Strategic Resilience Evaluation of Neural Networks Within Autonomous Vehicle Software

Anna Schmedding[1(✉)], Philip Schowitz[2], Xugui Zhou[3], Yiyang Lu[1], Lishan Yang[4], Homa Alemzadeh[3], and Evgenia Smirni[1]

[1] William and Mary, Williamsburg, USA
{akschmedding,ylu21,exsmir}@wm.edu
[2] University of British Columbia, Vancouver, Canada
philipns@cs.ubc.ca
[3] University of Virginia, Charlottesville, USA
{xz6cz,ha4d}@virginia.edu
[4] George Mason University, Fairfax, USA
lyang28@gmu.edu

Abstract. Self-driving technology has become increasingly advanced over the past decade due to the rapid development of deep neural networks (DNNs). In this paper, we evaluate the effects of transient faults in DNNs and present a methodology to efficiently locate critical fault sites in DNNs deployed within two cases of autonomous vehicle (AV) agents: Learning by Cheating (LBC) and OpenPilot. We locate the DNN fault sites using a modified Taylor criterion and strategically inject faults that can affect the functioning of AVs in different road and weather scenarios. Our fault injection methodology identifies corner cases of DNN vulnerabilities that can cause hazards and accidents and therefore dramatically affect AV safety. Additionally, we evaluate mitigation mechanisms of such vulnerabilities for both AV agents and discuss the insights of this study.

Keywords: Autonomous Vehicles · Fault Tolerance · DNNs

1 Introduction

Autonomous vehicles (AVs) are real-world safety-critical systems of increasing importance. With the growing complexity of software and use of deep neural networks (DNNs) for perception and control in AVs, many factors can threaten their safe operation, such as software bugs [1] and transient faults in hardware [2], leading to mis-classifications by DNNs and potential safety hazards.

Transient faults (i.e., soft errors) originating from cosmic radiation [3] or from operating under low voltage [4] have been shown to threaten the functionality of DNN hardware and software [5]. Transient faults in the main memory (DRAM) can manifest as single- or multi-bit flips, specifically in neurons or weights of DNN models [5] and may cause silent data corruption (SDC) where the output

A. Ceccarelli et al. (Eds.): SAFECOMP 2024, LNCS 14988, pp. 33–48, 2024.
https://doi.org/10.1007/978-3-031-68606-1_3

is faulty despite a seemingly "correct" execution. Since DRAM is used in AVs[1], this safety-critical application inherits the reliability challenges of DRAM faults. Transient faults have already contributed to vehicle crashes [7].

Table 1. Fault Space for OpenPilot Supercombo and LBC.

	LBC	Supercombo
# CONV Layers	40	70
# Weights	21,268,928	5,811,616
# Single-Bit Fault Sites	680,605,696	185,971,712
# Double-Bit Fault Sites	21,098,776,576	5,765,123,072
# Triple-Bit Fault Sites	632,963,297,280	172,953,692,160

Locating critical faults that cause safety hazards or accidents is necessary in such safety-critical systems. A major challenge here is the vast fault site space, often in the order of billions (see Table 1) which would require thousands of years to exhaustively analyze. This makes identifying corner cases where faults may affect the functional safety of a self-driving vehicle [8] similar to searching for a needle in a haystack. Within the AV ecosystem, the classical statistical fault injection [9], cannot discover the critical corner cases that could lead to safety violations. Past works focus on specific DNN tasks (e.g., image classification) and examine DNN resilience *without the context of the entire AV system* [10, 11]. Recent AV resilience assessment works have focused on input, models, or outputs [12–14], but not on its DNN components that are at the core of AV operation. *Our aim is to develop a method to efficiently locate these critical fault sites in the DNN components of AVs.* We evaluate these fault sites by injecting transient faults in DNN weights and determining whether their effects propagate to other AV components and eventually result in hazards or accidents.

We present a *strategic fault injection* method, called *Taylor-Guided Fault Injection (TGFI)*, that identifies and targets the DNN weights that are of high importance to reliable inference. This is done using a modified Taylor criterion [15] which ranks all the DNN weights with respect to their relative importance to inference accuracy. We inject faults in those important weights and show that our strategic fault injection method can efficiently discover safety-critical vulnerabilities in AVs. We specifically focus on locating critical fault sites within two AV systems: (i) Learning-by-Cheating (LBC), a fully autonomous self-driving agent [16] that is widely used in academic studies [2], and (ii) OpenPilot, a popular driver-assistance system that is used in over 250 existing car models on the road [17]. By using two systems with different levels of autonomy, we demonstrate the ability of TGFI to generalize to different AV DNNs.

[1] For example, the NVIDIA Jetson AGX Orin Series, which is used by NVIDIA DRIVE, uses 32GB or 64GB of DRAM [6].

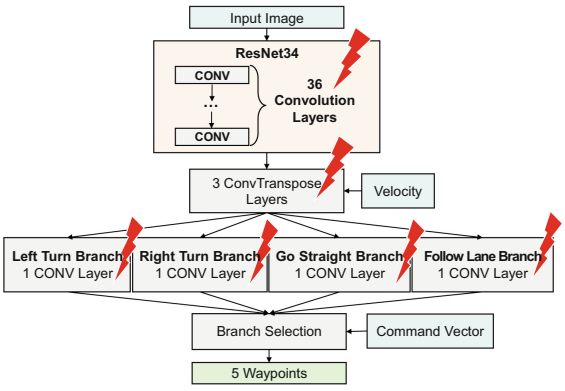
We also characterize the effect of mitigation in the cases where these critical faults occur. For LBC, we consider a state-of-the-art fault tolerance method based on neuron value range restriction for CNNs, called Ranger [11]. For OpenPilot, we examine the existing system safety checks which return control to the human driver. Additionally, we examine the effects of considering contextual factors such as the location of faults and environmental conditions that impact the input in order to offer insights into the practical reliability challenges of deploying AVs on the road. Both mitigation methods show improvement in resilience, while TGFI is still able to find critical corner cases.

2 Autonomous Driving Frameworks

The Society of Automotive Engineers (SAE) defines 6 levels of driving automation for AVs, from Level 0 (L0, no driving automation) to Level 5 (L5, full driving automation) [18]. L0 to L2 assume that there is a human in the loop who controls the automotive environment by supervising or taking over the autonomous features. For higher levels, the car autonomously controls the driving environment without human involvement. With DNN models incorporated, an L4 AV may use end-to-end ML models for perception and planning without human intervention, while L0-L2 levels use DNNs for driver assistance.

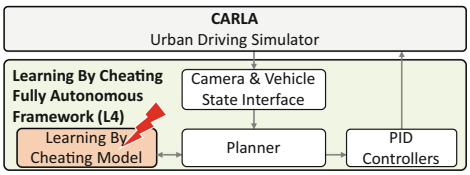
2.1 L4 System: LBC

Learning by Cheating (LBC) is a pretrained end-to-end agent for L4 autonomous driving and is widely used in research studies [2,19,20]. Figure 1a shows the model structure of LBC, which is built on a ResNet34 [21] backbone to process input images from a front-facing camera on the vehicle. The model takes two additional inputs: vehicle speed and a high-level command vector generated by the planner, instructing the vehicle to follow the lane, turn left/right



(a) LBC model structure.

(b) LBC screenshot. Waypoints are red dots.



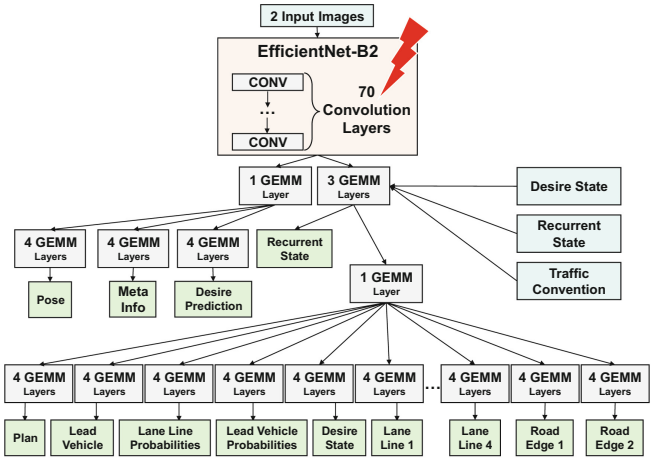
(c) LBC integration in the Carla simulator.

Fig. 1. LBC framework.

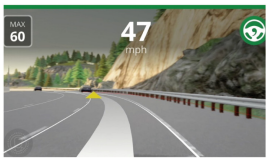
or go straight at an intersection. After the ResNet34 backbone, the model splits into four parallel branches that each corresponds to a high-level command. The final output is a set of five waypoints of the AV path (see Fig. 1b), which are passed to a low-level controller that produces the steering, throttle, and braking commands.

2.2 L2 System: OpenPilot

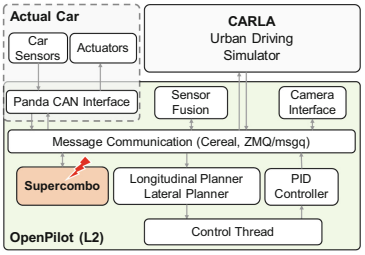
OpenPilot is an L2 Autonomous Driving Assistance System (ADAS) that supports more than 250 popular makes and models of cars [17]. A high-level overview of OpenPilot is shown in Fig. 2c. Car sensor data such as images and vehicle state information are passed into the Supercombo model. The output of Supercombo is sent to the planners. The PID Controller uses the results from the planner to decide actuator actions. The generated commands (e.g., brake) are passed through the Panda CAN interface to the actuators to perform the commands.



(a) OpenPilot Supercombo model structure.



(b) OpenPilot screenshot.



(c) OpenPilot Overview.

Fig. 2. OpenPilot framework.

The Supercombo model is at the core of the perception module and provides fifteen output fields. As shown in Fig. 2a, Supercombo utilizes an EfficientNet-B2 [22] base CNN for processing the incoming images from the car sensors. Then it uses additional inputs of the traffic convention, desire state, and recurrent state to incorporate the state of the vehicle and environment. Once all inputs have been tied in, Supercombo branches into separate general matrix multiplication (GEMM) computations to generate the plan, lanelines, laneline probabilities, road edges, lead vehicles, lead probabilities, desire state, meta information, vehicle pose, and recurrent state. Figure 2b shows an example screenshot.

2.3 Driving Simulator: CARLA

CARLA is an open-source simulator for autonomous driving research, design, and testing [23]. It provides a realistic urban environment for a vehicle to navigate with features such as variable road and weather conditions. CARLA provides a wealth of data at each simulation timestamp including whether a collision, lane invasion, or red light violation has occurred. CARLA is integrated to Openpilot and LBC, see Fig. 2c and Fig. 1c, respectively.

3 Methodology

Fault Model. We use fault injection in a single 32-bit floating point weight of the DNN to simulate commonly occurring transient faults in DRAM (Dynamic Random Access Memory). DNN weights typically reside in DRAM. A fault site in a neural network weight is defined by the weight id and the bit position(s) of the weight to be flipped. The size of the fault site space for single-, double-, and triple-bit faults for OpenPilot Supercombo and LBC is tremendous, see Table 1 and prevents its exhaustive exploration.

For the majority of the analysis (Sects. 4 and 5) we focus on double-bit flips that are detectable but not correctable by ECC. This is consistent with other reliability studies [11,24] and also consistent with DRAM faults in the wild [25]. For a broader view of the effect of bit flips, we also do experiments with single-bit flip (detectable and correctable) and triple-bit flips (undetectable, their safety implications are similar to double-bit ones, see Sect. 6).

Fault Injection Method. Fault injection is implemented as a two-stage process: *Preparation* and *Injection*. In the *Preparation* stage, before the actual simulation run, we load the (correct) neural network and select an injection site. Portions of the network where faults may be injected are denoted by a lightning bolt in Fig. 1a for LBC and Fig. 2a for OpenPilot Supercombo. We corrupt a weight tensor of the neural network by altering the values of an individual weight (depending on the type of experiment, we induce one single-bit, one double-bit, or one triple-bit fault), and save the corrupted tensor.[2] In the *Injection* stage, the corrupted model generated by the fault injector is used by the AV control software while performing the driving task. The corrupted Pytorch or ONNX files are loaded at the beginning of each LBC or OpenPilot experiment.

[2] Once the weight is altered, the modified model is written to a file in the necessary format (ONNX [26] or PyTorch [27], for OpenPilot and LBC, respectively). To corrupt Pytorch models, we load the model, alter the state dictionary associated with it and then save the new faulty model. For ONNX models, we read the model file as bytes, locate the plain text layer ID, and modify the bits corresponding to the target weight in the binary data.

Fault Injection Outcomes. Throughout the simulation, we focus on events that indicate abnormal behavior. A *lane invasion* occurs when the vehicle crosses into a neighboring lane erroneously. Since a lane invasion can occur when the car barely crosses the lane line, a lane invasion alone is not a hazard. We define a *hazard* to be one of the following situations:
H1: The vehicle violates safe following-distance constraints with the lead vehicle: *Relative Distance* / *Speed* $\leqslant t_{safe}$ and *Speed* > *Lead Speed*.
H2: The vehicle drives out of lane beyond a threshold (e.g., 0.1 m) while speed is higher than β, these are predetermined values given by the driving scenario.

We record all hazards that occur within the experiment as well as their time stamps. These hazards can lead to the following *accidents*:
A1: Collision with the lead vehicle.
A2: Collision with road-side objects or other vehicles in the neighboring lane.

An accident terminates the simulation, at which point the time stamp and the nature of the accident are recorded. The simulation also terminates if the vehicle successfully reaches its goal location. These definitions are based on the STPA [28] hazard analysis method, which has been utilized in other studies [12, 29]. The hazards considered here are indicative of failures of lane keep assist (LKA) and adaptive cruise control (ACC), two main functionalities of L2 AVs.

3.1 Vulnerable Weights: Taylor Guided Fault Injection (TGFI)

In a fault space as vast as the one reported on Table 1, the odds that the standard practice of 1,000 random fault injections [30] capture rare corner cases that result in catastrophic driving scenarios are low. Since our target is the identification of the aforementioned corner cases, we rank the importance of the weights in the neural network using a modified Taylor criterion [15] which estimates the first-order Taylor expansion of the contribution of each weight to the accuracy of the neural network. The more a weight contributes to the accuracy of the network, the more critical it is and the more it can affect accuracy if a fault occurs there. We relatively rank all weights in the two target models using

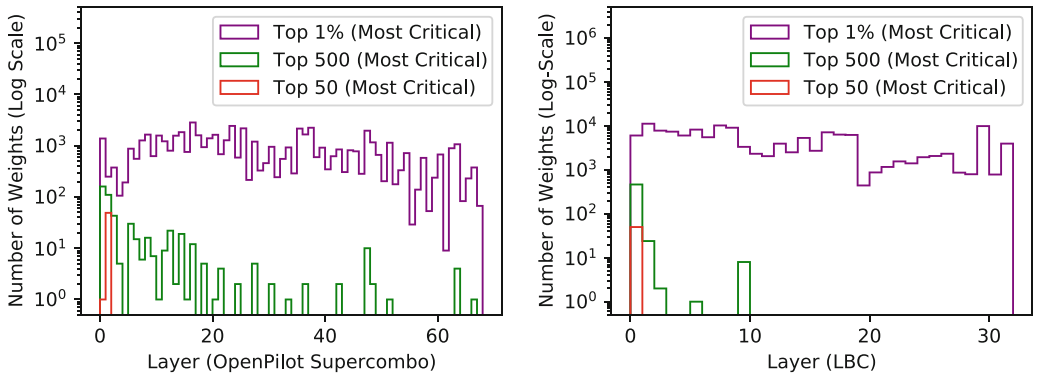


Fig. 3. Locations of top 50, top 500, and top 1% of important weights in OpenPilot Supercombo (left) and LBC (right).

the *importance scores* generated by the Taylor criterion [15]. The importance of a weight using this criterion can be estimated using the following equation:

$$I_m^{(1)}(W) = (g_m w_m)^2, \tag{1}$$

where I is the importance, W is the set of network parameters, g_m are elements of the gradient, and w_m are the weight values.

We use the Comma2k19 data set [31] of real-world driving footage as input to OpenPilot Supercombo and LBC, and compute the importance of their weights [15]. Figure 3 illustrates the location (layer, x-axis) and number (y-axis logscale) of the most critical weights for OpenPilot Supercombo and LBC. We make the following observations: 1) critical weights may be located in any layer and 2) the few *most* critical weights are concentrated in the earlier layers of both models. We perform fault injection experiments on the most critical weights as guided by Eq. 1, called *Taylor-Guided Fault Injection (TGFI)*.

3.2 Experimental Campaigns

Every distinct fault site uses the same map for the AV to traverse, the same starting location of the car(s) in the simulator, the same goal location, and the same random seed, initial velocity, and weather. The fault sites are selected according to the fault injection (FI) campaign: 1) **Random**: The weight where the double bit flip occurs is chosen using a uniform distribution [30]: we select 1,000 random fault sites to obtain results with 95% confidence intervals and ±3% error margins. 2) **TGFI-top500**: We select the top 500 most critical weights. 3) **TGFI-top50**: We select the top 50 most critical weights.

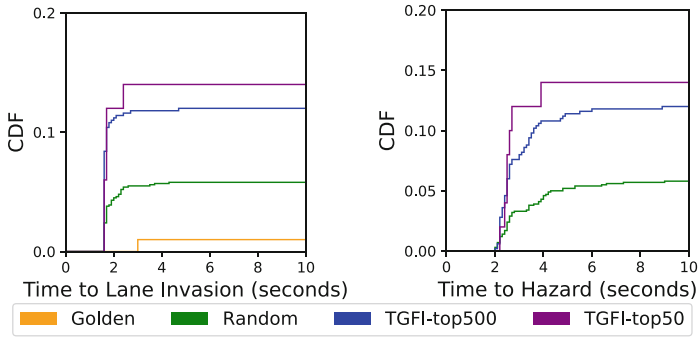


Fig. 4. LBC time to lane invasion and time to hazard. No time to driver intervention because of L4 autonomy. *Note that CDFs do not reach 1.0 since a portion of the experiments do not experience a lane invasion or hazard.*

4 Resilience Evaluation

Following the methodology laid out in Sect. 3, we perform three FI campaigns for LBC and OpenPilot. These FI campaigns highlight different methods for

selecting the target weight for fault injection (i.e., Random FI, TGFI-top500, TGFI-top50) using a two-bit fault model. We perform golden runs (i.e., fault-free) to capture normal behavior. The driving scenario is on a curved road in cloudy weather, which represents ideal driving conditions (i.e., good visibility with no glare). Variations of experimental setup are explored in Sect. 6.

4.1 Resilience of L4 LBC

Lane invasions are the least severe violation, since not every lane invasion leads to a hazard. Only 1% of experiments have lane invasions in golden runs, indicating that these are rare in fault-free cases. The time to lane invasion is shown in Fig. 4(a). Across all three fault injection campaigns, the lane invasion tends to occur early in the simulation. Table 2 shows the percentage: TGFI experiments double the percentage of lane invasions comparing to the random FI experiments.

Table 2. Percentage of various events. There is no front vehicle in LBC, hence H1 and A1 cannot happen. Driver intervention: the L2 system returns control to the driver.

ADS	Fault-Site Selection	H1	H2	A1	A2	Lane Inv.	Driver Int.
L4 (LBC)	Golden run	N/A	0%	N/A	0%	1%	N/A
	Random FI	N/A	5.9%	N/A	5.9%	6%	N/A
	TGFI-top500	N/A	12%	N/A	12%	12%	N/A
	TGFI-top50	N/A	14%	N/A	14%	14%	N/A
L2 (OpenPilot Supercombo)	Golden run	0%	0%	0%	0%	10%	0%
	Random FI	2.0%	0.6%	0%	0.1%	23.4%	21.4%
	TGFI-top500	3.3%	8.8%	0.1%	0.2%	29.62%	21.0%
	TGFI-top50	7.9%	16.3%	0.2%	0.3%	62.5%	18.3%

No hazards are detected in the golden fault-free runs, but Random FI and TGFI cause H2 hazards. The time to hazard is shown in Fig. 4. For both random FI and TGFI, hazards are encountered between the one and two second mark. There is a near-complete overlap between the experiments that have lane

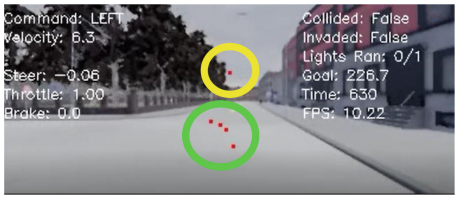


Fig. 5. Fault injection in left-turn control command branch in LBC. Four waypoints (circled in green) are correct but one (circled in yellow) is incorrect. (Color figure online)

invasions and the experiments that have hazards, i.e., if the fault causes a lane invasion, then the fault is also severe enough to cause a hazard shortly after. Similar to the lane invasions, TGFI finds more hazards than Random FI.

Every hazard for Random FI and TGFI scenarios results in an accident, indicating that LBC has a very limited ability to function under critical faults. Indeed, LBC does not have alerts or safety checks. TGFI triggers more accidents than Random FI.

We also evaluate the impact of faults in the convolution branches at the end of the network (see Fig. 1a) corresponding to high-level control commands. Figure 5 presents the results of a fault injected in the branch corresponding to the left turn control command: four waypoints are correct, but one (yellow circle) is clearly incorrect. When the car turns the corner, it tries to adhere to the trajectory generated by fitting a curve to the points, but in doing so turns the corner too widely and crashes into the wall on the side of the road.

4.2 Resilience of L2 OpenPilot

In OpenPilot, lane invasions occur often but do not always result in a hazard or accident. 10% of golden runs and 23.4% of Random FI experiments have lane invasions, see Table 2. TGFI-top500 shows slightly more lane invasions and TGFI-top50 nearly triples the number of lane invasions compared to Random FI. The time to lane invasion is shown in Fig. 6(a): most of the lane invasions occur between the 15 and 20 s marks.

The golden runs never result in a hazard, but 2.6% of Random FI experiments result in a hazard that typically occurs between the 5 and 10 s marks, see Fig. 6(b). 10.8% of TGFI-top500 experiments and 22.7% of TGFI-top50 experiments result in a hazard (1.3% and 1.5% of them experience both H1 & H2 hazards for TGFI-top500 and TGFI-top50, respectively). Hazards only occur after second 15 in the simulation and show a steeper increase towards second 50. The H1 hazard that occurs when the vehicle follows another car too closely

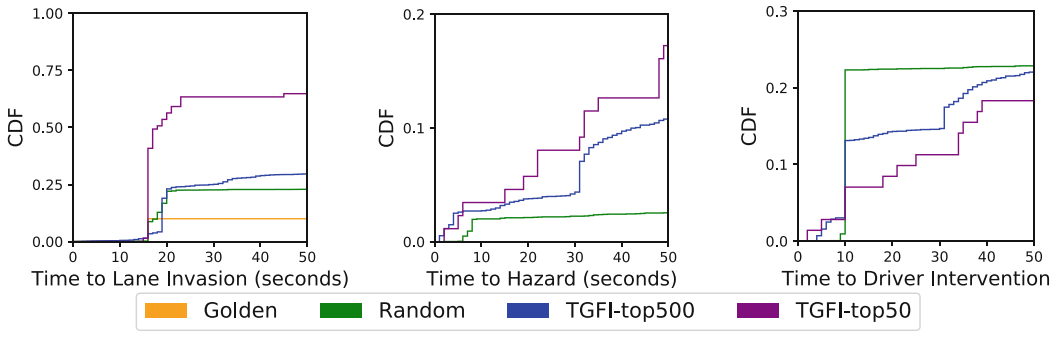


Fig. 6. OpenPilot Supercombo: Time to lane invasion, hazard, and driver intervention. No hazards or driver interventions were observed in golden runs. *CDFs do not reach 1.0 since some experiments do not have a lane invasion, hazard, or driver intervention.*

occurs in 7.9% of simulations for TGFI-top50 and the H2 hazard, which indicates a significant lane invasion, occurs in 16.3% of experiments for TGFI-top50, see Table 2. TGFI results in more hazards than Random FI.

Since OpenPilot requires the driver to resume control of the vehicle in the case of dangerous situations, hazards are often masked and accidents rarely happen, see Table 2. We will discuss this mitigation technique in detail in 5.2.

5 Mitigation

The effects of faults can be mitigated through several approaches, many of which are orthogonal to one another [11,24,32]. In this section, we examine different mitigation techniques for the two AV cases examined here.

5.1 L4 LBC: Ranger

Ranger [11] is a popular, state-of-the-art fault corrector which employs range restriction on neuron activation values to protect ML models from faults with negligible overhead. Here, we present a proof-of-concept mitigation of applying Ranger to LBC. To implement Ranger, we insert range restriction into the model following activation layers at crucial points in the network, such as after convolution layers. Each protection layer has a pair of minimum and maximum activation values to use as bounds, which are set after profiling through golden runs under cloudy (ideal) weather. This step is performed once, before the deployment of the protected model with Ranger. When Ranger is active, any activation values outside the ranges defined by Ranger are clipped to the bound.

We examine the effectiveness of Ranger using the TGFI-top500 experiments, see Fig. 7. We examine three CARLA driving scenarios: curved road, turn at intersection, and straight road. A combination of clear and inclement weather conditions (cloudy, rainy, sunset, wet) is also used to offer insight into how well the fault mitigation functions in unseen conditions. Ranger improves the resiliency of LBC across all three driving scenarios and all weather conditions. Significant improvements are observed under cloudy (ideal) weather conditions.

5.2 L2 OpenPilot: Driver Intervention

As an L2 ADAS, OpenPilot raises driver alerts and returns control to the driver when a problem is detected by its safety checker, i.e., *driver intervention*. We record the *time to driver intervention* as a CDF in Fig. 6(c). The golden runs never require driver intervention and are therefore not plotted. Random FI results in driver intervention in 21.4% of experiments, occurring close to the 10 s mark. The TGFI-top500 and TGFI-top50 experiments show an initial jump in driver intervention early in the simulation. OpenPilot has 3 types of driver alerts which may be displayed when control of the vehicle is returned to the driver, see Table 3. In fault injection experiments, PlannerError/noEntry alerts are the most

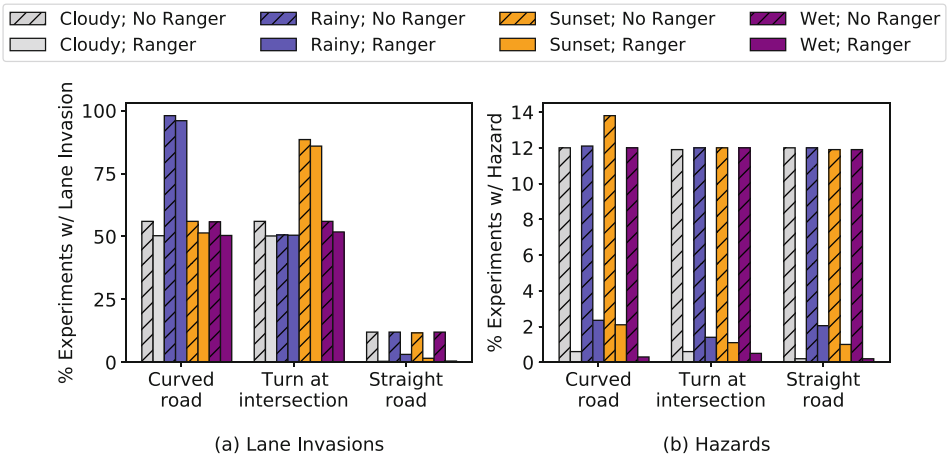


Fig. 7. Comparison of LBC without Ranger vs. LBC with Ranger for the TGFI-500 scenario.

common ones. These alerts indicate that the Model Predictive Control (MPC) cannot find a feasible solution for lateral planning (steering) and longitudinal planning (gas/brake) and releases control to the driver, successfully mitigating the fault before any hazard is triggered. The CanError/ImmediateDisable alerts indicate communication errors and occur the least frequently. The steerSaturatedWarning alerts also rarely occur and they indicate that the car is swerving sharply in the presence of the fault. *This indicates that TGFI finds faults that are more rare and more challenging for OpenPilot safety checks to detect and mitigate.*

Table 3. Experiments where the driver is alerted to a problem in OpenPilot.

Alert	Golden Runs	Random	TGFI-top500	TGFI-top50
plannerError/noEntry	0%	19.8%	10.88%	4.01%
canError/immediateDisable	0%	0.5%	0.1%	0.4%
steerSaturated/warning	0%	1.9%	2.98%	3.85%

6 Case Studies and Discussion

In this section, we use LBC as a case study to evaluate and discuss the impact of different faults, layers of DNN and bit positions on AV resilience.

6.1 Importance of Layer Depth for Resilience

Table 4 shows four distinct experiments with corrupted LBC models that identify the importance of the layer depth where the fault occurs. When Ranger is

Table 4. Case study of four distinct corrupted LBC models.

Exp. ID	Layer ID	Lane Invasion (No Ranger)	Accidents (No Ranger)	Lane Invasion (Ranger)	Accidents (Ranger)
1	10	100%	100%	0%	0%
2	30	100%	100%	0%	0%
3	37	92.5%	100%	91%	100%
4	38	53.3%	84.7%	32.7%	78.4%

not applied and if the fault is injected in the earlier DNN layers (experiments 1 and 2), resilience is severely affected: all faults result to accidents. Faults occurring earlier in the network have more time to propagate horizontally across the neurons, resulting in more severe corruption. This propagation still occurs even if the fault site is relatively deep into the ResNet34 section of the network, as experiment 2 with injection in layer 30 shows. The fault sites of experiments 3 and 4 are in a the final layers in the network, therefore error propagation is minimal and the severity of corruption in the final output is lessened.

With Ranger, results for faults injected in layer 10 and 30 improve from 100% accidents to 0% accidents. For faults in layer 37 and 38, applying Ranger still results in a majority of accidents. Since these two layers are at the end of the network, the clipped Ranger bound used as output, differs significantly from the ideal one.

6.2 Sensitivity to Single and Multi-bit Faults

We analyze the sensitivity of LBC to single- and multi-bit faults [33]. We evaluate how the bit position(s) and the number of bit flips affect hazards in the LBC model under cloudy ("easy") and rainy ("challenging") weather conditions using TGFI-500, as shown in Fig. 8. We compare fault injection outcomes using different numbers and locations of bit flips: single-bit exponent, single-bit mantissa, double-bit (the fault model majorly used in this paper), and triple-bit. These experiments show that single-bit exponent causes the most hazards, followed by triple-bit, double-bit, and finally single-bit mantissa. This experiment also shows that lane invasions and hazards for *non-detectable triple* bit faults are prominent, thus their mitigation for safety is of paramount importance.

6.3 Lessons Learned from L4 LBC and L2 OpenPilot

L4 LBC and the L2 OpenPilot have similarities: both ML models are structured around a backbone CNN which is more vulnerable to faults in earlier layers than later ones. Faults are commonly masked when they occur in weights of low importance. Meanwhile, structural differences of the DNNs affect their resilience: LBC has high-level command branches that activate or deactivate parts of the network depending on the driving scenario, thus possibly masking faults during

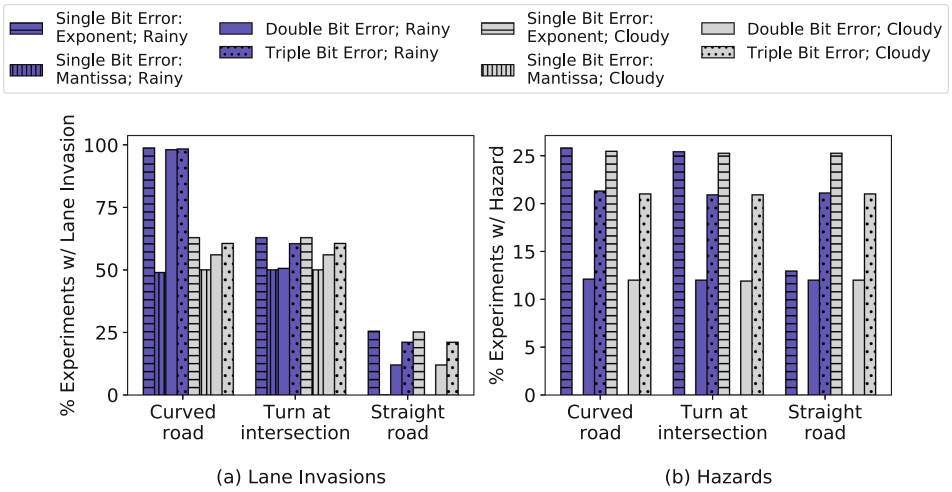


Fig. 8. Bit position analysis for LBC and the TGFI-top500 experimental campaign.

inference. OpenPilot uses all layers in every inference, meaning that faults can only be masked by the network itself, rather than circumstances around its use. Secondly, the format of the DNN output is different. LBC outputs waypoints while OpenPilot's Supercombo outputs information about lanelines, road edges, lead vehicle position, and many other variables. The availability of these different outputs in OpenPilot allows for the implementation of safety checks.

For OpenPilot, we find that many accidents are not mitigated by the system safety checks. This speaks to both the need for the improvement of existing checks, and for the implementation of new ones. Adding DNN-oriented error detection/correction mechanisms would improve AV resilience. High-level checks, like many of those employed in OpenPilot, cannot detect potential incorrect outputs but while imperfect, they provide a blueprint for developing resilience in L4 systems like LBC.

7 Related Work

Past studies have shown that faults [34] can cause hazardous behavior in AVs. DeepTest [1] focuses on testing the reliability of autonomous driving systems and safety engineering techniques for autonomous systems are investigated in [35], but these works do not consider soft errors. Other works examine the fault space for AVs through Bayesian fault injection [34] and rely on large amounts of random fault injection experiments. [36] explores the problem of effective safety checks for an ML-based AV.

[2] uses duplication of computation and temporal data diversity to improve the resilience of AVs with LBC but requires additional hardware. On the OpenPilot side, [12] focuses on hazard coverage and fault injection, but does not study its ML model. [37,38] present attacks on the CAN bus or camera input for a vehicle using OpenPilot. Unlike these works, we focus on the effects of unintended faults rather than attacks.

Most importantly, both for LBC and Openpilot, we identify corner cases that are otherwise hard to find: we identify which portions of their DNNs are vulnerable to faults and result in AV safety violations, i.e., we do not simply examine the accuracy of DNNs for classification but their holistic effect into the AV operation.

8 Conclusions

We perform strategic resilience evaluation of the neural networks of two autonomous vehicles against transient faults. We focus on an L4 system widely used in academia and an L2 ADAS system which is widely deployed on the road. We find that both systems are vulnerable to single- and multi-bit faults which may induce hazards/accidents. We use the Taylor criterion to strategically identify the most important weights for reliability in the DNNs used in the L4 LBC and L2 Openpilot and inject errors on those weights using TGFI. TGFI is efficient in identifying vulnerabilities that result in hazards and accidents. We also examine the effectiveness of mitigation in AVs. For L4 self-driving, mitigation techniques such as Ranger can be effective at minimizing the impact of faults. Driver intervention is a crucial contributing factor to the security of L2 systems.

Acknowledgments. This material is based upon work supported by two Commonwealth Cyber Initiative (CCI) grants (#HC-3Q24-047 and COVA C-Q122-WM-02).

References

1. Tian, Y., Pei, K., Jana, S., Ray, B.: DeepTest: automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th International Conference on Software Engineering, pp. 303–314 (2018)
2. Jha, S., et al.: Exploiting temporal data diversity for detecting safety-critical faults in AV compute systems. In: 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 88–100. IEEE (2022)
3. Fratin, V., Oliveira, D., Lunardi, C., Santos, F., Rodrigues, G., Rech, P.: Code-dependent and architecture-dependent reliability behaviors. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 13–26. IEEE (2018)
4. Ganapathy, S., Kalamatianos, J., Beckmann, B.M., Raasch, S., Szafaryn, L.G.: Killi: runtime fault classification to deploy low voltage caches without MBIST. In: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 304–316. IEEE (2019)
5. Li, G., et al.: Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In: Proceedings of Supercomputing, pp. 1–12 (2017)
6. Karumbunathan, L.S.: NVIDIA Jetson AGX Orin Series: A Giant Leap Forward for Robotics and Edge AI Applications (2022). https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf

7. Yoshida, J.: Toyota Case: Single Bit Flip That Killed (2013). https://www.eetimes.com/toyota-case-single-bit-flip-that-killed
8. Road vehicles - Functional safety. Standard, International Organization for Standardization, Geneva, CH, December (2018)
9. Leveugle, R., Calvez, A., Maistri, P., Vanhauwaert, P.: Statistical fault injection: quantified error and confidence. In: 2009 Design, Automation & Test in Europe Conference & Exhibition, pp. 502–506. IEEE (2009)
10. dos Santos, F.F., et al.: Analyzing and increasing the reliability of convolutional neural networks on GPUs. IEEE Trans. Reliab. **68**(2), 663–677 (2018)
11. Chen, Z., Li, G., Pattabiraman, K.: A low-cost fault corrector for deep neural networks through range restriction. In: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1–13. IEEE (2021)
12. Rubaiyat, A.H.M., Qin, Y., Alemzadeh, H.: Experimental resilience assessment of an open-source driving agent. In: 2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 54–63. IEEE (2018)
13. Jha, S., et al.: Kayotee: a fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors (2019). arXiv preprint arXiv:1907.01024
14. Jha, S., et al.: ML-driven malware that targets AV safety. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 113–124. IEEE (2020)
15. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11264–11272 (2019)
16. Chen, D., Zhou, B., Koltun, V., Krähenbühl, P.: Learning by cheating. In: Conference on Robot Learning, pp. 66–75. PMLR (2020)
17. Comma.ai. Supported Cars by OpenPilot. https://github.com/commaai/openpilot/blob/master/docs/CARS.md
18. SAE International. SAE Levels of Driving Automation™ Refined for Clarity and International Audience (2021). https://www.sae.org/blog/sae-j3016-update
19. Filos, A., Tigkas, P., McAllister, R., Rhinehart, N., Levine, S., Gal, Y.: Can autonomous vehicles identify, recover from, and adapt to distribution shifts? In: International Conference on Machine Learning, pp. 3145–3153 (2020)
20. Toromanoff, M., Wirbel, E., Moutarde, F.: End-to-end model-free reinforcement learning for urban driving using implicit affordances. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7153–7162 (2020)
21. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
22. Tan, M., Le, Q.: EfficientNet: rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning, pp. 6105–6114 (2019)
23. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: an open urban driving simulator. In: Conference on Robot Learning, pp. 1–16 (2017)
24. Kadam, G., Smirni, E., Jog, A.: Data-centric reliability management in GPUs. In: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 271–283. IEEE (2021)
25. Beigi, M.V., Cao, Y., Gurumurthi, S., Recchia, C., Walton, A., Sridharan, V.: A systematic study of DDR4 dram faults in the field. In: 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 991–1002. IEEE (2023)

26. The Linux Foundation. Open neural network exchange: The open standard for machine learning interoperability (2019). https://onnx.ai/
27. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
28. P Leveson, N., Thomas, J.: An STPA Primer. Cambridge, MA (2013)
29. Zhou, X., Ahmed, B., Aylor, J.H., Asare, P., Alemzadeh, H.: Data-driven design of context-aware monitors for hazard prediction in artificial pancreas systems. In: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 484–496. IEEE (2021)
30.) Nie, B., Yang, L., Jog, A., Smirni, E.: Fault site pruning for practical reliability analysis of GPGPU applications. In: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 749–761 (2018)
31. Schafer, H., Santana, E., Haden, A., Biasini, R.: A commute in data: The comma2k19 dataset (2018). arXiv preprint arXiv:1812.05752
32. Yang, L., Nie, B., Jog, A., Smirni, E.: Enabling software resilience in GPGPU applications via partial thread protection. In: 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021, pp. 1248–1259 (2021)
33. Yang, L., Nie, B., Jog, A., Smirni, E.: Practical resilience analysis of GPGPU applications in the presence of single- and multi-bit faults. IEEE Trans. Comput. **70**(1), 30–44 (2021)
34. Jha, S., et al.: ML-based fault injection for autonomous vehicles: a case for Bayesian fault injection. In: 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019, pp. 112–124 (2019)
35. Osborne, M., Hawkins, R., McDermid, J.: Analysing the safety of decision-making in autonomous systems. In: Trapp, M., Saglietti, F., Spisländer, M., Bitsch, F. (eds.) Computer Safety, Reliability, and Security. SAFECOMP 2022. LNCS, vol. 13414. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-14835-4_1
36. Terrosi, F., Strigini, L., Bondavalli, A.: Impact of machine learning on safety monitors. In: Trapp, M., Saglietti, F., Spisländer, M., Bitsch, F. (eds.) Computer Safety, Reliability, and Security. SAFECOMP 2022. LNCS, vol. 13414. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-14835-4_9
37. Zhou, X., et al.: Strategic safety-critical attacks against an advanced driver assistance system. In: 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 79–87. IEEE (2022)
38. Zhou, X., et al.: Runtime stealthy perception attacks against DNN-based adaptive cruise control systems (2024). arXiv preprint arXiv: 2307.08939

System and Software Safety Assurance

Reconciling Safety Measurement and Dynamic Assurance

Ewen Denney and Ganesh Pai(✉)

KBR/NASA Ames Research Center, Moffett Field, CA 94035, USA
{ewen.denney,ganesh.pai}@nasa.gov

Abstract. We propose a new framework to facilitate dynamic assurance within a safety case approach by associating safety performance measurement with the core assurance artifacts of a safety case. The focus is mainly on the *safety architecture*, whose underlying risk assessment model gives the concrete link from safety measurement to operational risk. Using an aviation domain example of autonomous taxiing, we describe our approach to derive safety indicators and revise the risk assessment based on safety measurement. We then outline a notion of *consistency* between a collection of safety indicators and the safety case, as a formal basis for implementing the proposed framework in our tool, AdvoCATE.

Keywords: Dynamic assurance · Safety cases · Safety measurement · Safety metrics · Safety performance · Safety risk assessment

1 Introduction

Software-based self-adaptation and machine learning (ML) technologies for enabling autonomy in complex systems—such as those in civil aviation—may induce new and unforeseen ways for operational safety performance to deviate from an approved baseline of acceptable risk. This phenomenon, known as *practical drift* [13], emerges from the inevitable variabilities in real-life operations to meet service expectations in an operating environment that is inherently dynamic. Conceptually, it can be understood as progressively imperceptible reductions in the safety margins built into a system in part due to initially benign operational tradeoffs between safety and performance. A system therefore appears to be operating safely but, in fact, is operating at a higher level of safety risk than what was originally considered acceptable, or approved for service. Left unchecked, practical drift may suddenly manifest as a serious *incident* or *accident*. Assessing the change in operational safety risk is thus key to identifying practical drift, its impact, and the mitigations needed.

Related Work. The conventional approach to operational safety assurance in aviation largely relies upon hazard tracking and safety performance monitoring and measurement, as part of a larger *safety management system* (SMS) [10].

A. Ceccarelli et al. (Eds.): SAFECOMP 2024, LNCS 14988, pp. 51–67, 2024.
https://doi.org/10.1007/978-3-031-68606-1_4

The contemporary safety case approach to assurance has similarly employed safety monitoring and measurement: for example, our earlier work on *dynamic safety cases* [5] first suggested connecting safety monitoring to assurance argument modification actions. Subsequently, an approach to defining performance metrics and monitors by identifying the defeaters and counterarguments to a safety case has been developed in [12]. The concept has since also been applied to safety assurance of self-adaptive software [4], and to detect operational exposure to previously unknown hazardous conditions [18]. More recently, the use of *safety performance indicators* (SPIs)—a concept with a well-established history of use in aviation safety [13]—has been proposed for evaluating safety cases for autonomous vehicles [15]. These approaches all share a common motivation: using measurement based assessment to confirm at deployment, and maintain in operation, the validity of the assurance arguments of a safety case.

Although such an approach suggests which parts of an argument may have been invalidated, and thus require changing, the nature and extent of the change to operational safety risk levels is left implicit. Such analyses can also meaningfully inform what modifications may be needed to the system and its safety case, especially when—due to practical drift—improved system performance is observed without detrimental safety effects, even though parts of the safety argument have become invalid. Current safety case approaches that use safety performance measurement to validate assurance arguments give limited guidance on how to facilitate what this paper considers as *dynamic* assurance (see Fig. 3): *continued, justified confidence that a system is operating at a safety risk level consistent with an approved risk baseline.*

There are other variations of the dynamic assurance concept [17,21] that aim to optimize operational system performance, and thus opt for situation-specific runtime tradeoffs between safety and functional performance, instead of designing for the worst case. However, such tradeoffs may result in the initiating conditions for practical drift. Our proposed framework rather aims to identify and contain practical drift, whilst considering that a safety case for a system is always for a design that accounts for the worst credible safety effects. In [18], dynamic assurance refers to the automated aspects of so-called *continuous assurance*: a concept that, in effect, extends our prior work [5], by using monitors for different kinds of uncertainty that then trigger modifications to the system and its assurance case. The relationship of testing and operational metrics to safety assurance has been explored in [19], similar to our work in this paper (see Sect. 4), though there the focus is on providing confidence that a system meets its safety target, given evidence of mishap-free operation. In contrast, our focus here is on determining how safety risk has changed given similar measurement evidence.

Contributions and Paper Organization. To facilitate a framework for dynamic assurance within a safety case approach, the focus of this paper is on associating safety performance measurement with the *safety architecture* of a system, in addition to assurance arguments (Sect. 2). Using an aviation domain system (Sect. 3) as motivation, we present our approach to define safety metrics and indicators, through a concept of *safety measurement basis* (SMB), then revise

the operational safety risk assessment based on safety measurement, and characterize the change to safety risk levels (Sect. 4). Additionally, we give illustrative numerical examples. Then (Sect. 5) we formalize a notion of *consistency* between the SMB for a system and the arguments of its safety case. We conclude (Sect. 6) by describing a preliminary implementation in AdvoCATE, and with a discussion of our future plans to further advance this work. The contributions above differentiate our work from prior related research.

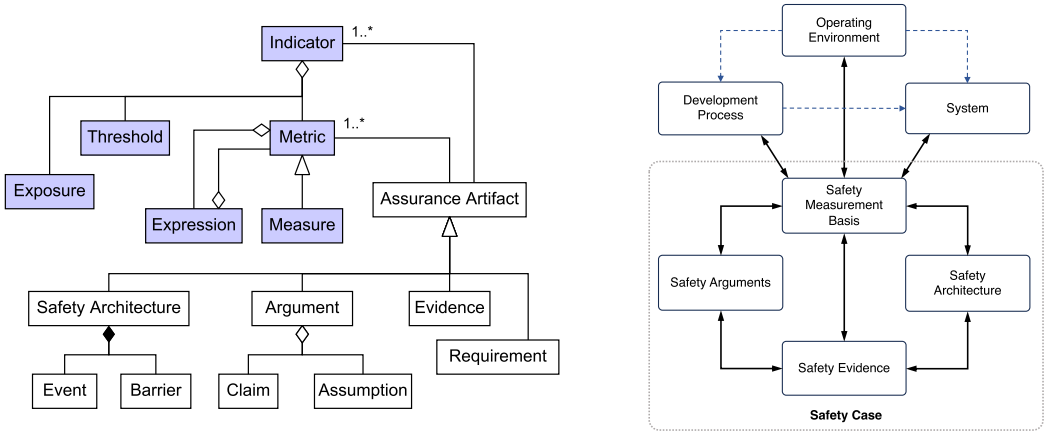


Fig. 1. Fragment of AdvoCATE safety case metamodel (on the left) extended with measurement concepts that are part of a safety measurement basis (shown on the right), which is the interface between quantities in the system, its environment, its development process, and the assurance artifacts comprising a safety case (solid arrows denote consistency relations).

2 Conceptual Background

Safety Case Metamodel. Our safety case concept [1] communicates confidence in safety through multiple viewpoints via a collection of core, interlinked *assurance artifacts*, namely: hazard, requirement, and evidence logs, a safety architecture, and an assurance rationale. Of those, the last two are particularly relevant for this paper. Assurance rationale captured as structured arguments expresses the reasoning why safety claims ought to be accepted on the basis of the evidence supplied. A safety architecture [6,8] models the mitigations (and their interrelations) to the events characterizing the operational *risk scenarios* for a system, thereby offering a system-level viewpoint on how safety risk is reduced.

Figure 1 shows a fragment of the metamodel associated with our safety case concept (as unshaded class nodes), for which we have a model-based implementation in our tool, AdvoCATE [7]. We use the *goal structuring notation* (GSN) [20] to represent structured arguments, and *bow tie diagrams* (BTDs)

to represent views of a safety architecture. Those views capture a causal chain (e.g., see Fig. 2) of *threats* (initiating events) causing a *top event* (a hazard) that can lead to *consequence events* (undesired safety effects), along with the *barriers* (mitigations) necessary to reduce the safety risk posed. Each such event chain requires a combination of *hazardous activity*, *environmental condition*, and *system state* (together representing the operating context[1]), and can admit an arbitrary number of intermediate events between the initiating threat and terminating consequence events. Each barrier is itself a system comprising underlying *controls*; thus, it can have its own associated safety architecture, giving the overall model a layered structure that can mirror the system hierarchy.

A risk assessment model underlying a safety architecture gives the formal basis to: (i) characterize the extent of risk reduction, and (ii) link safety metrics and indicators to operational safety risk (see Sect. 4). In brief, this model relates the risk of consequence events, i.e., their probability and severity, to that of the precursor events, and to the *integrity*[2] of the applicable barriers and their constituent controls. Depending on the stage of system development, we can interpret each of an event probability and barrier/control integrity both as a design target and verification goal. For the rest of this paper, we mainly consider the risk reduction contribution of barriers.

Safety Measurement. We extend the safety case metamodel in AdvoCATE with concepts for safety performance measurement (shown by the shaded class nodes in Fig. 1) as follows: we link the *indicators* to the core assurance artifacts—in particular, the event and barrier elements of a safety architecture, the claims and assumptions in arguments, to requirements, and to evidence artifacts. An *indicator* consists of a *metric* along with a *threshold*, representing the target that a metric should (or should not) reach, over a specified *exposure*, expressed either as a duration of continuous time or a specified number of occurrences of a discrete event. Indicators that have a bearing on safety can be called *safety indicators* (SIs) or *safety performance indicators* (SPIs). *Metrics* are computed values based on *measures*—directly observable parameters of the system, its environment, and its development process—and other metrics, which we represent using an expression language. Thus, they are arithmetic expressions over measured variables drawn from the most recent *mission*—which we term as a *data run*—or the missions conducted over the lifetime of the system. They can also refer to values referenced in assurance artifacts.

As shown in Fig. 1, a safety case can be seen as comprising a *dynamic* portion (indicators, metrics, and measures) and a static portion (safety arguments and safety architecture), with links associating the two. We refer to the set of interconnected indicators, metrics, and measures, along with their traceability links to the assurance artifacts of a safety case as a *safety measurement basis* (SMB). Roughly speaking, the connection between the dynamic and static por-

[1] Also known as an *operational design domain* (ODD) for systems integrating ML [14].
[2] Integrity is the probability that a barrier or control is not breached, i.e., it delivers its intended function for reducing risk in the specified operating context and scenario [8].

tions is that the indicators represent the objectively quantifiable content of the arguments and the safety architecture which, in turn, give the justification for how those indicators collectively provide safety substantiation. Put another way, we want the SMB to be *consistent* with the static portions of the safety case, especially the arguments and the safety architecture (see Sect. 5).

3 Motivating Example

We motivate this work using an aviation domain use case of autonomous aircraft taxiing [1]. This system uses a *runway centerline tracking* function comprising a classical controller coupled to a deep convolutional neural network that estimates aircraft position from optical sensor data. The functional objective is to maintain both the *cross-track error* (CTE) and the *heading error* (HE) within pre-defined bounds. CTE is the horizontal distance between the runway centerline and the aircraft body (or *roll*) axis; HE is the angle between the respective headings of the runway centerline and the roll axis. The safety objective is to avoid a lateral *runway overrun* (also known as a *runway excursion*), i.e., departing the sides of the runway.

Figure 2 shows a BTD fragment for this example (annotated to show its graphical elements and their identifiers) as a view of its wider safety architecture (not shown), which composes [8] similar such BTDs, albeit for different operating contexts, threats, top events, and consequences. Here, the operating context involves a relatively low speed (25 kn), low visibility taxi operation on a wet runway, at dusk, under no crosswind conditions. The hazard to be controlled (E_3) is a violation of the allowed lateral offset from the runway centerline, failing which a lateral runway overrun (E_4) could occur. Two (out of many) initiating causes for this hazard have been shown: a controller malfunction that steers the aircraft away from the centerline when not required (E_1); and runway centerline markings that are not visible, or are obscured (E_2).

3.1 Baseline Safety

To characterize the safety risk level of an operating scenario, we use the risk assessment model associated with the safety architecture to establish a *baseline level* of operational safety risk for the identified safety effects.

For the scenario in Fig. 2, the *initial risk level* (IRL) of the consequence event E_4 is labeled $\mathsf{4A(Medium)}$. That is, E_4 has a *medium* level of unmitigated risk, and is assigned the *risk classification category* $\mathsf{4A}$. That refers to a region of the overall risk space that has been discretized using a classical 5×5 risk matrix of categories of consequence event probability, ranging from *Frequent* (A) to *Extremely Improbable* (E), and consequence event severity, ranging from *Minimal* (5) to *Catastrophic* (1). For a definition of those categories, see [10]. A similar interpretation applies to *residual risk level* (RRL) which, for E_4, is shown as $\mathsf{4D(Low)}$, representing the risk remaining after mitigation using the indicated barriers and the associated controls. Specifically, B_1: *Runtime Monitoring*, B_2:

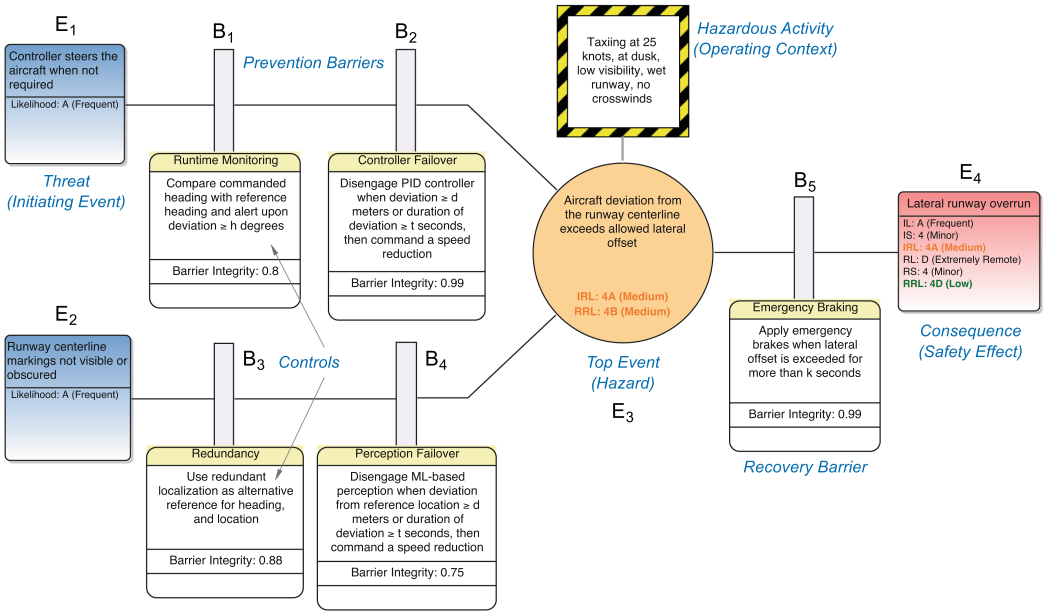


Fig. 2. Annotated BTD fragment for an autonomous taxiing capability, showing how a lateral runway overrun is mitigated under specific initiating events leading to centerline tracking violation.

Controller Failover, B_3: *Redundancy*, and B_4: *Perception Failover*, serve as prevention barriers for exceeding the allowed CTE, while B_5: *Emergency Braking* is a recovery barrier invoked after the top event occurs.

For aeronautical applications, civil aviation regulations and the associated certification or approval processes generally establish what constitutes *acceptable* and *approved* baseline risk levels respectively. The two can be the same (though they need not be) and, typically, are given in terms of a so-called *target level of safety* (TLOS), which specifies the (maximum acceptable) probability of the undesired safety effect per unit of operational exposure, e.g., 10^{-6} lateral runway excursions per taxi operation. How TLOS is established and approved is out of scope for this paper[3]; as such, in Fig. 2, either of the values of the IRL, $\mathsf{4A(Medium)}$, or the RRL, $\mathsf{4D(Low)}$, may plausibly meet the TLOS, and therefore could be an approved baseline level of safety risk. For the purposes of this example, we assume that the RRL shown is the approved baseline that meets the TLOS. Once a system is deployed, note that the RRL for an event is, in fact, *dynamic*, i.e., as a sequence of values starting from the approved baseline, it represents how the risk of that event evolves over the system lifetime (also see Fig. 3).

[3] Interested readers may refer to [3].

3.2 Practical Drift

Some barriers or controls in the safety architecture of a system may be relaxed in operation to improve the performance of system services and/or to make local optimizations that address the operating context. In our running example, for instance, to increase runway throughput whilst operating on large runways in better environmental conditions (e.g., clear weather, and dry runway surface), the time an aircraft spends on a runway could be reduced. For that purpose, suppose that disengagement of the perception function or the controller is delayed (see Fig. 2), or that more permissive CTE bounds are admitted. In those cases, the system may enter certain states that would have been prohibited otherwise. In particular, such states represent violations of the barrier/control requirements that were stated as claims in the pre-deployment safety case.

However, when there is improved system performance without observed safety consequences or mishaps, those states are not perceived as violations that increase residual risk. This can lead to misplaced assurance in operational safety when the system as operated deviates from its safety case. Practical drift can then emerge when multiple safety mitigations may be progressively loosened, and continued, incident-free system operations under such changes obscure the increase in operational safety risk. It is important to emphasize that relaxing mitigations to improve performance represents an operational tradeoff rather than a deliberate attempt to subvert safety. An analogy, for example, is highway driving at the speed of traffic that exceeds the posted speed limits—a practice that is not always unsafe, but poses higher risk in general.

4 Framework

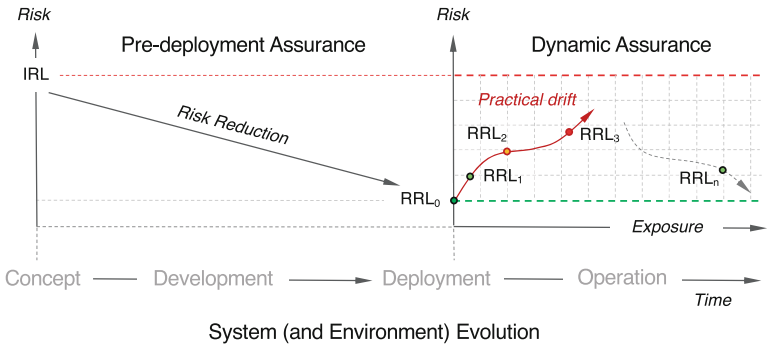


Fig. 3. Pre-deployment assurance gives justified confidence in the reduction of an initial risk level (IRL) of the safety effects in a system concept, through system development, to a baseline residual risk level (RRL_0) that meets the TLOS at deployment. Dynamic assurance provides confidence in operation that the approved baseline is maintained, by identifying and managing practical drift.

Dynamic assurance within a safety case approach gives a proactive means to assess and contain practical drift through continued assurance that the operational safety risk level for the system is aligned with its approved baseline (see Fig. 3). A framework that enables this must at least: (i) characterize how operational safety risk levels have changed; (ii) determine which mitigations, if any, may be legitimately relaxed without safety deteriorating; and (iii) identify the necessary modifications to both the system and its safety case, so that the two are mutually consistent during system operation. Next, we discuss how relating safety performance measurement to the safety architecture in a safety case, in addition to its arguments, gives the necessary elements and technical foundations for the first of the preceding three requirements—the main focus in this paper. The examples presented next are meant to be illustrative and not comprehensive.

4.1 Defining Safety Metrics and Indicators

A safety architecture and its associated risk assessment model [8] give a basis to allocate safety targets to the safety functions, and subsequently confirm them (analytically and empirically). TLOS is a system-level safety target always assigned to consequence event probability. Decomposing and allocating the TLOS across the elements of the safety architecture gives the safety integrity targets for barriers and controls, along with precursor event probabilities that we interpret as *scenario-specific safety targets*. Relating safety targets to safety performance measurement in general, and safety indicators (SIs) in particular, facilitates tracking and confirming that the mitigations are performing in operation as intended. One way to embed TLOS into an SI is by simply converting the corresponding probability value into an event frequency threshold applied to an appropriate safety metric used during development or in operation. In this section we focus on the operational safety metrics, addressing the metrics used during development in Sect. 4.2.

TLOS and the corresponding SIs can be *generic*, i.e., apply to all relevant operating contexts of a safety architecture, or *scenario-specific*, i.e., applicable to a particular operating context. For instance, let the TLOS for lateral runway overrun under all relevant operating conditions of the example system be 10^{-6} per taxi operation. We can then define a corresponding generic SI, $\mathcal{I}_{\texttt{LRE}}$: $\texttt{opLatRwyEx} \leq 1$ in 10^6 taxi operations, where `opLatRwyEx` is an operational safety metric[4] for the *number of lateral runway overrun events in operation*, whose threshold value is 1, measured over an exposure of 10^6 taxi operations. Another commonly used unit of exposure is *flight hours* [11], and the SI can be given equivalently as $\mathcal{I}_{\texttt{LRE}} : \texttt{opLatRwyEx} \leq 1$ in $10^6 \times t$ flight hours, where t is the average time in flight hours of a taxi operation.

Scenario-specific SI definition proceeds in the same way, but is applied to specific operating contexts after first decomposing and allocating the TLOS of a consequence event to its scenario-specific instance. For example, if 10% of all

[4] Henceforth, identifiers with the prefix '`dev`' refer to metrics used during system development, and the prefix '`op`' indicates an operational safety metric.

taxi operations occur under the operating context of Fig. 2, then we can modify the exposure of $\mathcal{I}_{\texttt{LRE}}$ to 10^5 taxi operations to get the scenario-specific SI for the consequence event E_4.

Similarly, we can define generic and scenario-specific SIs for the remaining safety architecture elements by converting the associated event probability and barrier integrity values as applicable. Moreover, recalling that a barrier can have its own safety architecture (Sect. 2), we can iteratively define SIs for the lower layers of a system hierarchy. Thus, in Fig. 2, we can define the scenario-specific SI for the barrier B_4: *Perception Failover* as $\mathcal{I}_{\texttt{PFO}}$: $\texttt{opPcpDisEngF} \leq y$ in n taxi operations. In Sect. 4.4, we illustrate one approach to instantiate y and n.

Here, `opPcpDisEngF` is a metric related to the integrity of B_4 (itself a metric) that counts the *number of failed disengagements of ML-based perception in operation*; its threshold value is y, to be measured over an exposure of n taxi operations conducted in the stated operating context for the specified scenario. This metric relies upon a precise definition of a *failed disengagement* (not given here), which may itself be given in terms of other metrics, e.g., those associated with its *functional deviations* (i.e., violation of the requirements for the barrier, its constituent controls, or their verification), and its *failure modes* (of the physical systems to which the barrier function is allocated). Additional operational safety metrics related to barrier integrity include `opTxLowVisW`, counting the *number of taxi operations conducted at dusk under low visibility, no crosswind, and wet runway conditions* (i.e., the operating context of Fig. 2), from which we may infer the *number of successful disengagements of ML-based perception* as the metric $\texttt{opPcpDisEngS} = \texttt{opTxLowVisW} - \texttt{opPcpDisEngF}$.

4.2 Updating and Revising the Operational Risk Assessment

A pre-deployment safety case represents what (we believe) a system design achieves at deployment, and will continue to achieve in operation. Some of the metrics and SIs applicable during system development constitute measurement evidence verifying safety performance, e.g., during pre-deployment system testing or flight testing. Thus, by associating those metrics and SIs with the safety architecture, we get the *prior* values of event probability and barrier integrity. For the scenario and operating context of Fig. 2, some of the metrics used during system development for the barrier B_4 are: `devTxLowVisW`: the *number of tests for* $\mathsf{B}_4 = t$ (say); `devPcpDisEngS`: the *number of successful disengagements of ML-based perception* $= s$; and `devPcpDisEngF`: the *number of failed disengagements of ML-based perception* $= (t - s) = f$.

If the test campaign during system development is designed as a Bernoulli process [16] then we can model the sequence of test results as a binomial distribution, $\texttt{Binom}\,(\chi : \eta, \theta)$, whose parameters are χ: the number of successes, η: the number of independent trials, and θ: the probability of success in each trial. Hence, we can assign the values of the metrics `devPcpDisEngS` and `devTxLowVisW`, respectively, to the first two parameters as $\chi \coloneqq s$, and $\eta \coloneqq t$. Let $\theta \coloneqq p$, the unknown (fixed) probability that each test produces a successful disengagement. We can model p as the conjugate prior beta distribution,

$\pi(p) \sim \texttt{Beta}\,(\alpha, \beta)$. The *hyperparameters* (i.e., the parameters of the prior distribution) represent our prior knowledge of the number of successful and failed tests during development. Hence we assign to them the values of the metrics `devPcpDisEngS` and `devPcpDisEngF`, respectively, as $\alpha := s$, and $\beta := f$. The beta distribution mean, $\mu_p = {}^{s}/_{t}$, gives a point estimate of the prior barrier integrity, and its variance, $\sigma_p^2 = {}^{sf}/_{t^2(t+1)}$, gives the uncertainty in that estimate.

In operation, safety performance measurement yields a sequence of observations of the state of the safety system. We can transform this data into a *likelihood function*, i.e., a joint probability of the observations given as a function of the parameters of a model of the underlying data generation process. In our example, a binomial probability density function (PDF) is a reasonable *initial* model (i) assuming that the pre-deployment safety case provides the argument and evidence that testing is representative of actual operations (as would likely be necessary), and (ii) since a binomial distribution models the sequence of test results. Thus, supposing that over n taxi operations conducted in the operating context of Fig. 2, there were x failures to disengage ML-based perception on demand. We now have the operational safety metrics $\texttt{opTxLowVisW} = n$, and $\texttt{opPcpDisEngF} = x$, so that $\texttt{opPcpDisEngS} = (n - x) = y$, and the likelihood function is $\mathcal{L}\,(p|n, y) = \binom{n}{y} p^y (1-p)^x$. As before, p is the unknown probability of a successful disengagement of ML-based perception on a random demand, representing a surrogate measure of barrier integrity.

Bayesian inference gives the formal procedure to update the priors into *posterior* values of barrier integrity (and event probability), which represent what the operational system *currently* achieves. Thus, for our running example, the posterior integrity for B_4 is given by (the proportional form of) Bayes' theorem as $\pi(p|y) \propto \mathcal{L}\,(p|n, y) \times \pi(p)$. Since the beta prior and the binomial likelihood are a conjugate pair, the posterior has a closed form solution, $\pi(p|y) \sim \texttt{Beta}\,(s + y, (t - s) + x)$. The distribution mean, $\mu_{p|y} = {}^{(s+y)}/_{(t+n)}$, is the updated point estimate of barrier integrity. To get a revised assessment of the operational safety risk level for the system, we propagate the posterior barrier integrity through the risk assessment model underlying the safety architecture.

4.3 Characterizing the Change to Safety Risk

We use *risk ratio* (RR), a metric of relative risk, to quantify the change in operational safety risk. In operation, the RR for a consequence event is the ratio of its current estimated probability of occurrence and the approved baseline. More generally, we will (re)compute the RR for any event of interest in the safety architecture, typically after the operational risk assessment has been revised (as in Sect. 4.2) as the ratio of its updated (i.e., prior or posterior, as appropriate) probability to its (scenario-specific) safety target. Denoting the RR for event E_i by $\texttt{RR}(\mathsf{E}_i)$, $\texttt{RR}(\mathsf{E}_i) > 1$ indicates an increase in the safety risk of E_i. Similarly, $\texttt{RR}(\mathsf{E}_i) < 1$ indicates a decrease, while $\texttt{RR}(\mathsf{E}_i) = 1$ indicates no change. By itself, RR reflects how effective the safety architecture is in reducing the risk of the

identified safety effects.[5] By considering the trend of RR over time, we can construct a powerful SI of practical drift, e.g., by fitting a linear trend line to a temporally ordered sequence of RR values computed over some pre-determined exposure, the sign and magnitude of the slope indicate, respectively, the direction and the rate of the change in safety risk.

4.4 Numerical Examples

We now give some numerical examples to concretize the preceding discussion.

Example 1 (Prior Barrier Integrity). During the development of our running example system and its pre-deployment safety case, assume we have a total of $\texttt{devTxLowVisW} = 32$ flight tests in which there are $\texttt{devPcpDisEngF} = 8$ failing tests for the *Perception Failover* barrier. Thus a prior distribution for its integrity is $\pi(p) \sim \texttt{Beta}(24, 8)$, whose mean is $\mu_p = 0.75$, and variance is $\sigma_p^2 = 0.0057$. The mean gives a point prior value of barrier integrity which we show in the corresponding node in the BTD of Fig. 2.

Example 2 (Scenario-specific Barrier Safety Indicator). Recall that a scenario-specific SI for the *Perception Failover* barrier is $\mathcal{I}_{\texttt{PFO}}$: $\texttt{opPcpDisEngF} \leq y$ in n taxi operations (Sect. 4.1). As before, opPcpDisEngF measures the number of failed disengagements of ML-based perception in operation. To determine a suitable exposure n and threshold y, consider that a conservative range of values for p that would provide the same, or better, risk reduction performance as its prior mean is the closed interval $[\mu_p + \sigma_p, 1] = [0.8254, 1]$. In other words, observing 8 or more successful disengagements or, equivalently, 2 or fewer failed disengagements on demand of ML-based perception over at least 10 taxi operations conducted in the specified operating context would validate the safety performance of the barrier. Thus, here, $n = 10$ and $y = 2$.

Example 3 (Likelihood of Data and Posterior Integrity). After system deployment, suppose that to improve runway utilization, the control in B_4 (see Fig. 2) is relaxed such that ML-based perception is disengaged after a larger distance (or duration) of position deviation than what was specified in the safety architecture. The metric that records the number of failed disengagements in operation, opPcpDisEngF (Sect. 4.2), includes violations of the barrier requirement as initially specified, which itself includes violations of the barrier requirement after operational modification. That is, the operational safety metric *should not be modified* even though the barrier function has been operationally changed. Supposing $\texttt{opPcpDisEngF} = 4$ violations have been observed over $\texttt{opTxLowVisW} = 10$ taxi operations. Given this data, the likelihood function is $\mathcal{L}(p|6) = \binom{10}{4} p^6 (1-p)^4$, and the posterior distribution is $\pi(p|6) \sim \texttt{Beta}(30, 12)$, whose mean is $\mu_{p|6} = 0.7143$ and variance is $\sigma_{p|6}^2 = 0.0047$.

[5] RR has also been used as a development safety metric, e.g., in designing aircraft collision avoidance systems [9].

Example 4 (Operational Safety Risk Update). We assume prior data is available (from characterizing the ODD [14] for the autonomous taxiing function) on how often runway markings are obscured during taxiing due to runway surface and weather conditions. Hence we can give a prior distribution, say $\pi(\mathsf{E}_2) \sim \texttt{Beta}(10, 190)$, whose mean is the prior point estimate $\texttt{Pr}(\mathsf{E}_2) = 0.05$. Similarly, let $\texttt{Pr}(\mathsf{E}_1) = 0.05$. Given these priors and the barrier integrity values as in Fig. 2, the prior probability of the consequence event is $\texttt{Pr}(\mathsf{E}_4) = 1.5998 \times 10^{-5}$ corresponding to an RRL of $\mathsf{4D}(\mathsf{Low})$. We recall from Example 3 that 4 barrier violations were observed in 10 taxi operations. Hence E_2 must have occurred on $z = 4$ occasions for B_4 to have been invoked and have failed on demand. Thus, we may reasonably model this event as a Bernoulli process with a binomial PDF as the likelihood function for the observed data. Thus, the posterior distribution over E_2 is $\pi(\mathsf{E}_2|z) \sim \texttt{Beta}(14, 196)$ so that $\mu_{\mathsf{E}_2|z} = 0.0667$ is the point posterior for $\texttt{Pr}(\mathsf{E}_2)$. Propagating both the posteriors for E_2 and B_4 through the risk assessment model of the safety architecture [8], we get the updated prior $\texttt{Pr}(\mathsf{E}_4) = 2.386 \times 10^{-5}$ for the consequence event. The corresponding RRL remains unchanged suggesting that the operational modification to B_4 may be acceptably safe.

Example 5 (Safety Risk Level Change and Practical Drift). The risk ratio for the consequence event E_4 given the change to barrier B_4 (as in Example 3) is $\texttt{RR}(\mathsf{E}_4) = {}^{2.386}/_{1.5998} \approx 1.49$. Thus, despite an unchanged risk level (see Example 4), the RR indicates increasing safety risk. Now, further suppose that to improve runway utilization, a greater deviation in CTE from the stated bounds is operationally admitted (see the top event E_3 in Fig. 2). Consequently barrier B_5 needs to be relaxed to be invoked after a longer duration than specified (see Fig. 2). Suppose that the posterior integrity computed from operational safety metrics (omitted here due to space constraints) is $\texttt{Pr}(\mathsf{B}_5) \approx 0.96$. In this case, the revised prior for E_4 is $\texttt{Pr}(\mathsf{E}_4) \approx 2.4 \times 10^{-4}$, the revised RRL is $\mathsf{4C}(\mathsf{Medium})$, and $\texttt{RR}(\mathsf{E}_4) \approx 10$. The updated RRL now violates the TLOS even if no safety effects may have been observed. Moreover, the modifications to the barriers B_4 and B_5 are at least *an order of magnitude more likely* to result in a lateral runway overrun, indicating an appreciable increase in safety risk relative to the approved baseline, and suggests practical drift.

5 Towards Formal Foundations

As mentioned earlier (Sect. 2), we want to formalize a notion of *consistency* between the static portion of the safety case (i.e., its assurance artifacts, see Fig. 1) and the collection of indicators that constitute the SMB. The safety metrics and indicators represent the objectively quantifiable content referenced in the arguments and the safety architecture, which in turn provide the justification for how the metrics and indicators collectively provide safety substantiation. Although operational safety measurement entails updating and revising the risk

assessment (Sect. 4), changing the SMB may not be necessary. However, in situations where replacing, modifying, or adding metrics and indicators is required—e.g., to reflect new observable phenomena in the environment—the SMB will change and so would the associated assurance artifacts to retain consistency. Note that currently we are not considering changes that would entail modification of the safety architecture (e.g., replacing a barrier). Hence we exclude that from our notion of consistency for now and focus on consistency with arguments.

We can achieve this consistency if the argument structure reflects the risk reduction rationale implicit in the safety architecture. That is, the form of the argument structure proceeds from all terminating consequence events in the safety architecture, working recursively backwards (i.e., leftwards) to all initiating (leftmost) threat events. Thus, each level of the argument has the following form: all consequence events are acceptably mitigated (i.e., the residual risk level meets the allocated TLOS), *which is supported by the argument that*: all their identified precursor events (causes) are acceptably mitigated, *which is supported by the argument that*: (a) all applicable barriers are operational and effective, and (b) all causes have the stated probability of occurrence. In a GSN representation of this argument, the leaves are *solution nodes* [20] that have the following *evidence assertion*: the initiating threat has the stated (assumed) probability.

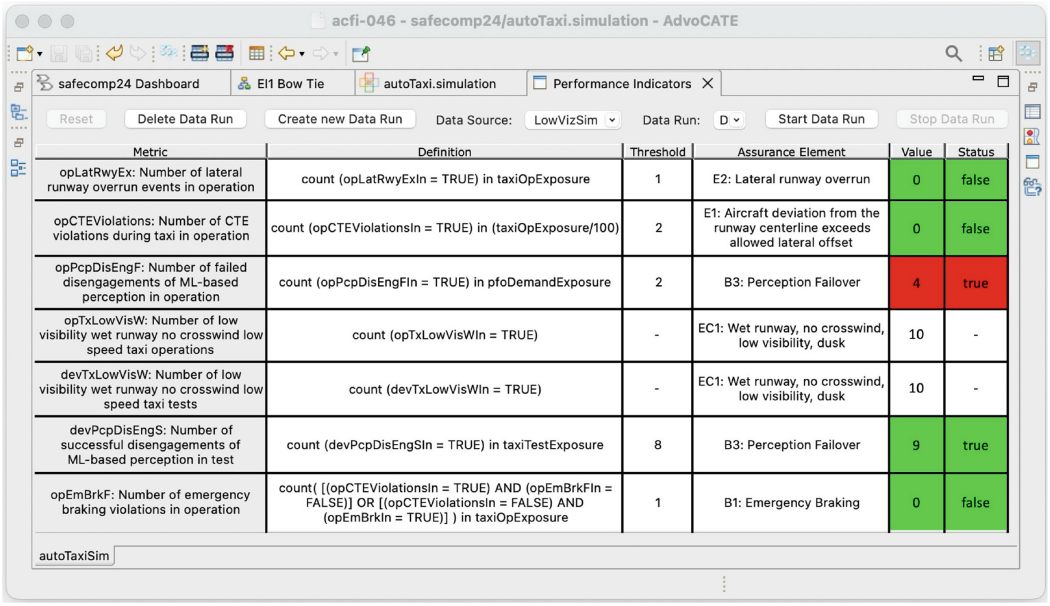
Thus, the overall argument states that if the barriers are effective and operational, and the events have the assumed probabilities, then the consequences have acceptable risk levels. Indicators map into the corresponding claims of barrier effectiveness and event probabilities, serving to monitor that those values are within the required limits.

Now we briefly outline how to place this consistency on a more rigorous basis. Let **Arg** and **SMB** represent the sets of well-formed arguments and SMBs, respectively, and define mappings $F : \mathbf{Arg} \to \mathbf{SMB}$ and $G : \mathbf{SMB} \to \mathbf{Arg}$, such that F extracts the associated indicators from an argument, and G embeds an SMB into a skeleton argument of the form outlined above. Then we require that $F; G \leq I$ and $G; F = I$ (where I is the identity mapping), where arguments are ordered by *refinement*. The first inequality ensures that the argument contains the necessary rationale for the SMB, with the refinement allowing that the argument can contain additional reasoning; the second ensures that all quantifiable components of the argument are represented in the SMB.

6 Concluding Remarks

We have a preliminary implementation of the SMB in AdvoCATE that currently supports the following functionality: real-time import of data (i.e., measures) from multiple data sources (simulations or feeds from external sensors); computation of derived metrics and indicators over multiple data runs; and tracing to assurance artifacts (events and barriers in the safety architecture, and goals and assumptions in the safety arguments). We display indicators and the associated assurance artifacts in a dynamically updated table (Fig. 4 shows an example) that highlights when the conditions on the indicator thresholds have been met

(in green) or have not been met (in red). A dashboard (not shown) allows selection between the various metrics of the SMB with charts displaying real-time updates of their values as well as other dynamically updated risk status, such as hazards ordered by risk level, and barriers ordered by integrity.



Metric	Definition	Threshold	Assurance Element	Value	Status
opLatRwyEx: Number of lateral runway overrun events in operation	count (opLatRwyExIn = TRUE) in taxiOpExposure	1	E2: Lateral runway overrun	0	false
opCTEViolations: Number of CTE violations during taxi in operation	count (opCTEViolationsIn = TRUE) in (taxiOpExposure/100)	2	E1: Aircraft deviation from the runway centerline exceeds allowed lateral offset	0	false
opPcpDisEngF: Number of failed disengagements of ML-based perception in operation	count (opPcpDisEngFIn = TRUE) in pfoDemandExposure	2	B3: Perception Failover	4	true
opTxLowVisW: Number of low visibility wet runway no crosswind low speed taxi operations	count (opTxLowVisWIn = TRUE)	-	EC1: Wet runway, no crosswind, low visibility, dusk	10	-
devTxLowVisW: Number of low visibility wet runway no crosswind low speed taxi tests	count (devTxLowVisWIn = TRUE)	-	EC1: Wet runway, no crosswind, low visibility, dusk	10	-
devPcpDisEngS: Number of successful disengagements of ML-based perception in test	count (devPcpDisEngSIn = TRUE) in taxiTestExposure	8	B3: Perception Failover	9	true
opEmBrkF: Number of emergency braking violations in operation	count([(opCTEViolationsIn = TRUE) AND (opEmBrkFIn = FALSE)] OR [(opCTEViolationsIn = FALSE) AND (opEmBrkIn = TRUE)]) in taxiOpExposure	1	B1: Emergency Braking	0	false

Fig. 4. AdvoCATE screenshot: Table of safety indicators for the example system in Sect. 3.

The goal of managing practical drift has mainly informed our choice of safety metrics and indicators. We plan to leverage the *Goal Question Metrics* (GQM) approach [2] to define additional metrics suitable for other dynamic assurance goals, e.g., improving functional performance whilst maintaining safety.

A binomial likelihood may be only initially appropriate for certain kinds of measurement data. Indeed, as more data is gathered, the preconditions for using a binomial PDF need to be reconfirmed. As such, it may be necessary to use other PDFs for the likelihood of the data, along with numerical methods for Bayesian inference. Our choice of beta priors is motivated, in part, by computational convenience, its flexibility to approximate a variety of distributions, and the domain-specific interpretation of the distribution parameters in different safety metrics. Although we represent the uncertainty in barrier integrity and event probability by specifying their distributions in the theoretical framework, our prototype implementation currently represents and propagates their point values (i.e., the distribution means) for both the pre-deployment risk assessment, and the revisions of the operational risk assessment. We plan to refine this approach by also propagating the uncertainties through the risk assessment model so as to quantify the corresponding uncertainty in the residual risk of the safety effects of interest. By so doing, we aim to ground the quantification of assurance in safety measurement.

Since TLOS is typically assigned to rare events, legitimate concerns can arise about the credibility of using quantitative methods as in this paper. Though we have yet to explore how *conservative Bayesian inference* [19] could be used in our approach, relative risk metrics such as risk ratio (RR) are a step towards circumventing those concerns.

Practical drift is distinct from *operating environment drift* in that the former results from changes within the system boundary, whereas the latter occurs outside that boundary. We reflect the assumptions about the operating environment in the pre-deployment safety case, for example, as the prior probabilities (conditional on the operating context) associated with the threat events. We can reflect environment drift via the posterior distributions of the corresponding event probabilities updated by operational safety metrics associated with the respective events (see Sect. 4.2, and Example 4). We additionally distinguish *runtime risk assessment* [1], from the update and revision of operational risk as described in this paper: the former occurs during the shorter time span of a mission (e.g., during a taxi operation), whereas the latter occurs over longer time intervals, between missions, and through the lifecycle of the system (e.g., over multiple taxiing operations, possibly involving an aircraft fleet).

The numerical examples (Sect. 4.4) have described a scenario-specific application of our approach, where the event probabilities and barrier integrities are *conditional* on the operating context. For a system-level characterization of how operational risk changes, we must consider the *marginal* probabilities and integrities in the overall safety architecture that composes different risk scenarios. However, we have not considered it in this paper, and it is one avenue for future work.

To further develop our proposed dynamic assurance framework we aim to explore how by thresholding, ranking, and comparing RR under changes made to individual mitigations or their combinations, we may infer: (i) which mitigations may be optimized for system performance whilst maintaining safety (possibly necessitating a change to the safety architecture itself); and, in turn, (ii) which system and safety case changes may be necessary. Some changes may be automated while will induce *tasks* requiring manual attention [5]. Additionally, we aim to define a tool-supported methodology on top of the main components of the framework. This will involve defining and formalizing the methods and procedures to decompose and allocate safety targets, derive safety indicators, and close the safety assurance loop, i.e., maintain consistency of the arguments with the SMB) through targeted changes to the system and its safety case.

Observations of system operations constitute one specific form of evidence that we can use to reason about system safety. We seek to systematize this through a notion of *evidence requirement* that will also cover *static* data. We are also extending the metrics expression language to express trends, although work remains to integrate it into our methodology and to relate it to the concept of *safety objective*. A need to update the SMB, e.g., modify indicators and possibly their thresholds, accompanies operational safety measurement. We aim to better understand the principles that underlie those modifications and, subsequently,

implement the corresponding tool features. However, practically deploying this framework will necessitate harmonizing with existing safety management system (SMS) [10] infrastructure, whilst carefully considering the roles of different stakeholders in safety performance monitoring, measurement, and assurance.

Acknowledgments. This work was performed under Contract No. 80ARC020D0010 with the National Aeronautics and Space Administration (NASA), with support from the System-wide Safety project, under the Airspace Operations and Safety Program of the NASA Aeronautics Research Mission Directorate.

References

1. Asaadi, E., Denney, E., Menzies, J., Pai, G., Petroff, D.: Dynamic assurance cases: a pathway to trusted autonomy. IEEE Comput. **53**(12), 35–46 (2020)
2. Basili, V., Caldiera, G., Rombach, D.: Goal Question Metric Paradigm, pp. 528–532. Encyclopedia of Software Engineering, John Wiley & Sons, Inc., 2nd edn. (1994)
3. Busch, A.C.: Methodology for Establishing a Target Level of Safety. Technical Report DOT/FAA/CT-TN85/36, US DOT, FAA Technical Center (1985)
4. Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M.U., Habli, I., Kelly, T.: Engineering trustworthy self-adaptive software with dynamic assurance cases. IEEE Trans. Softw. Eng. **44**(11), 1039–1069 (2018)
5. Denney, E., Habli, I., Pai, G.: Dynamic safety cases for through-life safety assurance. In: 37th International Conference on Software Engineering - Vol. 2, pp. 587–590. (2015)
6. Denney, E., Johnson, M., Pai, G.: Towards a rigorous basis for specific operations risk assessment of UAS. In: 37th IEEE/AIAA Digital Avionics Systems Conference (2018)
7. Denney, E., Pai, G.: Tool Support for Assurance Case Development. J. Autom. Softw. Eng. **25**(3), 435–499 (2018)
8. Denney, E., Pai, G., Whiteside, I.: The role of safety architectures in aviation safety cases. Reliab. Eng. Syst. Saf. **191**, 106502 (2019)
9. Edwards, M., Mackay, J.: Determining required surveillance performance for unmanned aircraft sense and avoid. In: 17th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference. AIAA 2017-4385 (2017)
10. FAA Air Traffic Organization: Safety Management System Manual (2022)
11. US Department of Transportation, FAA: Safety Risk Management Policy. Order 8040.4C (2023)
12. Hawkins, R., Ryan Conmy, P.: Identifying run-time monitoring requirements for autonomous systems through the analysis of safety arguments. In: Guiochet, J., Tonetta, S., Bitsch, F. (eds.) Computer Safety, Reliability, and Security. SAFECOMP 2023. LNCS, vol. 14181. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-40923-3_2

13. International Civil Aviation Organization (ICAO): Safety Management Manual (Doc 9859), 4 edn. (2018)
14. Kaakai, F., Adibhatla, S., Pai, G., Escorihuela, E.: Data-centric operational design domain characterization for machine learning-based aeronautical products. In: Guiochet, J., Tonetta, S., Bitsch, F. (eds.) Computer Safety, Reliability, and Security. SAFECOMP 2023. LNCS, vol. 14181. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-40923-3_17
15. Koopman, P.: How Safe is Safe Enough? Measuring and Predicting Autonomous Vehicle Safety. 1st edn. (2022)
16. Ladkin, P.: Evaluating software execution as a Bernoulli process. Saf. Crit. Syst. eJournal **1**(2) (2022)
17. Reich, J., Trapp, M.: SINADRA: towards a framework for Assurable situation-aware dynamic risk assessment of autonomous vehicles. In: 16th European Dependable Computing Conference (EDCC), pp. 47–50 (2020)
18. Schleiss, P., Carella, F., Kurzidem, I.: Towards continuous safety assurance for autonomous systems. In: 6th International Conference on System Reliability and Safety (ICSRS 2022), pp. 457–462 (2022)
19. Strigini, L.: Trustworthy quantitative arguments for the safety of AVs: challenges and some modest proposals. In: 1st IFIP Workshop on Intelligent Vehicle Dependability and Security (IVDS) (2021)
20. The Assurance Case Working Group (ACWG): Goal Structuring Notation Community Standard Version 3. SCSC-141C (2021)
21. Trapp, M., Weiss, G.: Towards dynamic safety management for autonomous systems. In: 27th Safety-Critical Systems Symposium (SSS), pp. 193–204 (2019)