What Makes It Testable? Conceptual Model for Safety Quantification

Edward Schwalb[1], Bernhard Bieder[2], Daniel Wiesenhütter[2]

Abstract

We propose a simple conceptual scenario based model to formalize and quantify safety validation. We introduce a failover model, whereby overall failure rates are reduced exponentially in the number of stages added, but the validation effort needed only increases linearly in the number of stages. We address validation of rare situations using synthetic data. We introduce multi-scenario and multi-model training and testing to quantify operational safety under poor conditions, failure and emergencies. Finally, we discuss regression introduced by a learning process, and hypothesize that safety-regression can be avoided in a broad class of system revisions.

1.0 Introduction

A number of organizations are conducting road tests for autonomous cars. Some of those tests and datasets, e.g., Waymo and Tesla, cover hundreds of thousands of miles. The effort level, cost and duration required for end-to-end testing renders validation and verification of (hardware or software) changes prohibitively expensive. The question we are asking is, how do we know when sufficient testing was done? How can we quantify safety?

In a recent position paper, Sanjit et al [1] describe many of the key challenges of *Verified AI*. Among those challenges included modeling of the environment and the systems that learn, quantifying metrics about the training data, lack of specification at various levels and models for run-time quantitative verification. In contrast, Tim Menzies and Charles Pecheur describe the approach NASA took for verification and validation of the Remote Agent Experiment (RAX) system which ran the deep-space probe without help from mission control [2]. The NASA effort clearly demonstrates that there are enough readily-available methods that enable V&V of AI systems. The control logic of RAX, however, was developed against handcrafted specifications and using handcrafted rules rather than by automated learning algorithms. The challenges facing us with autonomous vehicles are such that neither the specification nor the logic are handcrafted.

[1] MSC Software
[2] Vires

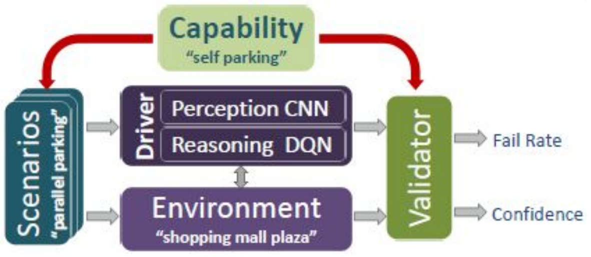

Figure 1: Basic concepts supporting collaboration.

Rather than present yet another learning model, algorithm, system architecture or framework, in this workshop paper we would like to present a simple conceptual model which enables quantifying safety, and further simplifies and accelerates collaboration among disparate teams.

Consider the conceptual model depicted in Figure 1. We introduce the notion of a capability **C**, such as"self parking" or "detecting stop sign". We define the notion of a scenario **S**, which specifies the content of a dataset originating from a simulation or a live test drive; see *www.OpenScenario.org* for a standardization approach to scenario specification. An environment **E** is always implied by a scenario **S**. Similarly, validation is performed by rules prescribed by the capability; not the scenario. The driving model M is conceptually comprised of a perception Convolutional Neural Network (CNN) and a reasoning Deep Q Network (DQN) component.

2.0 Scenario Based Failover Model

Our goal is to quantify safety. As an example, given **C=***"stop sign detection"* and **S** as the stop sign scenarios depicted in Figure 2, determine the rate of failing C, i.e. "stop sign detection", with a confidence of 95%. The notion of scenarios enables such quantification. Estimation of confidence further requires determining distribution of such scenarios based on real data such as the KITTI dataset [3], and others.

Figure 2: Example stop sign scenarios.

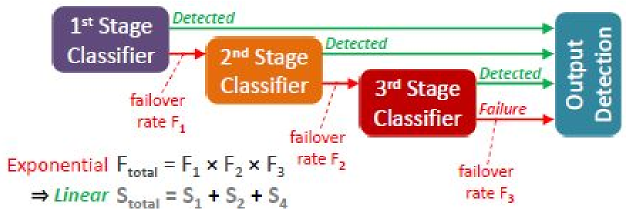

Figure 3: Failover model for a simple cascading classifier.

Definition 1: Given a capability **C** and a set ***S*** of scenarios:
a) The model **M passes C for s**∈S, denoted **M⇒Cs**, if the output of the simulation satisfies the requirement defined by capability **C** for **s**; otherwise, we say that **M fails C for s,** denoted **M⇏Cs.**
b) The failover rate of **M** on **C** for **s**, denoted ***failoverRate*(M,Cs)**, is the count of scenarios **s**∈S such that **M⇏Cs,** divided by the total number of scenarios |S|, namely |{ **s**∈S | **M⇏Cs** }| / |**S**|.
c) The confidence **P** of a failover rate **f** on **S** for capability **C**, denoted **Pc(S,f)** is an estimate of confidence that **f<failoverRate(M,Cs)**; the specific method for computing the confidence is not in scope for this paper.
d) A set of scenarios required to validate a model, for a specific confidence measurement heuristic, denoted ***scenarios*(C,M,f,p)**, is *any* set of scenarios satisfying **p < Pc(scenarios(M,f,p), f).** Note that many sets of scenarios may satisfy this condition.

Consider the simple cascading classifier depicted in Fig 3. Denote M(s) as the classification by M for a scenario **s,** and let M_i denote the model for stage i. The Fig 3 design is $M(s) = not(not(M_1(s))$ and $not(M_2(s))$ and $not(M_3(s)))$.

This means that, if, for example, the failover rate for M_i is $O(10^{-3})$, namely one of 1000 scenarios fails, then the overall fail rate of M would be proportional to $O(10^{-9})$ and thus require $O(10^9)$ scenarios to quantify the failure rate.

Definition 1 applies to the Fig 3 staged classifier design for C=”detect stop sign” as follows:
a) If (**M_1⇒Cs**) or (**M_2⇒Cs**) or (**M_3⇒Cs**) then **M⇒Cs**.
b) **failoverRate(M,Cs) = $\prod_i$ failoverRate(M_i,Cs)**.

To derive the overall failure rate for M, first quantify failover in each of the stages M_i separately, and then multiply the failover rates of the individual stages, namely $F_{total} = F_1 \times F_2 \times F_3$. In contrast, it is sufficient to validate each stage separately. Thus, the total number of scenarios required to quantify failure of M is proportional to the sum, namely $S_{total} = S_1+S_2+S_3$. Consequently, for a given p and f,

$$|scenarios(C,M,f,p)| \propto |\textstyle\sum_i scenarios(C,M_i,f,p)|.$$

Thus, as depicted in Fig 3, whereas the overall failure rate F_{total} decreases exponentially, the number of scenarios required for validation S_{total} increases linearly. As an example, for a 3-stage cascading classifier design with overall failover rate of $O(10^{-9})$, we can reduce the number of testing scenarios from $O(10^9)$ to $O(10^3)$, namely reduction of testing complexity by **6** orders of magnitude.

The more general case is a classifier DAG depicted in Figure 4. Each path along the arrows represents a cascade and each scenario propagates along all paths.

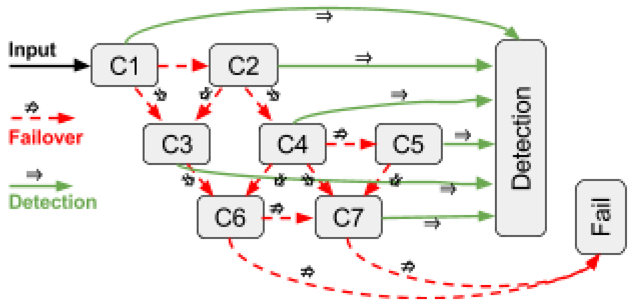

Figure 4: Failover DAG, whereby every path along the arrows represents a cascaded classifier as in Fig 3.

Proposition 1: A failover model in which failover paths form a DAG but detection passes directly to the output, can be validated using a number of scenarios which is proportional to the length of the longest failover path.

The implication of this DAG failover model is as follows: The data shown as input to a downstream classifier **C_j** is filtered to exclude items detected by an upstream classifier **C_i**. Further, each scenario such that **M⇒Cs** is passed by a specific classifier; thus, we can partition the set of scenarios according to the classifier which passes them.

In summary, the premise of this paper is that, through the use of scenarios and a DAG failover model, it is possible to standardize training, testing, and quantify safety of autonomous driving agents. Such quantification enables **reduction of the exponential amount of testing needed down to a linear amount proportional to the longest path on the DAG**.

3.0 Training Rare Events using Synthetic Data

Testing is needed to quantify safety after a system revision. The premise of a safe design is that failures are very rare. Consequently, encountering all failure scenarios in a live test drive would require testing millions of miles. It is not practical to perform such a testing for each software revision, or even upon release of a vehicle model.

This gives rise to the need for using simulation to quantify safety over synthetic data. Fig 5 depicts the results of using NVIDIA DIGIT DetectNet [4] and applying the techniques described in [5,6]. We trained a standard DetectNet CNN on synthetic images of self-balancing scooters, which are not commonly observed on streets and thus not present in the KITTI dataset. The training was basic and used uniform colors without data augmentation. The trained DetectNet CNN was subsequently tested on real data.

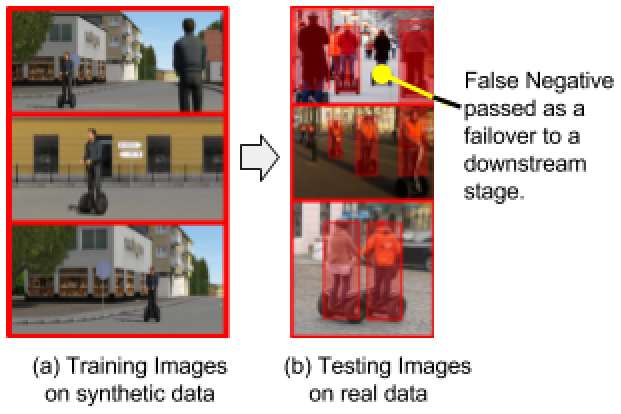
False Negative passed as a failover to a downstream stage.
(a) Training Images on synthetic data
(b) Testing Images on real data

Figure 5: Training rare events on synthetic data, and testing on specialized street data.

The results show that it is possible to obtain effective detection of self-balancing scooters within real images after training the DetectNet CNN on synthetic data.

The failover model DAG approach depicted in Figs 3,4 further allows replacing the output of a component with its corresponding ground truth, thus decoupling an individual component testing from others. More specifically, filtering the training and testing set to include only images which failed detection from upstream classifiers constitutes a simple extension to a training approach which sends all samples to all components. Note however that the samples fed into downstream classifiers due to a failover include all additional features accumulated from upstream classifiers beyond the input provided to the first stage classifier.

Once a DAG is defined, it is further possible to inject controlled failures at each component. For example, as depicted in Fig 6, it is possible to inject a failure of the perception layer, e.g., causing a specific vehicle or person to be undetected. Similarly, *Vires Test Drive* enables simulating poor conditions using plugins which render noise over each frame. To represent failing breaks or slippery roads, or blown out tires, *Vires Test Drive* allows changing the physical vehicle model based on the scenario.

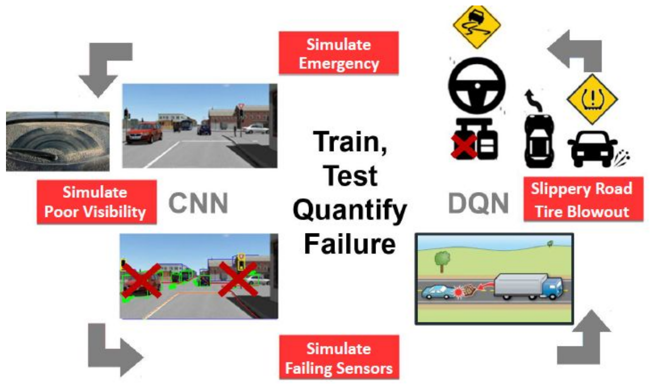

Figure 6: Quantifying safety of operation during poor conditions, emergency and malfunction.

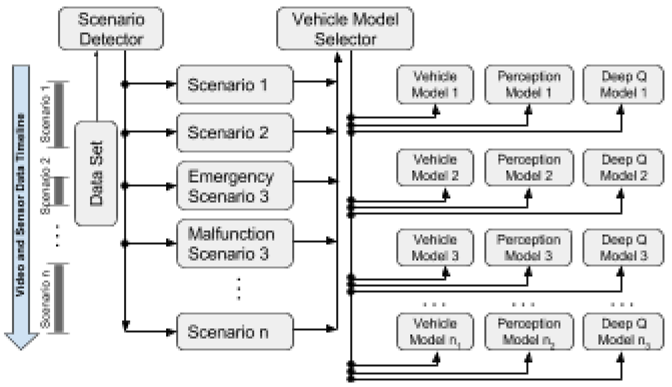

Figure 7: Ability to mapping of scenarios to models gives rise to the need for a model selector.

4.0 Multi-Model Training, Testing and Validation

A typical training dataset comprises of a stream of video frames or images associated with ground truth data. In many cases these streams describe multiple scenarios. As such, there is a need to identify the time-intervals of each scenario, and extract the images relevant to each scenario.

Fig 7 depicts an approach for extracting scenarios from a dataset stream. The dataset is fed into a scenario detector, which groups images according to the applicable scenarios. The scenario detector can be used to determine the distribution of scenarios in a dataset, which when combined with failure data, can be used to quantify the confidence output of the validator from Fig 1.

Emergency situations, such as failure of the braking system, would be represented by a separate physical vehicle model, and trained on a separate CNN and DQN. This gives rise to the a generalized model matrix depicted in Fig 7, whereby for each scenario a model selector associates a specific physical model, a specific perception CNN and a specific DQN reasoner. The model selection rules can be handcrafted, or may be learned automatically.

Each of the CNNs, and possibly the DQNs, in the model matrix of Fig 7 is a DAG following the failover model in Figs 3,4. The input images need to be partitioned according to the applicable models. Each of the stages can be decoupled from the other stages for the purpose of training and testing, further partitioning the images.

In summary, the overall partitioning of the training and testing images is performed over: a) scenarios, b) models in the matrix, c) and stages for each model. It is desirable to decrease exponentially the overall failure rates by adding stages. Using the DAG failover model, the total number partitions required, and the number of images and labels needed within each partition, grows linearly with the number of stages, rather than exponentially.

5.0 Regression Testing

Software development practices typically use regression testing and other means to regularly test functionality after revisions are made. It is common to find that functional regressions are introduced by changes. We therefore propose extending the state of the art training algorithms to accommodate regression testing, and continuous integration

For the purpose of the definitions berlow, we represent a system as a model, and thus a system revision reduces to a model revision.

Definition 2: Given a capability **C** and a scenario **s**, Model revision M_2 is regressed relative to M_2 if and only if

$$\mathbf{M_1 \Rightarrow Cs} \text{ and } \mathbf{M_2 \not\Rightarrow Cs} \qquad \text{(regression)}$$

To avoid regressive driving agents, we propose a new learning task, which reduces loss (to improve mAP or F-score) without "unlearning" correct behavior identified critical for specific capabilities and scenarios.

Definition 3: (avoid-unlearning) Given an initial set of parameters $\mathbf{\Theta}_{init}$ and a training data set **D** comprising of samples and ground truths, find a new set of parameters $\mathbf{\Theta_{new}}$ which minimize loss, but does not modify the classification of a subset $\mathbf{D'} \subset D$.

The initial set of parameters $\mathbf{\Theta}_{init}$ represents a pre-trained model. The subset $\mathbf{D'} \subset D$ represents the regression tests. The learning task is to minimize loss (and improve accuracy), but without breaking the regression tests.

Next, assuming that the safety testing of a revision M_1 is valid, we need to ensure that the same automated procedure can be used for safety testing of a subsequent revision M_2.

Definition 4: Model M_2 is **safety-test-regressive** (**STR**) compared to M_1 if and only if M_2 requires additional scenarios for validation as compared to M_1, namely it is *not* possible to satisfy

$$\forall\ \mathbf{C,f,p}\quad \mathbf{scenarios(C,M_2,f,p)} \nsubseteq \mathbf{scenarios(C,M_1,f,p)}$$

The premise of preventing STR is to prevent the need to change the (automated) safety testing procedures.

Hypothesis: It is possible to identify a wide range of system revision types which do not cause STR.

The intuition behind this hypothesis is that small changes, such as weight initialization and adjustments to training data, should not introduce STR. It should be possible to add new sensors and add classifier stages without introducing STR. If should further be possible to change physical characteristics of a vehicle, such as engine and tires, without introducing STR, if the physical system is encapsulated by an appropriate trajectory control model.

6.0 Conclusion

We propose to enable quantification of safety by formalizing scenario specifications. We further suggest to organize the scenarios according to the capabilities they are used to validate. As an example, the scenarios needed to quantify safety of lane changes are different from the scenarios needed to quantify safety of self parking.

We introduce a simple failover model, leading to a simple conceptual safety pass/fail decision. We argue that classifier failure rate reduces exponentially as stages are added. We then introduce a DAG failure model show how to reduce the number of scenarios needed to quantify safety from an exponential to linear in the number of stages along a path in the DAG.

Even after such reductions, quantifying safety requires processing of a large number of scenarios to ensure coverage of a wide range of rare events. For a given scenario, we argue that training and testing on synthetic data is an effective alternative to training on real data. We describe methods of training using dataset comprising multiple scenarios, and how partitioning can be used to train and test a model matrix when simulating emergencies and operation during poor conditions or failures.

Finally, we stress that identifying and avoiding regression is critical. Regressive behavior of learning agents occurs when a previously passing scenario fails. We propose a new learning task whereby loss is minimized without breaking regression tests. We further propose to focus on avoiding Safety Testing Regression (STR), which would invalidate the set of scenarios used for quantifying safety. Finally, we hypothesise that for a broad class of revisions would not cause STR.

References

[1] Seshia S.A, Sadigh D. and Sastry S.S, Towards Verified Artificial Intelligence, 2016

[2] Menzies, T., Pecheur, C.: Verification and validation and artificial intelligence. In: Zelkowitz, M. (ed.) Advances in Computers, vol. 65, Elsevier, Amsterdam (2005)

[3] A Geiger, P. Lenz, C. Stiller, and R. Urtasun, Vision meets robotics: The KTITI dataset. Inter.Jnational Journal of Robotics Research (IJRR), 32(11):1231-1237

[4] DetectNet: Deep Neural Network for Object Detection in DIGITS https://devblogs.nvidia.com

[5] Hoiem, D., Chodpathumwan, Y., and Dai, Q. 2012. Diagnosing Error in Object Detectors. Computer Vision – ECCV 2012, Springer Berlin Heidelberg, 340–353.

[6] Szegedy, C., Liu, W., Jia, Y., et al. 2014. Going Deeper with Convolutions. arXiv http://arxiv.org/abs/1409.4842.