Explaining Multimodal Errors in Autonomous Vehicles

Leilani H. Gilpin[§]
Department of Computer Science and Engineering
University of California Santa Cruz
Santa Cruz, CA
lgilpin@ucsc.edu

Vishnu Penubarthi and Lalana Kagal
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA
{vishnu, lkagal}@mit.edu

2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA) | 978-1-6654-2099-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/DSAA53316.2021.9564178

***Abstract*—Complex machines, such as autonomous vehicles, are unable to reconcile conflicting behaviors between their underlying subsystems, which leads to accidents and other negative consequences. Existing approaches to error and anomaly detection are not equipped to detect and mitigate *inconsistencies among parts*. In this paper, we present "Anomaly Detection through Explanations" or ADE, a multimodal monitoring architecture to reconcile critical discrepancies under uncertainty. ADE uses *symbolic explanations* as a debugging language, by examining underlying *reasons* for those decisions. Further, when decisions conflict, our method uses a *synthesizer*, along with a priority hierarchy, to process subsystem outputs along with their underlying reasons and transparently judges the conflicts. We show the accuracy and performance of ADE on autonomous vehicle scenarios and data, and discuss other error evaluations for future work.**

***Index Terms*—explainable artificial intelligence, complex systems, autonomous vehicles, anomaly detection**

I. Introduction

When complex machines are deployed in real world settings, they have failed; leading to injuries[1] and even deaths [1]. Such level of increased harm on human lives is undesirable and completely untenable. Consequently, assessing pre-deployment reliability is of essential importance for machines that are responsible for decisions that impact life and property. One key component to ensure pre-deployment verification of any sort of machine is *an explanation*; a model-dependent artifact providing an end-user (technical or otherwise) reason or justification for all decisions made by that machine. Unfortunately these explanations do not usually describe errors; they are simply positive justifications of *correct behavior*.

Instead, we focus on explanations of *erroneous* behavior. There are two ways that an existing, working, complex mechanism can fail. One way is a *local* failure, that can be pinpointed to a specific subsystem or component. There is a multitude of work on *anomaly detection* [2], [3] that tries to detect these types of errors. But the second type of failure, which is the common cause of failures, especially in self-driving vehicles [1], is due to a failed *communication* amongst subsystems. In this paper, we present a methodology: Anomaly Detection through Explanations (ADE), that applies explanatory subsystem monitoring to full system design. This is both an architecture and a reasoning methodology: local subsystems are encapsulated in a local monitor that imposes "sanity checks" to explain local decisions. These explanations (both symbolic and translated into human-readable text) motivate a new view of anomaly detection, where *symbolic explanations* facilitate a common debugging language.

In the case of inconsistent monitoring outputs, an explanation synthesizer, described in Section V, can reconcile inconsistencies amongst subsystems. This paper proceeds as follows: we motivate the problem in the next section (Section II), and discuss the technical contributions: the system architecture (Section IV) and the explanation synthesizer (Section V). We present evaluation results in Section VI and conclude with a discussion of limitations and future work.

II. The Problem

Complex machines have limited internal reasoning. In some cases, they have none; causing an inability to reason about their behavior. This is especially troublesome for self-driving vehicles; where adversarial attacks can trick a vision system into perceiving a stop sign as a 45 mph sign with a few pieces of tape [4], or overfit algorithms become increasingly concerning when they are applied to pedestrian detection [5]. When connected to self-driving applications or other mission-critical, safety-critical applications, this can lead to *consequences*.

In order to provide complex machines with more descriptive and interpretable internal reasoning, we present a system-wide architecture that facilitates better *communication* amongst parts. While the types of explanations that are prevalent in the literature are targeted to an end user, our explanations are *symbolic*, and they improve machine level understanding. The explanations are represented as symbolic triples, which are translated into natural language text for an end-user[2]. These local explanations are generated by querying commonsense rules and knowledge that are "related" to the local decision.

When there are inconsistencies amongst parts, our architecture automatically decides which subsystem or component

[§]Work was done as a PhD candidate at MIT CSAIL.

[1]Mall robot injures a toddler: https://qz.com/730086/a-robot-mall-cop-did-more-harm-than-good/

[2]Results are translated into natural language text for demonstration purposes. They are generated with templates, but represented as symbolic triples.

is the most "correct" or "reasonable". To deal with conflicting explanations, we built an explanation synthesizer; which examines the input explanations and validates that they do not violate any high-level rules from a priority hierarchy. The priority hierarchy defines (in order) which concerns are the most important. For this paper, we address self-driving vehicles and focus on passenger safety–avoiding threatening objects that can cause damage to the vehicle and its occupants. This priority hierarchy can be defined at various levels of detail.

In this paper, we demonstrate how using an *interpretable* framework can reconcile inconsistencies between subsystems using *symbolic explanations*. We provide a qualitative evaluation by simulating the Uber self-driving vehicle accident [6]. We also present results from quantitative analysis on existing self-driving data set [7] by inserting errors and using various synthesizer configurations in Section VI. Our work shows promising results to robustly detect errors in complex machines with safety or mission critical goals.

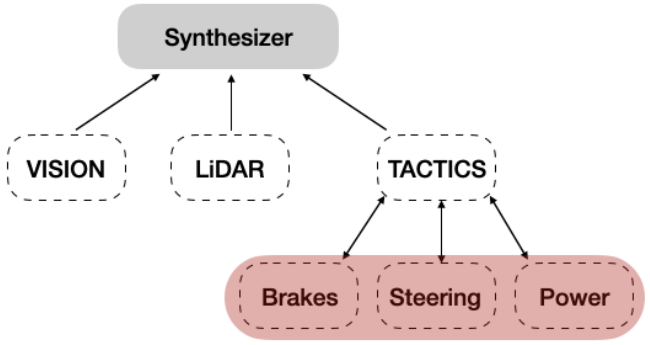

Fig. 1. The Anomaly Detection through Explanation (ADE) architecture for a simplified self-driving vehicle. The red region is the tactics committee; a 2-way communication between the brakes, steering, and power subsystems. Each subsystem is encapsulated in a reasonableness monitor (the dotted outline). The reasonableness monitor output at the higher level subsystems: the vision, LiDAR, and tactics subsystem's monitors are passed to the explanation synthesizer to reconcile inconsistencies between subsystems.

III. Related Work

The main goal of this work is to use *internal subsystem explanations* to handle uncertainty amongst the components of a complex system. While some solutions have developed novel testing protocols for cyber-physical systems like self-driving cars [8] during development, our system can be used during development and deployment to detect and monitor uncertainties between parts. Anomaly detection is a well-studied area in data science and machine learning [2], [3], even as a tactic to combat intrusion detection in networks [9]. In developing anomaly detection for autonomous systems, it is also necessary to develop real-time anomaly detection algorithms. Real-time anomaly detection needs a novel scoring algorithm designed for streaming data, including a series of benchmarks [10]. However, decreasing the number of false-positives and false-negatives in anomaly detection is a difficult problem. Some tactics include smoothing the output [11], or piece-wise approximations [12].

Many anomaly detection systems employ filtering methods at the sensor level [13], while others focus on communication between sensors and external roadside infrastructure [14]. However, these techniques are deficient as they only focus on detecting sensor anomalies at the local level, but do not account for errors due to failed cooperation between *internal subsystems*, which is an issue we address with our framework.

Our work is similar to work in diagnostics [15], [16] but extends these approaches with a flexible representation and commonsense knowledge. Although similar types of explanations and commonsense reasoning has been explored in robotics [17], our system is able to handle ambiguity through continuous introspection.

Another goal is to make autonomous systems naturally resistant to intrusion through monitoring and continuous introspection. Some approaches rely on a combination of topological vulnerability analysis and system alert data to detect attacks [18]. Other approaches are specifically for collections of autonomous flying vehicles, directly examine deviations from expected control algorithm behavior to detect faulty agents and route communication around them [19]. The primary deficiency of these approaches is that, while they provide fault detection, they do not attempt any explanation of how the faults might have arisen, which is a deficiency we hope to address through our work.

In order to ensure our framework is flexible and can be adapted to multiple platforms, we stored our commonsense rules in the RDF language. Other symbolic approaches include a confabulation network which stores symbolic reasoning within nodes [20], and an application of symbolic reasoning in the domain of smart cities [21]. Our approach and position is similar to that proposed in "explainable agency" [22]. This refers to the ability of autonomous agents to explain their decisions and be questioned. Although we adhere to many of the principles of explainable agency, our goal is to extend these principles to *full system design*.

IV. System Architecture

The ADE architecture is composed of local, explanatory monitors around each subsystem and an explanation synthesizer to reconcile inconsistencies. Based on the architecture, there are three distinct steps in anomaly detection: generating qualitative outputs for each subsystem or committee (groups of subsystems), followed by monitoring/explaining the outputs of each subsystem, and finally, reconciling inconsistencies or synthesizing explanations. A flow diagram of this process for a self-driving car is in Figure 1. In this application, there are three key subsystems: the vision, LiDAR, and tactics subsystems. Note that the tactics subsystem is composed of the brakes, steering and power controls. Although this representation is simplified, it encapsulates the key subsystems for self-driving decision making (and also the subsystems that are prevalent in self-driving car data).

The dotted borders around the components signify reasonableness monitors, the red component is the actuation committee, which reports to the tactics monitor. And finally, the grey component is the explanation synthesizer, which receives explanation outputs from the vision, LiDAR, and tactics components. Note, that this is a simplified architecture of a self-driving vehicle, and many more components could be added for more precise reasoning. However, this architecture represents all the components necessary for reconciling the inconsistencies in the Uber self-driving case and for reconciling the inconsistencies in our self-scrambled self-driving data set.

A. Interpretting Subsystem Output

The outputs of the subsystems of a self-driving vehicle, or any complex machine, vary in abstraction. The outputs can be a mostly symbolic list of high-level concepts, like the labels output by an opaque vision processing unit, or they can be uninterpretable logs of data points, like the "point clouds" output from the LiDAR subsystem. Note, that these sensor outputs are typical of other "lower level" subsystems of the vehicle, like braking, steering, power control, etc. In order to reconcile inconsistencies between subsystems, they need to be able to speak a common language. The first step towards providing subsystems with the common *explanation* language, is to provide them with a qualitative language to interpret their own outputs.

Each subsystem's output is translated into a qualitative description. For example, the raw data from the tactics systems is translated into the symbolic description: `(vehicle, moving, quickly), (vehicle, proceeding, straight)`. The LiDAR subsystem produces a similar description of the *surrounding environment* (e.g., `(pedestrian, inFrontOf, vehicle)`). For the vision system, these are the output labels from the vision processing subsystem. For the LiDAR system, this is sensor interpretation, which translates raw data into a symbolic description. This is similar to describing the "tactics committee" (which corresponds to the brakes, steering, and power control) with a model-based qualitative summary and analysis generated from previous work [23]. These qualitative descriptions are represented as a list of descriptions, which are then input into a monitoring system [24] for additional symbolic support: reasons supporting the output and an explanation of reasonableness.

B. Monitoring Local Subsystems

Reasonableness monitors examine the output of a subsystem, supplement this output with commonsense knowledge and rules, leading to a judgement and explanation of reasonableness (citation removed for anonymity). For subsystems that perceive the world (e.g., vision and LiDAR), these rules are commonsense knowledge pertaining to size and location of objects. For example, pedestrians and vehicles are found near the street (queried from commonsense as the symbolic triples: `(vehicle, locatedNear, street), (pedestrian, locatedNear, street)`. Or they can be deductions from commonsense: a mailbox is an object that cannot move on its own. The commonsense reasons are queried from ConceptNet [25], and the rules are built into a primitive representation.

For lower-level components (e.g., the Tactics subsystems) commonsense rules follow from safe-driving rules. For example, a vehicle traveling at a high speed should not make a sudden stop. We have also extended these rules to include qualitative physics knowledge for the vehicle. For example, safe turns occur when the vehicle slows down and changes steering slowing. The outputs of the reasonableness monitors are a judgement and explanation of reasonableness. Note that an unreasonable judgement does not necessarily indicate that a subsystem should be ignored; instead it indicates that the subsystem's output should be discounted in future decisions.

C. Conflicting Explanations

If there are conflicts between explanations, the explanation synthesizer will choose which subsystem's output to trust or discount. The synthesizer uses a priority hierarchy, which indicates, in order, which priorities are the most important. If any of those priorities are violated, then that subsystem is examined and its output is discounted in future decision making. Since the explanation synthesizer requires explanations from its underlying parts.

V. Explanation Synthesizer

Inconsistencies in driving occur all the time. Some of these inconsistencies are critical. If the vehicle operator cannot identify a large object in the road, it should stop, even if it unsure of the *type* of object. Large objects, especially ones that are moving, are threats to the vehicle and the people inside. Smaller unidentified objects are not usually harmful to the vehicle. Moving transparent objects, like plastic bags, should be ignored. But small sharp objects, like nails, or potholes and uneven paving can can flat tires, leading to accidents and discomfort for the vehicle and those inside. It is important to determine, at a high-level, which objects are threats, and which are not. Most autonomous vehicles struggle with this sort of reasoning, causing consistent starting and stopping behavior[3].

Therefore we created an explanation synthesizer reconcile these types of inconsistencies. It processes the explanations from its underlying parts, synthesizes (or summarizes) the explanations, and automatically determines which subsystems to trust or discount in critical situations. Since the input to the synthesizer are explanations with symbolic reasons, the synthesizer automatically constructs an argument tree to support its reasoning and judgement. If more information is available, or an auditor is able to validate or remove information, the argument tree can be automatically augmented and queried for counterfactual support.

[3]"Herky jerky" or constant starting and stopping is common behavior in self-driving: https://www.wired.com/story/ride-general-motors-self-driving-car/. This discomfort lead many vehicle manufacturers to choose to ignore objects if they were identified confidently, instead of trying to reason about which objects are threats (or not).

A passenger is safe if:
The vehicle proceeds at the same speed and direction, AND
The vehicle avoids threatening objects.

Fig. 2. The goal of passenger safety in natural language.

A. Priority Hierarchy

We created a priority hierarchy to represent the key priorities for safety in self-driving. The priorities are as follows (in order):

1) Passenger Safety
2) Passenger Perceived Safety
3) Passenger Comfort
4) Efficiency (e.g., route efficiency)

Note that these priorities are *intentionally* vague, as to avoid some of the ethical questions about specific obstacles [26]. In this paper, we specifically focus on goals related to (1) Passenger Safety, which can be expressed as the abstract goal in Figure 2.

B. Underlying Logic

The abstract rules that correspond to each priority are implicitly represented in first-order logic. Let's focus on the high level goal for the first priority: passenger safety in Figure 2.

This goal can be strongly defined in first-order logic. Let s and t be successive states, and let v be a qualitative description of the vehicle's (self's) velocity.

Then $\forall s, t \in STATE, v \in VELOCITY$

$$\begin{aligned}&((self, moving, v), \text{state}, s) \wedge \\ &\quad (t, \text{isSuccesorState}, s) \wedge \\ &((self, moving, v), \text{state}, t) \wedge \\ &\quad \nexists x \in OBJECTS \text{ s.t.} \\ &\quad ((x, isA, threat), \text{state}, s) \\ &\vee ((x, isA, threat), \text{state}, t) \\ &\qquad \Rightarrow (\text{passenger, hasProperty, safe})\end{aligned} \tag{1}$$

Which means that as long as the vehicle is moving at the same speed, and there are no threatening objects in the way of the vehicle, then the passenger is safe (and satisfying the first rule in the priority hierarchy). However, it is left to define what is a threatening object. This can be similarly defined in first-order logic, using the same variables, s for the state, and let x be an object and v be a velocity.

Then $\forall s \in STATE, x \in OBJECT, v \in VELOCITY$:

$$\begin{aligned}&((x, moving, v), \text{state}, s) \wedge ((x, near, self), \text{state}, s) \wedge \\ &((x, isA, large\_object), \text{state}, s) \Leftrightarrow ((x, isA, threat), \text{state}, s)\end{aligned} \tag{2}$$

These first-order logic properties are used to define rules in the explanation synthesizer.

```
IF ( AND('moving (?v) at state (?y)',
         '(?z) succeeds (?y)',
         'moving (?v) at state (?z)'),
   THEN('safe driving at (?v)
        during (?y) and (?z)'))

IF (OR('(?x) is not moving',
       '(?x) is not located near',
       '(?x) is not a large object')),
   THEN('(?x) not a threat at (?y)'))

IF (AND('(?x) not a threat at (?y)',
        '(?x) not a threat at (?z)',
        '(?y) succeeds (?z)',
   THEN('(?x) is not a threat
        between (?y) and (?z)'))
```

Fig. 3. The set of python rules used to define passenger safety. Variables are designed with the `(?x)` syntax.

C. Abstract Rules

Using the logic described above, we can represent these rules in an abstract goal tree:

```
'passenger is safe',
    AND(
    'safe transitions',
    NOT('threatening objects')
```

Where this rule is used to discount subsystems that violate this goal in the case of inconsistent information. The other abstract rules for the other properties are similar; namely for the (2) priority: perceived safety.

```
'passenger perceived safe',
    AND(
    'safe transitions',
    NOT('reckless driving')
```

Where this rule is used to discount sudden stops, turns, and other driving tactics that are deemed reckless. Note, the "gliding through a stop sign" example falls into this category as well.

D. Goal Tree

ADE can be used to explore alternatives and counterfactual reasoning. Each of the rules is written in python with rule keywords: `IF`, `AND`, `OR`, `NOT`, `THEN`. An example of the passenger safety rules are in Figure 3. Forward chaining is used to determine if any constraints are violated, and backwards chaining is used to generate the goal tree in Figure 4.

VI. Evaluation

We present two studies. The first evaluation study is a simulation of the Uber Accident. From the simulation environment, we extracted logs, and we validated that the logs generated

```
passenger is safe at V between s and t
AND (AND (moving V at state s
          t succeeds s
          moving V at state t )
     AND (
          OR ( obj is not moving at s
               obj is not located near at s
               obj is not a large object at s)
          OR ( obj is not moving at t
               obj is not located near at t
               obj is not a large object at t)))
```

Fig. 4. The goal tree for a single object that satisfies passenger safety.

were similar to the accident data from their report [27]. We validated that the algorithm (1) detected the inconsistency and (2) could explain the inconsistency and why it was critical (in terms of the given priority hierarchy), and (3) that the explanation was true to the error explained in the Uber report [1].

The second study is a statistical experiment. We add errors to a subset of a self-driving car data set to determine the robustness of ADE on new samples.

A. Simulation in CARLA

The simulation modeled the events of the Uber accident scenario described in detail below. This simulation was performed using CARLA, an open-source simulator for self-driving research [28]. The primary data recorded in this simulation included raw images, semantic-segmented images, and LiDAR point clouds.

1) Uber accident scenario: On March 18, 2018, the first reported self-driving pedestrian fatality occurred. The vehicle was an Uber Technologies, Inc. test vehicle: a modified 2017 Volvo XC90 with a self-driving system in computer control mode. At approximately 9:58 p.m that evening, the vehicle struck a pedestrian on northbound Mill Avenue, in Tempe, Maricopa County, Arizona. Although the test vehicle had a human safety driver, they were not paying attention at the moments before impact.

The following is from the Uber accident preliminary report: "According to data obtained from the self-driving system, the system first registered radar and LiDAR observations of the pedestrian about 6 seconds before impact, when the vehicle was traveling at 43 mph. As the vehicle and pedestrian paths converged, the self-driving system software classified the pedestrian as an unknown object, as a vehicle, and then as a bicycle with varying expectations of future travel path. At 1.3 seconds before impact, the self-driving system determined that an emergency braking maneuver was needed to mitigate a collision. According to Uber, emergency braking maneuvers are not enabled while the vehicle is under computer control, to reduce the potential for erratic vehicle behavior. The vehicle operator is relied on to intervene and take action. The system is not designed to alert the operator.

We simulated this scenario in Carla [28], an autonomous vehicle simulator. Logs were output after each drive, and analyzed after-the-fact. Figures 6 and 7 are screenshots of the Uber accident simulation on the CARLA Platform.

2) Experimental setup: The simulation was performed on a machine running Ubuntu 18.04 LTS. The version of CARLA running on the machine at the time of testing was 0.9.6, and we recommend using this or a newer released stable version for testing as they contain important features necessary for recreating this simulation. In addition, CARLA renders the simulation using Unreal Engine, which has to be installed separately. We used Unreal Engine version 4.22 for testing.

3) Modeling the Uber case: In order to model the simulation after the Uber scenario, we created a copy of **manual_control.py**. We choose to manually drive the vehicle in the recreation of the accident rather than have it drive autonomously, which would have utilized **syncronous_mode.py**, because we were not able to replicate the motions of the Uber vehicle with enough precision with the vehicle driving autonomously.

The first modifications that needed to be made were selecting the virtual world and preparing log recordings, which was done within the `game_loop(args)` function of **manual_control.py**. Carla offers many virtual worlds to simulate, but we found **Town01** to most closely mirror the streets in which the Uber accident occurred, and we set this as the world within `game_loop(args)`. Next, we created a Python dictionary to store each `snapshot` produced by each iteration of the simulation within `game_loop(args)`, as each snapshot contains the LiDAR point clouds from that specific timestamp. Each key in the dictionary is the timestamp and the value is the timestamp's corresponding `snapshot`.

Most of the primary modifications were made under the `restart(self)` function, so that we could begin our model of the scenario upon hitting the *backspace* key while running CARLA. The main modifications were creating a car to represent the Uber vehicle and a pedestrian to represent the pedestrian:

```
pedestrian = random.choice(self.world.
    get_blueprint_library()
    .filter("walker.pedestrian.0007"))

vehicle = random.choice(self.world.
    get_blueprint_library()
    .filter("vehicle.nissan.micra"))
```

In the above code, the vehicle and the pedestrian chosen were closest models of the actual individual and vehicle we could find in the CARLA catalog. The vehicle was relatively similar in terms of the color and shape, but the pedestrian model is not walking with her bike, as was the case in the actual Uber accident. It is also important to set the `is_invincible` attribute of both of these blueprints to *false* so that the motions after collision follows the set simulated physics rules.

Next, we placed the pedestrian at (-81.5,23.0,100.0), which is a (x,y,z) position within the CARLA map representing a gap between a median strip on a highway. In addition, using `carla.WalkerControl()`, we had the pedestrian walk from the median strip towards the edge of the highway at a constant 1.7 m/s, as this closely mirrored the motion of the pedestrian in the Uber scenario.

We placed the vehicle at (24.2,-170.6,5.0), which is a location on a lane of the highway where the walker is initially placed. We chose this initial location for the vehicle as it allowed for the proper acceleration to mirror the speeds of the vehicle of the Uber accident. We also attached a LiDAR sensor, camera, and a semantic-segmentation camera to the vehicle in order to record the necessary data.

4) Running the Simulation: There were two runs of the simulation: one for the raw camera images and one for the semantic-segmented images, as we could not get CARLA to properly output both in one run. In both of the runs, the images were saved to a local folder with the name of the image being the timestamp of the `snapshot` the image was taken. In both the simulations, we tried to have the motion of the vehicle mirror that of the Uber vehicle. This included the position of the vehicle at the time of collision relative to the pedestrian in order to collect simulated LiDAR and image data that correlates to the actual data that is unreleased by Uber.
Consider The Uber scenario approximately 6 seconds before impact. The explanation synthesizer evaluates and explains these high-level plans as follows:

1) Continue straight.
2) Slow down to a stop.
3) Veer to the side of the road.

Note that these intended decisions are not necessarily output in this human-understandable way. But using edge-detection and interval analysis with explanations which was explored in previous work (citation removed for anonymity), we can directly generate these kinds of text explanations from symbolic descriptions.

5) Simulation results: In this scenario, the explanation synthesizer receives input from the computer vision (perception) system, the LiDAR/radar system, and the driving tactics (consisting of the brakes, steering, gas, etc.). The system-wide monitoring diagram for this example is shown in Figure 1. The vision system output is a set of segmentations and their corresponding labels (e.g. person, tree, road, etc.) For this Uber example case, we focus on the segmentation in the upper left (from the point of view of the car). In the seconds before impact, the output of the reasonableness monitor for the vision processing component is shown below:

```
This vision perception is unreasonable.
There is no commonsense data supporting
the similarity between a bike, vehicle and
unknown object except that they can be
located at the same location.
This component should be ignored.
```

But there is more sensory information: the LiDAR sensor data log. The LiDAR reasonableness monitor first *interprets* the sensor log. Using edge detection and interval analysis, the raw sensor data is abstracted into a list of symbolic descriptions that can be passed into the reasonableness monitor. The symbolic list produced for the LiDAR data in this scenario is `('object, 'moving, '5-ft-tall, 'top-left-quadrant, ...)`. In the seconds before impact, the output of the reasonableness monitor for the vision processing component is shown below:

```
This LiDAR perception is reasonable.
An object moving of this size is a large
moving object that should be avoided.
```

Finally, the tactics system is similarly interpreted into a symbolic, qualitative description: `('moving-quickly, 'straight, 'continued-straight, ...)` signifying that the vehicle has been proceeding straight, quickly for the last 5-10 second horizon. The reasonableness monitor for the tactics system deduces that that system state is reasonable: `The system state is reasonable given that the vehicle has been moving quickly and proceeding straight for the last 10 second history.`

With these three subsystem explanations, the high-level reasoner processes the explanations (which are also stored as a list of symbolic triples). The reasoner examines and assesses at the strengths of each explanation, and compares it to a hierarchy of needs to see which intended decision does not violate the the needs hierarchy. Several iterations of this process may be necessary for more complex decisions (or a more complicated needs hierarchy). For this proof-of-concept, the high-level reasoner explains each of the intended plans against the component explanations and hierarchy of needs:

1) Continue forward (straight): this would result in injuring the object detected by the LiDAR system. The vision system cannot confirm this detection and is deemed unreliable due to misjudgements. Therefore, the vehicle should not continue forward.
2) Stop: It is unclear if stopping would guarantee limited harm to the object detected by the LiDAR system. A sudden stop at the speed of the vehicle may injure its occupants. Therefore, the vehicle should not stop. (Although, this intended decision will remain a possible choice since it does not produce as much damage as the first option).
3) Veer and slow down: this would result in avoiding the object detected by the LiDAR system. The vision system cannot confirm this detection and is deemed unreliable due to misjudgements. This is consistent to safely avoid the object. Veering and slowing down causes less damage to the vehicle occupants.

And the final explanation produced by the high-level reasoner is shown in Figure 5.

6) Generating Qualitative Descriptions: From the Uber accident data analysis, the vehicle "vehicle was traveling at

```
The best option is to veer and slow down.
The vehicle is traveling too fast to
suddenly stop.
The vision system is inconsistent, but the
LiDAR  system has provided a  reasonable
and strong claim to avoid the object
moving across the street.
```

Fig. 5. The explanation synthesizer output for the Uber self-driving vehicle example.

43 mph" and "the self-driving system software classified the pedestrian as an unknown object, as a vehicle, and then as a bicycle with varying expectations of future travel path," and for the sensor information LiDAR had detected the pedestrian "about 6 seconds before impact."

Note, that due to limitations in Carla, it was not possible to simulate a pedestrian with a bicycle. For the LiDAR data, we manually augmented the detection to capture the length of the bicycle. Otherwise, the other raw data from the simulation is processed with edge a qualitative analysis. Since the vision system outputs symbolic concepts, we represented the outputs of the vision system as oscillating among: vehicle, bike, and unknown objects.

7) Monitoring: Each qualitative description is then run through a reasonableness monitor. The reasonableness monitors query commonsense knowledge and subsystem-specific rules to generate reasons and explanation of behavior. For example, consider the monitor for the vision system seconds before impact in the Uber accident case; where the vision system was oscillating between a vehicle, bike and unknown object. The monitor first supplements those outputs with commonsense knowledge: finding the objects' locations, sizes, and other relevant information. Then, the monitor checks this data against a set of rules: e.g. `sameLocationRule`, `reasonableSizeRule`, etc. If any rule consequents are not met, then the underlying subsystem is classified as unreasonable with a justification explaining of why. In this case, the monitor finds that the output labels are not similar enough: `There is no commonsense data supporting the similarity between a vehicle, bike and unknown object except that they can be located at the same location.`

Note, that these are also represented in symbolic triples to be used by the synthesizer.

8) Synthesizing: The explanations are also in terms of symbolic triples. Below are a few highlights of these reasons (the ones that are deemed "important" for the synthesizer). They are aggregated from querying commonsense from ConceptNet, and forward chaining a set of commonsense rules.

These symbolic reasons are passed to the explanation synthesizer, which finds that the LiDAR's data description violates the goal: `NOT ('threatening objects')`. The final output, is the following explanation:

```
The best option is to veer and slow down.
The vehicle is traveling too fast to
suddenly stop.

The vision system is inconsistent, but the
LiDAR system has provided a reasonable and
strong claim to avoid the object moving
across the street.
```

The alert in a real-time simulation environment is shown in Figure 7.

B. Adding Errors on Existing Data

Self-driving data sets [7], [29], [30] do not contain errors. They are largely hand-currated, so that the labelings and bounding boxes are perfect. The vehicle behavior is also smooth, so that there are not egregious mistakes in driving behavior. Since the data sets themselves do not have errors to explain, we decided to scramble labels on some of these data sets and attempt to detect those errors with our ADE architecture.

We randomly scramble the NuScenes data set [7] since it has data for dense traffic and challenging driving (focused on data in Boston and Singapore). It also has more camera data than other data sets: 360 view camera data, rather than the front facing camera data on KITTI [30]. NuScenes has 1000 different scenes (20 clips of driving) where each scene is split into frames (we refer to these as *snapshots*). Therefore, we differentiate between a frame error (a single point in time where obstacles are incorrectly labeled) and a scene error (a prolonged error that remains over several frames).

We invoked errors at the frame level. For each frame, 50% of the objects in that frame are chosen to be scrambled. This is performed randomly: when a frame is loaded, a `scrambleFlag` vector is randomized with 1's and 0's; corresponding to whether the object in question should be "scrambled" or not. We invoked two types of errors:

1) Label Errors: The "true" object label is replaced with another label.
2) "Noise" Errors: The object size is perturbed.

Label errors are chosen at random and retain the same label distribution on the data set. Noise errors are also chosen at random, but require careful manipulation. Naively adding random numbers to the object size clearly results in an error: for example, a car that is over 20 meters wide is clearly wrong. So instead, we added random noise from the range $[0, 2\sigma]$, where σ is the standard deviation of the object size in meters. Therefore we generate a set of noisy sizes, which are not "clearly wrong": they are near misses [31].

In order to demonstrate that using the synthesizer increases the accuracy of error detection, we present a series of experiments in Table I. We present three different configurations of the explanation synthesizer: no synthesizer (baseline), a naive preference (choosing one component over the other), and the "safety" priority. The accuracy and analysis is shown in Table I. This table contains the average of 10 randomized runs

Fig. 6. A screenshot of the Uber accident simulation (with similar lighting to the report) on the Carla Platform. The vehicle starts 25 meters away from the unknown object traveling at 43 mph.

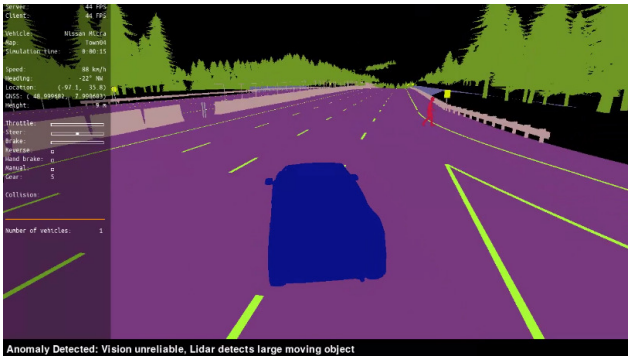

Fig. 7. Simulation snapshot of ADE on the Uber self-driving accident. The architecture indicates that the best option is to veer and slow down to avoid the large moving object highlighted in red: the pedestrian crossing the street.

on the 18,538 annotations (objects detected) in the NuScenes mini data set.

From the analysis, we see that synthesizing increases correctness of accuracy of detecting errors. Note that the method is susceptible to false positives and negatives. Since we are only adding a bit of noise to scramble the data, it is difficult to separate errors from noise. If instead, we added a lot of noise ($<< 2\sigma$), then the number of false positives and false negatives approaches zero, since the subsystem explanation *clearly* identifies a local error, and the synthesizer can confidently determine which system to discount.

Priority	Correctness	False +	False -
No synthesizer	85.6%	7.1%	7.3%
Single subsystem	88.9%	7.9%	3.2%
Safety	93.5%	4.8%	1.7%

TABLE I
A COMPARISON OF THE ADE RESULTS ON DIFFERENT SYNTHESIZER CONSTRAINTS. THE "NO SYNTHESIZER" OPTION SIGNIFIES THAT ALL SUBSYSTEM ERRORS ARE REPORTED. THE "SINGLE SUBYSTEM" OPTION MEANS THAT ONE SUBSYSTEM IS ARBITRARILY CHOSEN TO ALWAYS BE GIVEN PREFERENCE. THE "SAFETY" CONSTRAINT IS OUR DEFAULT, WHERE "THREATENING" OBJECTS ARE GIVEN A HIGHER PRIORITY (TO BE CLASSIFIED AS ERRONEOUS.)

We also show two examples of the explanation synthesizer output in Table II of the object detected in Figure 8. In the case that no errors are detected, the synthesizer outputs a summary. Note that we have focused on generating frame (or snapshot) errors because generating scene errors (video) consistently requires all frames to keep the same scrambled labels throughout each frame. This requires careful tracking of objects, which is out of the scope of this paper.

1) Examples: Now for the scene monitor, which uses the explanation synthesizer to summarize the findings.

```
This scene with with 225 distinct
annotations 2018-07-23 23:29:06.797517
is reasonable.

Explanation synthesized: Reasonable scene
with  a car proceeding forward, turning
left, and following  a van.
Perceived a parked truck, construction,
and intersection.
```

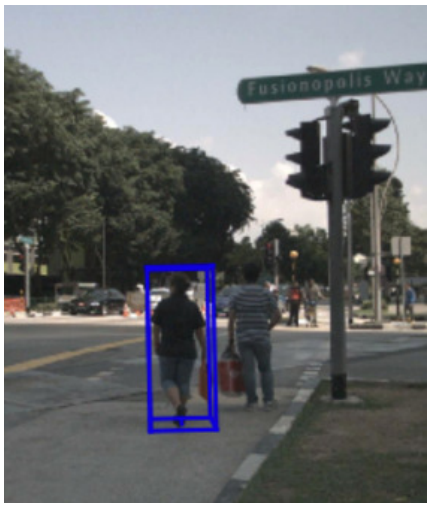

Fig. 8. The "true" annotation of a pedestrian shown in Table II. It is scrambled as a traffic cone, creating an error.

This explanation can be compared with the description from the data set; where parked truck, construction and intersection are part of the scene description (amongst other concepts).

VII. DISCUSSION

We evaluated ADE to detect failures by simulating the Uber Carla scenario and explaining it. We discussed preliminary results for a statistical test that simulates errors in Section VI-B. These are two of the three main ways to evaluate the ADE architecture:

1) Detection: Generate logs from scenarios to detect failures.
2) Invoke errors: Scrambling multiple labels on existing data sets.
3) Real errors: Examining errors on the validation data set of NuScenes leaderboard.

Regular Output	Scrambled Output
This sample's 69 labeled objects are reasonable. Sample explanations: The human.pedestrian.adult located at [373.256, 1130.419, 0.8] is approximately the right size. It is somewhat close, located to the right.	At least one of this sample's 69 labeled objects are unreasonable. Unreasonable explanations: The movable_object.trafficcone located at [373.256, 1130.419, 0.8] is not a reasonable size: it is too tall. There is no commonsense supporting this judgement. Ignoring objects detected to the right.
The vehicle.car located at is [353.794, 1132.355, 0.602] is approximately the right size. It is somewhat close, located to the right, past the movable_object.	The human.pedestrian.adult located at is [353.794, 1132.355, 0.602] is not a reasonable size: it is too wide and it is too deep. It's location is too far away to be close to a street. Ignoring objects detected to the right.

TABLE II
A COMPARISON OF THE ADE SYSTEM ON REGULAR AND SCRAMBLED OUTPUTS. THE REGULAR OUTPUT IS A *summary* SUPPORTING THAT THE JUDGEMENT OR ACTION IS REASONABLE. THE SCRAMBLED OUTPUT (WHERE A DIFFERENT LABEL THAN THE TRUE LABEL IS USED) SHOWS THAT THE ARCHITECTURE CAN CORRECTLY IDENTIFY JUDGEMENTS OR ACTIONS THAT ARE INCORRECT.

Another way to evaluate ADE is to examine the outputs of existing algorithms that are trained on some of these self-driving data sets. In particular, we wanted to investigate where existing algorithms failed on autonomous driving challenges, in hopes of finding common failure scenarios. NuScenes [7] has several different online challenges including detection, tracking, and prediction. Since the testing sets of these challenges are private, we started to examine if we could examine the validation errors of top-performing algorithms. First, we had to find if these algorithms were available. Unfortunately, most of these algorithms are not open source. Secondly, even if the source code was available, the code will require significant tweaking for different data sets.

NuScenes did provide the train, test, and validation sets for three of the top performing models. However, we ran into two issues with these models. The first was that the models used either LiDAR or camera and not both, which means that it would have been difficult for us to test the synthesis portion of the framework as there would only be one subsystem involved. The other issue is that the provided sets were not complete scenes. After we wrote a script to compile the image frames into a set of scenes, we found that there were some missing frames in the provided sets, so it would not have been possible to test our architecture at a higher-level across multiple frames or scenes.

A. Other Potential Evaluations

Other evaluations could be performed in simulation, similar the Uber scenario. There are a number of challenge scenarios proposed[4] that have yet to be simulated and released for research use. They are motivations for other problems.

[4]Carla challenges: https://carlachallenge.org/challenge/nhtsa/

We are currently working on a new set of challenge problems for anticipatory thinking in autonomous driving. In particular how does choosing the right representation, especially for self-driving, address the issues in "the long tail of rules," which we infer to mean, how does it scale. If you consider the number of rules required for safe driving, there are many. This becomes even larger if we consider corner cases: what if the pedestrian crossing the road is really a police-officer waving you through? We are interested in using these types of challenge problems as a new evaluation platform for explanatory error detection. Since the explanations are for a machine (in addition to an end-user or other human examiner) we are interested in creating tasks that show how this feedback mechanism can be used for more robust decision-making.

B. Limitations

There may be more than one plausible, reasonable behavior with a supporting, convincing explanation. It could be difficult to decide which behavior to accept. One way to tackle this situation is to simulate alternative futures, where the simulation is able to model the behavioral and physical consequences of acting on any set of premises that may be chosen by committee arbitration. This is particularly important in the case where it will have accepted premises that in fact represent the wrong situation. We have simulated scenarios to evaluate ADE in Carla [28], an autonomous driving simulation platform.

Our evaluation results rely on simulation. We were not able to obtain vehicle logs from accidents or failures. Since simulations are oversimplified, some scenarios and test cases may be missing key realistic aspects. In future work, we will work with real data.

A final limitation is the amount of hand-curated information. As stated in Section V-A, the ADE system needs a priority hierarchy to judge which subsystems to trust or discount in the case of inconsistent information. For self-driving, this was fairly straight-forward, as the priority is *safety* of the passenger inside the vehicle. But, in cases where there is variety of external distractions and near-perfect information, it may be important to consider which outside factors to prioritize; which reduces to the trolley problem [26]. One interesting area of future work would be to investigate whether these priorities can be learned from experience. This requires a *human-in-the-loop* to provide proper feedback. In the future, we will explore the right ways to incorporate human preferences into our architecture. Both for better system debugging, but also to learn preferences (and eventually, individualized priorities) for each passenger. There may be other meaningful priorities besides passenger safety (i.e. passenger comfort, or limited interruptions, or a smooth route). We will explore whether these can be learned through user feedback in future work.

VIII. CONCLUSION AND FUTURE WORK

The key idea of this paper is to utilize *dynamic* explanations along with a logic system to reason *between* subsystems of a complex machine. This will allow for better system-wide communication, even with limited information, uninter-

pretable information, or receiving conflicting information from the underlying subsystems. We motivated this approach for safety-critical autonomous decision making, especially self-driving. But this is also important for enabling communication, specifically, articulate communication, between machines and amongst machines with people.

Explanations can facilitate a common language for machines to communicate amongst each other. While we motivated this communication internally amongst the parts of a complex machine, this can also scale between machines that excel at different tasks. This is key for redundancy. We have systems that can solve problems in multiple ways: like a machine learning system versus a logic-based system. They have different strengths and weakness, but perhaps, using explanations, they can learn from each other for a truly hybrid process. But we may also wish for the same process between a human and a machine.

One of our future directions is exploring the role of *learning*. Many of the rules and preferences are manually generated. We are now investigating how to make this process more dynamic: using explanations for feedback. But the foundation for this - a common explanation language - already exists.

As machines and humans share control of tasks, communication ensures robustness and reliability. This is important for debugging, so that humans can improve complex systems, but also for education, where complex could "improve" or teach humans. Imagine a system where human interventions are well-communicated, and those explanations supporting the interventions are used as feedback to improve the system moving forward.

This paper motivates a new view of anomaly detection. A view in which failure and errors are not necessarily outliers, but inexplicable instances. This is grounded in the argument that a failure or an anomaly occurs when an explanation is inadequate or inappropriate; thus, indicating that the underlying subsystem or process should be corrected or disabled. This is supported by the argument that explanations are a *debugging language*. We have shown the applicability of this in critical scenarios like self-driving vehicle accidents, but we intend to extend ADE to other applications, and between other systems; to facilitate communication between machines and to work better with and among human counterparts.

REFERENCES

[1] "Uber's Self-Driving Car Saw the Woman It Killed, Report Says," https://www.wired.com/story/uber-self-driving-crash-arizona-ntsb-report/.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[3] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[4] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," 2018.

[5] B. Wilson, J. Hoffman, and J. Morgenstern, "Predictive inequity in object detection," *arXiv preprint arXiv:1902.11097*, 2019.

[6] J. Claybrook and S. Kildare, "Autonomous vehicles: No driver... no regulation?" *Science*, vol. 361, no. 6397, pp. 36–37, 2018.

[7] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[8] D. J. Fremont, E. Kim, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: A language for scenario specification and data generation," *arXiv preprint arXiv:2010.06580*, 2020.

[9] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[10] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark," in *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*. IEEE, 2015, pp. 38–44.

[11] M. Grill, T. Pevnỳ, and M. Rehak, "Reducing false positives of network anomaly detection by local adaptive multivariate smoothing," *Journal of Computer and System Sciences*, vol. 83, no. 1, pp. 43–57, 2017.

[12] O. Vallis, J. Hochenbaum, and A. Kejariwal, "A novel technique for long-term anomaly detection in the cloud." in *HotCloud*, 2014.

[13] E. Lampiri, "Sensor anomaly detection and recovery in a nonlinear autonomous ground vehicle model," in *2017 11th Asian Control Conference (ASCC)*, 2017, pp. 430–435.

[14] F. Wyk, Y. Wang, A. Khojandi, and N. Masoud, "Real-time sensor anomaly detection and identification in automated vehicles," 08 2018.

[15] J. De Kleer and B. C. Williams, "Diagnosing multiple faults," *Artificial intelligence*, vol. 32, no. 1, pp. 97–130, 1987.

[16] J. De Kleer, A. K. Mackworth, and R. Reiter, "Characterizing diagnoses and systems," *Artificial intelligence*, vol. 56, no. 2-3, pp. 197–222, 1992.

[17] T. Mota, M. Sridharan, and A. Leonardis, "Integrated commonsense reasoning and deep learning for transparent decision making in robotics," *SN Computer Science*, vol. 2, no. 4, pp. 1–18, 2021.

[18] M. Albanese, S. Jajodia, A. Pugliese, and V. S. Subrahmanian, *Scalable Detection of Cyber Attacks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 9–18. [Online]. Available: https://doi.org/10.1007/978-3-642-27245-5_4

[19] L. Negash, S.-H. Kim, and H.-L. Choi, "Distributed unknown-input-observers for cyber attack detection and isolation in formation flying uavs."

[20] Q. Chen, Q. Wu, M. Bishop, R. Linderman, and Q. Qiu, "Self-structured confabulation network for fast anomaly detection and reasoning," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–8.

[21] P. Hammer, T. Lofthouse, E. Fenoglio, and H. Latapie, "A reasoning based model for anomaly detection in the smart city domain," 2019.

[22] P. Langley, B. Meadows, M. Sridharan, and D. Choi, "Explainable agency for intelligent autonomous systems." in *AAAI*, 2017, pp. 4762–4764.

[23] L. H. Gilpin and B. Z. Yuan, "Getting up to speed on vehicle intelligence," *2017 AAAI Spring Symposium Series*.

[24] L. Gilpin, "Reasonableness monitors," in *The Twenty-Third AAAI/SIGAI Doctoral Consortium at AAAI-18*. New Orleans, LA: AAAI Press, 2018.

[25] R. Speer and C. Havasi, "ConceptNet 5: A large semantic network for relational knowledge," in *The People's Web Meets NLP*. New York: Springer, 2013, pp. 161–176.

[26] E. Awad, S. Dsouza, R. Kim, J. Schulz, J. Henrich, A. Shariff, J.-F. Bonnefon, and I. Rahwan, "The moral machine experiment," *Nature*, vol. 563, no. 7729, pp. 59–64, 2018.

[27] "Preliminary Report Highway HWY18MH010," https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf.

[28] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[29] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet, "Lyft level 5 av dataset 2019," urlhttps://level5.lyft.com/dataset/, 2019.

[30] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[31] P. H. Winston, "Learning structural descriptions from examples," Ph.D. dissertation, Massachusetts Institute of Technology, 1970.