Hazard Contribution Modes of Machine Learning Components

Colin Smith
Oak Ridge National Laboratory
Oak Ridge, TN
smithca@ornl.gov

Ewen Denney and Ganesh Pai
KBR / NASA Ames Research Center
Moffett Field, CA
{ewen.denney, ganesh.pai}@nasa.gov

Abstract

Amongst the essential steps to be taken towards developing and deploying safe systems with embedded learning-enabled components (LECs)—i.e., software components that use machine learning (ML)—are to analyze and understand the contribution of the constituent LECs to safety, and to assure that those contributions have been appropriately managed. This paper addresses both steps by, first, introducing the notion of *hazard contribution modes* (HCMs)—a categorization of the ways in which the ML elements of LECs can contribute to hazardous system states; and, second, describing how argumentation patterns can capture the reasoning that can be used to assure HCM mitigation. Our framework is generic in the sense that the categories of HCMs developed *i*) can admit different learning schemes, i.e., supervised, unsupervised, and reinforcement learning, and *ii*) are not dependent on the type of system in which the LECs are embedded, i.e., both cyber and cyber-physical systems. One of the goals of this work is to serve a starting point for systematizing LEC safety analysis towards eventually automating it in a tool.

1 Introduction

Learning-enabled components (LECs) are software that leverage knowledge acquisition and machine learning (ML) processes to implement a function or service. The dramatic increase in the capabilities of ML algorithms over the past decade has motivated the inclusion of LECs into larger systems, to accomplish tasks traditionally performed by human operators (in the case of physical systems), engineered heuristics (in the case of software and mechanical systems), or to perform functions that might otherwise have been infeasible. When integrated into a safety-critical context, e.g., transportation systems, or surgical robots, LECs can directly cause or contribute to harm (National Transportation Safety Board 2017).

Safety assurance arguments are structured reasoning that relate safety claims to auditable and verifiable items of evidence, and are often a core element of modern safety assurance cases—a comprehensive, defensible, and valid justification of the safety of a system for a given application in a defined operating environment (Denney and Pai 2018). Safety cases have been extensively used as a means of safety assurance in various safety-critical domains, and recent standardization efforts[1] feature safety argumentation as a core element of autonomy safety assurance. Safety arguments can be captured in a number of forms, one of which uses graphical notations such as the Goal Structuring Notation (GSN) (The Assurance Case Working Group (ACWG) 2018). Towards enabling reuse and ease of comprehension, GSN-based argument structures can be abstracted and represented in the form of *argumentation patterns*.

The key starting point for safety analysis and assurance is a systematic hazard analysis. Although one would usually conduct this analysis at the higher-levels of the system hierarchy, a variety of lower-level, *bottom-up* analyses applied at a component-level stand in support. A *failure modes and effects analysis* (FMEA) is a typical example of such lower-level component-focused analysis.

(Salay and Czarnecki 2018) suggest that the difficulty of assuring LEC safety is due to two distinct differences between an ML algorithm and a traditional software component. First, the problems to which ML algorithms are applied are often difficult if not impossible to specify. This condition is often by design—if it were possible to write a specification for a problem, an ML algorithm might not be necessary or useful in the first place. Second, ML algorithms and their internal logic are often uninterpretable to humans, complicating their oversight. This difficulty of assuring LEC safety presents a gap in the safety assurance practice that we seek to address in the work presented in this paper. The goals are to analyze and understand the contribution of the ML elements of LECs[2] to safety, when embedded into a wider system, and to assure that those contributions have been appropriately managed.

This paper addresses both objectives by, first, introducing the notion of *hazard contribution modes* (HCMs)—a categorization of the ways in which LEC outputs can lead to hazardous system states; and, second, describing how argu-

[1] For example, the forthcoming *Standard for Safety for the Evaluation of Autonomous Products*, UL 4600, from Underwriters Laboratories, Inc.

[2] In the rest of this paper, we will not distinguish between LECs and their ML elements.

mentation patterns can capture the reasoning that provides assurance of HCM mitigation. These two aspects of our framework are generic in the sense that the categories of HCMs developed *i*) can admit different learning schemes, i.e., supervised, unsupervised, and reinforcement learning, and *ii*) are not dependent on the type of system in which the LECs are embedded, i.e., both cyber and cyber-physical systems. Additionally, this work represents the first steps towards a coherent framework to integrate applicable (and possibly existing) analysis and verification techniques, along with mitigation mechanisms, towards the ultimate goal of developing assurance cases for systems embedding LECs. Another goal is to serve a starting point for systematizing LEC safety analysis towards eventually automating it in our assurance case tool, AdvoCATE (Denney and Pai 2018).

2 Hazard Contribution Modes

An LEC embedded and integrated into a wider system can migrate it to undesired states that pose the potential for harm—hazards—through its outputs and interactions with other system components, and the role each plays in the overall system state. This *contribution* of an LEC to a hazard may assume different forms, which we term as (LEC) *hazard contribution modes* (HCMs). An HCM thus characterizes the safety impact of an LEC, in terms of the relationship between (possibly) observable LEC output and the observable output (or behavior) of the wider system in which it is embedded.

We identify two types of LEC output, namely *expected performance* and *unexpected performance* (Figure 1). The characterization of LEC output for both output types is the same, though only shown fully for expected performance. We refine each output type into modes concerning *accuracy* and *error* (see Section 2.1 for more clarification).

As we will see later, the two hazardous modes shown under the "with error" category in Figure 1 can be subsumed under one generalized mode[3]. Thus, there are broadly four categories of HCMs: *Expected and Accurate* (EA), *Expected with Error* (EE), *Unexpected and Accurate* (UA), and *Unexpected with Error* (UE).

We can apply this categorization of HCMs independently of the type of system in which LECs are embedded, be it a physical system such as a road vehicle, or an intangible cyber system such as enterprise software. Recognizing that LEC output is produced from ML processes—effectively the application of optimization algorithms with respect to some objective functions—we hypothesize that the proposed HCMs are agnostic to the learning scheme used, i.e., supervised, unsupervised, and reinforcement learning (RL). We give examples to bolster this hypothesis in the discussion that follows later in this section.

Applying the HCM categories can be viewed as a *bottom-up* analysis similar to an FMEA. However, we distinguish HCMs from failure modes since not all failure modes need contribute to a hazard, whereas by definition all HCMs do,

[3]Henceforth, we will simply use *mode* when we mean hazard contribution mode (HCM), qualifying its usage when unclear from context.

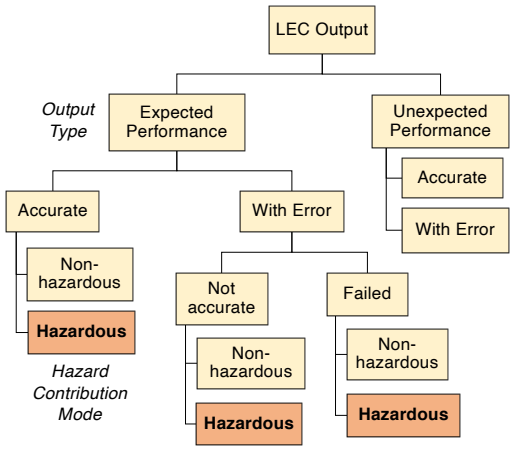

Figure 1: Characterization of LEC output, showing hazard contribution modes in **boldface** in the highlighted boxes.

although some HCMs may indeed be failure modes (see Figure 2). HCMs are more abstract than failure modes and, from a safety standpoint, they are broader in the sense that they include non-failing outputs that can be hazardous. Methodologically, we envision that HCMs would be used alongside, rather than as a replacement for FMEA.

Before elaborating each HCM in detail, first we discuss relevant terminology and concepts.

2.1 Preliminaries

LEC accuracy represents the property that its outputs are globally optimal, be it in the form of correct classification, near-exact regression, or a selection of the reward maximizing actions in RL. Likewise, by *LEC error*, we mean the divergence between LEC outputs and the reference values, i.e., either *ground truth* (if available, e.g., as in supervised learning), or the optimal values produced according to the loss, cost, or objective functions of the optimization algorithm (e.g., as in RL using reward functions). LEC accuracy and error are distinct concepts but intuitively related: the greater the error, the lower the accuracy. However, determining that which constitutes *accurate* output requires a reference for comparison (which we elaborate on in further detail later in this section).

It is worth contrasting LEC error, with the concept of (software) error from dependability terminology (Avižienis et al. 2004): although both capture a notion of divergence (or deviation), the reference for comparison used is different. Specifically, for the latter deviation is determined from a *required external system state*, while for the former the reference need not be a requirements specification.

Additionally, we differentiate LEC accuracy from *classification accuracy*. For classification tasks, LEC accuracy refers to the classification performance on a particular sample input and is deterministic—i.e., either the classification is accurate, or it is not. In contrast, classification accuracy gives a frequentist probability of correct classification, and is a metric that characterizes the (expected) classification performance based on collection of sample inputs.

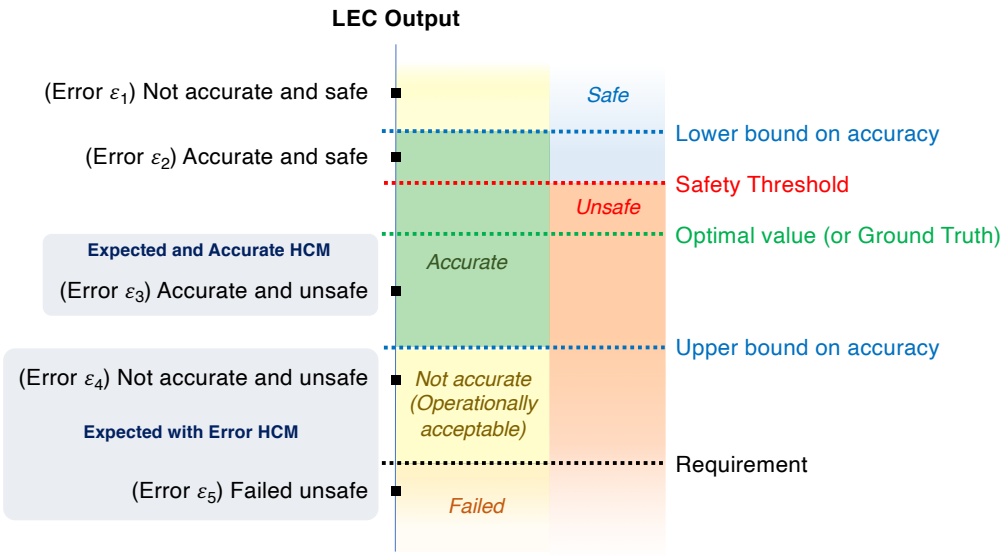

Figure 2: Simplified (notional) concept for LEC output, showing different LEC error magnitudes. In reference to the bounds on accuracy, the output is either accurate or not accurate. In reference to a requirement, an output is failed, while safety is determined in reference to a safety threshold. Together, the *Expected and Accurate* (EA), and the *Expected with Error* (EE) hazard contribution modes can be distinguished. The safety threshold shown is derived and applicable to the LEC.

Depending on the learning scheme, LEC error can assume different forms. For instance, error in supervised learning emerges from a combination of model assumptions (*bias*), the type and quantity of data used (*variance*), and *noise*. In unsupervised learning, error may emerge from an inadequately specified optimization objective (*approximation error*), model parameter choices that complicate differentiation between distinct inputs (*identifiability error*), inadequate data (*estimation error*), and algorithmic insufficiencies (*optimization error*) (Liang and Klein 2008).

Practically, various error bounds can be specified. The bounds on accuracy are such that LEC outputs that fall within those bounds are considered to be accurate. In other words, when the magnitude of LEC error does not exceed the bounds on accuracy, LEC output is considered accurate. When LEC error magnitude exceeds the bounds on accuracy, the output may nevertheless meet the applicable requirements. Such LEC output is *not accurate* but *operationally acceptable*, e.g., when the output is imprecise.[4] When the LEC error magnitude exceeds that which is acceptable according to the requirements, it can be considered to have *failed*.[5]

Figure 2 shows a notional concept that illustrates this characterization of LEC output. As shown in the figure, the bounds on accuracy and the requirement, together with the magnitude of LEC error (labeled from ε_1 to ε_5) determine when LEC output can be considered to be accurate, not accurate, and failed. For example, the outputs with LEC errors ε_1 and ε_4 are not accurate, those with LEC errors ε_2 and ε_3 are accurate, while the output with LEC error ε_5 is failed. For this paper, we will consider those outputs that are not accurate or failed to be of the type "with error" (see Figure 1).

We assume that safety thresholds can be defined on the wider system state, e.g., using a *safety envelope* approach (Tiwari et al. 2014). As shown in Figure 2, the safety threshold determines when LEC output is *unsafe*. For example, the LEC outputs with LEC errors ε_3, ε_4, and ε_5 are all unsafe. The safety threshold shown has been arbitrarily chosen. Practically, it is to be derived from the system-level safety threshold and apportioned appropriately to the LEC; although, for illustrative purposes, here we consider them to be equivalent. Violating the safety thresholds can result in a transition of the system from a safe state to a hazardous state (represented in Figure 2 by the region labeled *unsafe*).

Note that Figure 2 is intentionally simplified to illustrate the HCM concept. Indeed, we have assumed here that a range of acceptable error suffices to express the requirement and safety threshold; instead, bounds, regions, or more complicated manifolds—e.g., in the case of classification problems—may be more appropriate. Suitable mechanisms to determine and express those, are not in scope for this paper, though the framework we give here provides a principled way to integrate them.

The combination of whether LEC output is safe, and its output type is what we use to distinguish HCMs. Next, we describe the main LEC output types and the associated HCMs, giving representative examples towards typifying what the mode is, and identifying plausible causes. Subsequently, we will discuss how we envision providing assurance that the HCMs can be acceptably managed towards the broader goal of system safety.

[4]Note that accuracy refers to the closeness of the output to the reference, whereas precision refers to the variation in the output.

[5]In this case, error from dependability terminology and LEC error are equivalent.

2.2 Expected Performance

Expected performance is simply the required runtime LEC output or behavior. It is *expected* in the sense that, for inputs as captured within the system requirements, one anticipates observing LEC outputs[6] corroborating that it is at least as performant in operation, as it was during model validation and calibration. This is reflective of the input-output relationships learnt during the ML model (re)training, validation, and calibration efforts that are iteratively performed as part of the lifecycle of LEC development, prior to deployment. As such, expected performance of an LEC is either accurate, or with error.

Expected and Accurate Hazard Contribution Mode The *Expected and Accurate* (EA) mode of an LEC is one where its outputs are accurate and can induce hazardous system states. Conceptually, this can be viewed as the condition where the safe operational state space of the system has been over-approximated resulting in an overlap with the unsafe state space. In Figure 2, the LEC output with error ϵ_3 is representative of the EA mode.

Consider, for example, a lane keeping function in automated driving, whose implementation uses a perception LEC to classify and localize lanes whilst providing distance estimates to lane boundaries. In this case, not considering temporary or removed lane markings[7] in the LEC training data, may result in *correct* localization and *accurate* distance estimation from the *wrong* (i.e., removed) lane markings during operation. This is an instance of the EA mode that, when unmitigated, in turn poses the potential for unintended lane departure and intrusion into neighboring lanes.

The EA mode can manifest when known precursor or contributory conditions to hazards identified during system safety analysis—e.g., sensor malfunctions, off-nominal environmental conditions, etc.—have not been explicitly accounted for during LEC development, i.e., when they have not been *i*) included, or insufficiently represented, in the data used for training and testing the ML model underpinning an LEC, or *ii*) reflected in the optimization algorithm used.

Expected with Error Hazard Contribution Mode The *Expected with Error* (EE) mode of an LEC is one where the magnitude of LEC error is such that one or more safety thresholds are violated, due to which potentially hazardous system states are inducible. For the EE mode, the main issue is whether the amount of error produced beyond the bounds on accuracy is such that the output induces a hazardous system state. As shown in Figure 2, when an LEC is in the EE mode it produces outputs with error magnitudes (labeled ϵ_4 and ϵ_5) that are, respectively, *i*) not accurate (but operationally acceptable) and unsafe; and, *ii*) failed unsafe.

The EE mode can emerge in a number of scenarios, with diverse local and global effects. Consider, again, our earlier example of a lane keeping function using a perception LEC, although under nominal road conditions, i.e., without temporary or removed lane markings as before. If the errors in physical sensor values, as well as other errors introduced due to sensor placement, scene preprocessing, sensor fusion, etc., and their impact on violating the system-level safety threshold are not accounted for during LEC development, then the output error may be such that lanes are estimated to be farther away than they actually are. As a result, a subsequent control action intended to maintain the current lane may inadvertently result in a lane departure. Thus, errors in the input (including adversarial perturbations) can cause LEC output error magnitude to be modified such that an otherwise safe output becomes hazardous.

LECs can be susceptible to the EE mode owing to the numerous error generating processes involved in LEC development, and how they account for the applicable safety thresholds. For example, due to

a) inadequate sampling practices in preparing the training data. One consequence of this can be unbalanced data that insufficiently represents the relationship between the inputs, outputs, and the safety thresholds;
b) assumptions made in building the ML model. This may be reflected in an optimization objective function that insufficiently accounts for the relevance and impact of the applicable safety thresholds;
c) inadequate representation or coverage—of the conditions of the operating environment that impact, or are impacted by the safety thresholds—during model validation.

2.3 Unexpected Performance

Unexpected performance represents emergent runtime LEC output, behavior, or effects at the system level, e.g., through unanticipated feature interaction, that is not otherwise anticipated to be seen in operation, and that was also not previously observed during model validation. Like expected performance, unexpected LEC output can be considered as accurate, or with error. Also, as mentioned earlier (also see Figure 1), the HCMs associated with unexpected performance mirror those identified for expected performance, i.e., they concern hazardous LEC outputs that are both accurate—i.e., the Unexpected and Accurate (UA) mode—and have error—i.e., the Unexpected with Error (UE) mode.

Intrinsically, as well as from the standpoint of the impact in inducing hazardous system states, there is conceptually little difference between the UA and UE modes, and those that we identified earlier. Consequently we expect that the processes and causes from which the UA and UE modes stem, will overlap to some degree with many of those responsible for generating the EA and EE modes.

However, unexpected LEC performance, in general, is an operational manifestation of misconceptions about *i*) the nature of the operating environment and *ii*) the system in which the LEC is deployed, reflected in the ML algorithms used, and the data used to train them. This, in conjunction with potential inadequacies in the steps of LEC development result in a divergence between what is required (i.e., expected performance), and what is experienced at runtime (unexpected performance). From this, we hypothesize that two particular

[6] In general, it may not be possible to observe LEC performance directly; rather it would be inferred from the system output or behavior, to which LEC output would propagate.

[7] Possibly introduced due to roadwork, this condition of the road environment can confuse even experienced human drivers.

reasons for the UA and UE modes include dataset shift, and validation insufficiency.

Additionally, due to the inherently uncertain nature of unexpected (and possibly hazardous) LEC outputs, the approaches used for mitigation and assurance are where the UA and UE modes are likely to differ from the EA and EE modes, respectively. In particular, although the latter two modes may be manageable through a largely preventative approach, mitigation of the former two modes is likely to be more reliant on runtime detection, recovery, and architecture-level safety mechanisms.

Unexpected and Accurate Hazard Contribution Mode The *Unexpected and Accurate* (UA) mode characterizes unanticipated LEC output that are nevertheless optimal or accurate, and also hazardous.

For example, consider a collision avoidance function in an unmanned aircraft system implemented using an LEC trained using reinforcement learning (RL). In this case, a legitimate reward maximizing action such as *banking to the left* to avoid a collision can be learnt since this is an admissible action for some states and situations. However, in some collision scenarios, and in particular, under conditions where another aircraft rapidly emerges on a head-on course within a short range (for instance, in certain types of uncontrolled airspace where human-piloted general aviation also co-operate), the established rules of air require both aircraft to bank to the right. In this case, an otherwise optimal control action is unexpected for the operating scenario, and can be potentially catastrophic, i.e., if the aircraft controlled by the LEC banks to the left while the other aircraft banks to the right, as required.

The UA mode can emerge in operating contexts where there is no notion of ground truth, and potentially also from the conditions and causes that precipitate the EA mode. That is, when conditions and events known to contribute to identified hazards are not suitably accounted for, during LEC development, either in the training and validation data, or in the optimization objective function. Other differentiating causes include, as mentioned earlier, dataset shift, and insufficient model validation before deployment.

In our aircraft system example above, dataset shift may present as unbalanced data, for example, itself potentially due to inappropriate sampling, or due to assumptions made whilst modeling the environment. Likewise, incomplete coverage of various collision geometries, with variable aircraft types and speeds, is reflective of inadequate model validation.

Unexpected with Error Hazard Contribution Mode The *Unexpected with Error* (UE) mode categorizes unanticipated LEC output that has a magnitude of error such that it contributes to hazardous system states.

Consider, again, a perception LEC deployed in an automated driving scenario used for detecting and tracking external entities, in support of collision avoidance. Although in training and validation, the average detection performance to identify and localize pedestrians—as measured by sensitivity and specificity metrics—may indicate that the LEC will perform as expected in operation, false positives/negatives in specific context-dependent environmental situations reflect unexpected outputs with an error magnitude that could be potentially hazardous, e.g., a false negative classification of a pedestrian with dark clothing who may be camouflaged by an equally dark background, in a limited visibility environment, due to which vehicle brakes are not engaged, potentially resulting in a collision.

In this case, the false negative is the consequence of an *edge case* (Koopman 2019), a rare situation representing a special form of *covariate shift* that may elude non-comprehensive model validation. Covariate shift is one specific type of the more general dataset shift condition, where the distribution of the inputs to the LEC in operation is different from that which was used for its training. Other forms of dataset shift include *prior shift*, where the output distributions differ between training and operation, and *concept shift*, where the joint distribution of inputs and outputs changes in operation from that which was represented by the training data.

More generally, and as mentioned earlier in this section, the UE mode can occur due to various error generating processes in LEC development steps, including inadequate training and validation data, and assumptions encoded in the optimization loss, error, or objective functions, in addition to the two particular causes of unexpected performance: dataset shift, and inadequate validation.

3 Assuring Acceptable Safety Contribution

Recall that, first, the EA and EE modes are similar to the UA and UE modes, respectively, from the standpoints of: *i*) their impact on system safety, and *ii*) the specific aspects of the LEC output, i.e., accuracy and error magnitude. Also recall that there is a degree of overlap between the causes of the modes, due to which there is a consequent overlap in the way mitigation is addressed. Thirdly, we hypothesized two particular reasons for unexpected hazard contribution.

We now describe some mechanisms towards acceptably managing the safety contribution of the identified hazard contribution modes, which can also be considered as generic and high-level evidence requirements.

3.1 Managing Hazard Contribution Modes

Accurate and Hazardous Output Assurance of acceptably mitigating the EA and UA modes can be provided, in part, through evidence of a combination of the following:

- sufficiently reflecting in the training and validation data, all known and identified conditions that are both known hazard precursors and that can be associated to functions allocated to LECs;
- penalizing the relevant hazard precursors in the loss, cost, or objective functions of the optimization algorithms used, e.g., as parameterized variables of a regularization term added to a loss function. Contextually relevant robustness testing of the LEC can then provide additional evidence assuring that the revised objective functions are suitable to reduce the occurrence of the EA mode;
- comprehensive model testing and verification-based coverage of the identified precursor conditions of the hazards induced by the EA mode.

Hazardous Output with Error To provide assurance that the EE and UE modes have been acceptably managed, requiring evidence of at least a combination of the following can be useful:

– valid safety thresholds (or bounds) associated with the LEC outputs, derived from system-level safety bounds (or a specification of the safety envelope and associated conditions);
– adequately reflecting the LEC specific safety thresholds in the training and validation data;
– appropriately reflecting the LEC specific safety thresholds in the loss, cost, or objective functions of the optimization algorithms used. Reflecting appropriateness of the ML algorithms, in turn, relates to demonstrating that they are performant and robust (Ashmore, Calinescu, and Paterson 2019), with the proviso that these assurance properties account for the applicable safety thresholds;
– comprehensive model testing and verification that considers error performance in relation to the relevant safety thresholds.

Prevention and Recovery The mitigations proposed above are preventative, broadly focusing on reducing the opportunities for the EA and EE modes to occur in operation. In so doing, the idea is also that they may prevent the runtime occurrence of the UA and UE modes, since some of the same processes are responsible. To additionally assure that the particular causes of the UA and UE modes have been managed, assurance mechanisms can additionally focus on gathering evidence of a combination of the following:

– comprehensive definition of the operating environment, and the system context, e.g., by leveraging a rich specification of the operating constraints for the system (Koopman and Fratrik 2019);
– sufficiency of the training and validation data, that is assurable through evidence that the data used are relevant, balanced, complete, and accurate (Ashmore, Calinescu, and Paterson 2019).
– model validation techniques that are comprehensive and contextually relevant (Ashmore, Calinescu, and Paterson 2019).

Of the above, the first two items seek to provide assurance of managing dataset shift, while the third attempts to provide evidence that model validation approaches used are adequate.

Layered Mitigation For wider safety assurance, it is not sufficient only to show that LEC HCMs have been managed; rather a layered approach is prudent, including architectural mechanisms to reduce or mask the impact of the HCM. As such, from the standpoint of recovery from the HCMs once they have occurred, and specifically to manage the UA and UE modes, evidence of mitigation mechanisms deployed at an architecture level can provide additional assurance. For instance,

– runtime monitoring for detection of LEC outputs that violate the applicable safety bounds,
– fail-safe/failure tolerance mechanisms that disengage hazardous LEC outputs, or the entire LEC as appropriate, by leveraging assurance measures (Asaadi, Denney, and Pai 2019) that provide a quantified notion of confidence in specific LEC assurance properties;
– using redundant and sufficiently diverse implementations of the functions that LECs implement, along with runtime monitors for detecting function disagreement.

Collectively, we believe that satisfying the generic evidence requirements described above, can form part of the wider basis for assurance that the system-level effects of the LEC output are acceptable and do not contribute to hazardous system states.

The caveat in the preceding presentation of some general mitigations for HCMs is that some of those mitigations have their own challenges (that we do not address in this paper). For example, diverse data is generally useful but its inclusion requires care, without which the optimization algorithms used for ML may not converge on a solution. Likewise, the application of formal analysis techniques for robustness assurance may have limited utility if the risks being examined have small associated probabilities. A more complete examination of HCM mitigations is outside the scope of this paper. We note that a wide variety of such techniques have been previously surveyed, e.g., by (Ashmore, Calinescu, and Paterson 2019). Our emphasis here is, rather, the framework to marshaled those mitigations in a structured way, towards LEC assurance.

3.2 Safety Assurance Argument Patterns

Argument Pattern Notation We give a brief overview of the syntactic elements of the Goal Structuring Notation (GSN) that we have used here for specifying argument patterns. For comprehensive details on argument patterns and their use, we refer to our prior work (Denney and Pai 2013).

Safety claims are represented using rectangular elements termed as *goal* nodes (or, simply, goals). References to contextual information are given in rounded rectangles termed as *context* nodes. Parallelograms are *strategy* nodes, specifying how higher-level claims are developed into lower-level claims, e.g., using inference rules. Assumptions are recorded within ellipses annotated with the character 'A'. The triangular and/or diamond decoration on nodes indicate that those nodes can be *instantiated*, i.e., by replacing the abstract parameters specified within braces '{ }' in the node descriptions, with concrete data.

Nodes are generally connected by two types of links. Those with solid arrowheads represent an inferential relation interpreted as '*is supported by*', while those with hollow arrowheads denote a contextual relation, interpreted as '*in context of*'. Links can be annotated with *multiplicity*, filled circles with labels that specify how many times that link and connected target node are instantiated. Additionally, there is a special type of link for capturing a *choice*, which is shown using a filled diamond symbol on a link between a source and multiple target nodes. Such choice links can also be annotated with labels indicating how many of the target nodes are to be instantiated.

Example Argument Pattern Now, we discuss how the reasoning associated with the assurance mechanisms discussed in the preceding section may be presented in the form

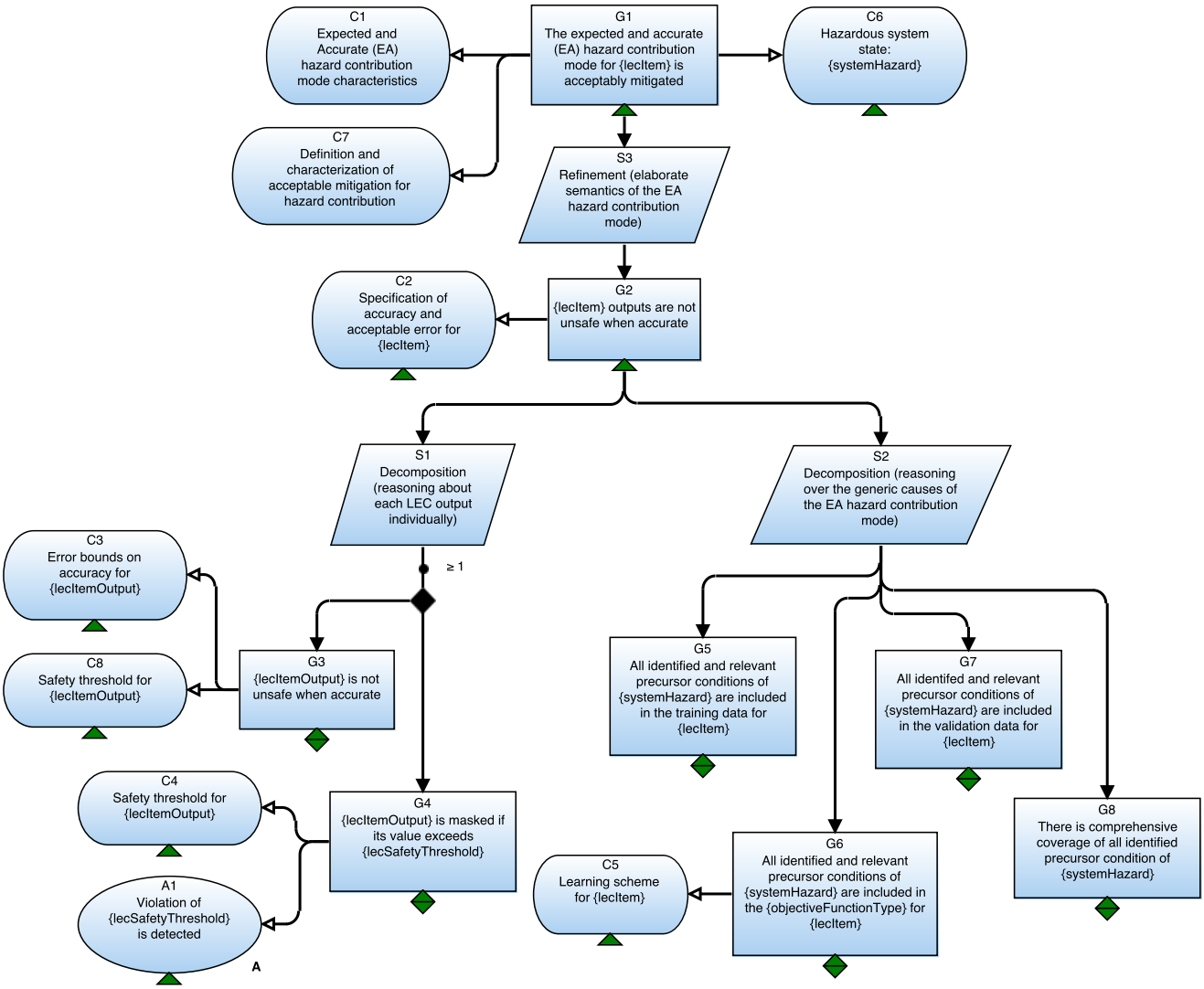

Figure 3: Argument pattern for assurance that the EA mode for a specific LEC is acceptably managed.

of patterns of assurance argumentation. Figure 3 shows the assurance argument pattern capturing the reasoning why it is the case that the EA mode is acceptably mitigated for a specific LEC (shown as the root goal node G1, referencing the parameter "`lecItem`"). This claim references the hazardous system state to which it contributes, as a contextual element (node C6). The pattern then essentially provides two strands of assurance reasoning to support the root claim.

The first leg (strategy node S1 and downwards) reasons individually about the specific LEC outputs, requiring evidence either that *i*) either each output is not hazardous when it is accurate (goal node G3), or that *ii*) if each output is, in fact, hazardous, its value is masked upon exceeding the associated safety threshold (goal node G4), assuming that this violation can be detected (assumption node A1). The choice link between the strategy node S1 and the goal nodes G1 and G2 captures these alternatives. The second branch (strategy node S2 and downwards) reasons about mitigation of each of the generic identified causes of the HCM (goal nodes G5–G8), reflecting the generic mitigations identified in the previous section. Since there can be multiple system hazards to which an LEC could potentially contribute, we would instantiate this pattern for each such hazard.

There are three remaining patterns, one for each of the three remaining modes, following a similar rationale, but specific to the particular causes identified. Owing to space constraints, we do not show those patterns here. Additionally, an overarching pattern (also not shown here) relates the root claim of each of the four patterns for the relevant modes (e.g., goal node G1 for the EA mode, as shown in Figure 3) to a higher-level safety claim that the contribution of the associated LEC to identified system hazards is accept-

ably managed. The inference rule that we use to make this link is that the overall contribution of the LEC to the relevant system hazards is characterized by the individual hazard contributions modes and assurance of the latter entails that of the former. This is the essential basis of the reasoning underlying a bottom-up analysis such as FMEA that we have sought to replicate but with a safety focus.

We note that, in fact, this is only provides a part of the assurance since, in reality, component contribution often depends on other components and the associated interactions. However, that analysis is not in scope for this paper.

4 Related Work

Our taxonomy of hazard contribution modes (HCMs) shares some similarities with other efforts examining ML safety: For example, (Varshney and Alemzadeh 2017) characterizes ML outputs as either desired or undesired, in contrast with our characterization of LEC output types. Their work also proposes a number of high-level strategies for safety, including inherently safe design, safety reserves or factors, fail safety, and procedural safeguards. These are well aligned with most, if not all, of the mitigation mechanisms that we have identified earlier in this paper.

(Amodei et al. 2016) outlines five failure modes focusing on reinforcement learning, which (Faria 2018; 2017) builds upon, to address additional failure mode as applied to supervised and unsupervised learning. Their research also points to how the identified failure modes and safety issues may be managed, but the associated assurance rationale is left implicit. By contrast, our categorization of HCMs is broader, including both failure modes as well as non failing modes that have a safety impact. The associated assurance rationale is included as an argumentation pattern that can be conveniently examined, augmented, and improved as appropriate. Moreover, although the evidence requirements we identify are far from comprehensive, the framework is intended as a starting point for developing a more comprehensive safety assurance basis for LECs.

Other research has addressed the problem of creating GSN patterns for ML assurance generally: (McDermid, Jia, and Habli 2019) introduces a high-level framework for safety assurance of autonomous systems. This framework is based on the observation that there is a gap between the *real world*, the *world as observed* and the *world as imagined*. The gap is caused by problem-inherent, procedural, and engineering limitations of autonomous systems. Safety assurance under this proposed framework involves examining these gaps and assuring that the negative impacts on the safety of the system is limited. There are clear similarities between this model/reality gap approach and our understanding of expected and unexpected performance due to distributional shifts and validation.

(Bragg and Habli 2018) presents a high-level pattern for assurance of the general safety of a reinforcement learning system in a specific environment. This pattern is based on the concepts of safe configuration (the model construction and subsystem construction is safe) and failsafes.

(Picardi et al. 2019) presents a GSN pattern for assurance that an ML decision system achieves a specific level of performance, and applies this system to medical diagnosis ML, aggregating sub-arguments of suitable benchmarks, the operating environment, the learned model, the training data, and the test data. (Burton et al. 2019) builds upon this work, proposing a method to develop confidence arguments about model performance from testing evidence, and applies these methods to the problem of arguing sufficient performance in pedestrian detection for autonomous vehicles.

The argument pattern(s) we have introduced, explicitly capture the implicit reasoning that constitutes a systematic, bottom-up analysis of component-level safety contribution. We call out generic, high-level evidence requirements that emerge from this preliminary framework of analyzing HCMs, but we do not address the methods for assembling or assessing the concrete evidence items that would be required, nor do we address the numerous assurance techniques that may be leveraged.

For example, (Ashmore, Calinescu, and Paterson 2019) presents a theoretical framework of safety considerations in an ML component development lifecycle, defining specific assurance properties for constituent lifecycle stages, and an overview of the state of the art techniques that produce the required assurance evidence. The gap in their work that our paper fills, is linking the assurance properties of the lifecycle stages and, in turn, the evidence generating techniques to safety, via the HCMs.

Our work seeks to combine a systematic component-level assessment of ML safety with argumentation patterns to capture the associated assurance rationale. Unique to our work is the focus on ML embedded in a wider cyber- or cyber-physical system: understanding how these components contribute to system hazards as a starting point for creating the arguments required to assure that those contributions have been acceptably managed.

5 Conclusions

In this paper we have presented a framework for understanding the hazardous contributions of Learning-enabled Components (LECs) to system safety, in terms of hazard contribution modes (HCMs), characterized broadly in terms of LEC output type and specifically in terms of accuracy, error and performance expectations. Using examples we have elaborated some candidate causes and conditions under which they arise, and that which is entailed in providing assurance of mitigation, seeking to be as generic as possible. However, safety effects of HCMs cannot be generalized, since details about the specific LEC, its interfaces to the wider system, the system itself, and its operating context are necessary. Hence, here we have only given illustrative examples of potential effects of HCMs on safety. Moreover, our characterization of the modes is not intended to be complete, nor does it cover the full scope of mode causes and mitigations. Thus, this should be seen less as an exhaustive enumeration of all possible hazard-inducing scenarios but rather as a guide to thinking systematically about causes, effects, and hazard conditions (including failure modes) during a component-focused hazard analysis.

HCMs are applicable to many situations where the outputs, behavior, and effects of a software component at the

system level may be difficult to specify in advance and, hence, apply more generally than for LECs. However, traditional (i.e., non-ML) software components have traditional hazard analysis techniques at their disposal; hence, here we have proposed HCMs as a generic analysis framework that can be coupled with mitigations techniques specific to ML.

We have given argument patterns (one of which has been elaborated in detail) that captures the above characteristics for the different HCMs, along with generic, high-level mitigations for the identified associated causes. Additionally, we have elaborated the reasoning that would be used to compose each mode-specific pattern into a pattern that addresses the overall contribution of the LEC. In practice, the mode-specific patterns would be subsequently refined and augmented with more details that are specific to the particular LEC, the learning scheme that is implemented, the system in which it is integrated, and the environment within which the LEC and its containing system are deployed. These also form the data source from which the patterns would be instantiated to give concrete assurance arguments. Building on existing hazard analysis and argument generation functionality in our assurance case tool, AdvoCATE (Denney and Pai 2018), we plan to develop templates for supporting the analysis that goes with HCMs, a library of application/ML-specific patterns, implement a guided decision process to assist users with pattern selection, composition, and instantiation, towards automating the steps from fundamental analysis to assurance argument creation.

Finally, as future work, we plan to refine the HCMs for LECs from a methodological standpoint, towards refining the high-level evidence requirements into lower-level ones that can be linked to objective quality evidence.

Acknowledgments

This work was supported by the System-wide Safety project in the Airspace Operations and Safety program of the NASA Aeronautics Research Mission Directorate.

References

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. arXiv preprint. arXiv:1606.06565v2 [cs.AI].

Asaadi, E.; Denney, E.; and Pai, G. 2019. Towards Quantification of Assurance for Learning-Enabled Components. In *Proceedings of the 2019 15th European Dependable Computing Conference (EDCC)*, 55–62. IEEE.

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

Avižienis, A.; Laprie, J.-C.; Randell, B.; and Landwehr, C. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1(1):11–33.

Bragg, J. E., and Habli, I. 2018. What Is Acceptably Safe for Reinforcement Learning? In Gallina, B.; Skavhaug, A.; Schoitsch, E.; and Bitsch, F., eds., *Computer Safety, Reliability, and Security. SAFECOMP 2018. Lecture Notes in Computer Science*, volume 11094. Cham, Switzerland: Springer.

Burton, S.; Gauerhof, L.; Sethy, B. B.; Habli, I.; and Hawkins, R. 2019. Confidence Arguments for Evidence of Performance in Machine Learning for Highly Automated Driving Functions. In Romanovsky, A.; Troubitsyna, E.; Gashi, I.; Schoitsch, E.; and Bitsch, F., eds., *Computer Safety, Reliability, and Security. SAFECOMP 2019. Lecture Notes in Computer Science*, volume 11699. Cham, Switzerland: Springer.

Denney, E., and Pai, G. 2013. A Formal Basis for Safety Case Patterns. In Bitsch, F.; Guiochet, J.; and Kaâniche, M., eds., *Computer Safety, Reliability and Security (SAFECOMP 2013)*, volume 8153 of *LNCS*, 21–32.

Denney, E., and Pai, G. 2018. Tool Support for Assurance Case Development. *Automated Software Engineering* 25(3):435–499.

Faria, J. M. 2017. Non-determinism and Failure Modes in Machine Learning. In *Proceedings of the 2017 IEEE International Symposium on Software Reliability Engineering Workshops*. Toulouse, France: IEEE.

Faria, J. M. 2018. Machine Learning Safety: An Overview. In *Proceedings of the 26th Annual Safety-Critical Systems Symposium*. York, UK: Safety Critical Systems Club.

Koopman, P., and Fratrik, F. 2019. How Many Operational Design Domains, Objects, and Events? In Espinoza, H.; Yu, H.; Huang, X.; Lecue, F.; Chen, C.; Hernandez-Orallo, J.; hÉigeartaigh, S. O.; and Mallah, R., eds., *Proceedings of the Workshop on Artificial Intelligence Safety 2019*. Macao, China: CEUR Workshop Proceedings.

Koopman, P. 2019. Edge Cases and Autonomous Vehicle Safety. Presented at the 27th Safety-Critical Systems Symposium (SSS).

Liang, P., and Klein, D. 2008. Analyzing the Errors of Unsupervised Learning. In *Proceedings of the Human Language Technology and Association for Computational Linguistics (HLT/ACL) Conference*.

McDermid, J. E.; Jia, Y.; and Habli, I. 2019. Towards a Framework for Safety Assurance of Autonomous Systems. In Espinoza, H.; Yu, H.; Huang, X.; Lecue, F.; Chen, C.; Hernandez-Orallo, J.; hÉigeartaigh, S. O.; and Mallah, R., eds., *Proceedings of the Workshop on Artificial Intelligence Safety 2019*. Macao, China: CEUR Workshop Proceedings.

National Transportation Safety Board. 2017. Collision Between a Car Operating With Automated Vehicle Control Systems and a Tractor-Semitrailer Truck Near Williston, Florida, May 7, 2016. Highway Accident Report NTSB/HAR-17/02, NTSB, Washington, DC.

Picardi, C.; Hawkins, R.; Paterson, C.; and Habli, I. 2019. A pattern for arguing the assurance of machine learning in medical diagnosis systems. In Romanovsky, A.; Troubitsyna, E.; and Bitsch, F., eds., *Computer Safety, Reliability, and Security*, 165–179. Cham: Springer International Publishing.

Salay, R., and Czarnecki, K. 2018. Using machine learning safely in automotive software: An assessment and adaption of software process requirements in iso 26262.

The Assurance Case Working Group (ACWG). 2018. Goal Structuring Notation Community Standard Version 2.

Tiwari, A.; Dutertre, B.; Jovanović, D.; de Candia, T.; Lincoln, P. D.; Rushby, J.; Sadigh, D.; and Seshia, S. 2014. Safety envelope for security. In *Proceedings of the 3rd International Conference on High Confidence Networked Systems*, HiCoNS '14, 85–94. New York, NY, USA: ACM.

Varshney, K. R., and Alemzadeh, H. 2017. On the Safety of Machine Learning: Cyber-Physical Systems, Decision Sciences, and Data Products. *Big Data* 5(3):246–255.