Safety Validation

Autonomous systems are rapidly being developed and deployed in human centered environments with examples including:

Delivery robots

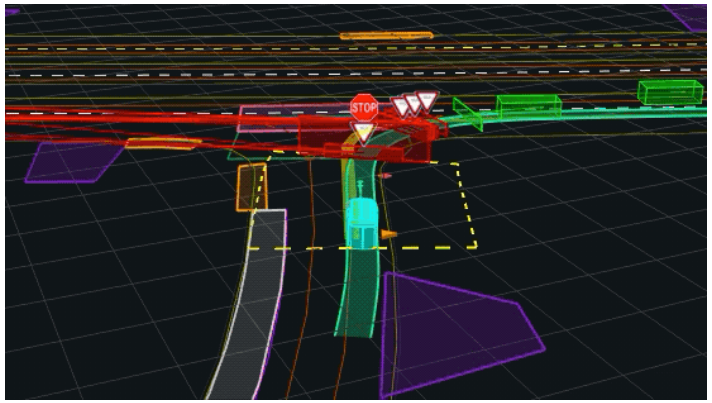
Human robot control

Autonomous driving

- Validation is an important step prior to deployment to ensure that these systems work in an expected and safe manner
- Real world testing, while accurate, can have a hard time comprehensively covering the space of failures
  - Estimates say human drivers experience one accident every ~480,000 miles on average. How many vehicles and miles driven does an autonomous vehicle tester need to cover a wide range of accident scenarios?

- To complement real world testing, we can turn to simulation and models of the environment to generate test scenarios for autonomous systems
- High fidelity simulators have allowed autonomous system developers to iterate at a pace that physical testing would not allow:
  - E.g. As of 2020, Waymo is reporting 20 million miles driven in simulation per day and over 15 billion simulation miles in total

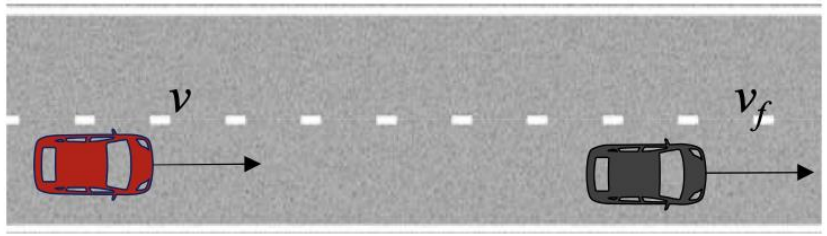
CARLA



Waymo

Test Matrix Approach

- Our goal in validation is to subject the system to various scenarios it may face during actual deployment and ensure that it performs "correctly" under all conditions
- The test matrix approach looks to generate concrete testing scenarios by defining rows of all possible parameter values, thus forming a matrix of scenarios



Name	Description	Min	Max	Offset
V_f	Speed of FV (m/s)	0	30	0.5
Dif_v	Speed of FV minus speed of AV (m/s)	-10	10	0.5
Dis	Distance between two vehicles (m)	5	50	1

~100,000 scenarios in this test matrix

Test Matrix Approach cont.

- Coupled with simulation resources, the test matrix approach can be used in low dimensional settings to get full coverage of all testing scenarios
- However, running all scenarios from a test matrix quickly becomes infeasible in high dimensional systems even with simulation (curse of dimensionality)
- Failure scenarios are also often characterised by long tail distributions where majority of scenarios in the test matrix do not lead to critical configurations

We would like to determine test scenarios that are high in criticality (likely to lead to failure) without spending our limited simulation/physical resources analysing safe configurations.

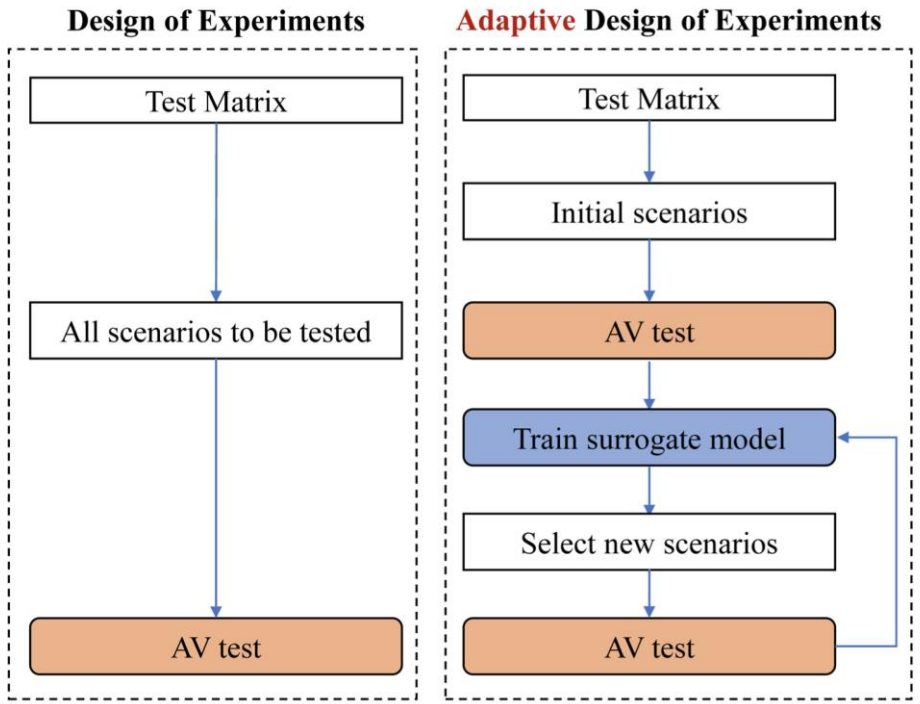
Adaptive Validation Strategies

1. Adaptive design of experiments
2. Scenario generation
3. Adaptive failure search

1. Adaptive Design of Experiments

Adaptive Design of Experiments

- The adaptive design of experiments approach aims to reduce the effects of the curse of dimensionality associated with using test matrices for validation
- Instead of iteratively evaluating all scenarios in the test matrix, we iteratively train a surrogate model to guide which scenarios to select for testing
- In general, the surrogate model can be anything that is able to make predictions of unlabeled data



J. Sun, H, Zhou, H, Zhang, Y, Tian, and Q, Ji, "Adaptive Design of Experiments for Accelerated Safety Evaluation of Automated Vehicles"

Adaptive Design of Experiments

The process is intialised by:

- Randomly select an initial batch of scenarios N_i from the test matrix X_t
- Run the scenarios through simulation validation tool $F(\cdot)$ to generate an importance score S_i for each scenario
- Initialise surrogate model with dataset $D = [N_i, S_i]$

Scenarios are selected from the test matrix iteratively as follows:

- Use $F(\cdot)$ to obtain importance score S_k for concrete scenario N_k
- Append N_k, S_k to training dataset D
- Train surrogate model on D
- Randomly select a batch of scenarios N_t from the test matrix X_t
- Use surrogate model to predict an importance score Y for each scenario in N_t
- Pick concrete scenario N_k according to predicted importance scores

Adaptive Design of Experiments

3-dimensional Intelligent Drivers Model

Name	Coverage	Runtime(s)	Name	Coverage	Runtime(s)
Random	4.48%	45.5	XGB	94.01%	507.6
RBF	72.14%	4209.6	KNN	91.42%	50.7
KRG	65.11%	11183.7	SVR	87.95%	881.6
QP	82.00%	62.8			

9-dimensional Intelligent Drivers Model

Name	Coverage	Runtime(s)
Random	0.53%	771.53
QP	27.50%	1183.67
XGB	28.14%	3766.47
KNN	22.67%	1020.05
SVR	27.35%	11997.62

2. Scenario Generation

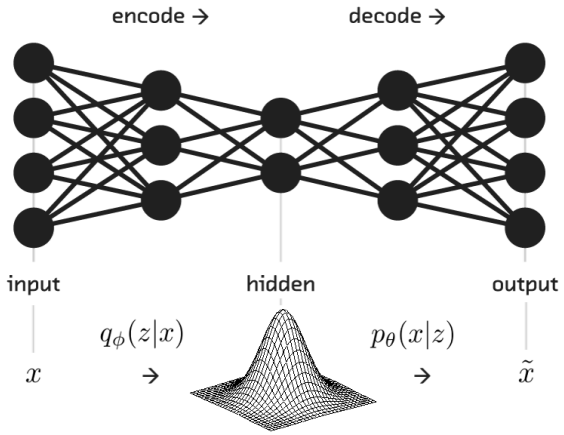
Scenario Generation

- Developing models to generate testing scenarios from limited existing datasets can provide a rich source of data for autonomous system testing and validation.
- The goal is to take a set of existing data (that may include a small amount of desirable test scenarios) and use it to build a generative model where we can sample a latent space for new test scenarios.
- We can employ deep generative approaches such as Variational auto-encoders (VAEs) and Generative adversarial networks (GANs) for this.

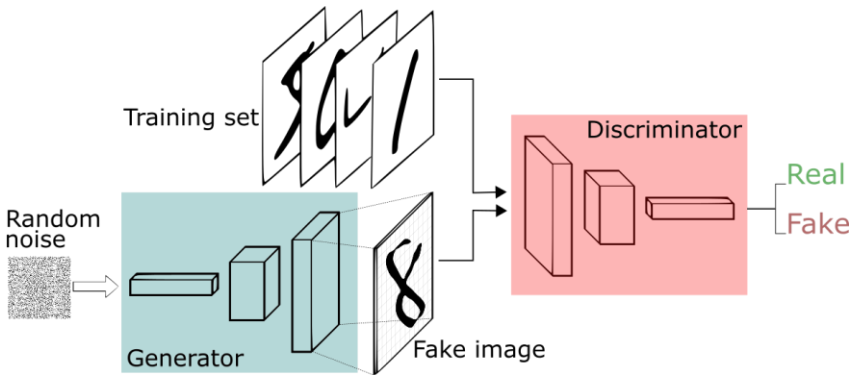
Scenario Generation

The goal of these methods is to learn a parametric model for the distribution of testing scenarios through maximizing the likelihood of some set of empirical data.

- VAEs use an encoder/decoder architecture to learn a mapping from a low dimensional latent variable to the output space.
- GANs train a generator/discriminator in a two-player game where the goal is to make the discriminator unable to distinguish between outputs of the generator and the training dataset.



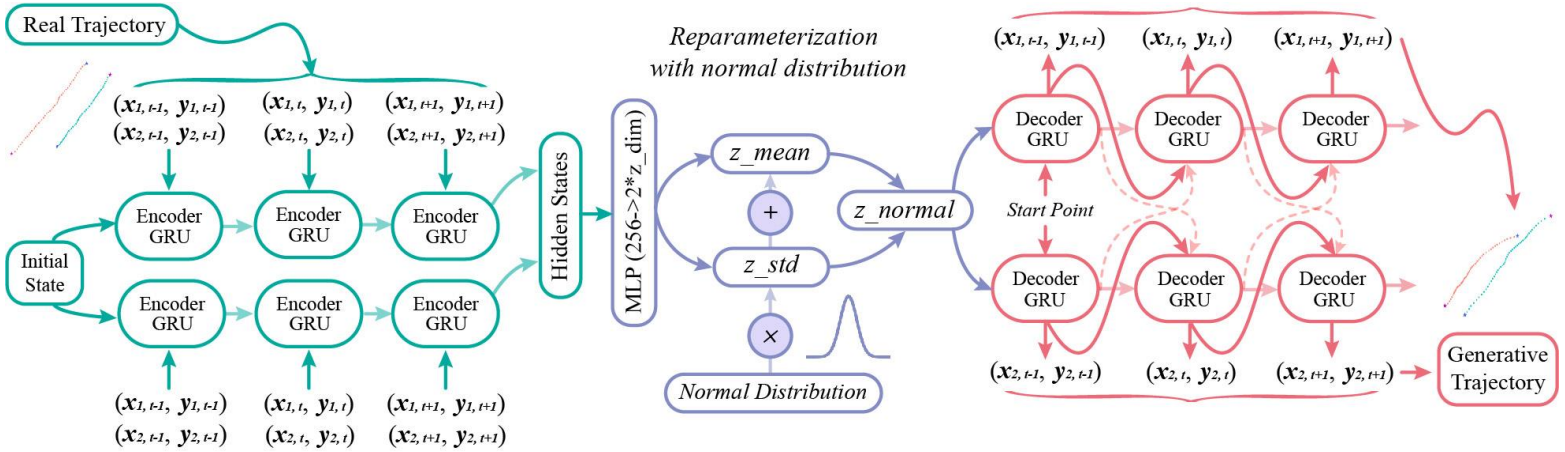
VAE [1]



GAN [1]

VAE Trajectory Generator

- Ding et al. employ this methodology for generating autonomous vehicle testing scenarios.
- The method encodes multi-vehicle interaction scenarios into a latent space which can then be sampled and passed through a decoder to generate new driving encounters.



W. Ding, B. Chen, B. Li, K. J. Eun, and D. Zhao, "Multimodal Safety-Critical Scenarios Generation for Decision-Making Algorithms Evaluation"

Safety Critical Scenario Generation

- The previous VAE approach could generate new testing scenarios; however, it did not have a mechanism for encoding which scenarios are more desirable.
  - For example, a tester may want to generate test scenarios that are both new *and* sufficiently dangerous.
- We can account for this by weighting the likelihood maximization by an appropriate risk metric that quantifies the criticality of the scenario.
  - Recall that the essence of these generative approaches is to fit a parametric distribution model that maximizes the likelihood of some empirical dataset.

Weighted likelihood model:
$w(x_i)$ depends on risk metric

$$L(\boldsymbol{x}_i|\boldsymbol{y}_i;\boldsymbol{\theta}) = p(\boldsymbol{x}_i|\boldsymbol{y}_i;\boldsymbol{\theta})^{w(\boldsymbol{x}_i)}$$

$$\log L(\boldsymbol{x}_i|\boldsymbol{y}_i;\boldsymbol{\theta}) = w(\boldsymbol{x}_i)\log p(\boldsymbol{x}_i|\boldsymbol{y}_i;\boldsymbol{\theta})$$

W. Ding, W. Wang, and D. Zhao, "A New Multi-vehicle Trajectory Generator to Simulate Vehicle-to-Vehicle Encounters," in IEEE International Conference on Robotics and Automation (ICRA)

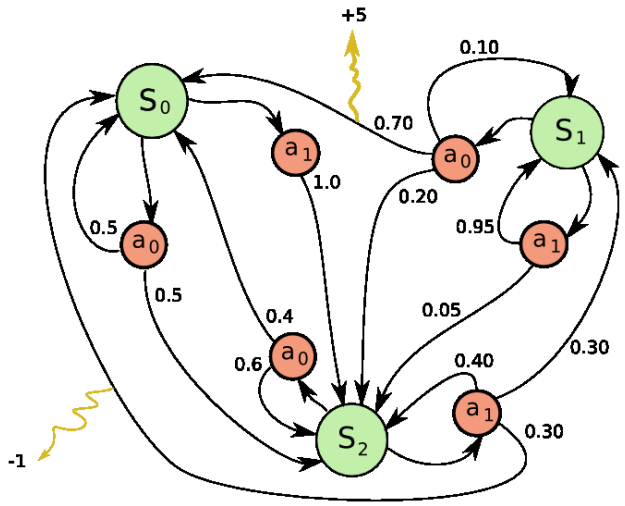
3. Adaptive Failure Search

Adaptive Failure Search

- Suppose we don't have empirical data available for scenario generation, but we instead have a model (simulation) of the autonomous system and environment
- Adaptive failure search techniques can leverage simulation to explore the space of scenarios for failure conditions
- One approach is to treat the problem of finding failures as a Markov decision process (MDP) and solve it using reinforcement learning (RL) techniques

MDPs & RL

- An MDP is a way of characterizing a sequential decision-making task as a tuple (S, A, T, R) where:
  - S is a set of states
  - A is a set of actions
  - T is a transition probability of going from state s to s' after action a: $T: S \times \mathrm{A} \times S \rightarrow Prob(S)$
  - R is a reward function when transitioning from state s to s' after action a
- Given an MDP, we can use standard RL solvers (MCTS, DQN, etc) to train a decision policy whose goal is to maximize the reward obtained by an agent acting in the environment

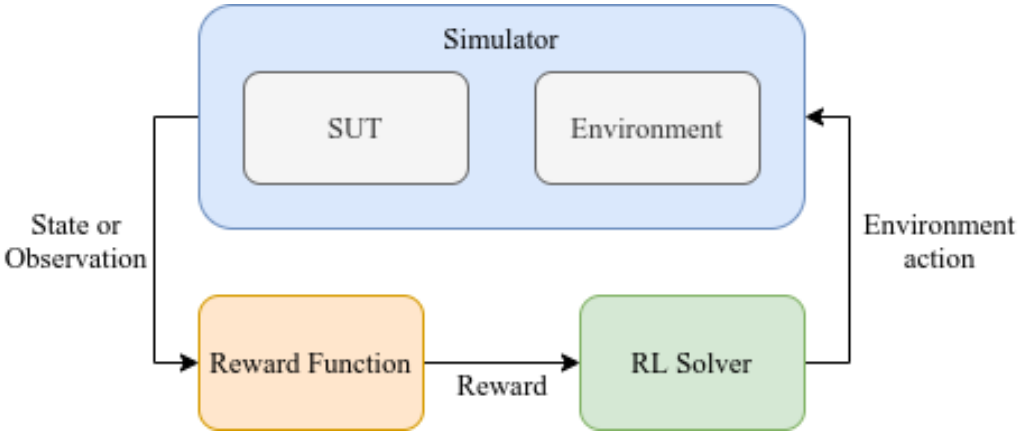


Visualization of an MDP[1]

1. https://towardsdatascience.com

Adaptive Failure Search

- Given a simulator, we can train an RL policy to generate failure scenarios by defining an agent and reward function.
  - The agent explores the environment and collects reward
  - At each step, the agent's policy is updated reflecting the reward it just obtained
  - The reward here is tuned to promote system failures and penalize safe operation



Penalty for no failure — Heuristic to guide search

$$R(s) = \begin{cases} 0 & s \in E \\ -\alpha - \beta \mathcal{D}(r_v, r_p) & s \notin E, t \geq T \\ -\mathcal{M}(a \mid s) & s \notin E, t < T \end{cases}$$

Penalty for rare actions

Reward Function

- In particular, the reward is sensitive to
  - The occurrence of a failure
  - The probability of system action

Penalty for no failure

Heuristic to guide search

$$R(s) = \begin{cases} 0 & s \in E \\ -\alpha - \beta \mathcal{D}(r_v, r_p) & s \notin E, t \geq T \\ -\mathcal{M}(a \mid s) & s \notin E, t < T \end{cases}$$

Penalty for rare actions

- This reward formulation can lead to failures that are often:
  - Low in relevance
  - Lack diversity

Reward Augmentation

- Classify each timestep as safe/dangerous and proper/improper
- Redefine failure to include scenarios when the system behaves improperly prior to a collision
- Higher reward for more improper system behaviour

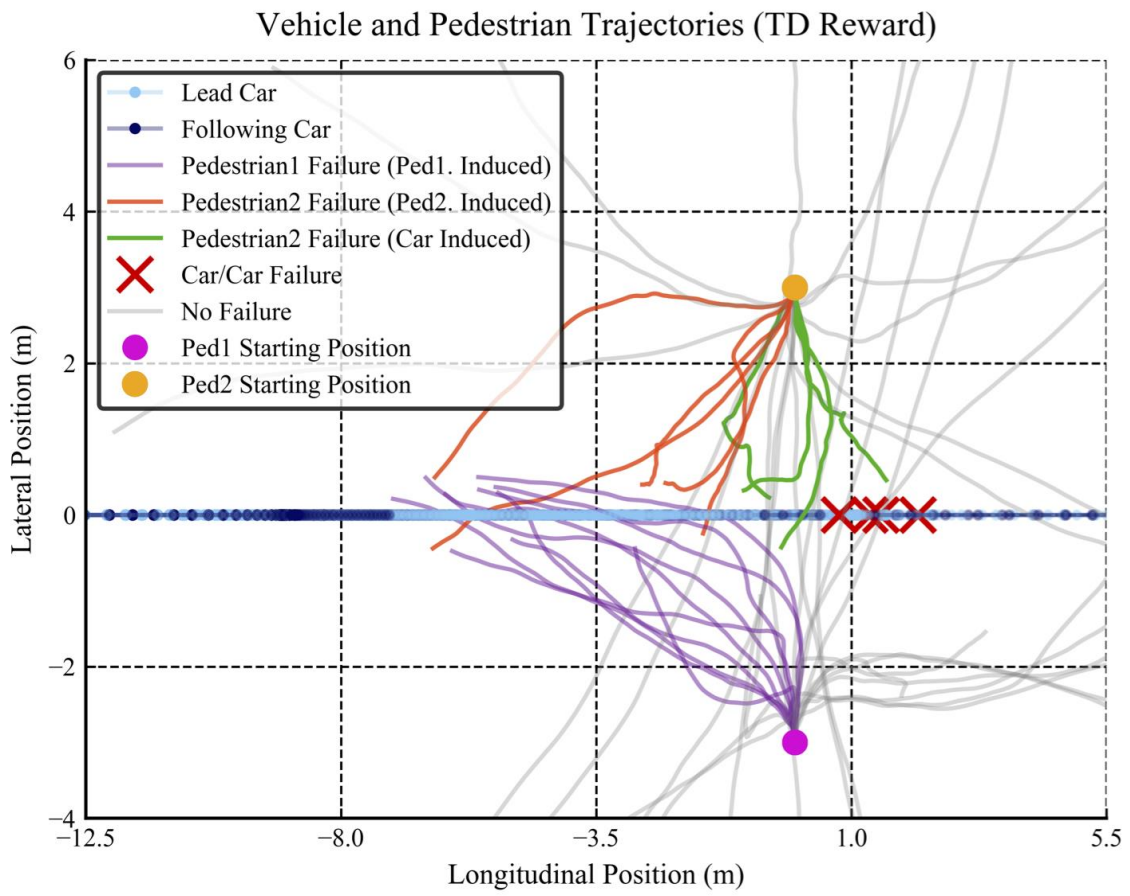
$$E_{\text{RSS}} = \{\tau \mid \tau \in E \text{ and } \mathrm{f}_{\text{imp}}(\tau) > \mathrm{f}_{\text{crit}}\}$$

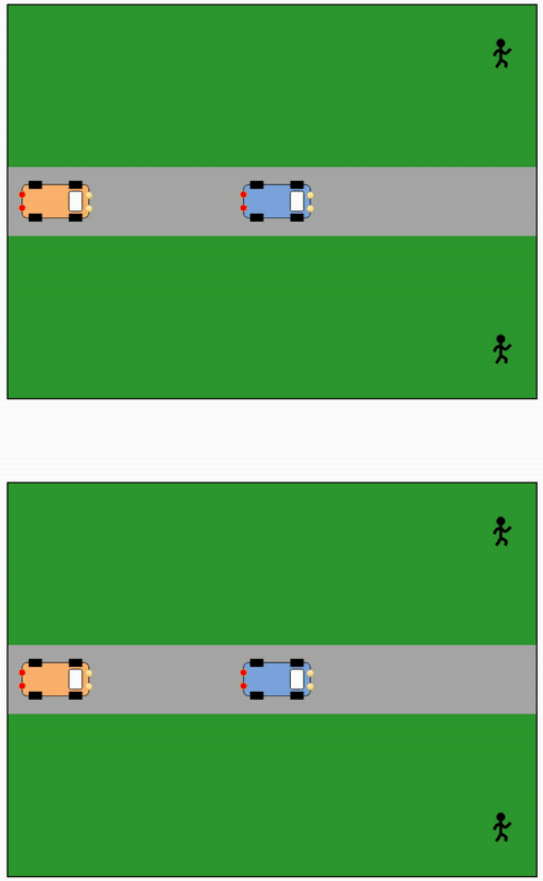
$$R_{\text{RSS}}(s) = \begin{cases} 0 & s \in E_{\text{RSS}} \\ -\alpha - \beta f_{\text{imp}}(\tau) & s \notin E_{\text{RSS}}, t \geq T \\ -\mathcal{M}(a \mid s) & s \notin E_{\text{RSS}}, t < T \end{cases}$$

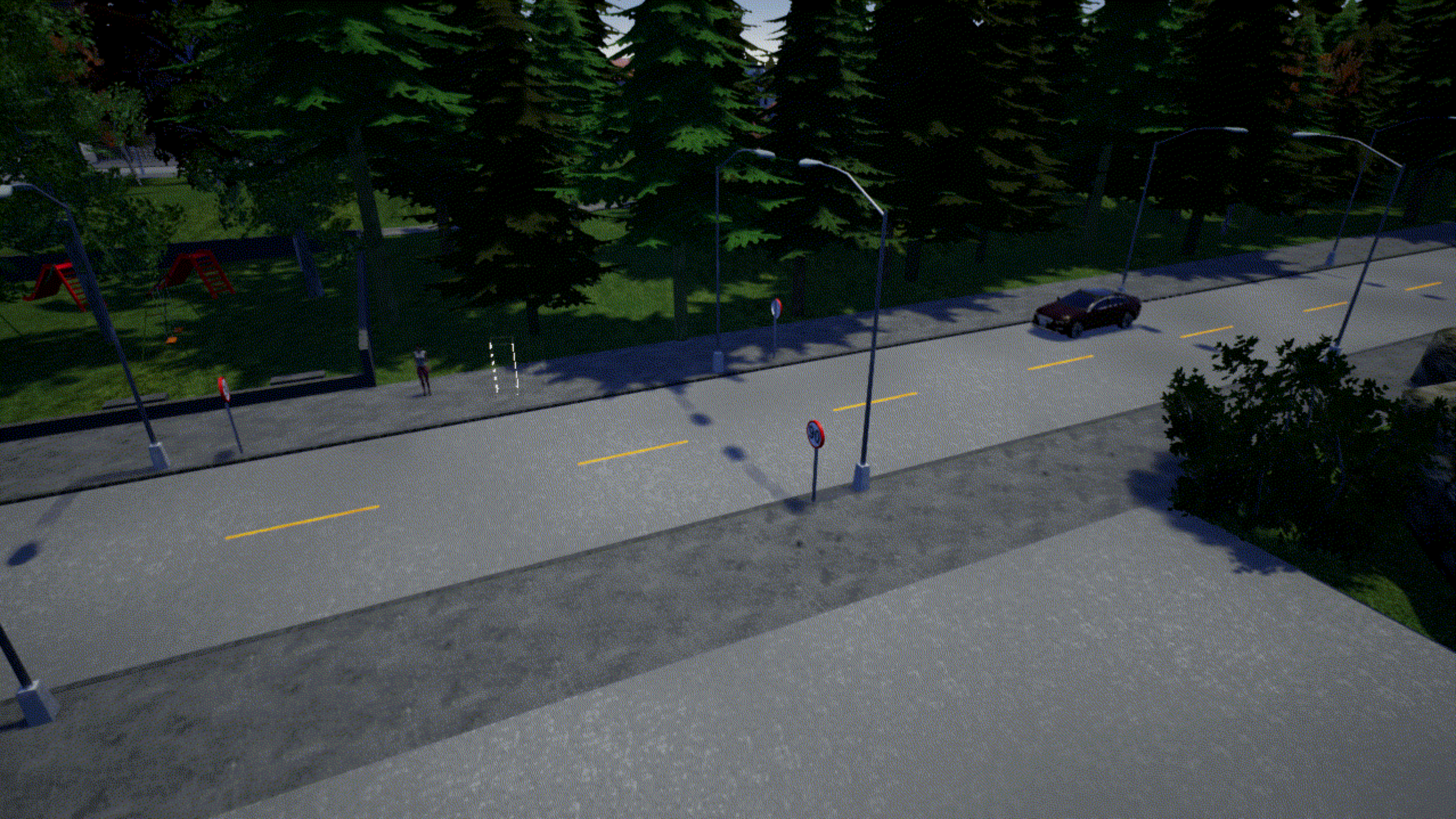
- The dissimilarity term encourages search over spaces where unique failures occur without the efficiency penalty incurred by using random search

$$R_{TD}(s) = \begin{cases} \frac{\gamma}{\mu} \sum_{i=1}^{\mu} D(t_s, t_i) & s \in E \\ -\alpha - \beta \mathcal{D}(r_v, r_p) & s \notin E, t \geq T \\ -\mathcal{M}(a \mid s) & s \notin E, t < T \end{cases}$$

Adaptive Search Applied to Vehicle/Pedestrian Scenario

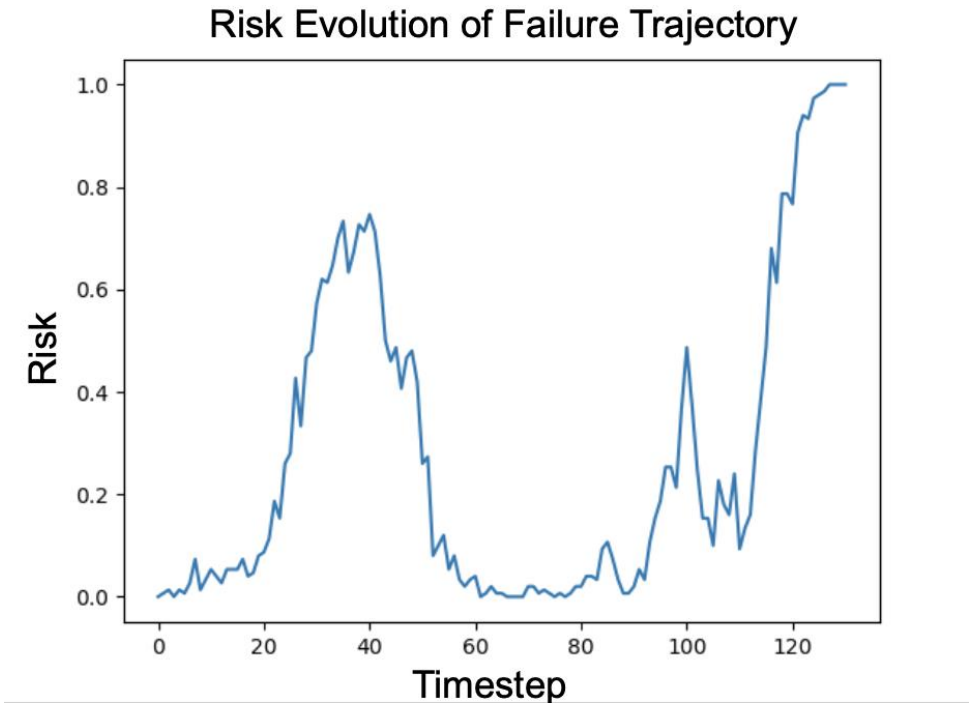




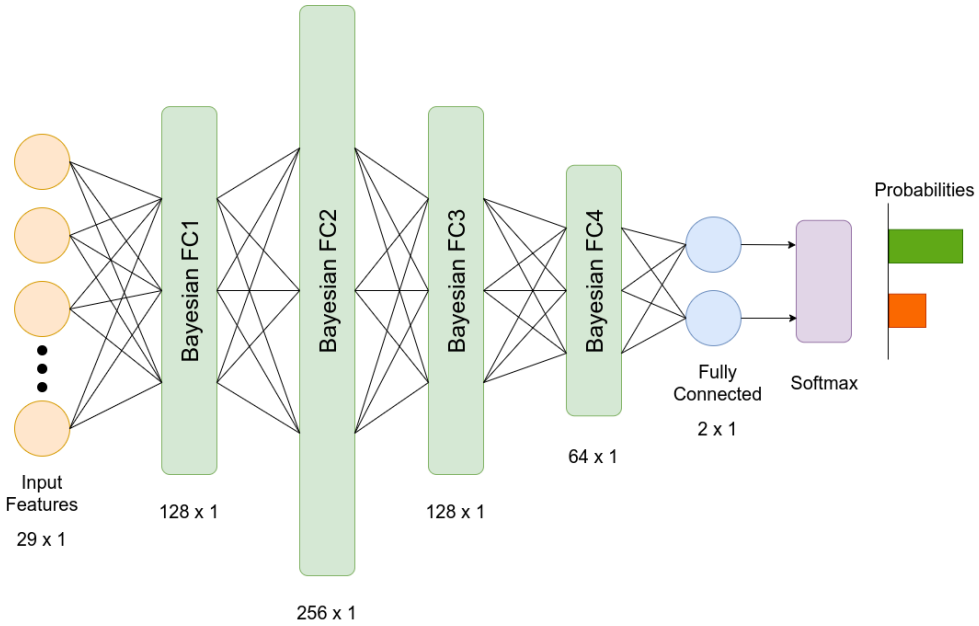


Incorporating Human Insight

A single failure trajectory may stem from multiple high-risk scenarios. To identify these "critical states", we propose incorporating a Bayesian network trained to perform classification of critical states.



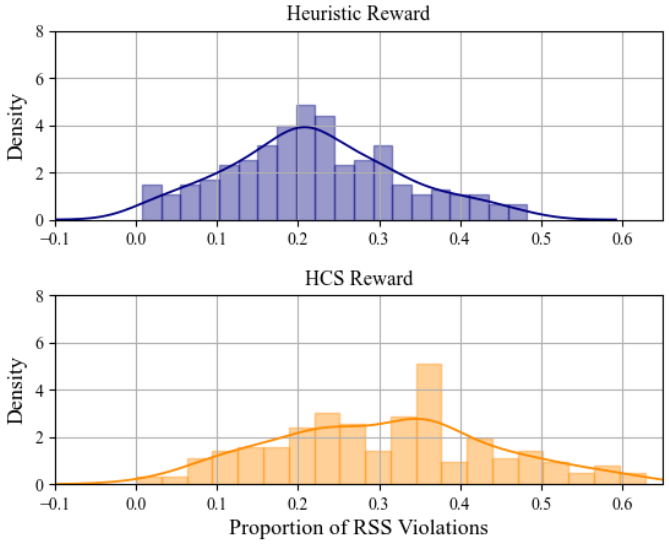
We seek to incorporate the notion of critical states from an *onlooker's perspective* into the failure search framework with *human expert labels*

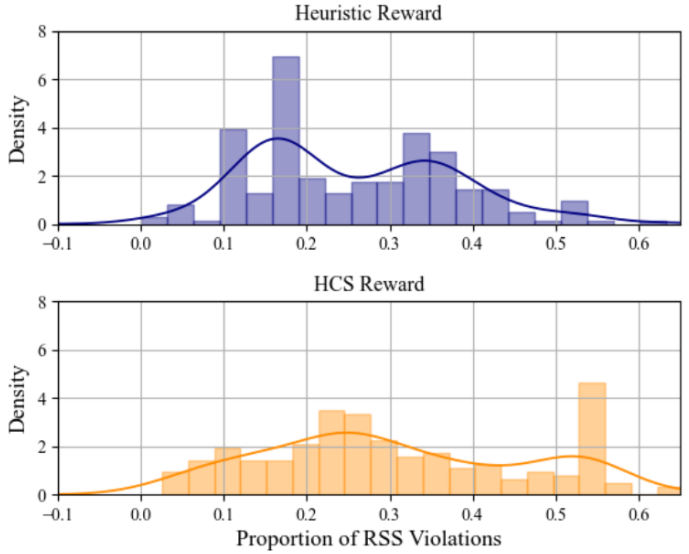


Human critical state (HCS) classifier

A Bayesian neural network is used to predict the probability of a given state representing a critical scenario.

We compare the proportion of unsafe states (deemed by RSS) in failure trajectories and see an increase in the proportion of dangerous behavior found when considering critical states





Formal Verification

Formal Verification

- The prior approaches looked at generating failure and test cases which can then be used see how the autonomous system reacts in the real world/simulation
- We can also consider the task of validation in a more direct manner:
  - Suppose you are given a system; how can we design a module that takes in the specification of the system, the safety conditions it should satisfy, and returns a yes or no answer to us guaranteeing whether the system is safe?
- High level description of formal verification:
  - Prove the safety (or unsafety) of a system based on a mathematically rigourous model
  - Verify the properties of the system via mathematically sound proofs instead of through empirical methods

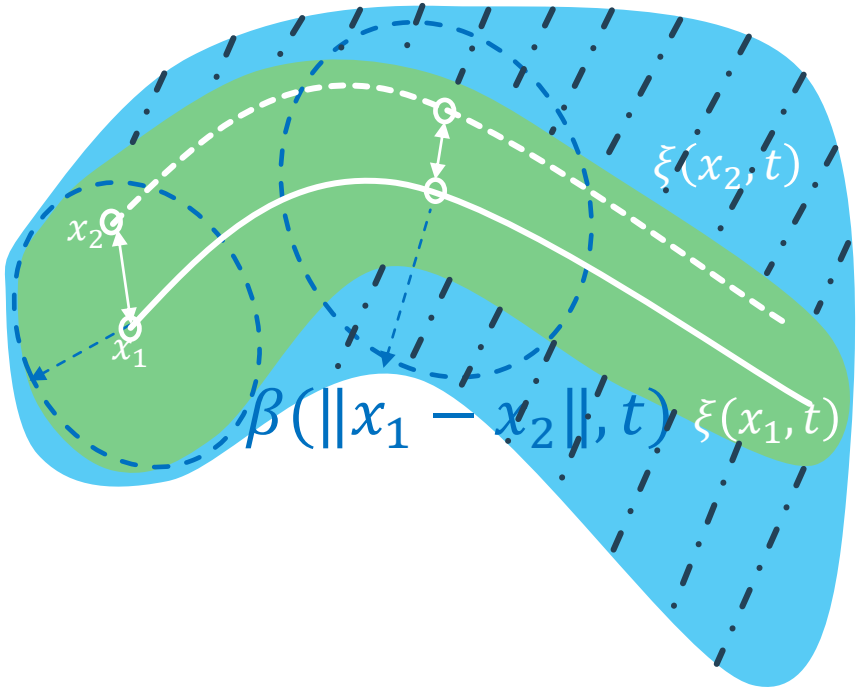
Reachability

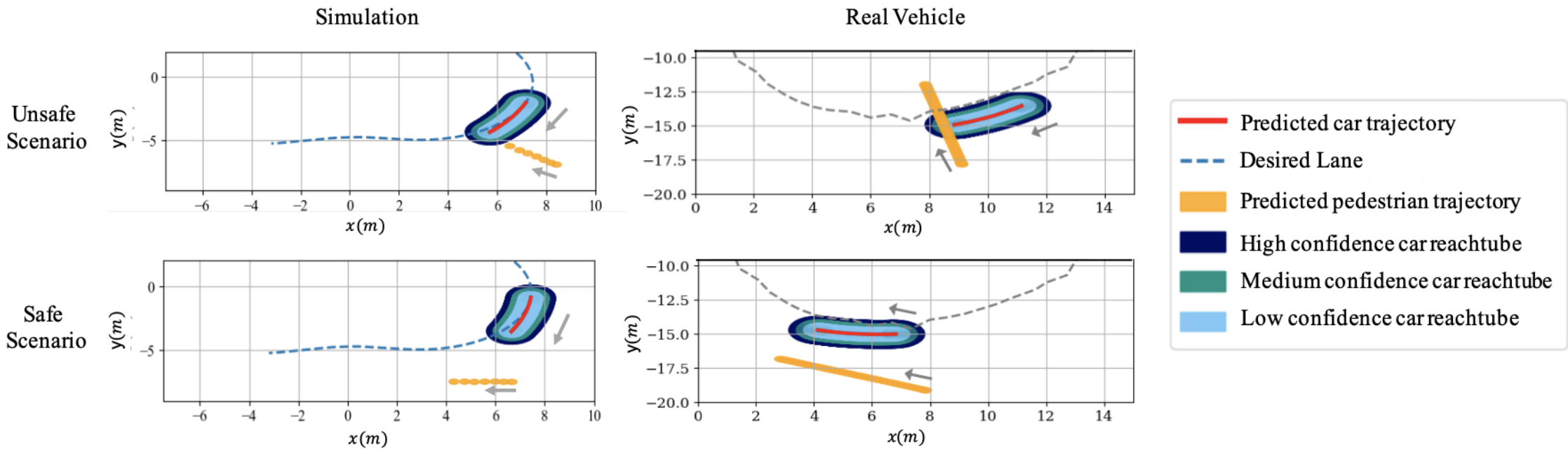
- To address the task of validating autonomous systems, we can pose it as a reachability problem:
  - Given a system model, an initial set of states, and an unsafe set, the reachability problem asks whether any behaviour of the system starting from the initial set can reach the unsafe set.
  - More formally: given a dynamical system A over a state space X, a mode space P, a set of initial states Θ, a mode $p \in P$, and a look-ahead time T, the set of reachable states is denoted as $Reach_A(\Theta, p, T) \in X$.
- DryVR is an example of this: A data driven verification engine that employs reachability to verify hybrid dynamical systems

DryVR: Data-driven verification and compositional reasoning for automotive systems, C. Fan, B. Qi, S. Mitra & M. Viswanathan, *CAV 2017*

DryVR

- DryVR uses a "sensitivity function" β that bounds the distance between trajectories of the system A as function of initial state
  - β is learned offline through sampling trajectories from a simulator
- During verification, the initial set is partitioned into m regions. For each region i, a numerical simulation $\xi(x_i, t)$ from a representative state x_i is computed up to time T_{Look}
- An over-approximation of the reachable set $Reach_A(\Theta, p, T_{Look})$ is obtained by bloating $\xi(x_i, t)$ with β and then taking a union




Simulation
Real Vehicle
Unsafe Scenario
Safe Scenario
y(m)
x(m)
Predicted car trajectory
Desired Lane
Predicted pedestrian trajectory
High confidence car reachtube
Medium confidence car reachtube
Low confidence car reachtube

Wrap Up

- There's a trade off that one needs to make when considering the level of formal guarantees provided by a validation method
- Typically, the stronger the safety guarantees, the more restrictions are required
  - Eg. More computation, ability to handle complex/higher dimension systems etc.
- As a result, many of the techniques mentioned here are not independent validation methods. Rather they can each serve a purpose during a particular phase or application
  - Eg. Apply adaptive search during early stages of testing to quickly get a sense of the scope of failure space. Use formal verification for safety critical tasks where the system dynamics and model can be explicitly specified.