neural network (DNN) that keeps track of state, allowing it to build connections between layers to learn sequences in time-series data. Traditional RNNs can suffer from vanishing (or exploding) gradients due to extremely large or small weights at each layer. To solve this issue we used a type of RNN, called a Long Short Term Memory (LSTM), which is better able to learn long-term dependencies and find patterns across sequences in time for many domains including machine translation, language processing, and weather forecasting. We see that the LSTM doesn't perform as well as the RDF though it performs better than the SVM, but it requires several minutes to train on a high-performance computing cluster making it less well-suited for conducting ML at the edge, a planned focus of future research. The results from each model are shown in Table 3.1:

Table 3.1. Machine Learning Results

	RDF	**SVM**	**LSTM**
Accuracy	0.92	0.66	0.82
Average Absolute Error	0.08	0.34	0.18
True Positive Rate (TPR)/Recall	0.91	0.91	0.89
True Negative Rate (TNR)	0.93	0.44	0.74
False Negative Rate (FNR)	0.09	0.09	0.11
False Positive Rate (FPR)	0.07	0.56	0.26
Positive Predictive Value (PPV)	0.91	0.58	0.77
F1 Score	0.91	0.71	0.83

Fig. 3.8 shows the structure of the LSTM model used for the results reported in Table 3.1.

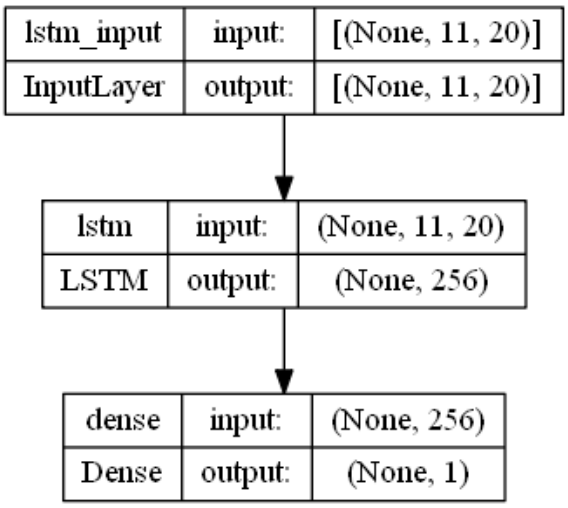


Figure 3.8. The structure of the Long Short Term Memory model used for experimentation.

Table 3.2 shows the features used for all previously discussed ML/DL models (i.e., RDF, SVM, and LSTM) discussed in the experiments. The 20 features shows are sampled at each time step.

Table 3.2. Features Used for Machine Learning

distance	distance between the ego and adversary
angle	angle of the adversary relative to the ego
ego_vel_x	ego velocity in the x direction
ego_vel_y	ego velocity in the y direction
ego_vel_z	ego velocity in the z direction
ego_accel_x	ego acceleration in the x direction
ego_accel_y	ego acceleration in the y direction
ego_accel_z	ego acceleration in the z direction
ego_ang_vel_x	ego angular velocity in the x direction
ego_ang_vel_y	ego angular velocity in the y direction
ego_ang_vel_z	ego angular velocity in the z direction
adv_vel_x	adversary velocity in the x direction
adv_vel_y	adversary velocity in the y direction
adv_vel_z	adversary velocity in the z direction
adv_accel_x	adversary acceleration in the x direction
adv_accel_y	adversary acceleration in the y direction
adv_accel_z	adversary acceleration in the z direction
adv_ang_vel_x	adversary angular velocity in the x direction
adv_ang_vel_y	adversary angular velocity in the y direction
adv_ang_vel_z	adversary angular velocity in the z direction

3.3 Discussion of Results

Therefore, the answer to the question "Given data from 3 seconds before the potential collision up to 1 second before the potential collision, can I predict whether or not there will actually be a collision?" is "Yes, with over 90% accuracy." More generally, this technique demonstrates that ML-assertions can be utilized for behavior prediction in complex multi-agent systems such as self-driving vehicles or an unmanned maritime swarm described in Section 1.1.1.

The question then becomes, "what then?" Now that we have these assertions, what do we do with them? There are two types of RM which could be conducted to utilize these assertions: online and offline.

3.3.1 Offline Runtime Monitoring

Offline RM is conducted retroactively, usually on log files recorded from the SUT. One of the most significant challenges to the widespread deployment of AVs is establishing liability for crashes and setting insurance rates. For human drivers, this is typically done by looking at a driver's pattern of behavior to determine whether there is a high risk (and therefore should demand higher insurance rates), or whether it represents a lower risk of collisions. When a collision does occur, the insurance company uses the data available to establish financial liability. Specifications could play a key role in establishing risk metrics for AVs and assigning liability for collisions. For example, suppose an insurer wants to determine how risk-taking an AV's driving behavior is; the insurer could train a ML-assertion on naturalistic driving data that demonstrates human driving behavior, and then use log files from AVs to determine how often the AVs exhibits risky driving beavhior to better assess appropriate insurance rates.

Similarly, suppose an AV swerves suddenly on a highway to avoid a vehicle encroaching in its lane, and strikes a guardrail. The insurer could run the assertion discussed in Section 3.2 above to show that 1 second before the AV swerved, a collision was imminent and the vehicle encroaching in the AV's lane was at fault and should bear the financial liability. We do not attempt to weigh in on the discussion of *who* exactly is liable for a fully autonomous system (e.g., the manufacturer, the owner), but many such assertions could be run offline for retrospective analysis to provide key risk metrics and behavior analysis to insurers,

policymakers, vehicle manufacturers, regulators, and other stakeholders.

3.3.2 Online Runtime Monitoring

Offline RM is often the preferred approach for safety-critical systems because the computational power required to analyze the specifications in real time can interfere with the operation of the underlying system. However, advances in edge computing, networking, and storage have mitigated this issue in many types of systems including AVs. Online RM can enable systems to respond to assertion violations in real time which in turn can aid in advancing cooperative autonomy, also known as human-machine teaming.

As autonomous systems continue to advance and proliferate, humans who interact with such system need to understand the tasks that can be delegated to them, along with the risks of delegating such tasks. When humans place too much trust in the dependability of an autonomous system to perform a task, the human is not likely to maintain an adequate level of situational awareness in order to be able to intervene and take control such as in an emergency. When humans place too little trust in autonomous systems, the cognitive burden (e.g., task overload) on humans is increased and the intended benefits of autonomy are reduced. Therefore, understanding what tasks can be delegated to an autonomous system, the risks of delegating such tasks, and how that risk changes over time is critical to enabling humans to effectively trust autonomous systems.

This paradigm is well-illustrate by the current state of the automotive industry. The Society of Automotive Engineers (SAE) defines six levels of driving automation ranging from Level 0 (no automation) up to Level 5 (full autonomation) [16].

- Level 0 - No automation: The human performs all driving tasks.
- Level 1 - Driver Assistance: The vehicle has a single automated system (e.g., conventional cruise control).
- Level 2 - Partial Automation: Also known as advanced driver assistance system (ADAS). The vehicle can automate aspects of longitudinal and lateral control (e.g., through lane keeping assist (LKA) and/or adaptive cruise control (ACC)), but the human is still responsible for monitoring the environment and regularly making or correcting control actions.
- Level 3 - Conditional Automation: The vehicle monitors the driving environment and

perform most driving tasks, but human backup is still required.

- Level 4 - High Automation: The vehicle performs all driving tasks under specific circumstances, and either in-vehicle or remove human override is still an option.
- Level 5 - Full Automation: The vehicle performs perception and control under all conditions. Human attention and interaction is not required.

Nearly all new vehicles in 2023 can be categorized as Level 2, in which the vehicle can perform steering and acceleration, but the human is solely responsible for monitoring the environment and must stay consistently engaged in the vehicle's control. However, while the transition from Level 0 through 2 has proceeded relatively smoothly, and has even been welcomed by users and policymakers alike, the potential transition from Level 2 to Level 3 has been widely contested and the subject of much debate. This is primarily because in Level 2, the human monitors the driving environment and is well-equipped to takeover at any time. In Level 3, the autonomous system monitors the driving environment, and the challenge of alerting a human concerning the need to take over control, provide awareness of the current state of the environment, and detect the need to do so far enough ahead of time, is a significant challenge. Each of these factors: when to alert a human driver, whether or not the driver is professionally trained as an AV backup driver, what to alert them about, how many alerts to provide, conditions under which they should be alerted, and time between alerts and takeover are all factors that can (and should) be encoded in specifications.

ML specifications, described in Section 3.2 can be well-suited for this purpose with additional human factors analysis. Suppose an AV manufacturer who is deploying Level 3 vehicles wants to warn a backup driver when a nearby vehicle is making an unsafe lane change. In that case, the AV would need to first detect the fact that a nearby vehicle is making an unsafe lane change, and then it would need to alert the backup driver through visual or haptic feedback in enough time for the driver to safely gain situational awareness, and resume control to avoid a potential accident. In the previously described ML-assertion, the lead time between prediction and potential collision is only one second, which would certainly not be enough time from alert to takeover, but the exact amount of time required is outside the scope of this research.

As a quick experiment, we took the same preprocessing steps as described in Section 3.2 above but set $X = 2$ (attempt to predict whether or not there will be a collision two

seconds prior rather than one second prior). Because many of the collisions occurred relatively quickly, this yielded a dataset of 50 ($< vanilla, perturbation >$) pairs (i.e., 50 0-labeled sequences, and 50 1-labeled sequences). The classifier still performs better than random guessing, but as expected, and due to the realistic similarity between the vanilla and perturbed paths, the accuracy of the prediction drops. Due to the decrease in data-set size, it is difficult to make a direct comparison to the RDF performance reported in Table 3.1, yet the results are shown in Table 3.3:

Table 3.3. Random Decision Forest Performance: Two Seconds Prior

Accuracy	0.68
Average Absolute Error	0.32%
True Positive Rate (TPR)/Recall	0.76
True Negative Rate (TNR)	0.61
False Negative Rate (FNR)	0.24
False Positive Rate (FPR)	0.39
Positive Predictive Value (PPV)	0.59
F1 Score	0.67

3.3.3 AI On the Edge

The research described in this report is focused on the extraction of ML training and testing datasets from simulations. The sophisticated CE search based extraction process consumes non-trivial resources, both in time and computational power. Therefore, this process is not expected to perform well on the edge. Nevertheless, the ML classifiers trained on the resulting datasets can execute on the edge. Moreover, the abovementioned CE search-based dataset generation process can be refined to generate datasets suitable for training classifiers applicable to important on-the-edge applications, such as advance notice systems (ADNs). A relevant example of an ADN is a system that alerts an inattentive human inside an AV when that AV is heading towards a collision (see Section 3.3.2). ML classifiers for ADNs

train with datasets in which vehicle paths end (i.e., are truncated) several seconds before the point of collision. To that end, when applying the CE search process to the creation of training datasets for ADNs, the CE cost function is tuned such that the resulting HybridPair paths are long enough to contain meaningful training paths even when truncated.

While the CE process for training data-set extraction is not suitable for edge computations, some of the classifiers used in this research are lightweight enough to be trainable on the edge (e.g., RDF); this fact enables a mixed training approach, as follows. A classifier is initially trained using datasets generated from simulations. Thereafter, whenever the edge device detects an incident labelled as a collision, it is added to the training set and the classifier is retrained - on the edge.

3.4 Challenges

Though we believe this method provides a useful, state-of-the-art method for conducting V&V on autonomous cyberphysical systems, there are multiple challenges associated with its use.

3.4.1 Challenges with Simulation

Conducting a large number of visually and physically realistic simulations is compute and time intensive. The more realistic the simulator is to the system's real-world environment, the more resource-intensive it is to run such simulations. For example, CARLA provides visually realistic renderings using ray-tracing as well as physically realistic behaviors to include engine and road friction calculations. While such realism enhances the ability of simulated findings to translate to real-world outcomes, it also makes running hundreds of thousands of simulations relatively slow.

To generate the 94 vanilla/perturbed pairs discussed above, it took approximately 108 hours (4.5 days) to run the number of simulations required. Other domains may have simulations that run significantly faster or slower depending on the system being simulated. We are investigating ways to parallelize the simulations and/or use cloud computing resources to speed up this process.

3.4.2 Challenges with Variance

The ability of a ML classifier to generalize from the training data is not only dependent on the amount of data, but also on the variance within the data set. Even if the classifier learns the mapping of inputs to output, if the inputs do not encompass the variance that can be seen in the system's real-world operational environment, the classifier will not be able to generalize well to unseen data.

Even though Fig. 3.7 displays some differences in perturbed path positions, many of the paths are very close and the overall variance is low. We are working to add additional complexity to the environment to increase the dataset variance.

CHAPTER 4:
Some DoD Technology-Transfer Opportunities

In 2021, the autonomy portfolio manager at the Office of Naval Research and the warfare integration branch (N9IX) at the Deputy Chief of Naval Operations for Warfighting Requirements and Capabilities sponsored a report called "Dimensions of Autonomous Decision-making: A First Step in Transforming the Policies and Ethics Principles Regarding Autonomous Systems into Practical System Engineering Requirements" [17]. This study attempts to address the question:

> How can DOD incorporate the factors that must be considered when transferring decision making ability from humans to autonomous systems into a framework to guide the development, fielding, integration, and employment of militarily effective, legally compliant, and ethically conforming Intelligent Autonomous Systems?

As part of the research conducted by the Center for Naval Analyses (CNA), they identify 13 "categories of potential risk that one should consider before transferring decision-making capabilities to an intelligent autonomous system." The categories are referred to as dimensions of autonomous decision-making (DADs). Each of these categories is meant to encompass the factors that should be considered by the acquisition community and military commanders before transferring decision-making capabilities to an autonomous system to ensure its use is "legal, ethical, and militarily effective." [17].

One identified DAD is:

> **DAD #9: Human-machine teaming**
> *Does design of the human-machine team (particularly when the use of force is involved) leverage the human's superior ability of exercise judgement in the use of IAS in order to reduce the risks associated with its use?* [17]

If the intelligent autonomous system (IAS) is equipped with the ability to deliver kinetic or nonkinetic effects, program managers should use this question to identify areas of risk and

consider if/how those risks should be mitigated/accepted. For example, for an armed MQ-9 unmanned aerial system (UAS), one risk identified might be that an adversary could jam the system's legitimate command and control (C2) link, and inject a spoofed command to launch munitions. To mitigate that risk, the program manager may define a requirement that says, "If the system experiences link loss due to jamming, it will immediately disable all weapons systems and return to base." This requirement ensures that the potential legal and ethical risk identified in the IAS is reduced by preventing unintended weapons deployment in the absence of legitimate operator approval.

The question then becomes, "How does one define, and then how can one **verify**, that an autonomous system is used in a manner that is legal, ethical, and militarily effective?". The authors of the report conclude that broad requirements similar to that stated above must be translated into a representation that is measurable and testable [17]. In fact, the authors go as far to say that the formulation of measurable and testable requirements is the key factor for the DoD to implement their commitment to the legal, ethical, and responsible use of artificial intelligence (AI). We assert that the foundation for ML-assertions laid out in this report and other previously published works could serve as a mechanism for translating these requirements into a measurable and testable form. Such specifications would enable DoD program managers and their assurance teams to conduct V&V on autonomous systems during acquisition and sustainment, as well as for warfigthers to assess the risk associated with the use of such systems.

This research is still nascent and ripe for future work that has applications for multiple speciality areas of Computer Science and Engineering.

First, this research can be extended to other autonomous systems outside of AVs. There are many realistic simulators for other vehicles to include airborne, surface, and subsurface systems. In addition, non-vehicle autonomous systems such as medical devices, power-grid controllers, and intelligent decision-making and battle management systems.

Second, this research focuses primarily on the training and testing of safety-focused ML-assertions because these are the specifications of primary interest to the AV V&V community. Other communities, such as the Department of the Navy (DON), would be more interested in liveness properties (i.e., what the system *should* do rather than what it *should*

not do) for the autonomous systems they want to integrate in military Concept of Operations (CONOPs).

Third, as increasingly autonomous systems are deployed, the machine learning portion of this technique can be applied to the data generated from their operation as it becomes available. We expect the amount of real-world data available from autonomous systems for research and analysis to significantly increase over the next several years.

Finally, this research aligns directly with multiple recent USN-driven efforts to push AI to the edge and utilize cloud computing resources. Just as our research focused on using ML to analyze large volumes of operational data for V&V, Falkonry AI is taking advantage of edge processing and cloud computing resources for anomaly detection on Navy vessels [18]. Amazon also provides multiple use cases which take advantage of increasingly prevalent edge-to-cloud linkages, one of which focuses on an ADAS for pedestrian avoidance (see Use case 2 in [19]).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A:
Cross Entropy Algorithm

The Cross Entropy Method is an algorithm primarily used for combinatorial optimization and rare even simulation. For optimization, it can be applied as a generic method for solving NP-hard problems. For rare event simulation, it can be used to estimate the probability of rare events for guaranteeing adequate performance of various types of systems (to include autonomous systems) [15]. For example, a relatively mature AV may experience an at-fault collision with another vehicle every 100 million miles. Rather than driving an AV 100 million miles to study the collision (or simulating 100 million miles of driving), CE can be used to *search* through various parameters of the vehicle's perception, control, and environment that would be most likely to cause the collision, and quickly converge on the parameter distributions of interest.

An excellent resource on the theory and application of the CE method can be found below:

- P.-T. de Boer, D. P. Kroese, M. Shie, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, Feb. 2005. Available: https://doi.org/10.1007/s10479-005-5724-z

At its most basic level, the CE method is an iterative procedure consisting of two phases:

Algorithm 1 Basic Cross Entropy Method [15]

1. Generate a random data sample (trajectories, paths, etc.) according to some specified mechanism
2. Update the parameters of the random mechanism based on the data to produce "better" samples in the next iteration

In our implementation of CE, the "random mechanism" is a probability distribution, specifically the *hybrid distribution* discussed in Section 2.3. The parameters are updated to produce "better" samples according to scores assigned to the randomly generated paths by the score function discussed in Section 2.5.

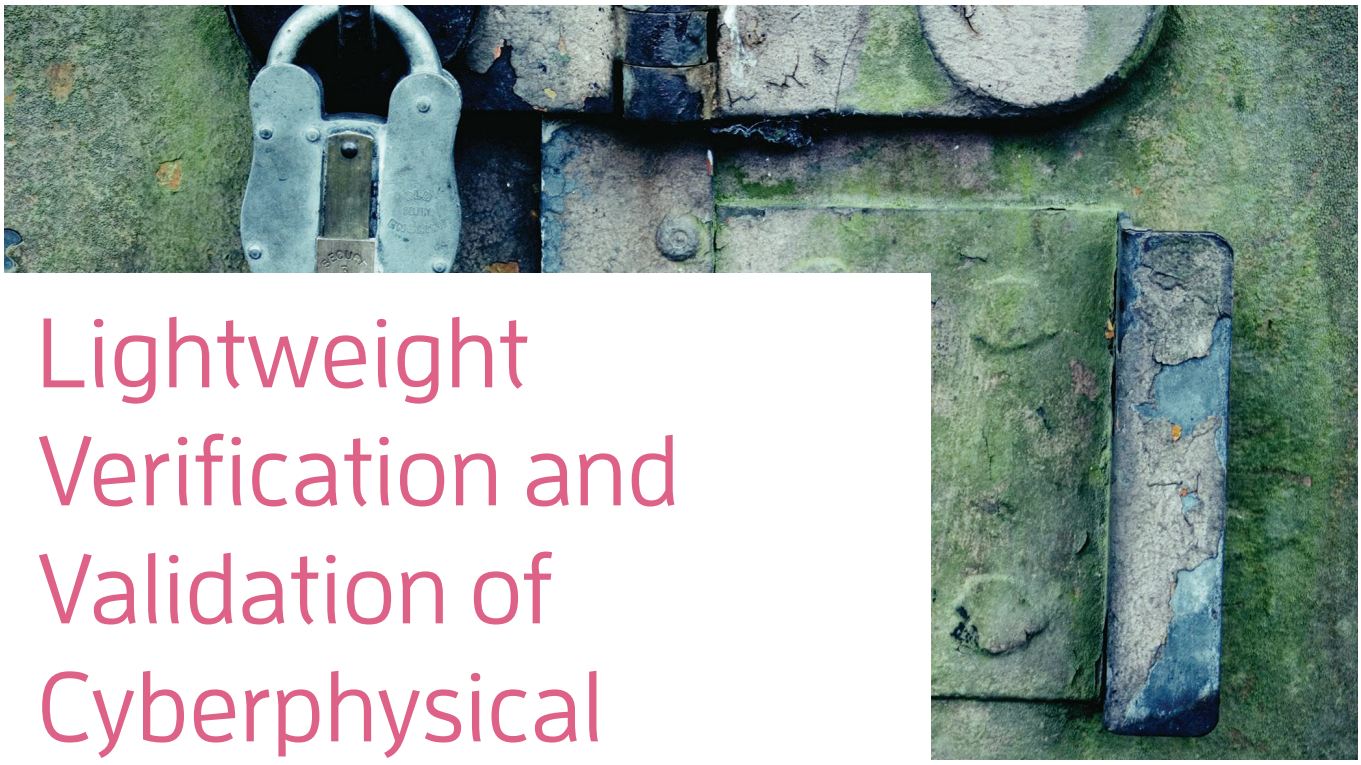
Therefore, the algorithm used in this specific case can be described as shown below:

Algorithm 2 Cross Entropy Method used in CARLA

1. Initialize the adversary's categorical (position) and normal (speed) probability distributions (called the *hybrid distribution*)
2. Generate a sequence of positions and speeds for the adversary based on the hybrid distribution
3. Execute the scenario in the simulator using the AV and the adversary(ies)
4. Score the scenario according to some criteria
5. Once a pre-defined number of scenarios has been scored, sort them according to score
6. Update the hybrid distribution based on the scenarios with the lowest scores
   NOTE: The lowest scored scenarios are called the *elite set*, and typically consist of a percentage of all scored scenarios between one and 10 percent.
7. If the stopping criteria is met, end. Otherwise, use the updated distribuions and reiterate from Step 2
   NOTE: The stopping criteria is often that the *gamma element* (the highest scored scenario in the elite set) has remained constant for a certain number of iterations, indicating that the distributions have stabilized.

APPENDIX B:
Lightweight Verification and Validation of Cyberphysical Systems Using Machine-Learned Correctness Properties

The following article was published in IEEE's *Computer* Magazine, the flagship publication of the IEEE Computer Society. This publication contains peer-reviewed articles written for and by professionals representing the full spectrum of computing technology from hardware to software and from current research to new applications.



Lightweight Verification and Validation of Cyberphysical Systems Using Machine-Learned Correctness Properties

Doron Drusinsky, Matthew Litton, and James Bret Michael, Naval Postgraduate School

Applying classical formal methods to cyberphysical systems is inherently difficult, but using machine learning for lightweight verification and validation can provide assurances for the operation of such systems, even at the edge.

Digital Object Identifier 10.1109/MC.2021.3132469
Date of current version: 14 February 2022

In 2021, two of the authors (Michael and Drusinsky) conducted a virtual roundtable with several subject matter experts (SMEs) to explore the efficacy of applying formal methods in the verification of systems that contain physical and software components, also known as cyberphysical systems (CPSs).[1] All the experts acknowledged the lack of widespread adoption of formal verification (FV) in software design in general and CPSs in particular. Interestingly, rather than advocating for the immediate adoption of classical formal methods for CPSs, such as autonomous vehicles, nearly all the experts proposed alternatives to those practices, including theorem proving and exhaustive model checking.

With the increasing complexity and proliferation of CPSs in safety-critical systems, including

EDITOR JAMES BRET MICHAEL
IEEE Senior Member; bmichael@nps.edu

those that utilize machine learning (ML) in their operation, the experts admit that such systems are often not amenable to traditional FV and instead advocate for the use of more pragmatic approaches. This article begins with a brief overview of the issues related to applying classical FV to CPSs, and possible alternatives. We suggest such an alternative: using ML classifiers for lightweight verification and validation (LV&V). To help advance this approach, we outline the needed areas of active research. Our aim is to apply this approach to conducting LV&V on edge devices.

A QUESTION OF TRUST

The ability to predict the acceptance and integration of CPSs into our societal norms has proved to be difficult, especially for CPSs that are safety critical. In April 2019, at Tesla's "Autonomy Day," Elon Musk predicted that the company would have millions of fully autonomous vehicles operating on public roads by the end of 2021; Uber and others made similar projections.[2] In fact, companies developing autonomous driving have demonstrated much of the technological maturity needed to make Musk's predictions come true.

In August 2021, a U.S.-based company demonstrated level 4 autonomous operation (without a safety driver, remote operator, and any other form of human intervention) of a semitrailer truck in heavy traffic on a Chinese highway.[3] Yet, the National Highway Traffic Safety Administration has not allowed such unconstrained testing on U.S. highways. This lack of progress suggests that while the technology is quite advanced, there are more factors at play than just feasibility and maturity. Then what is the holdup? Although there is certainly some additional technological development required, the crux of the issue lies in the willingness of humans to place trust in the dependability of machines.

Policy makers and regulatory bodies continually point to the need to quantify the safety risks of operating such systems, understand the safeguards that are in place, and clearly communicate the risk assessments and mitigation strategies to decision makers, many of whom often lack understanding of the underlying technology. Also, groups such as insurers and manufacturers demand assurances about the design and implementation of underlying vehicle control algorithms as well as the reliability and performance characteristics of the sensors, actuators, computing devices, and other components comprising the systems. In addition, there is a need for information about the behavior of the systems in different modes of operation in a wide variety of environments and environmental conditions (for example, mixed traffic on a freeway at night with icy road conditions). While some stakeholders prioritize maximizing safety, performance, resilience, technological advancement, and profits, Benjamin Kuipers observes that "the value of trust is difficult to fit into this maximization framework, so it is tempting to leave it out."[4] For stakeholders to get onboard with the widespread deployment of such technology, proven trust will have to be the centerpiece and not an afterthought.

However, Nancy Leveson states that we need to assume that "highly reliable software is not necessarily safe. Increasing software reliability or reducing implementation errors will have little impact on safety."[5] Leveson also posits that "systems will tend to migrate toward states of higher risk. Such migration is predictable and can be prevented by appropriate system design or detected during operations using leading indicators of increasing risk."[5] In the LV&V approach that we introduce, we propose to go beyond purely reliability-based assurances by conducting monitoring to verify that the bounds introduced in design also hold in operation. Formal methods, classical and otherwise, provide well-tested ways to perform oversight of safety-critical systems and ensure their correct operation. Bringing high-risk CPSs into widespread deployment will require V&V techniques that are themselves dependable and trustworthy and that can provide human-understandable assurances concerning their correct operation, all while enjoying levels of machine intelligence similar to the operational software.

CLASSICAL FV TECHNIQUES

Efforts to formally verify properties of computing systems gave birth to the field of formal methods and a subset, FV, that uses mathematics and logical expressions to provide rigorous assurances about correct operation. Until the 1990s, FV was solely an academic exercise and not adopted in any computing industry; the high cost associated with performing rigorous analysis was seen as an unnecessary expense and a barrier to getting products to market. However, Intel incurred a significant revenue loss in 1994 after discovering a flaw in widely distributed processors, leading it to pioneer the implementation of FV for the entire electronic design automation industry.[6]

Today, FV is a standard part of the engineering workflow process for the specification, design, and verification of hardware, but except for certain mission- and safety-critical applications, including avionics and astronautics, and deep-pocket enterprises, such as Amazon, the software industry has not seen the same cost incentive to

adopt these processes. This is because it is often more economical to simply push out a patch to fix software flaws. Nevertheless, the proliferation of CPSs during the past decade has blurred the lines between hardware and software, and as CPSs make their way into even more highly regulated and safety-critical industries, the cost of design and implementation flaws is increasingly measured in human lives rather than dollars. The price of neglecting to provide rigorous guarantees about the behavior of CPSs is simply too high to ignore. Although FV has had success at NASA as well as in sectors of the automotive industry, we have not seen the ability to widely apply these methodologies to many CPS and Internet of Things devices in production today.

Classical FV is divided into heavy- and lightweight methods. The former consists of theorem proving and model checking, and the latter includes runtime monitoring (RM) (also known as *runtime verification* and *runtime execution monitoring*) combined with automatic test generation or falsification.[7,8] Model checking methods do not scale well from the small subsystems they were originally designed to handle because constructing models of complex systems can result in an exponential increase in possible states, a phenomenon referred to as *state explosion*, which renders methods such as model checking significantly less useful.[9,10] The alternative is to condense the models to decrease the burden on verification tools, but this runs the risk of abstracting away key details, producing models that do not accurately represent the true physical behavior of a system. Theorem provers do not suffer from the state explosion problem but require human guidance, hindering their ability to automate the verification proces.

Considering the issues with heavyweight methods, researchers have proposed the increased use of lightweight approaches—RM and falsification—to test a system's adherence to formal specifications.[7] RM observes the sequence and temporal behavior of an underlying application and compares it to the correct behavior as encoded in a formal specification language. It has the advantage of being able to observe a system under test (SUT) under a variety of inputs and edge cases, and it can be a necessity when FV at the code level is not feasible or possible, depending on the system's implementation language.[11] Additionally, the complexity and size of a CPS's input space has rendered the problem of checking whether all inputs satisfy a specification practically intractable. Therefore, falsification considers the opposite problem and searches for inputs that violate the specification. This method has recently been viewed as "a practical alternative to exhaustive formal verification" and has the potential to greatly reduce the time to identify undesirable behavior.[10]

SHORTFALLS IN FV FOR CPSs

Even with the increased use of RM and falsification, the community still faces two primary problems when conducting V&V on CPSs: 1) developing and encoding the specifications themselves and 2) applying these processes to stochastic and ML-enabled systems.

SPECIFICATIONS

Whether light- or heavyweight methods are used, they are still dependent on the specifications (also known as *temporal assertions* or *correctness properties*) over which they reason.[7] Such specifications are usually expressed via some dialect of temporal logic or by using diagrams such as state chart assertions;[7,8] therefore, the success of these LV&V methods is still typically dependent on the use of SMEs to write formal specifications. The specifications themselves can contain bugs due to specification ambiguity and coding errors when using the formal language at hand. Moreover, correctness properties for CPSs are often too complex to be captured by heavyweight, FV-enabled specification languages.[12] Even for simple systems, encoding "nontrivial specifications [is] challenging even for experts," and applying linear-time temporal logic (LTL) that is well suited to the finite state space of hardware is much less tractable for modern software systems.[1]

Developing formal specifications is an exploratory process that requires considering all possible interpretations of a natural language requirement, picking one that removes ambiguity, and then mapping it into a formalism of choice. The complexity of the languages and tools designed to support this process can pose a high barrier to entry, so experts advocate the use of "domain-specific specification patterns" to aid in specification writing, which enables users to capture recurring solutions that have been developed and tested by expert system designers.[1] Dwyer et al. collected more than 500 examples of property specifications in various temporal logics such as LTL and computation tree logic. They discovered that 92% of them could be specified as variations of a small set of regularly recurring patterns.[13] Based on this, they constructed a system that would enable practitioners, rather than experts, to utilize patterns that address requirements for their systems, then modify the patterns for use cases to more easily translate their requirements into formal specifications.[13] Even with these tools and others, many researchers still echo the sentiment that "specification is difficult, unglamorous, and arguably the biggest bottleneck facing verification and validation of aerospace, and other, autonomous systems."[14] We aim to investigate the possibility of using machine-learned correctness properties to aid in alleviating this burden.

STOCHASTIC AND ML-ENABLED SYSTEMS

Besides the complexity of specifications, there is additional difficulty with applying FV to systems that exhibit uncertain behavior and operate in unpredictable environments. For example, designers of an autonomous

vehicle may specify a set of safety properties for behavior when approaching a stop sign, such as the maximum deceleration, time to remain stopped, and conditions under which the car may proceed. These properties can be proved when those of all other vehicles and parameters in the environment are deterministically known. However, the vehicle's sensors and underlying algorithms may provide only probabilistic guarantees for artifacts that are not directly observed, including the intent of human operators of the other cars, the upcoming actions to be taken by the vehicles operating under autonomous control, the road friction coefficient (algorithms estimate this value, and it can change as a car moves from its starting point to its destination), and changes in the behavior of the vehicle's sensors due to inclement weather. These nondeterministic observations cannot be readily used to determine whether a rigid specification is satisfied, making it problematic to prove that specifications hold for systems that operate within such uncertain environments.

Additionally, ML is rapidly expanding into safety-critical applications, due to its ability to assimilate large amounts of data and aid humans in decision making for purposes such as medical diagnosis and aircraft collision avoidance systems.[1] While classical FV analyzes the well-defined logical flow of execution branches and function calls, the opaque nature of machine-learned functions makes it more difficult to reason about their correctness. In addition, the very nature of ML algorithms relegates most of the complexity to the training data rather than the implementation code. This training phase is highly nondeterministic, which is at odds with the transparency and traceability requirements specified in traditional safety-critical software development processes. There is research being conducted on adapting FV techniques to ML systems. However, the current methods for conducting FV on algorithms such as neural networks generally do not scale well for more than a few hundred neurons and are limited to specific architectures (for example, neural networks with piecewise linear layers).[15]

The primary methodology for conducting FV on neural networks is to frame the problem as a constraint satisfiability one, referred to as *satisfiability solving*, or an extension called *satisfiability modulo theory solving*. This approach requires specifying (the complement of) a safety property as well as a model of a neural network, encoded together as a set of constraints. If a satisfiability solver determines that the constraints can be met, the safety property holds; otherwise, the solution represents a counterexample, and the safety property is violated in that instance.[15] Nevertheless, this approach suffers from many of the same weakness found in conducting classical FV on traditional (non-ML) software, namely, the difficulty of developing formal specifications as well as the complexity of modeling the underlying system. The challenges associated with "state explosion" and model abstraction in traditional software are only magnified in neural networks, which may contain thousands of neurons per layer, especially when considering the number of inputs to a CPS such as an autonomous vehicle. Because of the widespread proliferation of ML-enabled systems in safety-critical industries, experts are unanimous in their consensus that "it is most pressing to identify and formulate requirements for the correctness of adaptive and ML algorithms."[1]

RECOMMENDATIONS

To address these issues, the authors propose using ML classifiers as the correctness properties. Rather than having human experts generate formal specifications, we intend to use machine-learned correctness properties called *monitors*. We do not mean for a machine to learn to generate formal specifications in a human-centered formal specification language, such as temporal logic. Rather, ML models will serve as the LV&V "monitors," alleviating the need for humans to write formal specifications.

An LV&V process checks for two primary requirements: a system functions in a way that meets the needs of the end user and other stakeholders (that is, validation), and it satisfies a set of specifications (that is, verification). At a high level, the process involves constructing the specifications, assessing the SUT using the specifications (through automatically and manually generated trials), and then running regression testing on the instrumented program and log files. RM enables us to use the same correctness properties on the system in operation to determine if it is behaving correctly (and the environment is as we expect). For critical systems, if the monitor detects misbehavior, it can take emergency actions, such as initiating a shutdown, giving control to the user, and alerting the user that the environment is not as expected.

The classical LV&V life cycle consists of the following three phases:

1. Develop formal specifications.
   - Assertions are created for important requirements. Typically, SMEs write these by hand.
   - Unit tests simulate these specifications in isolation to ensure that they behave as expected.
2. Test a SUT using the formal specifications.
   - Manually write or automatically generate tests for the SUT. These tests exercise the SUT as well as the previously mentioned assertions.
   - The runtime monitor observes the SUT's behavior under the given assertions. Alternatively, the monitor examines log files generated by the program in its operating environment.
3. Perform regression testing.
   - Repeatedly verify the behavior of the SUT as its code base evolves through time, using the assertion monitors.

As with that of traditional LV&V, an ML-based LV&V life cycle consists of three phases, of which only phase 1 differs, as follows:

1. Train the ML classifier.
   - Either manually create or automatically generate scenarios for an SME to observe and label as "good" (a zero label) or "bad" (a one label). Note that one-labeled scenarios are those that are of interest; that is, they capture a behavioral violation.
   - Use those labeled data as the ML training set, creating one or more ML-based correctness property models that serve as counterparts to formal specifications.
   - Evaluate the ML-based correctness property models using an ML testing set. For safety-related scenarios, training should attempt to minimize the number of false negatives, that is, scenarios for which there is an expectation that the ML model should flag a violation yet does not.

This ML-based approach addresses the two main concerns with classical FV: 1) developing and encoding the specifications and 2) applying these processes to ML-enabled systems and systems operating in uncertain environments. For the first concern, the ML classifier model serves as a function that maps system behavior scenarios to classes: acceptable and unacceptable. This mirrors the fact that traditional deterministic assertions map a system's behavior to a Boolean output. The primary benefit of using ML-based correctness properties is that rather than having experts painstakingly specify complex correctness properties, the ML classifier learns the underlying model from an SME's labeled feedback. The SME's role becomes similar to a driving instructor who is teaching a student; knowing that it is not possible up front to specify all actions the student should take to operate a vehicle safely, the instructor provides continuous feedback amid a dynamic environment.

For the second concern, many CPSs are systems of systems, containing traditional (non-ML) software as well as systems that use ML algorithms, such as neural networks. Because models for traditional software (for example, regular expressions, temporal logics, and finite-state machines) are difficult or impossible to apply to neural networks, conducting classical V&V on these systems requires using a plurality of approaches tailored to the underlying algorithms, thereby impeding verification of a system as a whole entity. In contrast, our proposed solution for conducting LV&V does not need to be tailored to the underlying system. Our approach entails wholistically monitoring the system's behavior during runtime, without potentially abstracting away critical details about operation and functionality.

Further research is needed to make the ML-based correctness property generation approach usable by industry, including but not limited to obtaining labeled data for one- and zero-labeled training scenarios, performing nonbinary classification, conducting LV&V on edge devices, and evaluating the ML classifier's performance. A particular area of future investigation is the generation of one-labeled scenarios, which are usually much less abundant than their zero counterparts.

Labeling data sets

To train the ML classifier, which will serve as the LV&V monitor, it is necessary to first label system behavior scenarios under various conditions as acceptable or unacceptable to obtain data sets for training. Unmanned cars and aircraft are particularly conducive to interactive simulation, and advances in virtual and augmented reality have enhanced the realism of these models. Pappas et. al. developed a low-cost, gamified simulator to generate realistic training data and validate a self-driving algorithm.[16] They discovered that by crowdsourcing the gameplay of average drivers, randomizing driving conditions (for example, lighting, scenery, and road conditions), and incentivizing desired driving behavior (for instance, collecting coins to "win" by staying within lane boundaries and fulfilling other safety criteria), they were able to obtain data sets that transferred well from synthetic environments to real-world tests.[16]

ML of sequences

Many ML classifiers take static data (for example, the dimensions, color, and shape of fruit) to classify an object into one of a known set of classes (for instance, oranges, cherries, or apples). However, CPSs are not static objects but continuous systems, so to categorize their operation as acceptable or unacceptable, the classifier needs to receive data that are sequenced in time. This idea is part of languages such as signal temporal logic, where temporal operators, including *always* and *eventually*, are sequence-related adverbs, potentially also scoped using intervals of time. Hence, sequencing and time should be made explicit in the training data set for the classifier to account for time sequences as a system feature.

Nonbinary classification

For many safety-critical systems, it is not always desirable to binarily classify behaviors as acceptable and unacceptable. Returning to our consideration of the operation of autonomous cars, due to complex traffic conditions, unexpected environmental interference, and unpredictable actions of other manned and unmanned vehicles, a self-driving vehicle system must be able to identify potential collision conditions, estimate and assess the level of risk for every potential action, and mitigate its level of risk by taking an action, such as reducing its speed and stopping if appropriate. Stopping in the middle of a highway is an unacceptable condition with no other confounding factors, but when weighed against the potential harm caused to a vehicle's passengers by striking a large animal that

has wandered onto the road, coming to a halt may be the condition that best minimizes risk. To conduct robust RM on a system, the V&V monitor needs to take these nonbinary risk calculations into account. For this purpose, there are multiple classes of ML methods that can be extended from binary to multiclass classification, including multilabel decision trees, random forests, and gradient boosting algorithms. By categorizing data in a nonbinary way, the LV&V monitor can ensure that the system takes actions that maximize functionality while minimizing risk.

Emphasis on fast classification training at the edge

When employing ML classifiers as correctness properties, we prefer the use of lightweight classifiers, which can be 1) trained and tested within 1 min and 2) hosted on edge devices instead of relying on pushing data to remote centers for processing. The focus on lightweight classifiers is motivated by the fact that using ML correctness properties should enable a simpler LV&V workflow than relying on SME-created properties. To that end, using resource-intensive classifiers such as deep neural networks with TensorFlow (DNNT), along with their long training time and dependence on cloud-based computing resources, is counterproductive. In addition, we envision that for some applications, labeling and training must be done in the field, on edge devices and servers, which are secure deployed machines with limited computational power. Such a device might not be powerful enough to train a DNNT without incurring delays that are operationally unacceptable.

Evaluating ML classifiers for LV&V

When thinking about using machine-learned correctness properties for ML-based CPSs, many have asked "who will guard the guards?"[1] In other words, if we propose to use ML to conduct LV&V on systems that already employ the technology in their operation, how can we be sure the same errors we are attempting to identify in an underlying SUT are not also inherent to the LV&V monitor? To answer that, recall that there is a precedent for such a configuration in the software engineering community. The traditional model uses the same technology (human software developers) split into multiple teams (development and testing, among others) to deliver software that is functional and safe. The traditional model evolved into transitional LV&V, where human coders develop a system and are supplemented by formal specification assertions in the automated monitoring and evaluation of the system. Our proposal is a logical extension, where development-oriented SMEs train the ML algorithms that constitute a system and V&V-oriented SMEs collect and process data for ML algorithms that serve as correctness properties for testing.

Additionally, though ML classifiers are more opaque than explicit formal specifications, we have tools to evaluate their performance and ensure that they are accurate in performing the LV&V function. One such tool, the confusion matrix, enables the simple visualization of a classifier's performance through the observation of its errors. False negatives, also known as *type 2 errors*, are particularly dangerous in safety-critical systems. They occur when the monitor fails to raise an exception for an unacceptable system state. In the case of the ML classifier, a type 2 error would give current conditions a zero label ("good") when a system was actually in a "bad" state that should have received a one label. Training the ML classifier will seek to minimize these types of errors for safety-critical systems.

For assertions that underwent proper validation testing, false positives, also referred to as *type 1 errors*, are usually due to preconditions for the assertions themselves. Preconditions are the circumstances under which an assertion (or in this case, the ML model) is evaluated; for example, a certain model might be meant for a vehicle in motion, not a stopped one. While preconditions can be encoded into the assertions, it is common practice to program the test suite to invoke certain specifications based on preconditions and not others. Therefore, explicit feedback from the confusion matrix enables the evaluation of the ML model's performance and reveals the need for modifications to the model, test cases, and test suite preconditions.

Positioning

In this article, we discussed the challenge of applying classical FV to CPSs, and we proposed an LV&V approach for validating systems with uncertainties, including those that use ML. Our approach treats ML classifiers as correctness properties instead of relying on manually encoding complex formal specifications. In the December 2021 issue of *Computer*, Philip Koopman wrote:

> *Where we are now is that we hope that the promise of AVs [autonomous vehicles] to save lives will work out. AVs will make different mistakes than human drivers, and nobody yet knows how long it will take to get the balance in favor of AVs. I'd really like to see a credible safety case backed by solid evidence showing that AVs will be at least as safe as human drivers—with ample margin for error—before deploying.*[4]

Our approach addresses Koopman's desire. We have discussed how to efficiently train and test ML classifiers on embedded systems, making it possible to conduct RM on edge devices and, in turn, use the results of monitoring to build assurance arguments that a system can operate in a safe enough manner for the intended contexts of use and uncertain environmental conditions.

A wide range of FV methods has been suggested through the years, accompanied by a corresponding assortment of formal specification languages. These methods extend from user-assisted theorem

proving, through various model checking techniques, to LV&V. Each has a potential role in the verification of systems. For example, heavyweight V&V methods are a necessary tool for the verification of safety-critical CPSs such as nuclear power plants, while LV&V has a useful role verifying properties of an embedded system by using log files. Hence, the approach advocated in this article is not expected to replace the more traditional method, where properties are manually created. Rather, we believe it will be added to the V&V toolbox as a first-class citizen.

REFERENCES

1. J. B. Michael, D. Drusinsky, and D. Wijesekera, "Formal verification of cyberphysical systems," *Computer*, vol. 54, no. 9, pp. 15–24, 2021, doi: 10.1109/MC.2021.3055883.
2. M. Ford, "Elon Musk's failed Tesla robotaxi promise is the height of self-driving hype," Fast Company, Sep. 18, 2021. [Online]. Available: https://www.fastcompany.com/90677822/elon-musks-tesla-robotaxi-promise-typifies-self-driving-overexuberance
3. B. Haensel, "Driverless semi reaches milestone in test, startup Plus says," Bloomberg, Aug. 5, 2021. [Online]. Available: https://www.bloomberg.com/news/articles/2021-08-05/driverless-semi-reaches-milestone-in-test-startup-plus-says#:~:text=Driverless%20Semi%20Reaches%20Milestone%20in%20Test%2C%20Startup%20Plus,with%20no%20driver%2C%20even%20as%20a%20safety%20backup
4. P. Koopman, B. Kuipers, W. H. Widen, and M. Wolf, "Ethics, safety, and autonomous vehicles," *Comput.*, vol. 54, no. 12, pp. 28–37, 2021, doi: 10.1109/MC.2021.3122781.
5. N. Leveson, *Engineering A Safer World: Systems Thinking Applied to Safety*. Cambridge, MA, USA: MIT Press, 2011.
6. J. Harrison, "Formal verification at Intel," in *Proc. 18th Annu. IEEE Symp. Logic Comput. Sci.*, 2003, pp. 45–54, doi: 10.1109/LICS.2003.1210044.
7. D. Drusinsky, *Modeling and Verification Using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-Based Model Checking*, Oxford, U.K.: Elsevier Science & Technology, 2006.
8. D. Drusinsky *et al.*, *Practical UML-Based Specification, Validation, and Verification of Mission-Critical Software: Space Exploration and Defense Software Examples in Practice*. Carmel, IN, USA: Dog Ear Publishing, 2011.
9. J. B. Michael, G. W. Dinolt, and D. Drusinsky, "Open questions in formal methods," *Computer*, vol. 53, no. 5, pp. 81–84, 2020, doi: 10.1109/MC.2020.2978567.
10. Z. Zhang, P. Arcaini, and I. Hasuo, "Constraining counterexamples in hybrid system falsification: Penalty-based approaches," in *Proc. 12th Int. Symp. NASA Formal Methods, LNCS*, vol. 12229, Cham, Switzerland: Springer International Publishing, 2020, pp. 401-419, doi: 10.1007/978-3-030-55754-6_24.
11. R. C. Cardoso, M. Farrell, M. Luckcuck, A. Ferrando, and M. Fisher, "Heterogeneous verification of an autonomous curiosity rover," in *Proc. 12th Int. Symp. NASA Formal Methods, LNCS*, vol. 12229, Cham, Switzerland: Springer International Publishing, 2020, pp. 353–360, doi: 10.1007/978-3-030-55754-6_20.
12. D. Drusinsky, J. B. Michael, and M.-T. Shing, "A visual tradeoff space for formal verification and validation techniques," *IEEE Syst. J*, vol. 2, no. 4, pp. 513–519, 2008, doi: 10.1109/JSYST.2008.2009190.
13. M. B.Dwyer, G. S. Avrunin, and J.C. Corbett, "Property specification patterns for finite-state verification," in *Proc. 2nd Workshop Formal Methods Softw. Practice*, 1998, pp. 7–15, doi: 10.1145/298595.298598.
14. K. Y. Rozier, "Specification: The biggest bottleneck in formal methods and autonomy," in *Verified Software. Theories, Tools, and Experiments, LNCS*, vol. 9971, S. Blazy and M. Chechik, eds. Cham, Switzerland: Springer International Publishing, 2016, pp. 8–26, doi: 10.1007/978-3-319-48869-1_2.
15. C. Urban and A. Miné, "A review of formal methods applied to machine learning," Apr. 2021. [Online]. Available: http://arxiv.org/abs/2104.02466
16. G. Pappas, J. E. Siegel, K. Politopoulos, and Y. Sun, "A gamified simulator and physical platform for self-driving algorithm training and validation," *Electronics*, vol. 10, no. 9, p. 1112, 2021, doi: 10.3390/electronics10091112.

DISCLAIMER

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies and endorsements, either expressed or implied, of the U.S. government. The U.S. government is authorized to reproduce and distribute reprints for government purposes, notwithstanding any copyright annotations thereon.

DORON DRUSINSKY is a professor in the Department of Computer Science, Naval Postgraduate School, Monterey, California, 93943, USA, and chief science officer at Aerendir, Inc., Mountain View, California, 94040, USA. Contact him at ddrusins@nps.edu.

MATTHEW LITTON is a doctoral student in the Department of Computer Science, Naval Postgraduate School, Monterey, California, 93943, USA. Contact him at matthew.litton@nps.edu.

JAMES BRET MICHAEL is a professor in the Departments of Computer Science and Electrical and Computer Engineering, Naval Postgraduate School, Monterey, California, 93943, USA. Contact him at bmichael@nps.edu.

APPENDIX C:
Machine-Learned Specifications for the Verification and Validation of Autonomous Cyberphysical Systems

The following article was submitted to the IEEE International Symposium on Software Reliability Engineering (ISSRE) Workshop on Assured Autonomy, Artificial Intelligence and Machine Learning (WAAM). It was presented at the live conference workshop on October 31, 2022 in Charlotte, NC, and included as an archival publication in the associated conference proceedings.

2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) | 978-1-6654-7679-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISSREW55968.2022.00089

Machine-Learned Specifications for the Verification and Validation of Autonomous Cyberphysical Systems

Doron Drusinsky
Dept. of Computer Science
Naval Postgraduate School
Monterey, CA USA
ddrusins@nps.edu

James Bret Michael
Dept. of Computer Science
Naval Postgraduate School
Monterey, CA USA
bmichael@nps.edu

Matthew Litton
Dept. of Computer Science
Naval Postgraduate School
Monterey, CA USA
matthew.litton@nps.edu

***Abstract*—Machine learning classifiers can be used as specifications for runtime monitoring (RM), which in turn supports evaluating autonomous systems during design-time and detecting/responding to exceptional situations during system operation. In this paper we describe how the use of machine-learned specifications enhances the effectiveness of RM for verification and validation (V&V) of autonomous cyberphysical systems (CPSs). In addition, we show that the development of machine-learned specifications has a predictable cost, at less than $100 per specification, using 2022 cloud computing pricing. Finally, a key benefit of our approach is that developing specifications by training ML models brings the task of developing robust specifications from the realm of doctoral-level experts into the domain of system developers and engineers.**

***Index Terms*—verification and validation, formal methods, machine learning, specifications, autonomous systems, cyberphysical systems**

I. Introduction

Individuals and corporations are increasingly developing cyberphysical systems (CPSs) on whose services they must place significant trust. Many of these systems enjoy longstanding acceptance due to their proven record of dependability (e.g., traffic lights, nuclear power plants, electronic vehicle transmissions). Currently, there is a push to introduce autonomous CPSs into widespread use to reduce the physical and cognitive burden on humans by delegating certain tasks to autonomous systems, while humans retain the ability to perform high-level risk management and decision-making. Whether autonomous CPSs aid doctors in medical diagnosis and treatment, or assist commuters in navigating a busy highway, our willingness to trust in human-machine teaming rests on the ability of autonomous systems to behave in a dependable manner. Unlike purely deterministic systems, autonomous CPSs do not currently enjoy the same level of trust due to the lack of tools for performing verification and validation (V&V) on systems specifically designed to adapt.

Avižienis et al. define *dependability* as "the ability to avoid service failures that are more frequent and more severe than is acceptable." They define *trust* as "accepted dependability" [1]. *Dependability* is a generic term that encompasses all areas by which a system's behavior can be considered, such as safety, performance, maintenance, availability, integrity, and reliability.

Formal methods (FM) can be used to produce evidence supporting claims that a system meets stakeholders' criteria for being sufficiently trustworthy. During the rapid design, test, and deployment of the National Aeronautics and Space Administration's (NASA) Lunar Atmosphere Dust Environment Explorer (LADEE), FM experts integrated V&V into the early development cycles of software development, and used specifications refined throughout the development phase to perform operational and integration testing prior to a successful seven-month mission to gather data about the lunar atmosphere [2]. FM tend to be applied to the V&V of safety- and mission-critical systems, such as those for manned and unmanned space flight missions, industrial process control [3], development of semiconductor and analog/mixed-signal systems ([4], [5]), and security and network protocol analysis [6]. Unfortunately, FM have seen little use throughout the software industry relative to their adoption by the hardware engineering community [7].

Specifications (sometimes called "assertions" or "correctness properties") are the key to establishing correct behavior through the V&V process. Both heavyweight methods (e.g., model checkers, theorem provers) and lightweight methods (e.g., runtime monitors) automate the mechanical aspects of demonstrating that a system model satisfies a set of given rules about correct behavior. The success of these methods is predicated, in part, on how well the engineer captures the specifications for the target system. For systems that are not learning-enabled (i.e., systems that only exhibit behavior in operation that was explicitly programmed in its design), using specifications is critical for ensuring that subsystems and integrated components meet the stakeholders' requirements and performance markers throughout the design process. As the code base of the system evolves, the specifications can be integrated into regression test suites to ensure the system continues to meet the design requirements as the prototyping

This research is sponsored by the U.S. Department of the Navy.

978-1-6654-7679-9/22/$31.00 ©2022 IEEE
DOI 10.1109/ISSREW55968.2022.00089

progresses. For an autonomous system, systems which "have a set of intelligence-based capabilities that allow it to respond to situations that were not pre-programmed or anticipated in the design," having specifications that can be used at design-time, but also raise onboard exceptions during runtime, becomes crucial [8]. Without specification-based automated monitoring, ensuring that autonomous systems continue to abide by the requirements specified in design as they adapt to their operational environment is a challenging, time-consuming process that increases the cognitive burden on humans, detracting from the intended benefits of autonomy.

II. Specifications

Specifications have served as a basis from which to perform testing as part of the formal verification and validation (FV&V) process. For many years, software testing involved manually creating test cases, executing each test case, and then, with the use of an oracle, determining whether the test case passed. Manual testing is a labor-intensive process, does not scale well, and is incompatible with modern software development paradigms such as agile development and DevSecOps continuous integration and continuous deployment. The development of computer-readable specifications made it possible to automate the testing process, or replace it with mathematical proofs.

No matter which tool is used in the FV&V process, its success is highly dependent on the specifications over which it reasons. Model checkers determine whether the model ($\mathcal{M}$) of the system satisfies a formal specification (ψ) (i.e., $\mathcal{M} \models \psi$). Theorem provers, such as ACL2 [9], manipulate first, second, or higher order logic to reason about the correctness of a system with respect to its specified properties. Runtime monitoring (RM) methods observe the system model's output given its inputs to determine if its behavior complies with the given assertions. Falsification is a technique for automatically searching through a system's possible state transitions for paths which violate the specifications. Falsification is, in effect, a fast automatic test generator for RM methods.

In the 1950s, Arthur Prior introduced what he called "tense logic" which forms the basis for modern-day temporal logic, and began advocating for a more formal approach to specifying properties of dynamic, real-world systems [10]. Today, temporal logic is widely used, with signal temporal logic (STL) being the most commonly used specification language for CPSs. However, developing formal specifications is still largely an academic exercise not frequently utilized in the software industry. This is because specification languages remain in the realm of FV&V experts and are not easily utilized by system designers.

In a virtual roundtable on FV&V for CPSs, experts discussed reasons why specifications provide such a barrier to the adoption of FM by the software industry including: disconnect between system implementation and the specification process, and issues with specification languages themselves [11]. The first issue is the difficulty in convincing software engineers of the benefit of mastering yet another language (i.e., the specification language as well as the implementation language). Without support from management, emphasis on specification development, and proper tools to integrate specification into the system design lifecycle, the process is viewed as another useless exercise that delays a product from getting to market. Some companies have addressed this by bringing in a team of specification developers (usually people with doctoral degrees specializing in FM), but the associated cost is often prohibitive. The second issue is that despite the wide range of formal specification languages available, writing formal specifications that accurately and completely capture requirements is still a manual, time-intensive, and error-prone process even for FM experts.

To illustrate the difficulty of writing formal specifications, consider the following requirement for an aircraft collision avoidance system written in temporal logic (TL)[1]:

$$\Box_{[0,\text{inf}]}\Big((d_h < 500 \wedge d_v < 200) \implies \Diamond_{[0,10]} \left(\Box_{[0,120]} \left(d_h > 500 \wedge d_v > 200\right)\right)\Big) \tag{1}$$

Formula 1 uses metric temporal logic (MTL) to state, "Always, if the horizontal and vertical distance between two aircraft ever become less than 500 and 200 feet respectively, then at some point within the next 10 seconds, the aircraft must remain at a horizontal distance of more than 500 feet and a vertical distance of more than 200 feet for at least 120 seconds." Note that the distance between aircraft needs to increase above the specified threshold within 10 seconds after violating the safety margin, but the aircraft are not considered to have recovered from an unsafe distance until they have remained so for two minutes (possibly after two minutes and 10 seconds). The requirement can fail in two main ways. First the distance may not reach the given safe threshold within 10 seconds. Second, the recovery may be incomplete (e.g., the aircraft may only remain at the specified separation for one minute rather than the required two). The natural language (NL) specification is ambiguous in that it does not specify where the 120-second interval begins. Perhaps it is meant to begin whenever the aircraft minimum distance (500 and 200 feet) is first violated. Alternatively, it could be interpreted to begin whenever recovery began, that is, within the allowed 10 second recovery interval. The MTL specification is not ambiguous: it enforces the second interpretation because of the underlying MTL semantics of "always" ($\Box$) being nested under "eventually" ($\Diamond$). This, however, is not necessarily the intention behind the NL specification, or perhaps the author of the NL specification was unaware of the underlying ambiguity. This ambiguity, along with MTL's semantic interpretation bias, were discovered when the MTL specification itself was simulated and tested against the subject matter expert's (SME) interpretation, a process that is rarely done in practice. In contrast, when using machine-learned correctness properties

[1]A rewrite of the banking example in [12](p. 117)

as suggested in this paper, actual system behavior defines the property, without the intermediate human interpretation. Moreover, the testing of the resulting classifiers is a standard part of the classifier development process.

To illustrate the sheer time requirement of specification development, several researchers performed FV&V on an aircraft collision avoidance system, formulating specifications in linear-time temporal logic (LTL) to conduct symbolic model checking. Developing the formal specifications took approximately 100 person-hours, while the model checker took only about 10 hours to verify the full system model [13]. Additionally, those specifications were implemented exclusively by SMEs in FM, rendering them not easily modified by system designers and testers when changes are made to the requirements.

A. A New Approach to Specifications

There is a clear need to develop a new approach to specification generation that can be used to certify autonomous systems in design, and provide evidence to determine the level of trust humans can place in the dependability of such systems during their operation. Although there are industries that have benefited from using TL specifications and that have well-established processes for developing and integrating the specifications into their engineering workflows, generating TL for ensuring the dependability of autonomous systems is tricky. Researchers and practitioners alike have discussed the idea that specifications should be developed and explored through more intuitive and expressive means than higher-order logics (e.g., diagrams, simulations, natural language), and with advances in machine learning (ML), it may be possible that ML models can more easily extract the transformation from observed system behavior to a determination of correctness. Just as traditional formal specifications enabled a significant step forward from manual to automated testing, using ML specifications will bring a similar step-function increase in the effectiveness and usability of V&V techniques to certify dependable autonomous systems.

The primary benefit is that developing specifications by training ML models brings the task of developing robust specifications from the realm of PhD-level experts into the domain of system developers and engineers. Previously, formal specification SMEs had to translate (or in some cases, generate) human-readable requirements from NL into the formal specification of choice. This process is difficult, not only due to the ambiguity inherent to NL, but also due to the difficulty in translating the requirements into a formal specification language with complex syntax. On the other hand, generating ML specifications involves the more intuitive process of selecting representations of behavior that are easily understandable to humans to serve as training data for the ML model. Not only does this bypass the requirement for doctoral-level specialization, it does not require in-depth knowledge of the system's internal operation; any user who is familiar with what "correct" behavior looks like for the system in question can generate and select appropriate simulated scenarios for training. Moreover, autonomous vehicle (AV) companies are collecting huge amounts of data on a daily basis when driving their vehicles in test mode on U.S. and foreign roads; this data is ripe for use for training and testing ML-assertions models. This technique also provides a more flexible approach to specifications which evolve more easily with systems designed to learn and change to meet the demands of their environment. TL specifications are rigid, and may need regular revision to account for new behaviors that autonomous systems exhibit as well as unanticipated environmental factors, whereas ML specifications can generalize from the training data, better allowing them to detect violations even under slightly different conditions than those used to train them.

The general approach to generating and using machine-learned correctness properties was laid out in [14], and since that time, the authors have conducted further research into the suitability of this approach for conducting V&V on autonomous CPSs. The purpose of this paper is to discuss the current progress of that research and planned future work.

III. Related Work

NASA was an early adopter of FM and continues to conduct research and development (R&D) on FM and apply them. In 1990, NASA established the annual *Formal Methods Symposium* with the aim of spurring along the theory and practice of FM. In this section, we some discuss some of the past innovations and current R&D projects of NASA and their relationship to our approach to obtaining formal specifications of systems.

NASA's Copilot project began in 2008 with a recognition that applying heavyweight formal methods at the system level, especially to complex real-time systems like avionics, is impractical due to issues related to complexity theory [15]. Therefore, the project's principal investigators developed a RM framework in which specifications are written in a high-level domain-specific language (embedded in Haskell). The specifications are then compiled into C code for efficient integration into existing embedded systems. Such high-level specification representations with automatic code generation have been explored by previous researchers such as Drusinsky (e.g., automated Java code generation from statecharts [12]). The C executable monitors are generated as ".c" and ".h" files that must then be integrated into existing software, and they are currently being used with NASA's Core Flight System applications [15]. The authors acknowledge that while high-level domain-specific specification languages provide some benefit to improving their usability for RM, "finding the logic that is most appropriate to express the properties of autonomous vehicles remains an open problem in the runtime verification community" and is certainly not amenable to a "one-size-fits-all" solution [15]. Additionally, the Copilot research group designed the RM system to operate in real time alongside the underlying system, leading them to investigate critical areas such as fault-tolerant RM, efficient processor and memory usage to minimize latency, and isolation of the monitor from the system.

Dutle et al. attempted to extend the Copilot capability by adding a front-end to translate requirements from NL. This interface, called Formal Requirements Elicitation Tool (FRET), takes a NL requirement and translates it into TL for use by Copilot RM. The very fact that the authors recognized a need for another layer of abstraction beyond a domain-specific specification language illustrates the difficulty with writing clear, unambiguous specifications. To generate executable monitors, a user must first provide FRET awareness of the various observable data available from the underlying system so that references in NL to measures such as "distance" can be mapped to a concrete data stream or memory location. FRET then translates the NL requirement into MTL, which will serve as a formal specification to be compiled into C by Copilot for instrumenting the system's software. Though the process can assist in automating the translation from NL requirements to executable assertions, the multiple layers of translation required (natural language to MTL to C, with several intermediate representations in between) also necessitate conducting V&V on the toolchain, which in itself can be challenging because toolchains can be complex.

In recent work, NASA has recognized the proliferation of "increasingly autonomous" systems in aviation, as well as the fact that assurance must be gained at the system level, not the machine-learning enabled component (MLEC) level [16]. Traditional assurance processes require every possible path to be explored during testing, but the statistical nature of MLECs does not lend itself to similar certification processes. Therefore, researchers like Davies have expressed the need for RM architectures that can detect violations at the system-level during runtime, changing the mode of operation from a MLEC to a more deterministic component which can steer the system into a safe state [16]. These high-assurance RMs will also require high-assurance mechanisms of formulating specifications.

To address the specification issue, Zeller asked at the 2011 NASA FM Symposium, "Can we have specifications for free?" [17]. The motivation for such a question is similar to our motivation for this paper, namely that there has been significant research focusing on advancing the state-of-the-art in software V&V, with comparatively less effort in actually *specifying* the behavior to be validated and verified. To address this gap, Zeller proposes specification mining to extract specifications from existing systems in which the knowledge of how a system should function is already embedded in the code. He proposed to extract specifications into XML representations with pre- and post-conditions, exploring and enriching specifications through automated test generation [17]. Though our approach is similar in motivation to Zeller's, it takes a fundamentally different approach to representing specifications. We do not believe specifications can be obtained for free. Instead, we believe our approach may make obtaining specifications affordable.

IV. Method

In order to generate visually realistic scenarios, we selected CARLA (CARs Learning to Act) as our simulation engine, an open-source environment designed to facilitate the rapid evaluation of AVs [18]. We use AVs as our representative system under test (SUT) because their V&V process encompasses many of the categories of specification that other autonomous CPSs are also likely to utilize: safety, performance, legal, security, and hardware reliability among others. In addition, their tasks (e.g., geospatial navigation, interagent-coordination, sensor-based perception and control) are easily translatable to many other autonomous CPSs currently being tested including caregiver robots and autonomous aerial delivery vehicles.

In each simulation, we use an *ego vehicle*, which refers to the vehicle that is being tested and contains the sensors that perceive the environment around it. In addition, there are one or more *adversary vehicles* which exhibit various behaviors defined by underlying probability distribution functions in order to effectively model real-world phenomena. Because this research is focused on the generation of specifications for V&V, and not the design of the underlying SUT, we remain agnostic to the ego vehicle's actual implementation of autonomous perception and control functions. A simulation environment such as SmartPath would be more appropriate to use for design and parametric evaluations of the performance of autonomous vehicles based on changes to their specifications [19].

A. Data Generation Framework

In order to train and test ML classifiers to serve as specifications, the most important factor is how to obtain the labeled data sets that will serve as the training and testing sets. For a binary classifier, scenarios with a "1" label are those that the SME believes are of interest (e.g., accidents or close encounters), and "0"-labeled scenarios represent desired or benign behavior. During early stages of the research, we planned to have a SME observe scenarios in a simulation environment and manually label them as "1" or "0" to train the classifier. While this is still a feasible approach, and recommended by multiple researchers in cases where the number of specifications needed is relatively small ([13], [20]), our ML approach requires a volume of data that could make this technique prohibitive for a large body of assertions. Especially for systems that are relatively mature (and therefore aren't likely to generate sufficient number of "1" scenarios during normal operation), it is important to investigate a means to programmatically generate labeled training data.

Rather than simply generating random data through the use of stochastic techniques like Monte Carlo methods, the more interesting question is how to obtain the diversity of data expected. When considering specifications for critical events, scenarios that contain violations (e.g., crashes, reckless maneuvers, violations of traffic codes) are much less common than typical driving scenarios in which obviously unsafe events do not occur (human drivers cause only approximately 77 injuries and 1 fatality for every 100 million miles driven [21]).

Therefore, because CARLA provides the ability to generate scenarios programmatically, we use optimization techniques to search for the optimal distribution of scenario parameters that lead to higher percentages of critical scenarios ("1"-labeled scenarios). Specifically, we use cross entropy (CE) to optimize the search for critical scenarios in order to ensure sufficient coverage, beyond that obtainable from baseline driving behavior. CE is a sampling technique that allows one to construct a random sequence of solutions which converges probabilistically to the optimal one [22]. In this case, an optimal distribution is one which we assert has a higher occurrence of behavior that violates the specification(s) of interest, thereby yielding a more appropriate data set with which to train the ML classifier. CE operates iteratively across the following phases:

1) Generate samples of random data according to a specified mechanism. In our case, we use parameterized families of distributions including normal, exponential, and categorical, to generate the random samples. The statistical distributions represent the distribution of the random variables of interest (e.g., vehicle position/speed/acceleration, driving behaviors, and environmental factors).
2) Use those data samples to define a scenario with the AV (ego), other vehicles (adversaries), and corresponding environmental conditions. That scenario is "scored" according to a cost function that drives the search toward violations of the specification.
3) Take the fraction of samples with the lowest score (i.e., the elite set) and use them to update the parameters of the statistical distributions.
4) Iterate until the score of the entire elite set falls below some threshold. The elite set contains one or more samples that can be used to train the classifier to detect specification violations.

B. Data Generation Example

The following example illustrates an iteration of CE to generate "1" scenarios. Many of these scenarios will need to be generated for the training/testing sets for the the classifier, but this details the methodology. In this specific example, an ego vehicle must navigate an intersection in the presence of an adversary (bicycle). The assertion for which we are generating scenarios is *"The Time-to-Collision between the vehicle and the cyclist should never be less than β"* where β is some threshold, 1 second in this example. To demonstrate the method of controlling the adversary's position through CE, we define a 20x20 grid that extends throughout the intersection in which the scenario is executed (Fig. 1):

1) Sample random variables to define behavior of adversaries in the scenario: In this example, we use two different probability distributions to demonstrate the use of both continuous and discrete random variables: normal (Gaussian) and categorical. These distributions are used to model a variety of characteristics of the adversary vehicle such as position, speed, acceleration. They can also be used to define the state

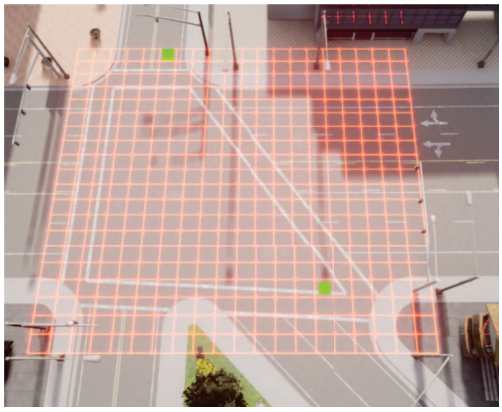


Fig. 1: Grid showing adversary start point (row 0, column 5) and end point (row 16, column 14)

of various other environmental factors (e.g., timing of traffic lights, weather conditions, road friction coefficients, etc.). In some cases, there may be multiple adversaries, each of which has an aspect of their operation dictated by a random variable. In the case of this example, a single adversary demonstrates both normal and categorical distributions. For each iteration of CE, the adversary first follows a given sequence of positions (specified by grid coordinates) drawn randomly from the categorical distribution. Then, the adversary selects the lowest-cost path from the categorically-distributed samples and uses random variables drawn from a normal distributions to control its acceleration between each point in the path. In a more sophisticated setting, CE performs a hybrid optimization of the categorical and normal distributions.

An example of a random path in the first iteration is shown in Fig. 2 which visualizes samples of random variables drawn from the categorical distribution. The adversary starts and finishes at the defined positions shown in Fig. 1, and the order of the path (not shown) is less important than the length of the path (number of points visited). In this iteration, the path score is poor (according to the cost function discussed in Section IV-B2) because the probability matrix for the categorical distribution is still random; the probabilities have not yet been iteratively updated through the CE process. The dark squares represent locations inside the crosswalk, and light squares represent locations outside the crosswalk.

After the lowest cost path is selected from the randomly generated paths, the adversary will traverse the path using randomly selected, normally distributed acceleration values.

2) Score the scenarios according to a cost function: In order for CE to improve at each iteration, it must receive a score value for each path in order to sort the paths and select a percentage of paths with the lowest scores. It will use this percentage (usually called the "elite set") to update the probability density function parameters to iteratively improve. The cost function defines the metrics to be optimized (i.e., the "goal" of the scenario). We discuss the use of cost functions for both categorical and normal distributions.

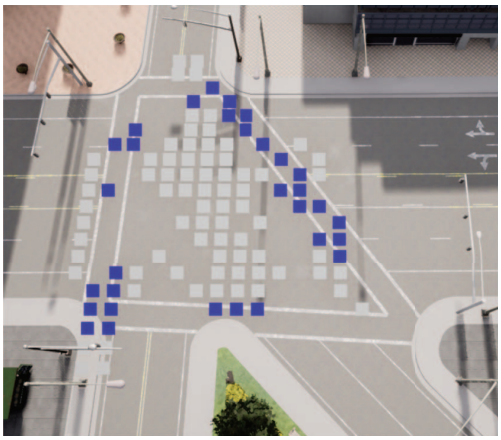


Fig. 2: Selection of random locations based on categorical distribution

For the categorical distribution, the cost function represents the realistic behavior of a cyclist by defining a goal of finding the shortest path between a start point and end point while prioritizing locations inside the crosswalk. Therefore, the cost function rewards shorter paths with path locations in the crosswalk (dark squares), and penalizes longer paths with locations outside the crosswalk (light squares). Using this cost function, CE converges quickly from high-cost paths (Fig. 2: 104 locations with 30% inside the crosswalk) to low-cost paths (Fig. 3: 19 locations, 74% of which are inside the crosswalk).

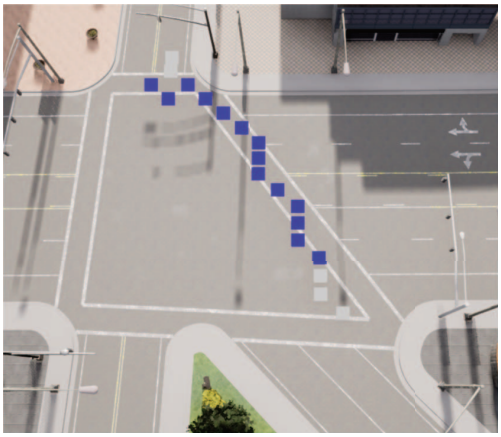
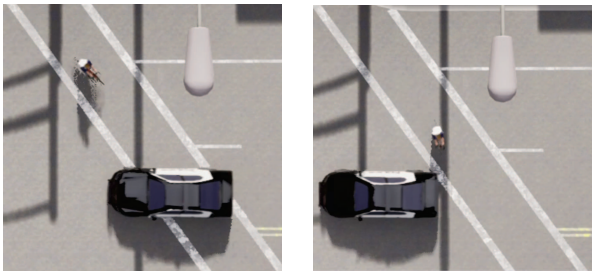


Fig. 3: Selection of random locations based on categorical distribution after multiple iterations of cross-entropy

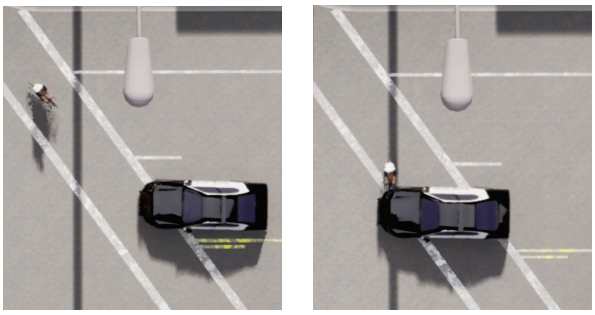
For the normal distribution, the cost function computes the time-to-collision (TTC) between the adversary and ego vehicle which drives CE to discover those scenarios in which each actor is attempting to optimize its own individual cost function, but there is either an accident or a near-miss (defined as a TTC below some threshold: $TTC < \beta$).

3) Update the probability distribution function parameters: After both simulations take place (i.e., categorically distributed random variables for position, normally distributed random variables for acceleration) and are scored according to their cost functions, the probability distribution parameters are updated to reflect the new data. For the categorical distribution, this means updating the 20x20 probability matrix to reflect the new sample of scored paths. For the normal distribution, this means updating the mean (μ) and variance (σ^2) to reflect the scored sequences contained in the elite set.



(a) TTC = 5.78 s (b) TTC = .98 s

Fig. 4: Path 1 ("Near Miss"): TTC_{min} = 0.98 seconds



(a) TTC = 5.13 s (b) TTC = 0 s (Crash)

Fig. 5: Path 2 (Accident): TTC_{min} = 0 seconds

4) Iterate until an acceptable level is reached: At the end, our elite set contains all of the scored paths below a certain score level, β. In one run, we set $\beta = 1$, so the final elite set contained a set of path which resulted in a TTC of less than 1 second. Relevant portions of two of the paths from that elite set are shown in Fig. 4 and Fig. 5.

At the end of the CE process, the elite set from the final iteration represent the most realistic "1" scenarios for training the classifier. The other scenarios, serve as "0" scenarios and represent behavior that complies with the specification of interest (in this case, $TTC > \beta$).

V. Positioning

With the software community's general reticence to develop and utilize specifications, we discuss several objections to traditional formal specifications. In addition, we consider several potential objections specifically concerned with our proposed use of machine-learned specifications.

A. Devil's Advocate Arguments to Manually-Written Formal Specifications

1) Why use specifications at all?: Without specifications, a human observer attempting to determine if a system is operating correctly must manually compare the system's behavior to a set of requirements. For more than one or two requirements, a human is unlikely to be able to evaluate them all repeatedly (e.g., as required by daily regression testing). In addition, many NL requirements are ambiguous. Formal specifications allow the automated evaluation of a system during testing by automatically evaluating and reporting specification violations. For autonomous systems where RM is necessary not only in testing but also during operation, manual observation is not possible and executable specifications which can be unambiguously evaluated to True or False are really the only remaining option.

2) Are you only interested in accidents?: The average person might consider there to be only a single relevant specification for the safe operation of AVs (or any safety-critical system) - "no accidents." However, such a trivial assertion does not accurately reflect the current methods used to certify and evaluate human driving behavior. To deem a fully autonomous vehicle to be safe, or at least legal, one needs to make a convincing argument that it satisfies all the rules expected of a human driver and exhibits behavior comparable or superior to vehicles driven by a human, not simply that it is not involved in any accidents. Human drivers are monitored for compliance with legal as well as socially-accepted rules during testing (driving with a learner's permit), certification (driving tests), and even after they are licensed (subject to traffic tickets, remedial training, license suspension). For example, not coming to a complete stop at a stop sign is a violation of traffic laws, even though it may directly lead to an accident only in some circumstances. However, detecting such behavior is critical because such violations degrade the ability of drivers to have common expectations about the driving behavior of others which in turn increases uncertainty in the decision-making process, making incidents and mishaps more likely. Therefore, we aim to develop specifications to detect behavior that not only directly leads to mishaps, but also that may serve as upstream indicators of future failure states. Such an approach is equally applicable to systems like space exploration vehicles for which there are concerns about system properties such as liveness, performance, accuracy, and precision.

3) Does an assertion flagging provide any useful information?: Yes, if the assertion is applied within a well-managed verification repository. Assertions in a well-managed verification repository are accompanied by three primary attributes: preconditions, organization, and unit testing. First, the assertion has pre-conditions which specify the conditions that must be true in order for that assertion to be applied. For example, minimum speed limits are typically only specified on highways in the United States, so if the minimum speed limit is calculated at 75% of the actual speed limit, reporting a speed of 12 mph in a 25 mph zone located in a city as a violation is not meaningful. Secondly, assertions in the repository should have some form of organization which allows specifications to be stored in an accessible way for automated analysis and reuse from design time to runtime. The organization will be domain specific, but for autonomous vehicles could include categories such as vehicle positioning, parking, response to traffic signs, pedestrian interaction, etc. Each of those categories could be further split into environmental categories such as city driving, highway driving, slippery roads, and driving in fog. Organization allows for common pre-conditions to be applied to a set of assertions, as well as provides relevant context when an assertions pops. Lastly, an often overlooked aspect of creating usable formal specifications is testing the specification itself. Without evaluation isolated from the system, testers will have difficulty determining if violations result from a true system violation rather than an incorrect specification. Goodloe argues that "Ideally, specification verification capabilities should be integrated into the RV framework so engineers could write specifications, verify their correctness, and generate monitors in a seamless fashion" [23]. Simulation can play a vital role in ensuring an assertion is correct and makes sense in the context for which it is applied [20].

4) Will assertion checking introduce significant latency?: Many CPSs depend on high-data-rate sensors and near-real-time response thresholds to accomplish safety- and mission-critical tasks. Therefore, many developers write off specification-based RM as unrealistic for their system due to the anticipated latency introduced by evaluating specifications. There are two primary ways to conduct RM: online and offline. Online RM actively checks assertions as the simulation runs (or the system operates in its environment). If a violation is detected, system designers can use such information to execute a "safing maneuver" in which the system executes a predetermined routine to reach a safe state, or issue a warning or alert to a human operator who uses his or her judgement to take some action, such as retaking manual control or instructing the system on how to proceed. Depending on how assertion checking is implemented, online RM may introduce unacceptable latency into the system's operation, though there are many methods to ensure noninterference between the monitor and system [23]. Offline assertion checking uses captured data that is recorded and then passed to the monitoring system to evaluate retrospectively. Such data can then be used to modify the system's operation on a specified interval (e.g., daily) or to investigate failures. For safety-critical systems like AVs, offline assertion checking can be particularly useful for establishing cause-effect relationships, since conducting assertion checking against the system's data logs can reveal violations that may be causal factors relevant to accident investigations.

B. Devil's Advocate Objections to Machine-Learned Specifications

1) Since you are generating a rather large set of "1"-labeled scenarios for training, can they be applied directly to test the SUT, thereby rendering the ML specification re-

dundant?: More specifically, the objection is as follows. A common approach to testing is to identify failures by searching for operational modes that violate the specifications, a process known as falsification. These operational modes, often called "critical paths," are then used to modify the system's operation to reduce the frequency and severity of failures. In this case, the search for violations is done prior to testing; it is done as part of the process of generating data to train the ML classifiers (Section IV-A). The question then arises, "will you reuse those scenarios used to train the classifiers on 'bad' behavior to detect that same 'bad' behavior when the system is tested? If so, doesn't that make the assertions redundant?"

The answer to this concern is as follows. In ML the underlying model uses training to learn the transformation that maps input data to a target, and that transformation becomes generally applicable to data *outside* the training set. In the AV case, the inputs are time-sequenced features that describe a driving scenario, and the target is a Boolean representing whether the behavior is correct or not. Hence, the "1"-labeled training data is but a limited subset of the overall domain of scenarios that contain incorrect vehicle behavior; a subset from which the classifier learns the general relationship between the driving scenario and the Boolean correctness determination. The operational scenarios witnessed during RM of the SUT will inevitably include environmental factors (e.g., rainy weather conditions instead of clear, smaller lane widths) and vehicle path and state sequences that differ from those in the training set. Because the machine-learned specifications generalize from the training data, a major benefit lies in their ability to capture and identify undesirable behavior in scenarios not explicitly defined in their training data. Note that, when a ML assertion does not extend to data it was not trained on, this is known as over-fitting, a concern common to all ML classifiers.

2) How do you ensure the assertions are explainable?: Though traditional TL assertions can suffer from poor readability, the fact that they are developed in a one-to-one relationship with a requirements document provides explainability by association. For ML models, their behavior is governed by their training data, and the iterative search for useful data transformations (the *learning* part of machine learning) is less transparent. In addition, one ML model may represent multiple NL requirements that cannot then be easily separated. The explainability of ML specifications depends on the explainability of the data used to train the ML classifiers. Fortunately, the use of simulation to generate training data provides an instant sanity check on whether or not that simulation represents "good" or "bad" behavior. It is more challenging to visualize a condition under which Formula 1 (The MTL formula in Section II) evaluates to True or False. Because the assertions are trained using an environment that is readily understandable by humans, the explainability of ML assertions builds trust in the V&V process and the underlying dependability of the system.

3) Is generating machine-learned assertions extremely resource-intensive?: Generating paths to train/test ML classifiers can be a resource-intensive process, especially with a simulator in-the-loop (e.g., CARLA). The average time required to simulate an individual path such as the one discussed in Section IV-B was approximately two seconds on a typical desktop machine (Intel®Core™i7 with 16GB of RAM). In our computing environment, each round of CE executed 2000 simulations and usually converged in about 10-15 rounds, meaning it took at least 10 hours to produce a single "1"-labeled example. We later implemented some optimizations in the CE code which reduced this time, but the wall-clock time remains on the order of hours. Some of the time requirement is simulator-specific and is a result of the realistic graphics and physics-based calculations done in CARLA, but generating data from simulation is resource-intensive.

This problem can be dealt with by parallelization using n machines each finding a "1" path for a specific cost function and/or specific assertion; using 100 low-cost machines would reduce the average time for producing a "1" path to 6 minutes, using the same hardware configuration. Moreover, the approach can be implemented on a cloud-based environment such as Google E2 virtual machines (VM) currently costing less than $0.25 per hour for an 8 CPU, 32 GB memory system, or $2.50 per "1" path; this amounts to $500 for a combined set of 100 train and 100 test "1" paths that some ML classifiers require. This cost drops to less than $100 per assertion when using spot pricing for 16GB machines (CARLA's memory requirement). We believe that the cost of developing and maintaining a correct formal specification assertion by a human is more than that. Moreover, AV companies already have readily available data that can be used for ML training, in the form of log-files obtained from driving their autonomous vehicles across the country and elsewhere in the world,

VI. Future Work

This paper details our current data generation framework, a necessary prerequisite to training and using machine-learned correctness properties. As our research continues, we intend to train ML classifiers to serve as specifications, and test their suitability to assert about vital properties of autonomous CPSs.

Additional areas of work include classification beyond simply binary "good" and "bad." Because the cognitive processes that humans desire to delegate to machines often involve levels of risk, we believe it's necessary to sometimes classify autonomous behavior into multiple classes. Finally, we believe that it is necessary to investigate the organization of specifications. We hypothesize that the way in which specifications are organized will influence their explainability and long-term knowledge management.

Disclaimer

REFERENCES

[1] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, Jan 2004. [Online]. Available: https://doi.org/10.1109/TDSC.2004.2

[2] K. Gundy-Burlet, "Validation and verification of ladee models and software," in *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2013. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2013-592

[3] S. Lampérière-Couffin, O. Rossi, J.-M. Roussel, and J.-J. Lesage, "Advances in formal methods for the design of analog/mixed-signal systems," in *Proceedings of the European Control Conference*, 1999. [Online]. Available: https://10.23919/ECC.1999.7099641

[4] S. Goel, A. Slobodova, R. Summers, and S. Swords, "Balancing automation and control for formal verfification of microprocessors," *LNCS*, vol. 12759, July 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-81685-8_2

[5] V. Dubikhin, C. Myers, D. Sokolov, I. Syranidis, and A. Yakovlev, "Advances in formal methods for the design of analog/mixed-signal systems," in *Proceedings of the 54th Annual Design Automation Conference*, 2017. [Online]. Available: https://doi.org/10.1145/3061639.3072945

[6] J. Voas and K. Schaffer, "Whatever happened to formal methods for security?" *Computer*, vol. 49, no. 8, Aug 2016. [Online]. Available: https://doi.org/10.1109/MC.2022.3142829

[7] J. B. Michael, G. W. Dinolt, and D. Drusinsky, "Open questions in formal methods," *Computer*, vol. 53, no. 5, May 2020. [Online]. Available: https://doi.org/10.1109/MC.2020.2978567

[8] M. Clark, K. Kearns, J. Overholt, K. Gross, B. Barthelemy, and C. Reed, "Air force research laboratory test and evaluation, verification and validation of autonomous systems challenge exploration," Air Force Research Lab, Wright-Patterson AFB, Tech. Rep., 2014. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA614199

[9] W. A. Hunt, Jr., M. Kaufmann, R. B. Krug, J. S. Moore, and E. W. Smith, "Meta reasoning in acl2," *LNCS*, vol. 3603, August 2005. [Online]. Available: https://doi.org/10.1007/978-3-030-81685-8_2

[10] A. N. Prior, *Time and Modality*. Oxford University Press, 1957.

[11] J. B. Michael, D. Drusinsky, and D. Wijesekera, "Formal verification of cyberphysical systems," *Computer*, vol. 54, no. 9, Sep 2021. [Online]. Available: https://doi.org/10.1109/MC.2021.3055883

[12] D. Drusinsky, *Modeling and Verification Using UML Statecharts*. Elsevier Science, 2011.

[13] Y. Zhao and K. Y. Rozier, "Formal specification and verification of a coordination protocol for an automated air traffic control system," *Science of Computer Programming*, vol. 96, no. 3, Dec 2014. [Online]. Available: https://doi.org/10.1016/j.scico.2014.04.002

[14] D. Drusinsky, M. Litton, and J. B. Michael, "Lightweight verification and validation of cyberphysical systems using machine-learned correctness properties," *Computer*, vol. 55, no. 2, Feb 2022. [Online]. Available: https://doi.org/10.1109/MC.2022.3142829

[15] I. Perez, F. Dedden, and A. Goodloe, "Copilot 3," NASA Langley Research Center, Hampton, VA, USA, Tech. Rep. TM-2020-220587, 2020. [Online]. Available: https://ntrs.nasa.gov/citations/20200003164

[16] M. Davies, "System wide safety workshop: NASA's complex autonomous systems assurance technical challenge," Online, Mar. 2021.

[17] A. Zeller, "Specifications for free," *NFM 2011: Proceedings of the NASA Third International Symposium on Formal Methods*, vol. 6617, Apr. 2011. [Online]. Available: https://doi.org/10.1007/978-3-642-20398-5_2

[18] "CARLA. *Open-source simulator for autonomous driving research*," Accessed June 23, 2022. [Online]. Available: https://carla.org/

[19] F. Eskafi, D. Khorramabadi, and P. Varaiya, "An automated highway system simulator," *Transportation Research Part C: Emerging Technologies*, vol. 3, no. 1, 1995. [Online]. Available: https://doi.org/10.1016/0968-090X(94)00013-U

[20] C. Heitmeyer, "On the need for practical formal methods," *LNCS*, vol. 1486, Dec 1998. [Online]. Available: https://doi.org/10.1007/BFb0055332

[21] N. Kalra and S. M. Paddock, "Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, no. C, 2016. [Online]. Available: https://ideas.repec.org/a/eee/transa/v94y2016icp182-193.html

[22] P. T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, Feb 2005. [Online]. Available: https://doi.org/10.1007/s10479-005-5724-z

[23] A. Goodloe, "Challenges in high-assurance runtime verification," in *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*, 2016. [Online]. Available: https://ntrs.nasa.gov/api/citations/20160012454/downloads/20160012454.pdf

List of References

[1] D. Drusinsky, M. Litton, and J. B. Michael, "Lightweight verification and validation of cyberphysical systems using machine-learned correctness properties," *Computer*, vol. 55, no. 2, Feb. 2022. Available: https://doi.org/10.1109/MC.2022.3142829

[2] D. Drusinsky, J. B. Michael, and M. Litton, "Machine-learned specifications for the verification and validation of autonomous cyberphysical systems," in *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2022 [Online]., pp. 333–341. Available: https://doi.org/10.1109/ISSREW55968.2022.00089

[3] J. M. Anderson, N. Kalra, K. D. Stanley, P. Sorensen, C. Samaras, and T. A. Oluwatola. *Autonomous Vehicle Technology: A Guide for Policymakers*. RAND Corporation, Santa Monica, CA, USA, 2016 [Online]. Available: https://www.rand.org/pubs/research_reports/RR443-2.html

[4] National Research Council, *Autonomy research for civil aviation: toward a new era of flight*. Washington, DC, USA: The National Academies Press, 2014.

[5] M. A. Clark, X. D. Koutsoukos, J. Porter, R. Kumar, G. J. Pappas, O. Sokolsky, I. Lee, and L. Pike, "A study on run time assurance for complex cyber physical systems," Air Force Research Labs, Tech. Rep. ADA585474, 2013 [Online]. Available: https://apps.dtic.mil/sti/citations/ADA585474

[6] W. Wetsig, "Congressional report commends AFRL for life-saving collision avoidance technology; integrated air and ground system remains near transition," AFRL, Jul. 21, 2021 [Online]. Available: https://www.afrl.af.mil/News/Article-Display/Article/2678754/congressional-report-commends-afrl-for-life-saving-collision-avoidance-technolo/

[7] Partners for Automated Vehicle Education, "Pave poll: fact sheet," McLean, VA, USA, Tech. Rep., 2020 [Online]. Available: https://pavecampaign.org/wp-content/uploads/2020/05/PAVE-Poll_Fact-Sheet.pdf

[8] J. Galliott and A. Wyatt, "Risks and benefits of autonomous weapon systems," *Journal of Indo-Pacific Affairs*, vol. 3, Dec 2020. Available: https://media.defense.gov/2020/Nov/23/2002540369/-1/-1/1/WYATT.PDF

[9] Lockheed Martin, "Saving the Good Guys with Auto GCAS Technology," Accessed Oct. 27, 2022 [Online]. Available: https://www.lockheedmartin.com/en-us/products/autogcas.html

[10] R. Stumpf, "Europe now requires all new cars to have anti-speeding monitors," The Drive, Jul. 6, 2022 [Online]. Available: https://www.thedrive.com/news/europe-now-requires-all-new-cars-to-have-anti-speeding-monitors

[11] Department of the Navy, "Department of the Navy science and technology strategy for intelligent autonomous systems," Washington, DC, USA, Tech. Rep., 2021 [Online]. Available: https://www.nre.navy.mil/media/document/department-navy-science-technology-strategy-intelligent-autonomous-systems

[12] R. Franklin, "U.S. 5th fleet launches new task force to integrate unmanned systems," CUSNC, Sep. 9, 2021 [Online]. Available: https://www.cusnc.navy.mil/Media/News/Display/Article/2768468/us-5th-fleet-launches-new-task-force-to-integrate-unmanned-systems/

[13] D. Drusinsky, *Modeling and verification using UML statecharts*, 1st ed. Burlington, MA, USA: Elsevier, 2006.

[14] California Department of Motor Vehicles, "Disengagement reports," Dec. 27, 2022 [Online]. Available: https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/disengagement-reports/

[15] P.-T. de Boer, D. P. Kroese, M. Shie, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, Feb. 2005. Available: https://doi.org/10.1007/s10479-005-5724-z

[16] SAE International Surface Vehicle Recommended Practice, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," SAE Standard J3016, Rev. Jun. 2018. Available: https://www.sae.org/standards/content/j3016_201806/

[17] M. F. Stumborg, B. Roh, and M. Rosen, "Dimensions of autonomous decision-making," Center for Naval Analyses, Tech. Rep. DRM-2021-U-030642-1Rev, 2021 [Online]. Available: https://www.cna.org/reports/2022/01/Dimensions-of-Autonomous-Decision-making.pdf

[18] S. Birla, "U.S. Navy takes Falkonry AI to the high eeas for increased equipment reliability and performance," businesswire, Oct. 19, 2022 [Online]. Available: https://www.businesswire.com/news/home/20221019005482/en/U.S.-Navy-Takes-Falkonry-AI-to-the-High-Seas-for-Increased-Equipment-Reliability-and-Performance

[19] D. Subramani and S. Araujo, "Demystifying machine learning at the edge through real use cases," AWS, Jun. 15, 2022 [Online]. Available: https://aws.amazon.com/

blogs/machine-learning/demystifying-machine-learning-at-the-edge-through-real-use-cases/

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. RDML Kurt J. Rothenhaus, USN
   Program Executive Officer, C4I
   San Diego, California

4. Dr. Robert E. Parker, Jr.
   PEO C4I Technical Director
   SSTM, Naval Information Warfare Center Pacific
   San Diego, California