Design and Assessment of Safe Autonomous Vehicles

Saurabh Jha

TJ Watson Research Center

Collaborators: Ravi Iyer, Zbigniew Kalbarczyk, Subho Banerjee, Shengkun Cui

Q1: What vulnerabilities exist in Autonomous Vehicles?

Q2: Are these vulnerabilities worse than existing vulnerabilities in non-AVs?

Q3: Can these vulnerabilities be used for attacks?

Q4: How do we make the AVs safer?

Q1: What vulnerabilities exist in Autonomous Vehicles?

Q2: Are these vulnerabilities worse than existing vulnerabilities in non-Avs?

Q3: Can these vulnerabilities be used for attacks?

Q4: How do we make the AVs safer?

Autonomous Systems in the Wild!

Promise AVs advertised as transformative

Reality Catastrophic accidents

Uber accident: Multiple failure points: (a) object flickering, (b) distracted safety operator, (c) disabled emergency braking

Autonomous Systems in the Wild!

AI-operated service robot fell on the elevator and knocks two people

Source: https://www.youtube.com/watch?v=4Pwx3U4vJKw

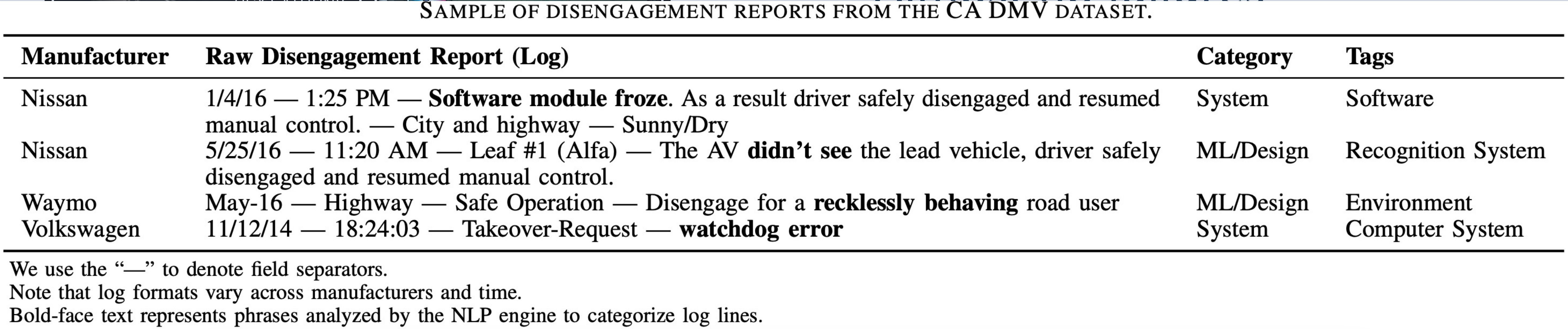
AI-operated service cart spins out-of-control nearly destroying an aircraft

Source: https://www.thrillist.com/news/nation/chicago-ohare-airport-cart-out-of-control

Challenges that Threaten Safety

Source: nvidia.com

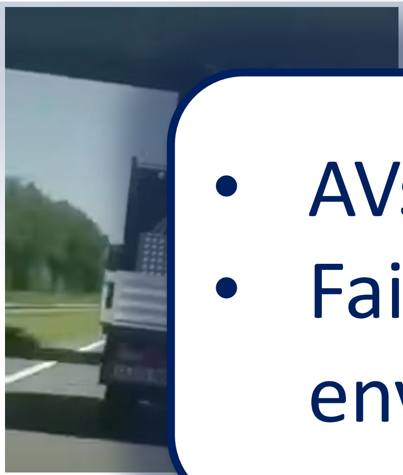
- Almost no failures

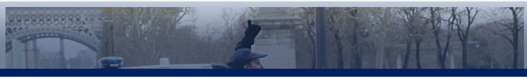


SAMPLE OF DISENGAGEMENT REPORTS FROM THE CA DMV DATASET.

Manufacturer	Raw Disengagement Report (Log)	Category	Tags
Nissan	1/4/16 — 1:25 PM — **Software module froze**. As a result driver safely disengaged and resumed manual control. — City and highway — Sunny/Dry	System	Software
Nissan	5/25/16 — 11:20 AM — Leaf #1 (Alfa) — The AV **didn't see** the lead vehicle, driver safely disengaged and resumed manual control.	ML/Design	Recognition System
Waymo	May-16 — Highway — Safe Operation — Disengage for a **recklessly behaving** road user	ML/Design	Environment
Volkswagen	11/12/14 — 18:24:03 — Takeover-Request — **watchdog error**	System	Computer System

We use the "—" to denote field separators.
Note that log formats vary across manufacturers and time.
Bold-face text represents phrases analyzed by the NLP engine to categorize log lines.





➢ Model uncertainty, weather, …

- AVs 15-4000x worse than humans
- Failures equally attributed to hardware/software, environment and ML for Waymo

rediction

model

Real world traffic

DSN 2018

Q1: What vulnerabilities exist in Autonomous Vehicles?

Q2: Are these vulnerabilities worse than existing vulnerabilities in non-Avs?

Q3: Can these vulnerabilities be used for attacks?

Q4: How do we make the AVs safer?

Are AVs more exposed than non-AVs?

Accessible in the open

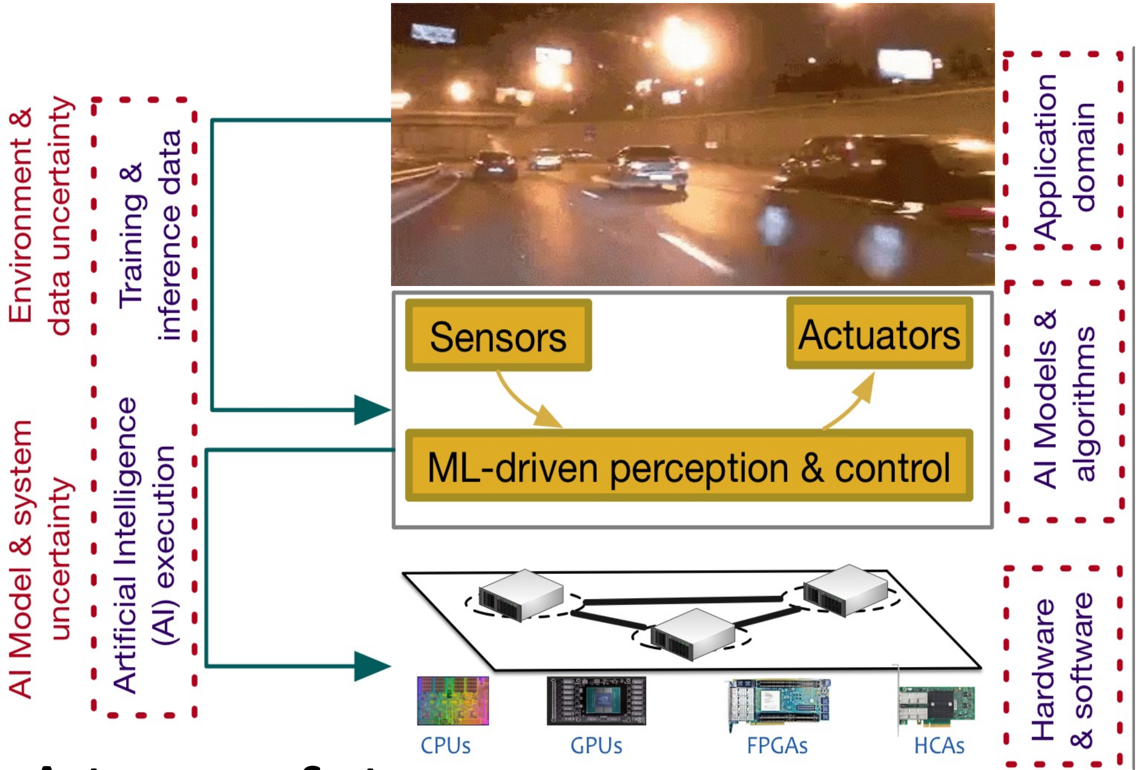
Connect to the internet with IoT devices and phones

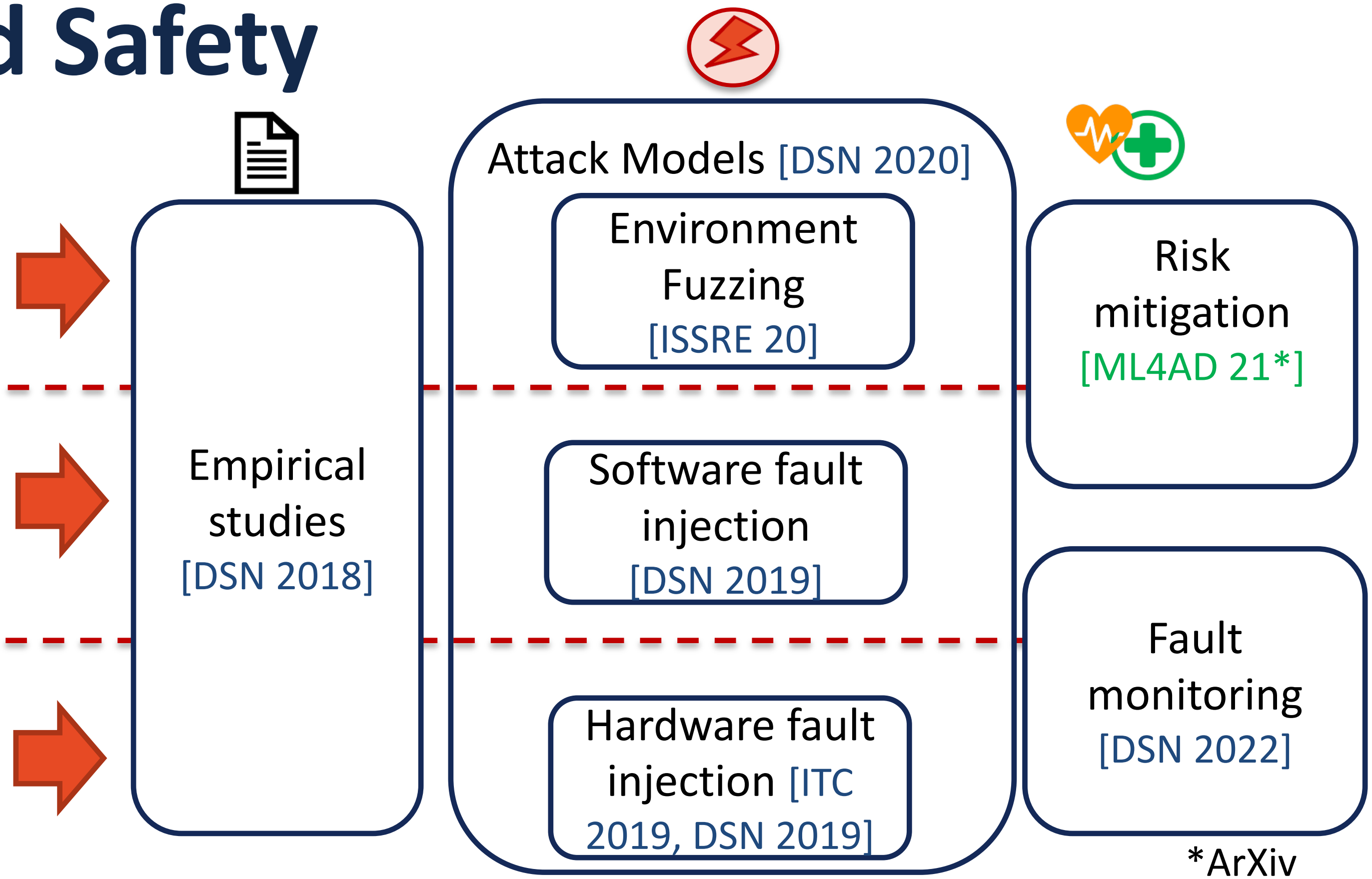
Vehicles in general are most exposed safety-critical system in the world!

Are AVs more exposed than non-AVs?

- Resounding **Yes!**
- Increased attack surface
- AV ecosystem is more decentralized
- More software components that are hard to verify
  - Trojans, ML uncertainty, training data quality, unknown unknowns, etc.

AV Research Overview for Vulnerability Assessment and Safety


Environment & data uncertainty
AI Model & system uncertainty
Training & inference data
Artificial Intelligence (AI) execution
Sensors
Actuators
ML-driven perception & control
CPUs
GPUs
FPGAs
HCAs
Application domain
AI Models & algorithms
Hardware & software



Empirical studies [DSN 2018]

Attack Models [DSN 2020]

Environment Fuzzing [ISSRE 20]

Software fault injection [DSN 2019]

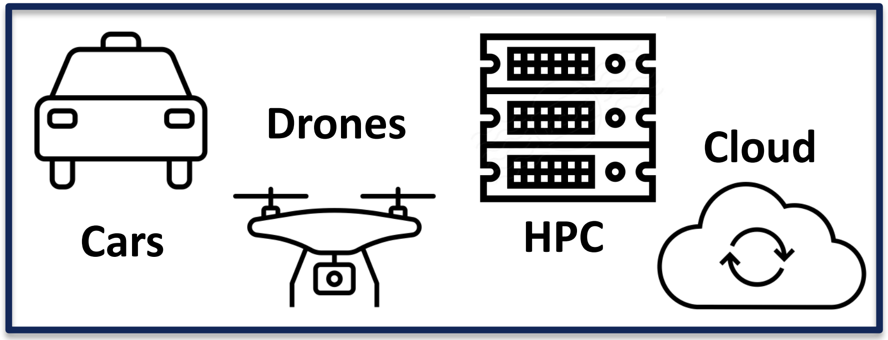
Hardware fault injection [ITC 2019, DSN 2019]

Risk mitigation [ML4AD 21*]

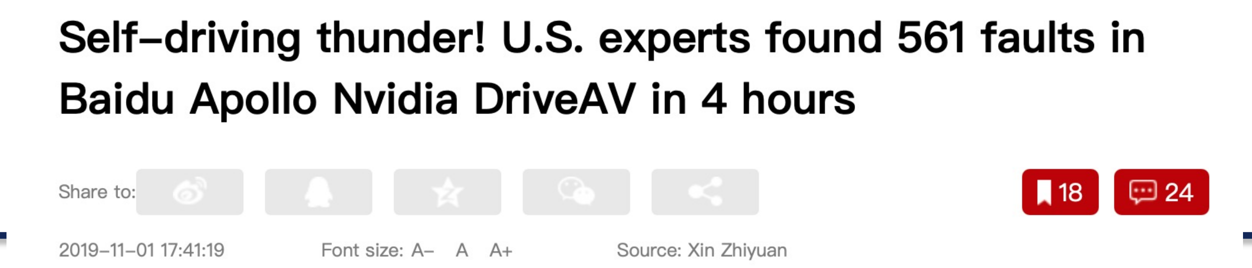
Fault monitoring [DSN 2022]

*ArXiv

Autonomous Systems



Science Daily, Daily Illini , Guancha.cn, Space Daily, Sina,…



Self-driving thunder! U.S. experts found 561 faults in Baidu Apollo Nvidia DriveAV in 4 hours

Share to:

18 24

2019-11-01 17:41:19 Font size: A- A A+ Source: Xin Zhiyuan

Q1: What vulnerabilities exist in Autonomous Vehicles?

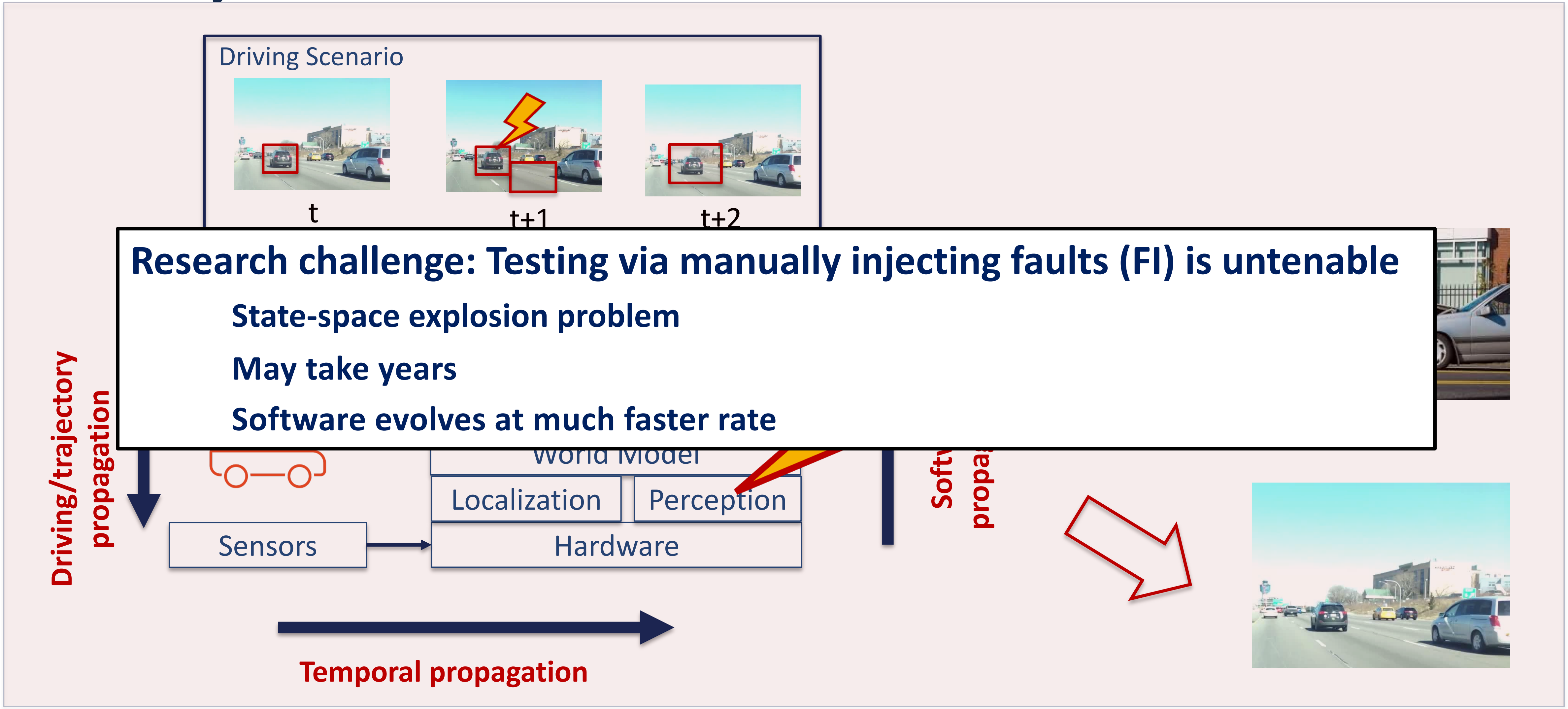
Q2: Are these vulnerabilities worse than existing vulnerabilities in non-Avs?

Q3: Can these vulnerabilities be used for attacks?

Q4: How do we make the AVs safer?

ML-Based Fault Injection for Autonomous Vehicles: A Case for Bayesian Fault Injection, DSN 2019

Identify vulnerabilities that lead to collision ?



Research challenge: Testing via manually injecting faults (FI) is untenable

State-space explosion problem

May take years

Software evolves at much faster rate

Our Solution: Bayesian Fault Injection (BFI)

Goals: To accelerate testing to identify faults (in hardware or software) that can lead to safety hazards

Key Ideas:

- ❑ Explicitly model fault propagation in software using Probabilistic Graph Models (PGMs)
- ❑ Explicitly model fault propagation in environment using kinematics and safety model
- ❑ Train with observational data from field tests (or simulations)
- ❑ Model fault injection as an inference query on the PGM model

Accelerated testing by:

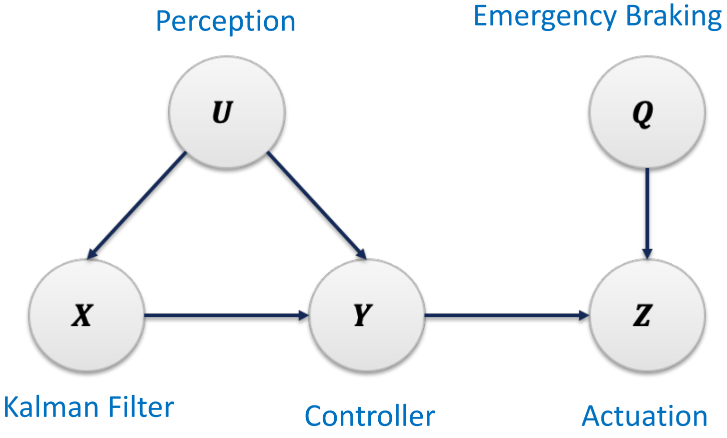
- ❑ Replacing FI with ML inference (conduct FI only when ML inference predicts safety violation)
- ❑ Pruning the state space

Results:

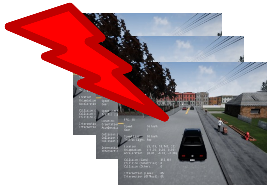
- ✓ Demonstrated on Baidu Apollo AV agent
- ✓ Test accelerated by 3690x (4 hours vs 2 years for traditional fault injection)

Approach Overview

S1: Construct and train probabilistic graph model



Golden simulation runs



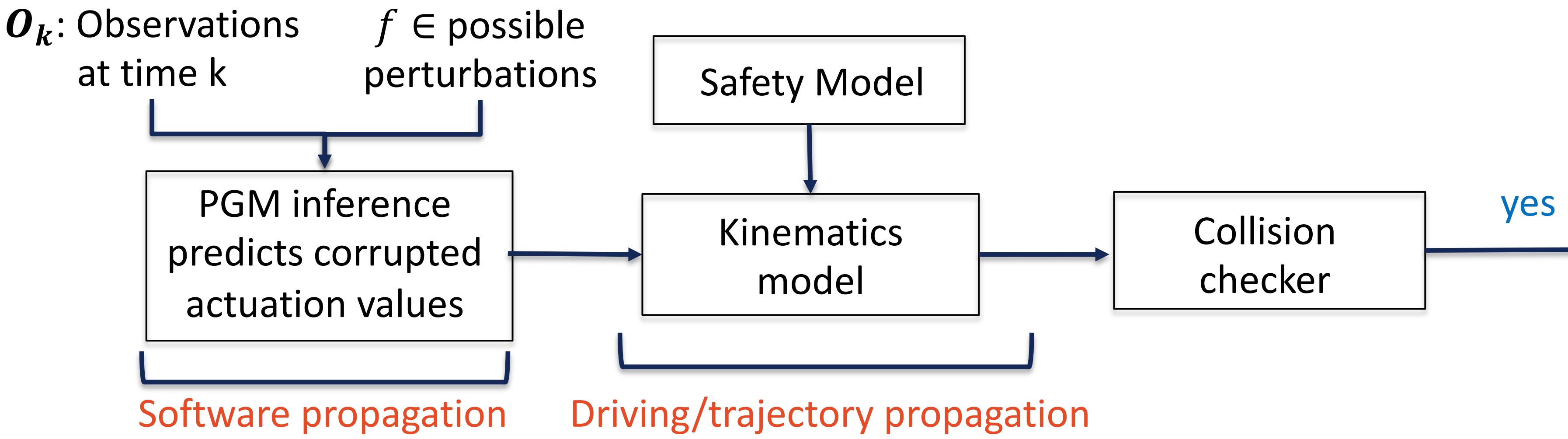
Fault injected simulation runs

Dataset

$$P(U,X,Y,Z,Q) = P(Z|Q,Y) \times P(Y|U,X) \times P(X|U) \times P(U) \times P(Q)$$

Learn Conditional Probability Table

S2: Predict outcome of fault injection using PGM inference


O_k: Observations at time k
f ∈ possible perturbations
Safety Model
PGM inference predicts corrupted actuation values
Kinematics model
Collision checker
yes
Software propagation
Driving/trajectory propagation

S3: Validate using real fault injection in simulation

Construct Probabilistic Graph Model

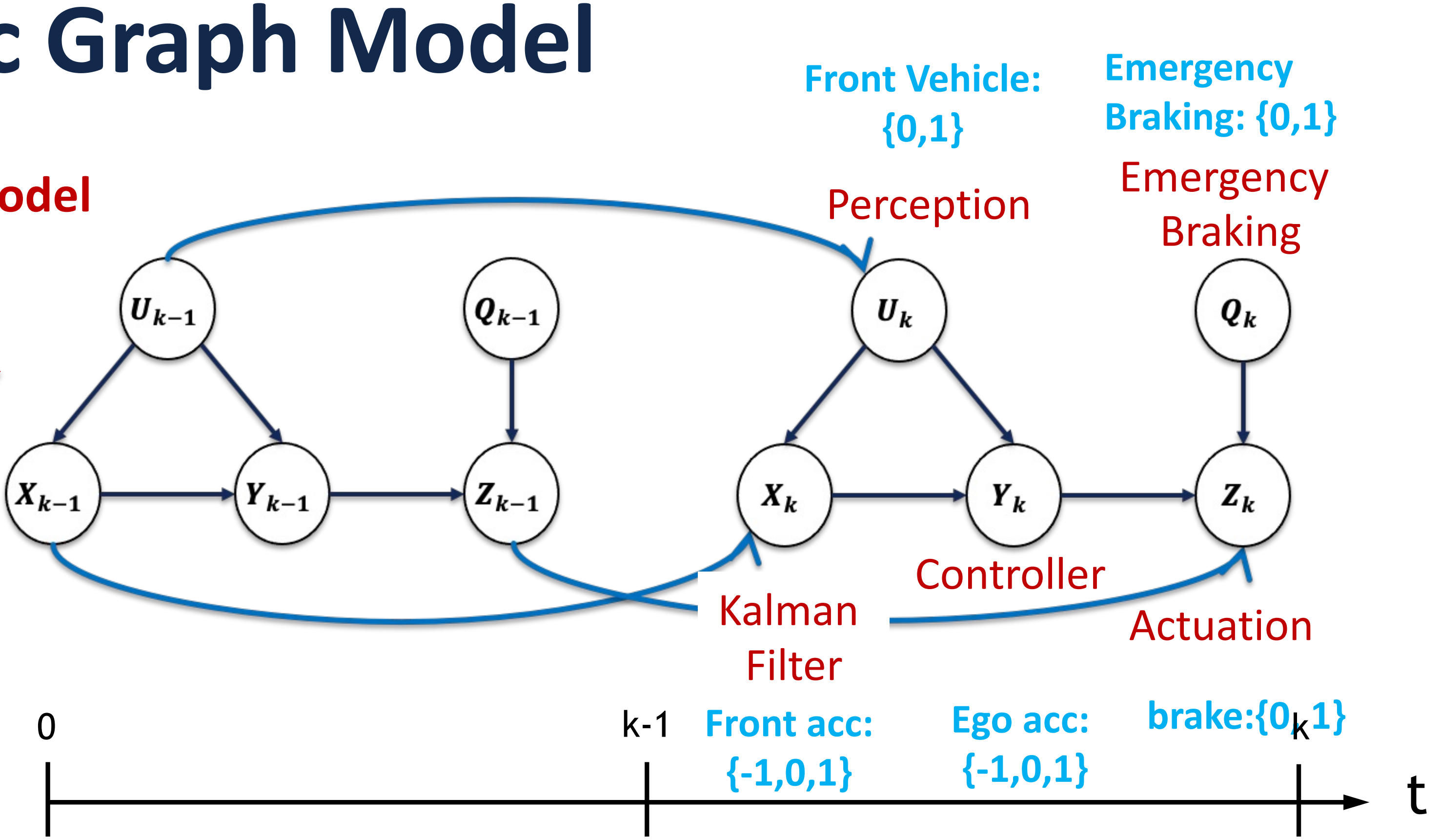
Step 1: Extract DFG and construct causal model

ADS Software

Static analysis

Code instrumentation

Leverage lineage and dependency information



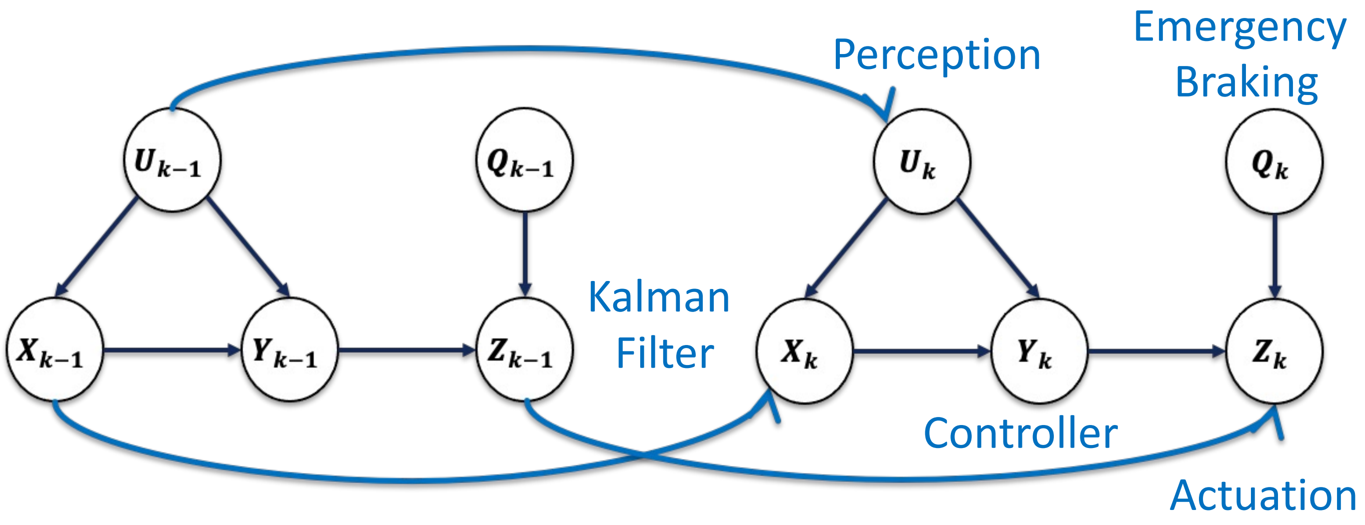
Step 2: Learn the parameters of the observational distribution

$$P(Z_k|Q_k, Y_k, Z_{k-1}) \times P(Y_k|U_k, X_k) \times P(X_k|U_k, X_{k-1}) \times P(U_k|U_{k-1}) \times P(Q_k) \times P(U_{k-1}) \times P(Q_{k-1})$$

Reduces state-space: Example, Z_k depends only on Z_{k-1}, Y_k and Q_k
Conditioning on Z_{k-1} and Y_k blocks the effect of other variables
If Y_k has no effect on Z_k=> X_k and U_k have no effect on Z_k

Training: Learning Conditional Distribution

Fill in node values from trace



Determine CPD values that will minimize difference between predicted and actual car measurements

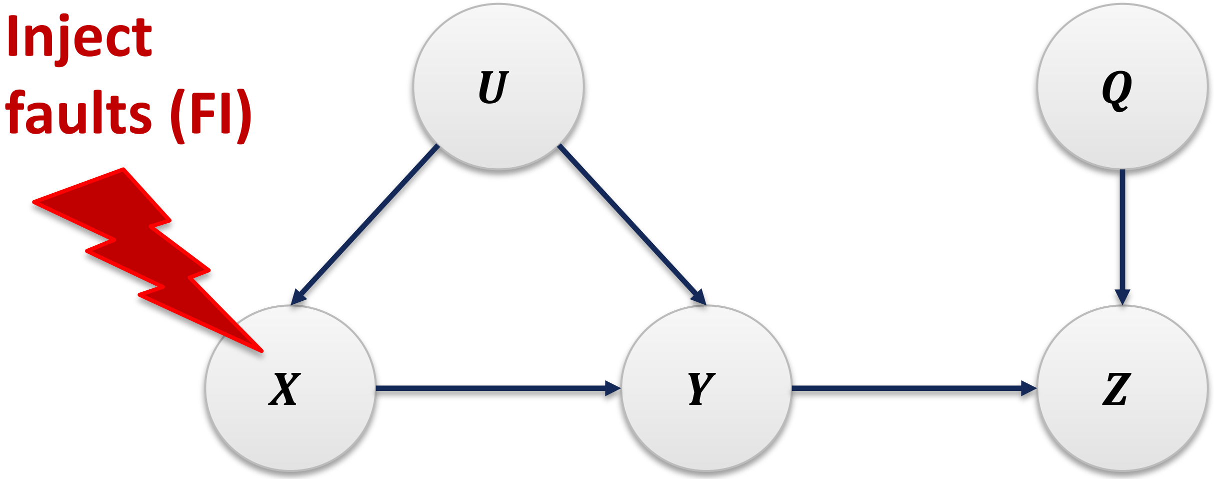
Gaussian distribution: Mean, variance

or

Multinomial: Normalized histogram

Conditional Probability Distribution: Parent to child matrix with probabilities (discretized estimation for continuous variables)

Inference: Fault injection as a conditional query

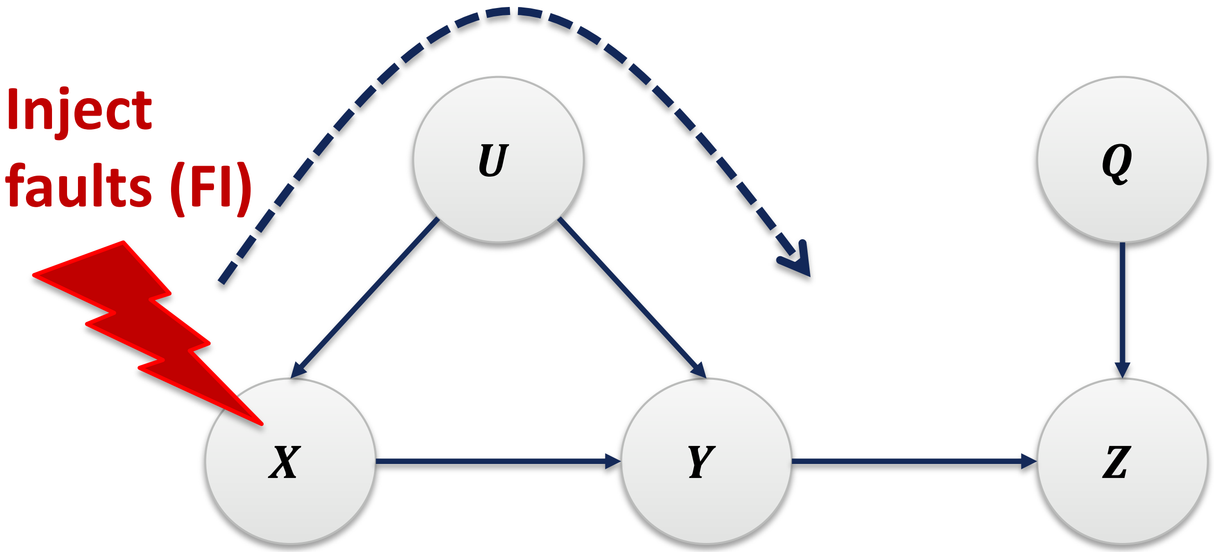


❑ Calculate conditional

$$P(Z|X='x')$$
$$= \sum_{U,Y,Q} P(Z|Q,Y) \times P(Y|U,X='x') \times P(X|U) \times P(U) \times P(Q)$$

No real fault injection is begin conducted on software or hardware!

Inference: Fault injection as a conditional query


Inject faults (FI)

□ Calculate conditional

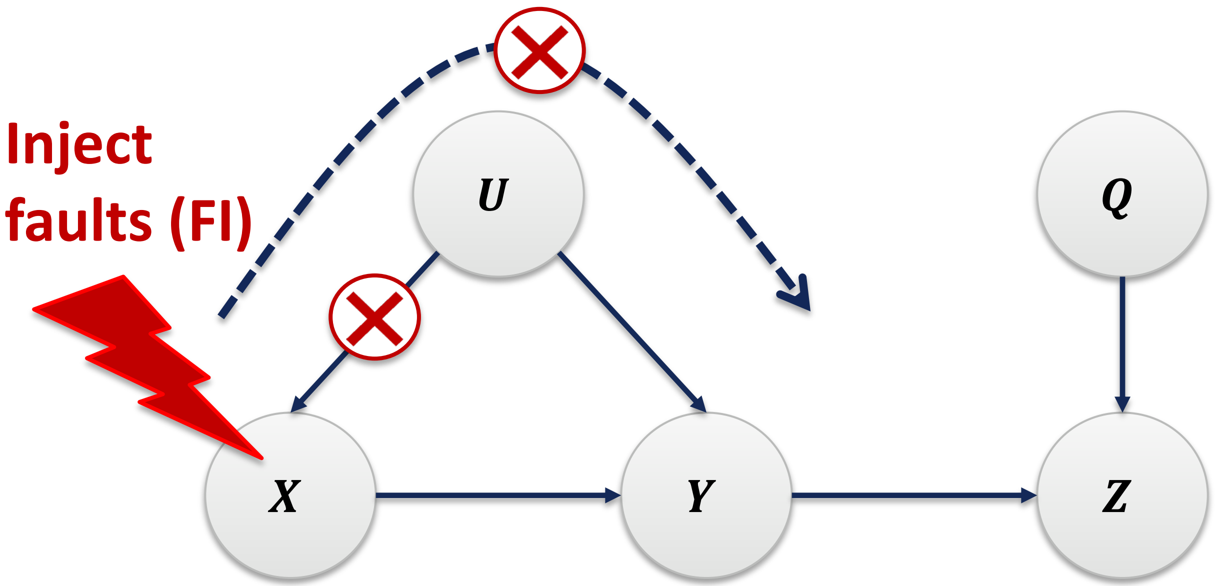
$$P(Z|X =' x')$$
$$= \sum_{U,Y,Q} P(Z|Q,Y) \times P(Y|U,X =' x') \times P(X|U) \times P(U) \times P(Q)$$

- ➢ Conditioning on X influences the distribution of U, which influences the distribution of Y
- ➢ In realty, FI does not influence U and only influences Y (as information does not flow backwards in function invocations)

Research challenge: adjust for confounding variables

Inference: Fault injection as an interventional query

Inject faults (FI)



Estimating intervention queries using observational data & causal analysis

- ❑ Leverage do-calculus [Pearl 95]
- ❑ Backdoor & front door criteria

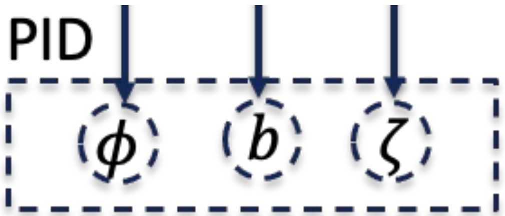
- Joint: $P(U,Y,Z,Q)|do(X=' x') = P(Z|Q,Y)\times P(Y|U,X=' x')\times \overset{P(X|U)}{1}\times p(U)\times P(Q)$
- $P(Z|do(X=' x')) = \sum_{U,Y,Q} p(Z|Q,Y) \times p(Y|U,X=' x')\times 1\times p(U)\times p(Q)$
- Leveraged Rule 2 of do-calculus (backdoor-criteria)

Modeling Propagation to Driving Behavior (forward simulation)

ϕ: *steer*

b: *brake*

ζ: *throttle*



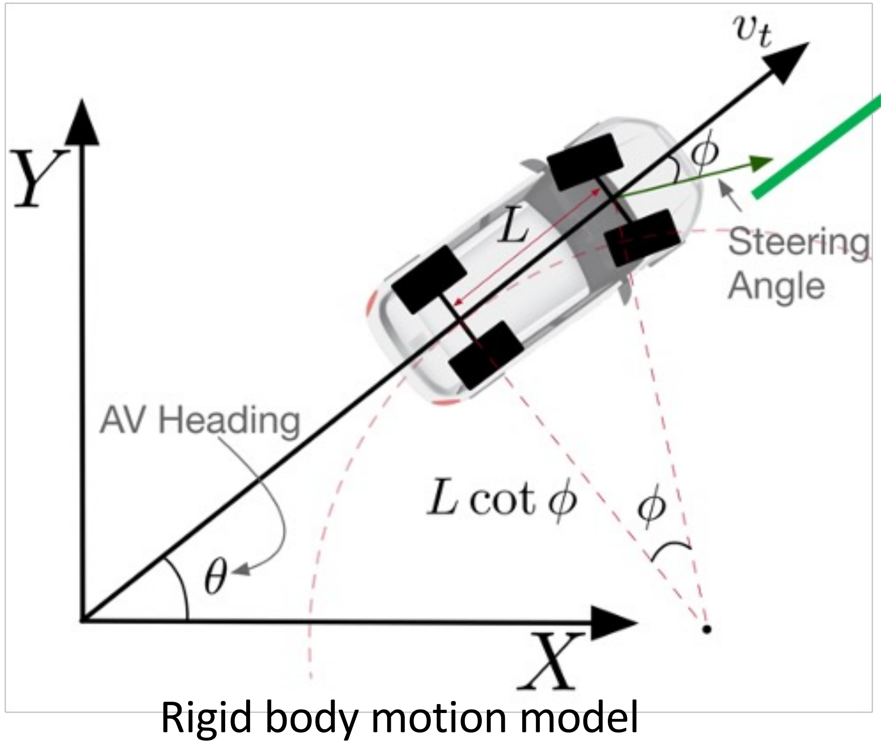
Kinematics model

$$v_t = f(\zeta_t, b_t, \phi_t)$$

$$dx_t/dt = v_t \cos\theta_t;$$

$$dy_t/dt = v_t \sin\theta_t$$

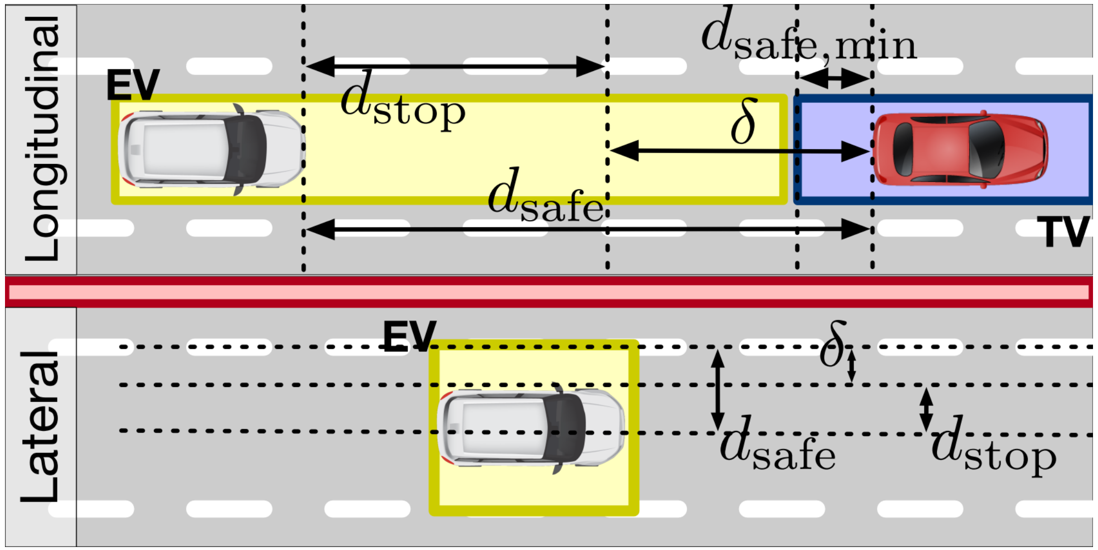
$$d\theta_t/dt = (v_t \tan\phi_t)/L,$$



Rigid body motion model

Forward simulation can be also modeled using NNs [DSN 2020]

Accounting for temporal compensation (Safety Model)



d_{safe}: Safe ego car distance from other objects

d_{stop}: Minimum distance ego car needs to stop

$\delta = d_{\text{safe}} - d_{\text{stop}}$

Emergency stop maneuver (assume max deceleration)

$$\left.\frac{dx_t}{dt}\right|_{t=t_{\text{stop}}} = 0 \text{ and } \left.\frac{dy_t}{dt}\right|_{t=t_{\text{stop}}} = 0.$$

$$\frac{dv_t}{dt} = -a_{\max} \text{ and } \frac{d\phi_t}{dt} = 0.$$

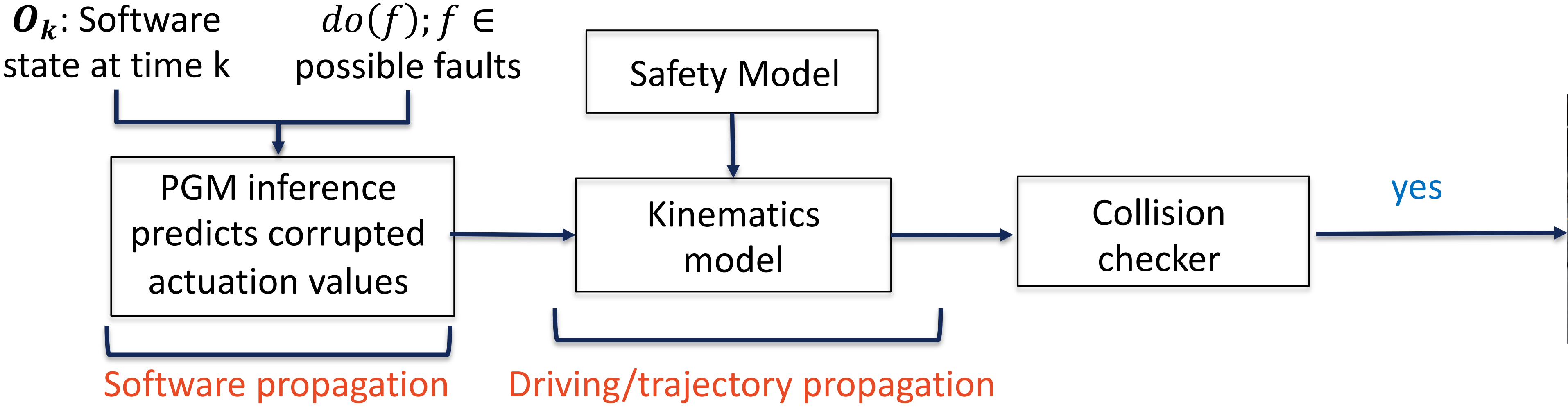
$$d_{\text{stop}} = \mathcal{P}(a_{\max}, v_0, \theta_0, \phi_0, x_0, y_0)$$

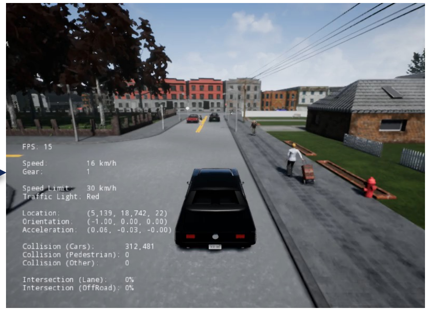
Other collision models: NVIDIA safety force field, Intel RSS

Summary: Inference and Validation

S2: Predict outcome of fault injection using PGM inference

S3: Validate using real fault injection in simulation


$\boldsymbol{O}_{\boldsymbol{k}}$: Software state at time k
$do(f); f \in$ possible faults
Safety Model
PGM inference predicts corrupted actuation values
Kinematics model
Collision checker
yes
Software propagation
Driving/trajectory propagation



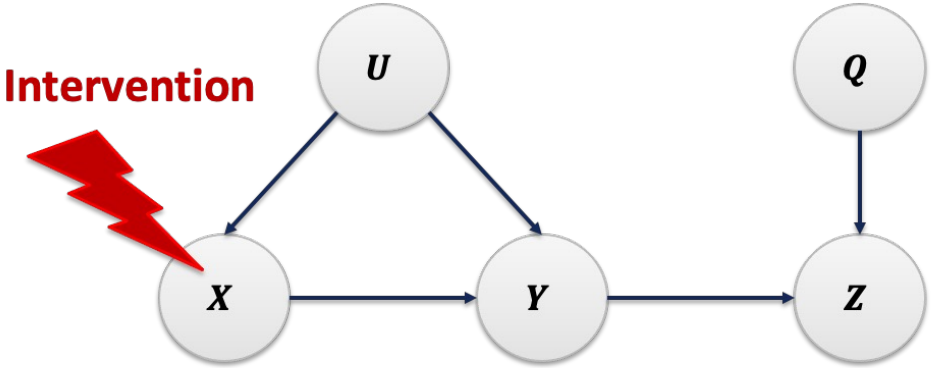
```
FPS: 15

Speed:          16 km/h
Gear:           1

Speed Limit:    30 km/h
Traffic Light:  Red

Location:       (5,139, 18,742, 22)
Orientation:    (-1.00, 0.00, 0.00)
Acceleration:   (0.06, -0.03, -0.00)

Collision (Cars):         312,481
Collision (Pedestrian):   0
Collision (Other):        0

Intersection (Lane):      0%
Intersection (OffRoad):   0%
```

Injection Targets: Fault Models Encapsulates What and Where

- Measured velocity (min, max)
- Tracked object velocity (max)
- Tracked object velocity (min)
- Tracked object acceleration (min)
- Tracked object acceleration (max)
- Measured acceleration (min)
- Measured acceleration (max)
- PID input (half)
- PID input (double)
- Throttle (min)
- Throttle (max)
- Steer (min)
- Steer (max)
- Brake (min)
- Brake (max)
- Obstacle classification (disappear)
- Obstacle classification (vehicle)
- Obstacle classification (pedestrian)
- Obstacle classification (cyclist)
- Obstacle distance (min)
- Obstacle distance (max)
- Camera object classification (disappear)
- Camera object classification (vehicle)
- Camera object classification (pedestrian)
- Camera Obstacle distance (min)
- Camera Obstacle distance (max)
- Camera object classification (cyclist)
- LiDAR classification (disappear)
- LiDAR classification (vehicle)
- LiDAR classification (pedestrian)
- LiDAR classification (cyclist)
- LiDAR Obstacle distance (min)
- LiDAR Obstacle distance (max)
- Lane disappear
- Lane type (dashed)
- Lane type (solid)
- Lane width (double)
- Lane width (half)
- Camera frame (noise)
- LiDAR frame (noise)

41 fault types in total

Results



- ❑ Modeling data/control flow as probabilistic graphs
- ❑ Model faults as interventions
- ❑ Leverage forward simulation techniques for checking safety

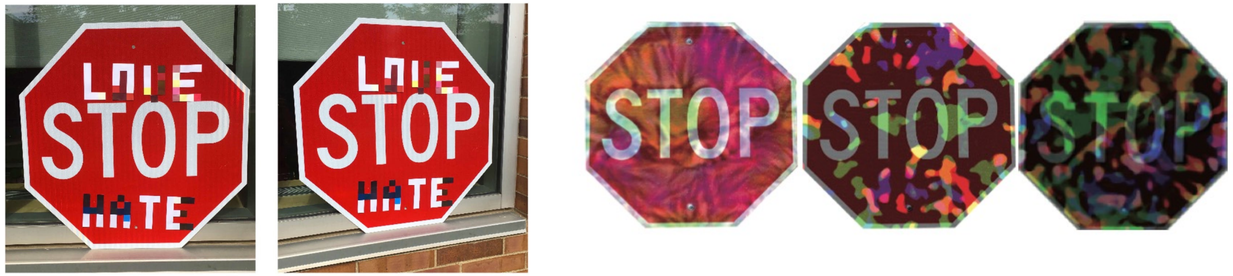
Results:

- ✓ Demonstrated on Baidu Apollo AV agent
- ✓ Able to find 561 faults that lead to collisions (found 0 via random sampling); >3690x speed up over enumeration
- ✓ Replicated 460 (82%) faults in “software-in-the-loop” simulation

ML-driven Malware that Targets AV Safety, DSN 2020

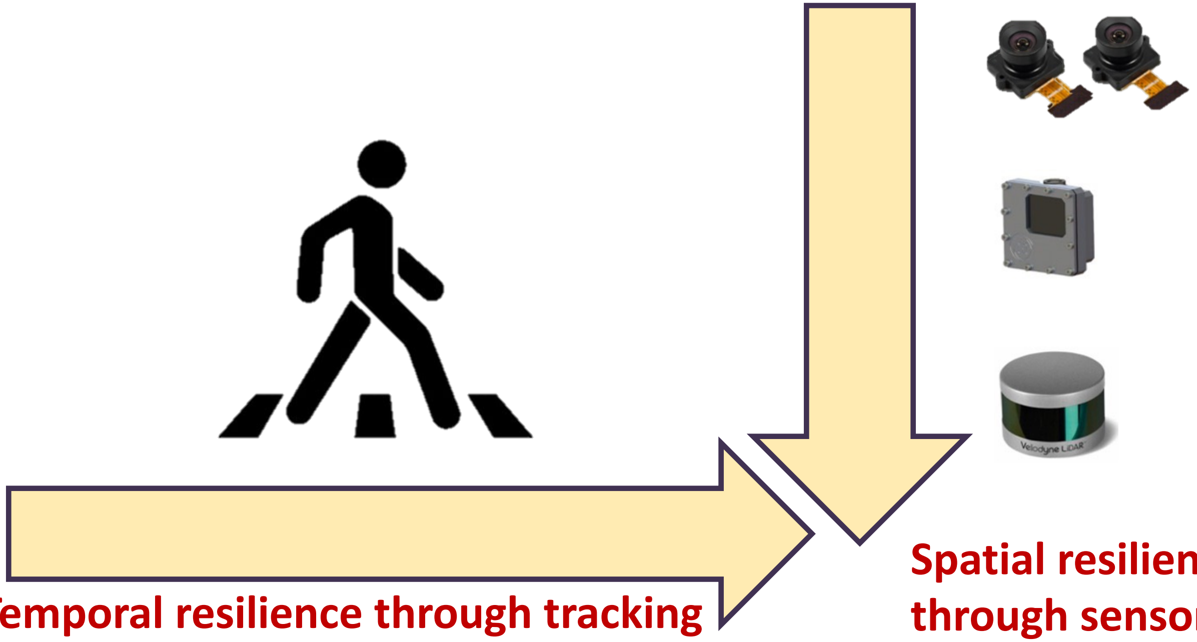
Curious Case of Stop Signs

- **Most attacks focus on altering detection outputs**
  - Misclassification or misdetection of objects
  - Boloor et al., Eykholt et al., and Lu et al.



Stop signs can be fooled

- **Low chance of causing safety hazards**
  - Kalman filter or other state-space model
    - Redundancy from multiple sensors (sensor fusion)
    - Redundancy in time (continuous tracking)
  - Timing of attack



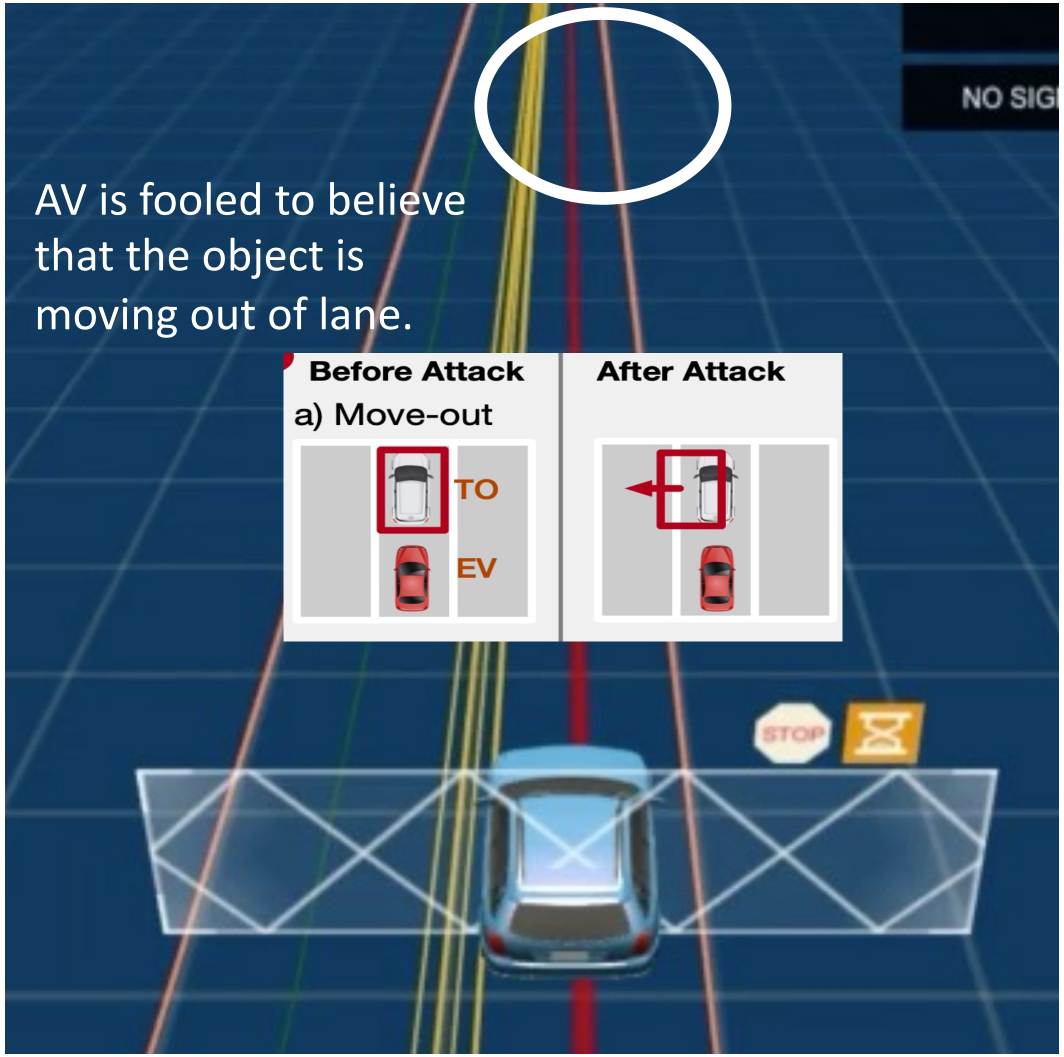
Temporal resilience through tracking

Spatial resilience through sensor fusion

Our Attack: Situationally Aware ML-driven Malware

Malware characteristics

- **End-to-end attack --- perturbs Kalman Filter to overcome both temporal and spatial resilience**
- **Masquerade attacks as failures**
- **Leads to safety hazards --- collisions and emergency braking**



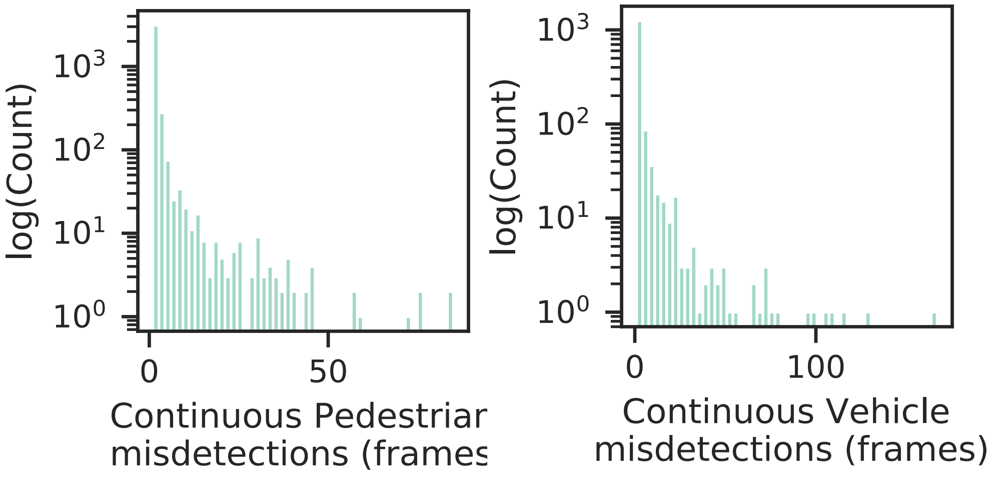
AV is fooled to believe that the object is moving out of lane.

Demonstration using **Apollo** autonomous driving agent and **LGSVL** world simulator

Key Insight

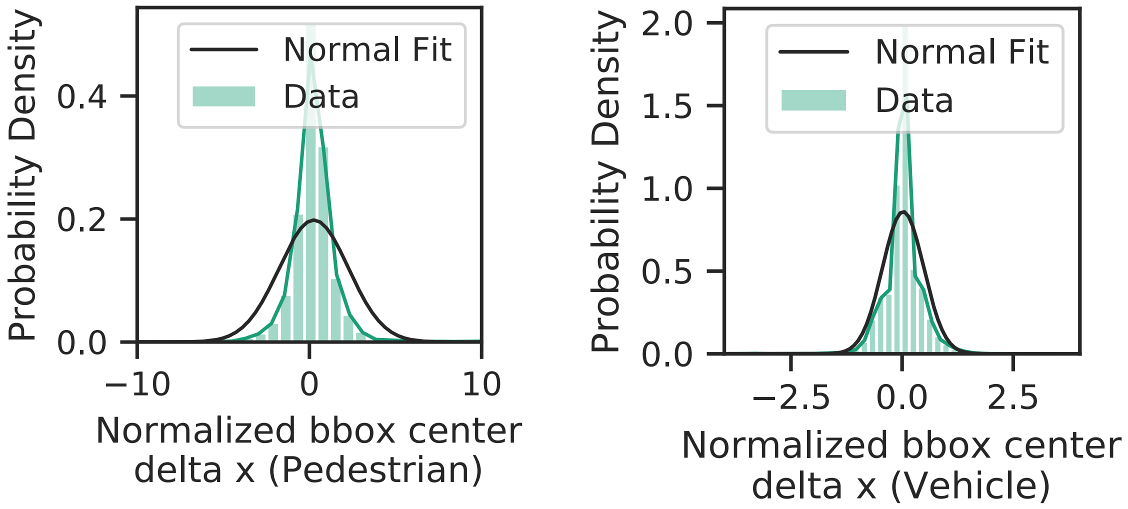
- State-of-the-art object detectors (Yolo and FCNN) are highly noise
    - **99th %ile continuous misdetection rate for pedestrian is 31 and vehicles is 59.4 frames**
    - **Position estimates (i.e, bounding box) can be miscalculated by up to 10 m**

- Perception systems rely on Kalman filters and sensor fusion models to compensate for errors/noise



Misdetections

Bounding box errors

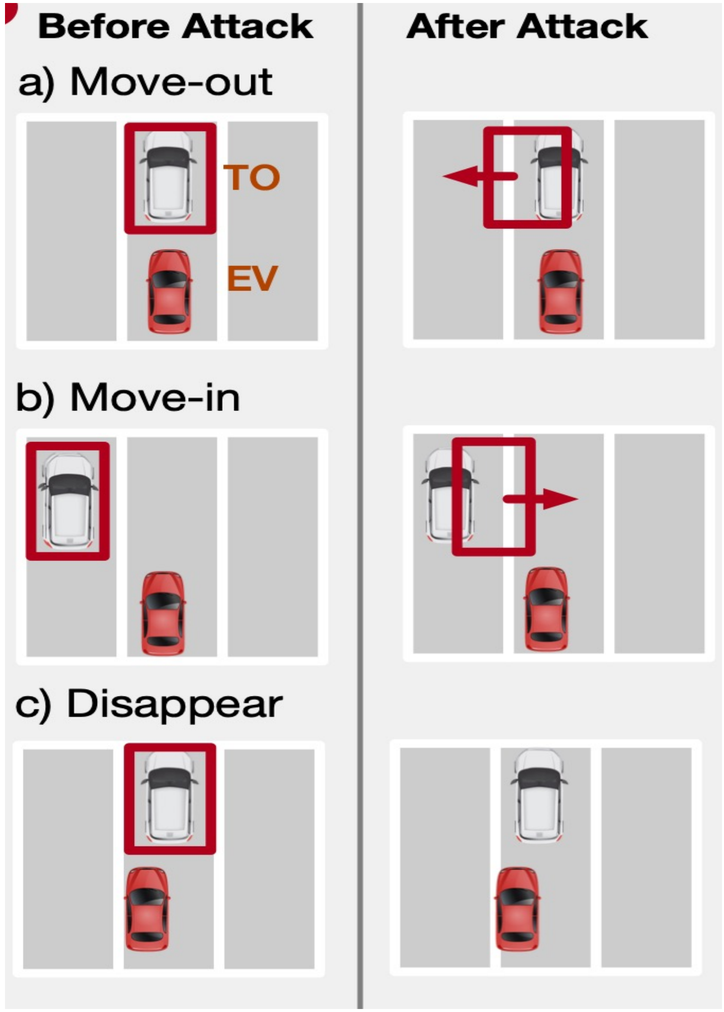


Launching Attacks is Not Easy!

- Gain access to the system
- Persist on the system without getting notice
- Monitor the state of the system to launch the attack
- Execute the attack such that safety hazard occurs
- Hold the attack for sufficiently small duration to evade detection and mitigation

Situationally Aware ML-driven Malware

Attack Goal



Methodology

What to attack ?

Perception module to alter perceived object trajectories

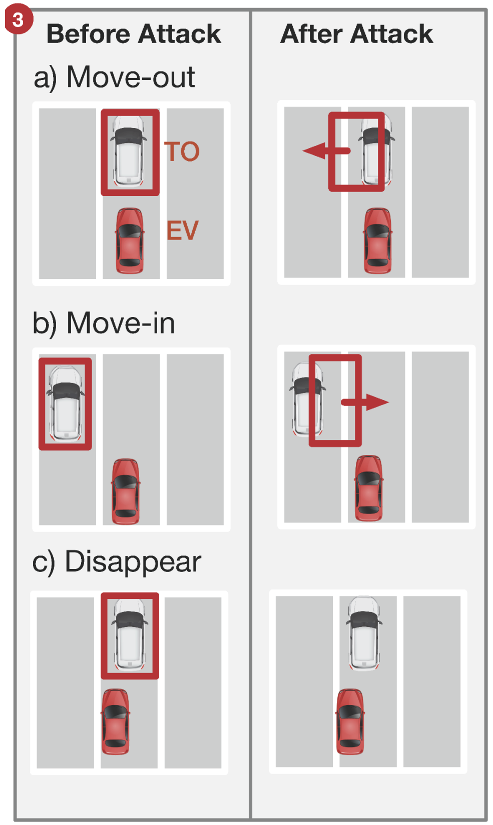
How to attack ?

Perturbs either camera pixels or object detection outputs

When to attack ?

Safety potential is low, and time needed to cause safety hazard is minimal.

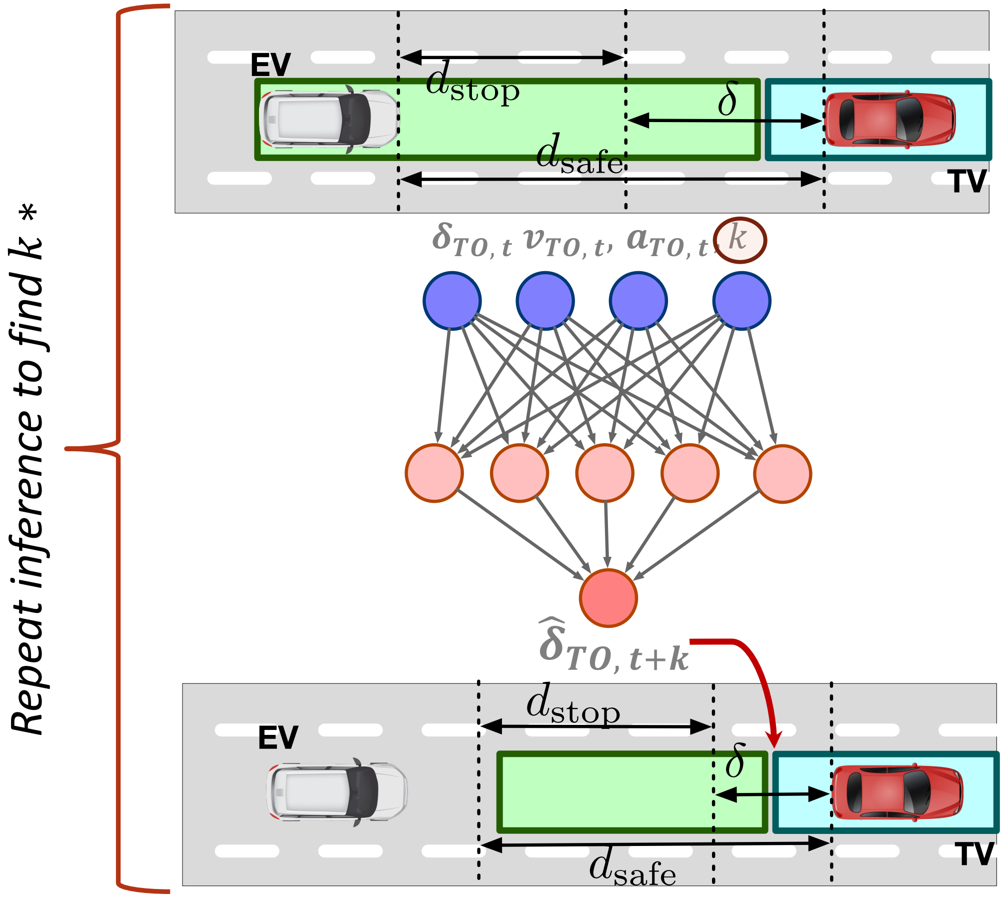
Step 1: Scenario Matching (What to attack ?)



- Infer the closest-in-path-object or target object (CIPO or TO)'s position and trajectory
- Choose the attack vector based on the table as follows
- Provides **Situational Awareness**

Target object trajectory	Target object location	
	AV lane	Non-AV lane
Moving in	---	Move_Out/Disappear
Lane Keeping	Move_Out/Disappear	Move_In
Moving out	Move_In	---

Step 2: Safety Hijacking (When to attack ?)


Repeat inference to find k *
EV
TV
d_{stop}
δ
d_{safe}
$\boldsymbol{\delta}_{\boldsymbol{TO},\,\boldsymbol{t}}\ \boldsymbol{v}_{\boldsymbol{TO},\,\boldsymbol{t}},\ \boldsymbol{a}_{\boldsymbol{TO},\,\boldsymbol{t}},\ k$
$\widehat{\boldsymbol{\delta}}_{\boldsymbol{TO},\,\boldsymbol{t+k}}$
EV
TV
d_{stop}
δ
d_{safe}

- Safety hijacking uses a 3-layer-fully-connected neural network to decide the attack launch time
  - **Learns:** ADS’s behavior and change in safety potential $\widehat{\boldsymbol{\delta}}$
  - **Input:** Target object (TO) kinematics (v, a) , safety state (δ) and attack duration (k)

$$input = (\boldsymbol{\delta}_{\boldsymbol{TO},\,\boldsymbol{t}}\ \boldsymbol{v}_{\boldsymbol{TO},\,\boldsymbol{t}},\ \boldsymbol{a}_{\boldsymbol{TO},\,\boldsymbol{t}},\ k)$$

  - **Output:** predicted safety potential after the attack

$$output = FCNN(input) = \ \widehat{\boldsymbol{\delta}}_{\boldsymbol{TO},\,\boldsymbol{t+k}}$$

- Optimal attack duration (k^{*}) search:
  - Given $input$, $FCNN$ model, and *targeted* safety distance $\boldsymbol{\delta}_{h}$, find the optimal k^{*}

$$k^{*} = min\,\{k\ |\widehat{\boldsymbol{\delta}}_{\boldsymbol{TO},\,\boldsymbol{t+k}} < \boldsymbol{\delta}_{\boldsymbol{h}}\}$$

Step 3: Trajectory Hijacking (How to attack ?)

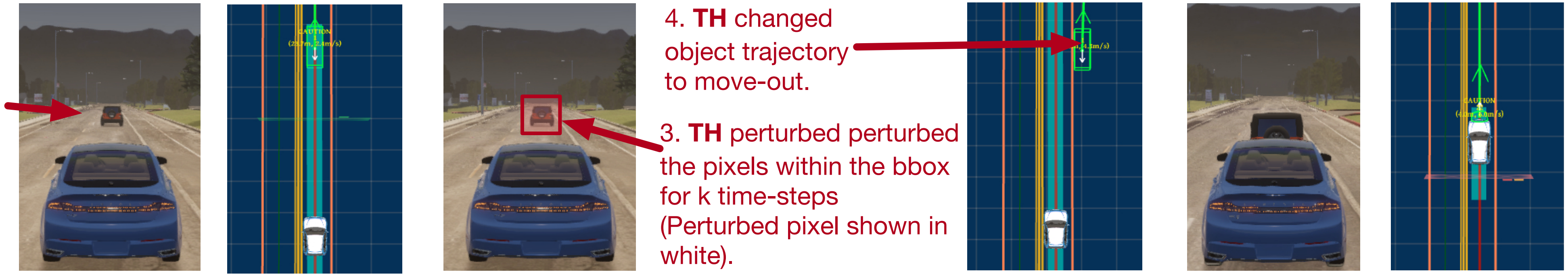
- Perturbs camera feed around target object (TO) with adversarial patches for k^* time steps to impact sensor fusion outputs
  - Modifies perceived object trajectories and impacts decision making
- Disguise such perturbation as noise
  - Small shift for **Move_In** and **Move_Out** to avoid tracker evasion (within detector noise characteristics)
- Trajectory hijacking can be skipped if attacker has access to onboard software

Attacking Kalman Filters (refer to the paper for details)

$$\begin{aligned} \underset{\vec{\omega}_t}{max}\ \mathbf{M}\ (o_t^i + \vec{\omega}_t,\ \hat{s}_{t-1}^i) \\ s.t.\ \mathbf{M} \leq \lambda, \\ IoU(o_t^i + \vec{\omega}_t,\ \text{patch}) \geq \gamma, \\ \vec{\omega}_t \in [\mu - \sigma, \mu + \sigma] \end{aligned}$$

1. **SM** chooses *move-out* attack vector as the target object is in-path in lane keep mode.

2. **SH** identifies the current state and time optimal for attack.



4. **TH** changed object trajectory to move-out.

3. **TH** perturbed perturbed the pixels within the bbox for k time-steps (Perturbed pixel shown in white).

(a) simulation view

(b) ADS view

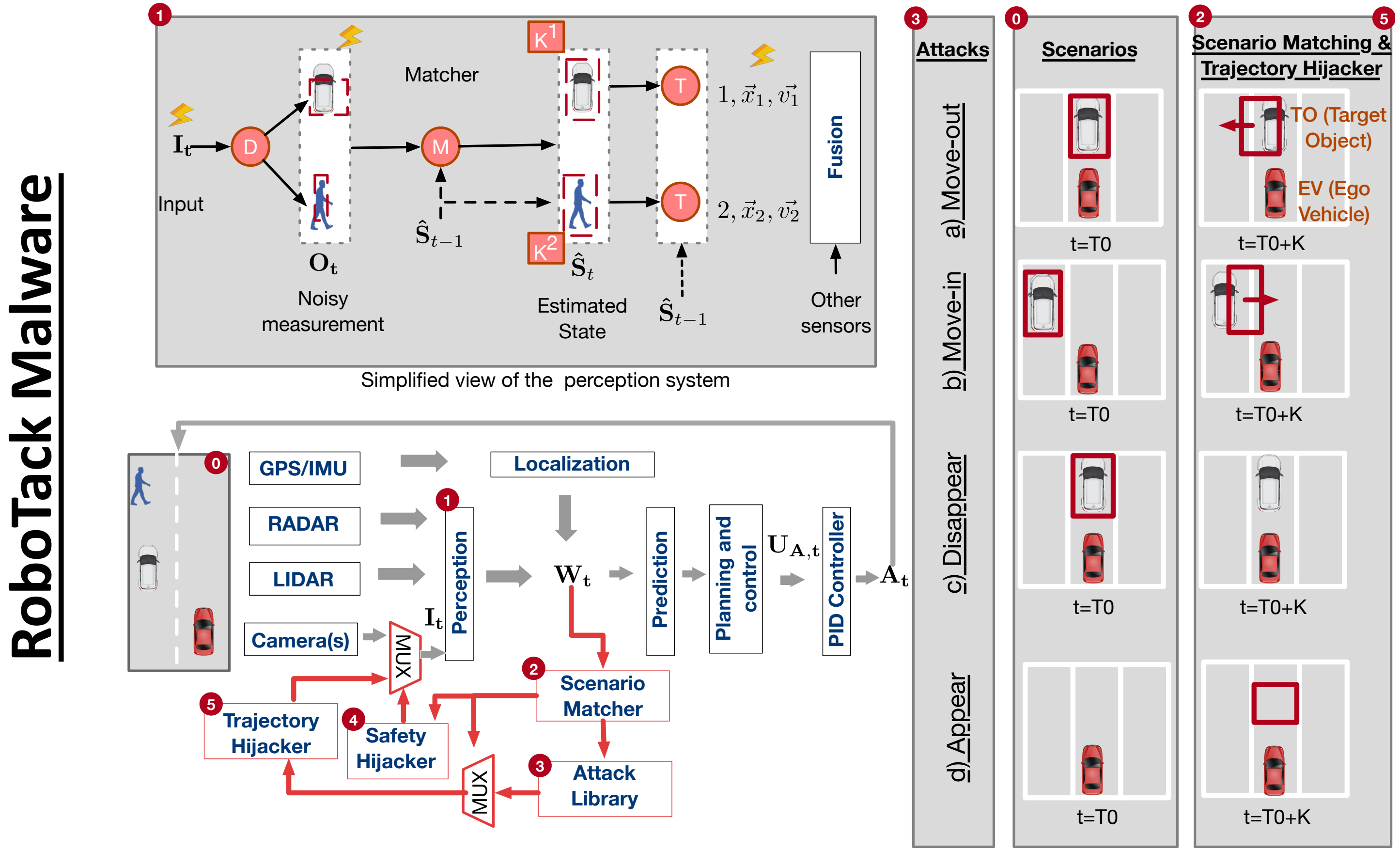
(c) simulation view (Perturbed pixels)

(d) ADS view (Hijacked trajectory)

(e) Simulation view (collision)

(f) ADS view (collision)

RoboTack: Putting it all together

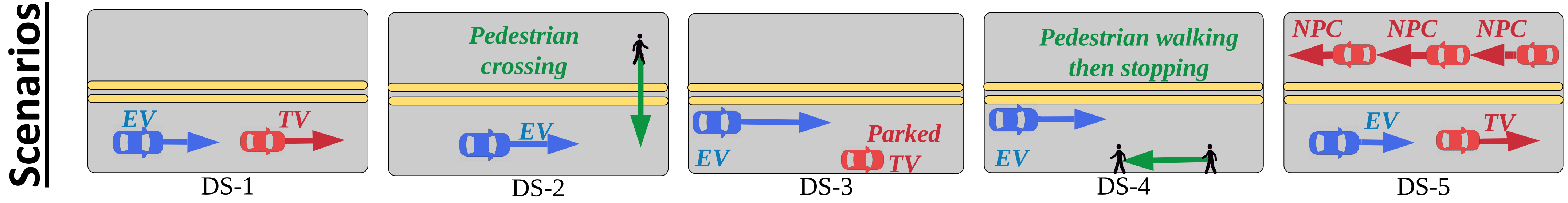


Evaluation platform

- Apollo 5.0 (AI agent)
- LGSVL 2019.2 (world-simulator)

Experiment details

- ~260 hours of simulation on 3 machines
- ~1400 evaluation experiments
- 5 driving scenarios



Evaluation and Results

Scenario ID	k^* (# frames)	% Emergency Braking			% Crashes		
		RoboTack	Scenario matching + Trajectory hijacking	Baseline	RoboTack	Scenario matching + Trajectory hijacking	Baseline
Vehicle - Move_Out	65	**37.3**	7.3	2.3%	**17.3**	2.8	0%
Pedestrian -Move_Out	32	**97.8**	6.6		**84.1**	3.5	

Baseline attack (random object, random duration, random time)

Note: complete evaluation on RoboTack performance of all the scenarios is in the paper

- Highly efficient in targeting AV safety compared to state-of-the-art methods; across all experiments
  - caused **forced emergency braking in 75.2% cases (33x more),** and
  - caused **accident in 52.6% of cases** compared to zero accidents

- Highlights the need for assuring safety of AI systems
  - Temporal and spatial resilience is not sufficient
  - Safety oriented co-design of AI and systems

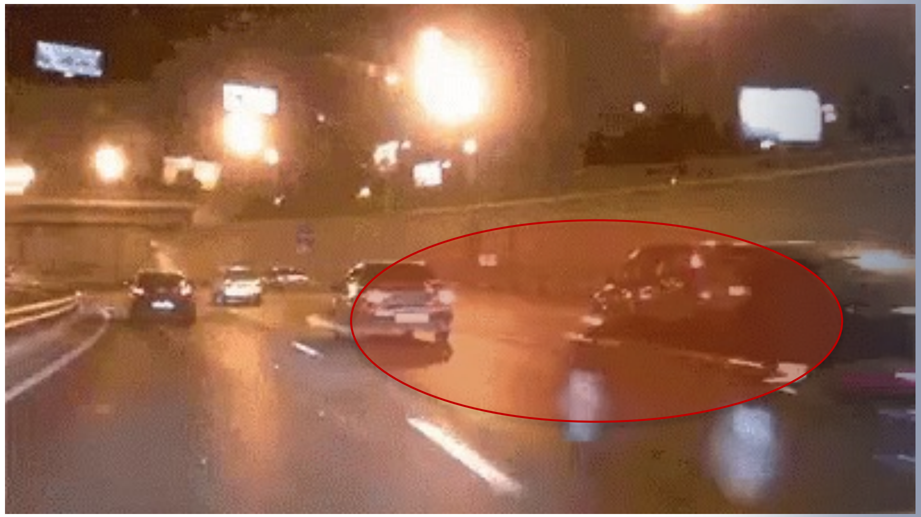
Q1: What vulnerabilities exist in Autonomous Vehicles?

Q2: Are these vulnerabilities worse than existing vulnerabilities in non-Avs?

Q3: Can these vulnerabilities be used for attacks?

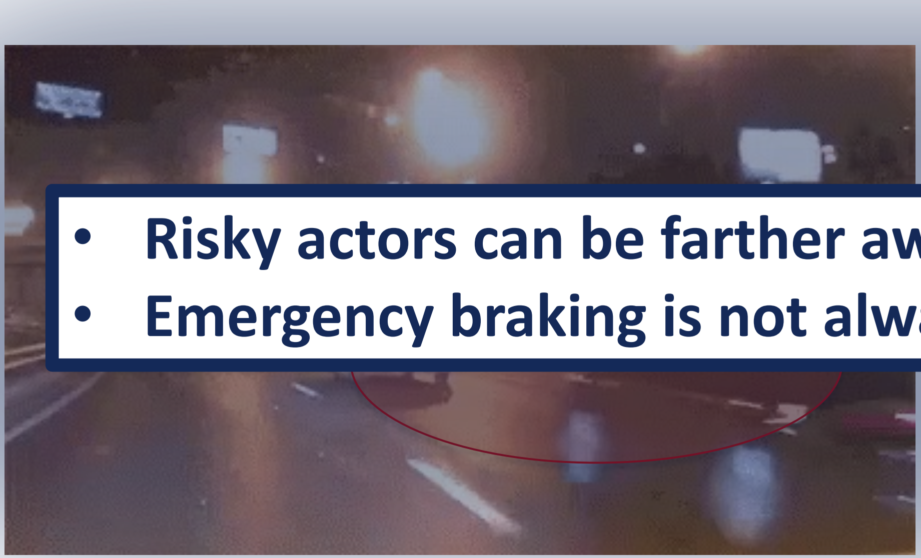
Q4: How do assess the threat at runtime for safety?

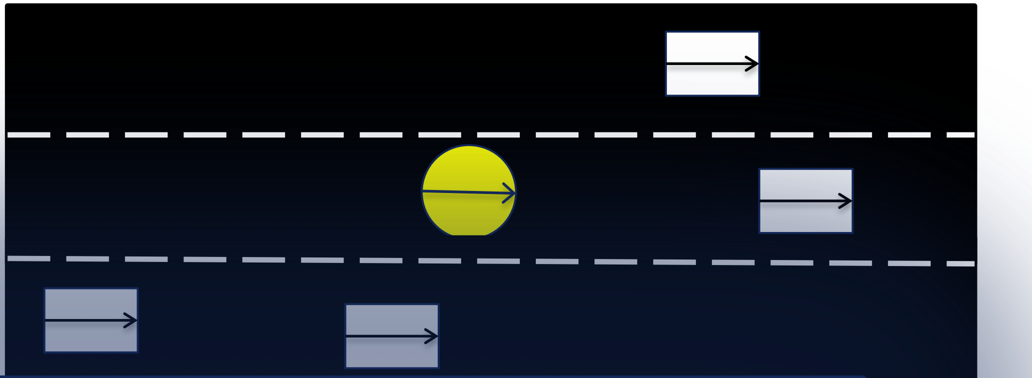
Watch Out for the Safety Threatening Actors: How do we make AVs safer?



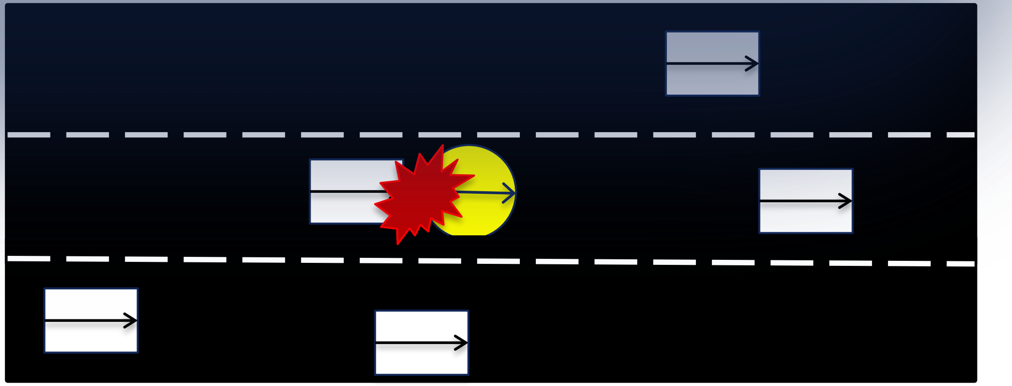
- Driving is invariably a risky task
- Availability of backup plans (multiple choices) key to dealing with threat posed by actors / what if analysis
- Predicting safety threatening actors is the key to safety assessment and mitigation

Watch Out for the Safety Threatening Actors: How do we make AVs safer?





- Risky actors can be farther away from the Ego actor
- Emergency braking is not always the safe solution



Only choice is to steer right

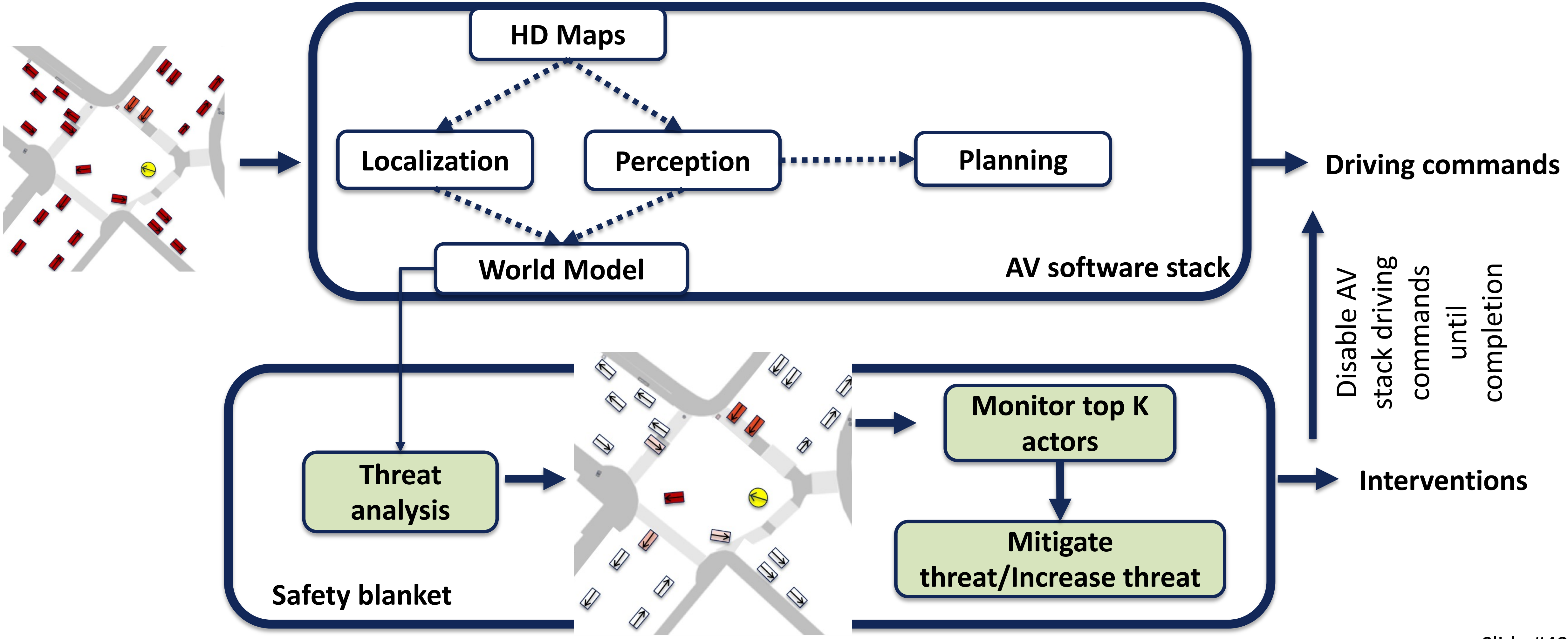
● AI vehicle

▭ Other actors

Humans continuously assess the importance of each actor and the risk posed by them while driving!

Question: How do we bring a similar actionable intelligence to autonomous driving?

Our Solution: Threat-aware Safety Blanket



HD Maps

Localization

Perception

Planning

World Model

AV software stack

Driving commands

Disable AV stack driving commands until completion

Threat analysis

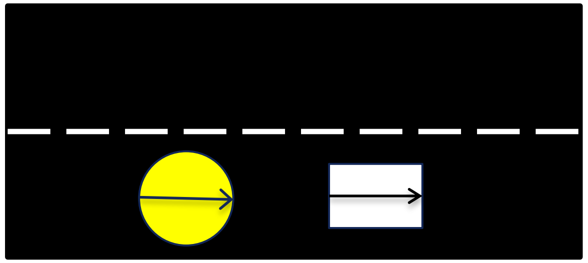
Monitor top K actors

Mitigate threat/Increase threat

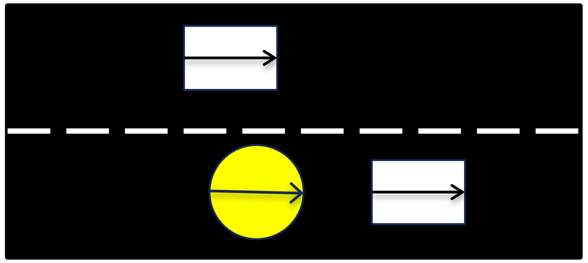
Safety blanket

Interventions

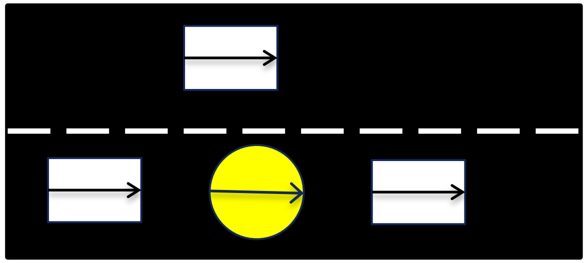
How Do We Assess Threat?



Possible trajectories:
- Follow vehicle
- Move to left lane
- Brake



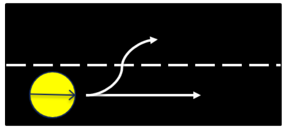
Possible trajectories:
- Follow vehicle
- ~~Move to left lane~~
- Brake

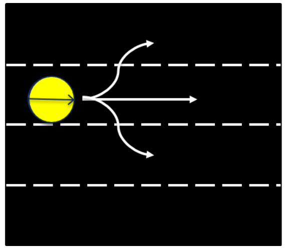


Possible trajectories:
- Follow vehicle
- ~~Move to left lane~~
- ~~Brake~~

Attention required to drive increases from left to right

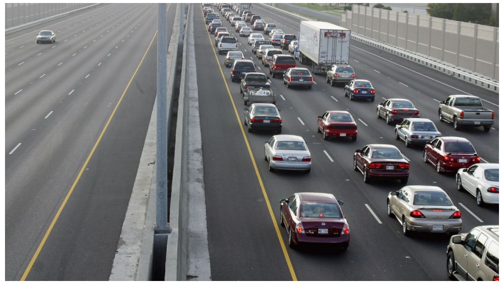
Intuition: available trajectories/drivable area decreases





#diverse trajectories also depends on the map

How Do We Assess Threat?



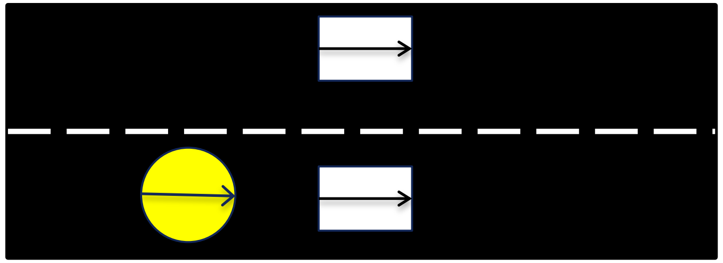
Attention required increases with the increase in uncertainty of another actor’s behavior

Actor’s Influence on Decision Making

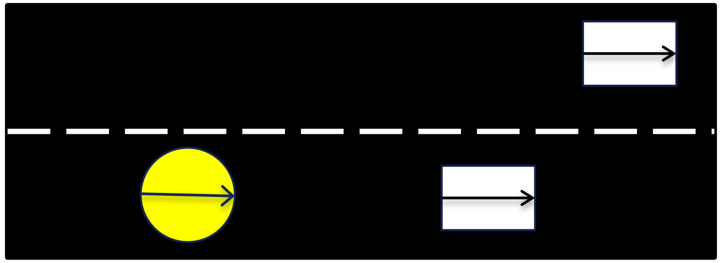
Current state

Predicted/given future states

Driving Scenario

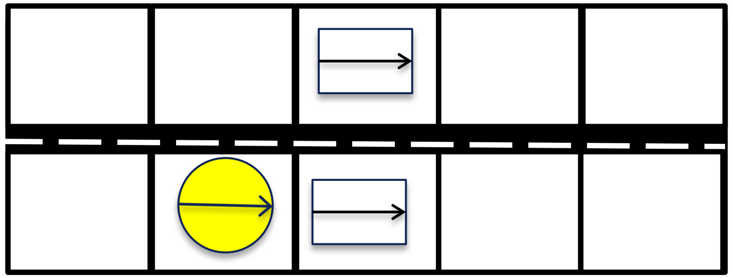


T

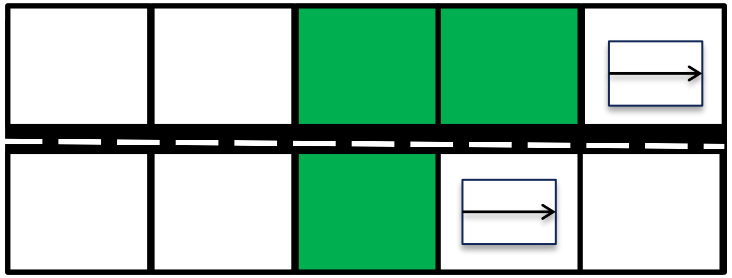


T+1

Internal representation



T

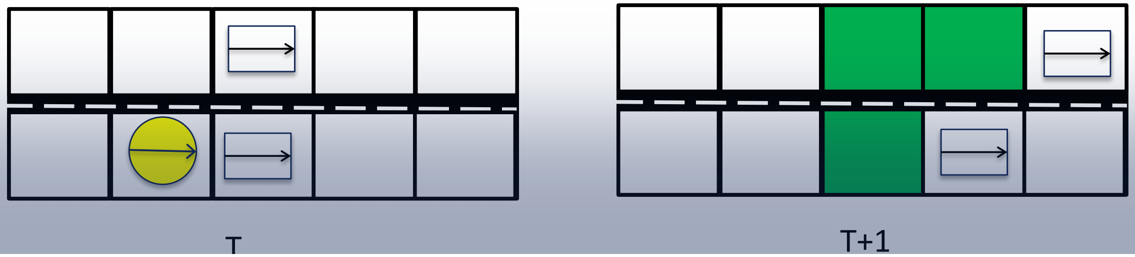


T+1

Actor’s Influence on Decision Making

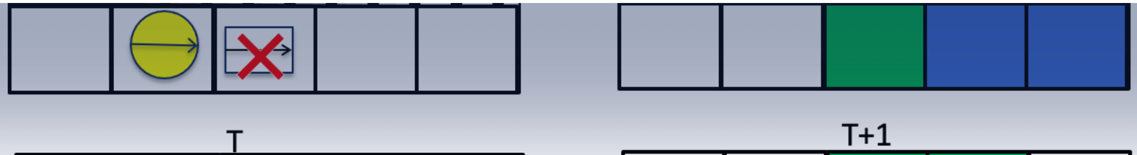
- Quantifying decrease in diverse trajectories due to an actor

Original driving scenario and trajectories



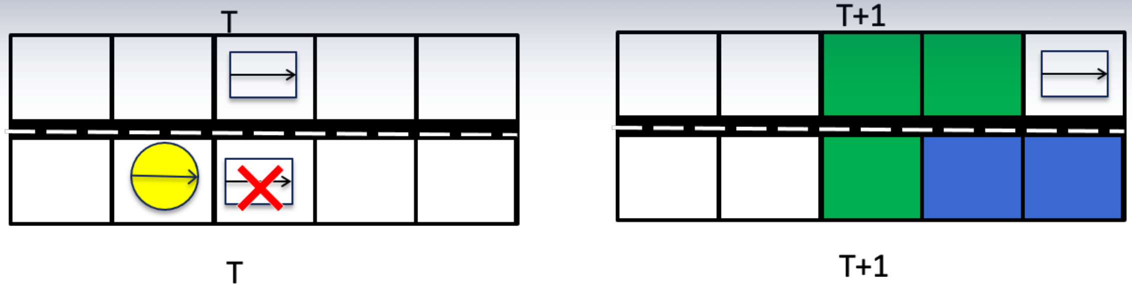
$$\textbf{Reduction in diverse trajectories} = \frac{\text{(total \#trajectories after deleting the actor(s)) – (total \#trajectories before deleting the actor)}}{\text{total \#trajectories after deleting all actors}}$$

Deleting all actors adds three extra trajectories



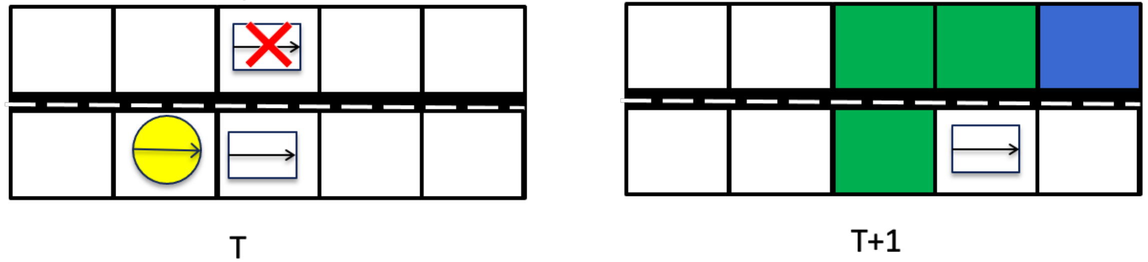
Deleting bottom actor adds one extra trajectory

More risky



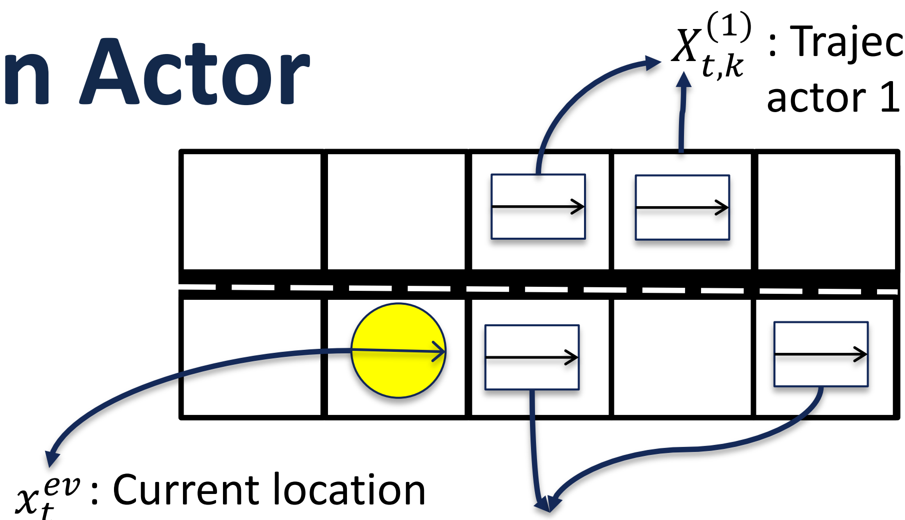
Deleting top actor adds two extra trajectories

Less risky



Formalizing Threat Posed by an Actor

Defining driving scenario: $\mathbb{S} =< \mathbb{M}, \mathbb{X}_{t,k}, x_t^{ev} >$



$X_{t,k}^{(1)}$: Trajectory of actor 1

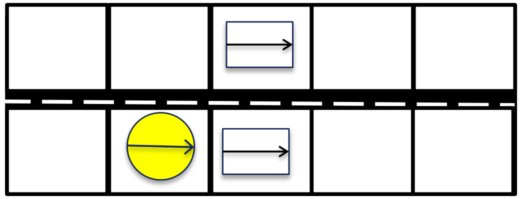
x_t^{ev} : Current location of AI actor

$X_{t,k}^{(2)}$ Trajectory of actor 2

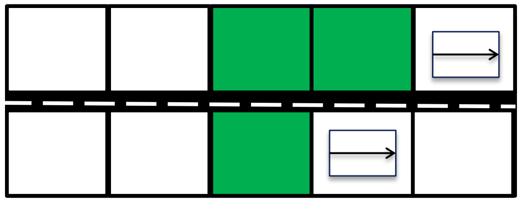
Possible driving trajectories from t to $t + k$:

$$\mathbb{Z}_{t,k} = f_p(\mathbb{M}, \mathbb{X}_{t,k}, x_t^{ev})$$

Planner that finds all reachable cells (we use RRT*)



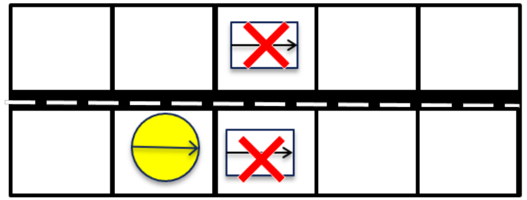
T



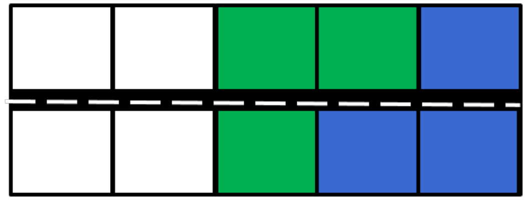
T+1

Possible driving trajectories when no actors are present :

$$\mathbb{Z}_{t,k}^{\phi} = f_p(\mathbb{M}, \phi, x_t^{ev})$$



T



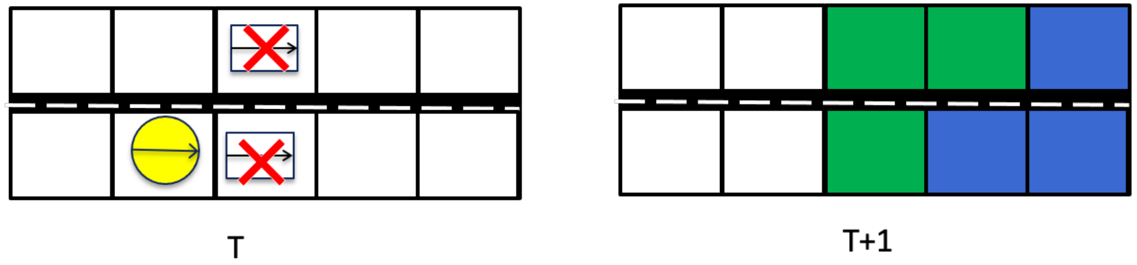
T+1

LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." (1998): 98-11.
Karaman, Sertac, et al. "Anytime motion planning using the RRT*." ICRA 2011.

Formalizing Threat Posed by an Actor

Quantifying total reduction in driving trajectories:

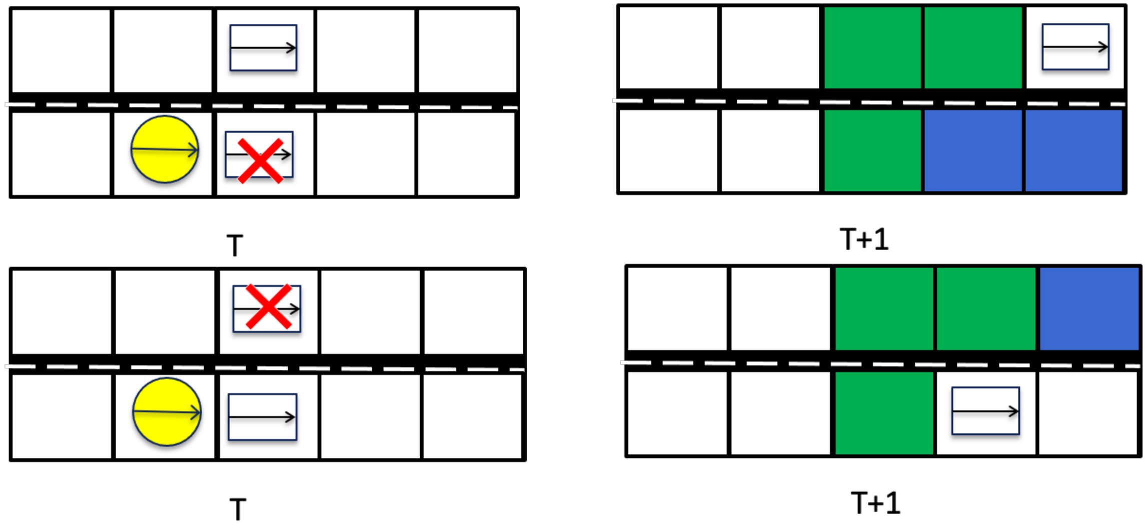
$$\rho_{t,k} = \frac{\mathbb{Z}^{\emptyset}_{t,k} - \mathbb{Z}_{t,k}}{\mathbb{Z}^{\emptyset}_{t,k}}$$



Quantifying reduction in driving trajectories due to an actor:

$$\mathbb{Z}_{t,k} = f_p(\mathbb{M}, \mathbb{X}_{t,k}, x^{ev}_t)$$

$$\rho^{i}_{t,k} = \frac{\mathbb{Z}^{/i}_{t,k} - \mathbb{Z}_{t,k}}{\mathbb{Z}^{\phi}_{t,k}}$$



Accounting for Uncertainty

Joint distribution modeling the uncertainty in the estimates

N actors

$$\bar{X}^{(i)}_{t,t+k} = p(\bar{x}^{(i)}_{t}, ..., \bar{x}^{(i)}_{t+k} | \underbrace{o^{(1)}_{1}, ..., o^{(N)}_{t}}_{\text{Output of DNN-based object detection model}})$$

Sample trajectories from the joint distribution and calculate possible set of future driving trajectories for the Ego actor

Risk with no uncertainty in future trajectories:

$$\rho^{i}_{t,k} = \frac{\mathbb{Z}^{/i}_{t,k} - \mathbb{Z}_{t,k}}{\mathbb{Z}^{\phi}_{t,k}}$$

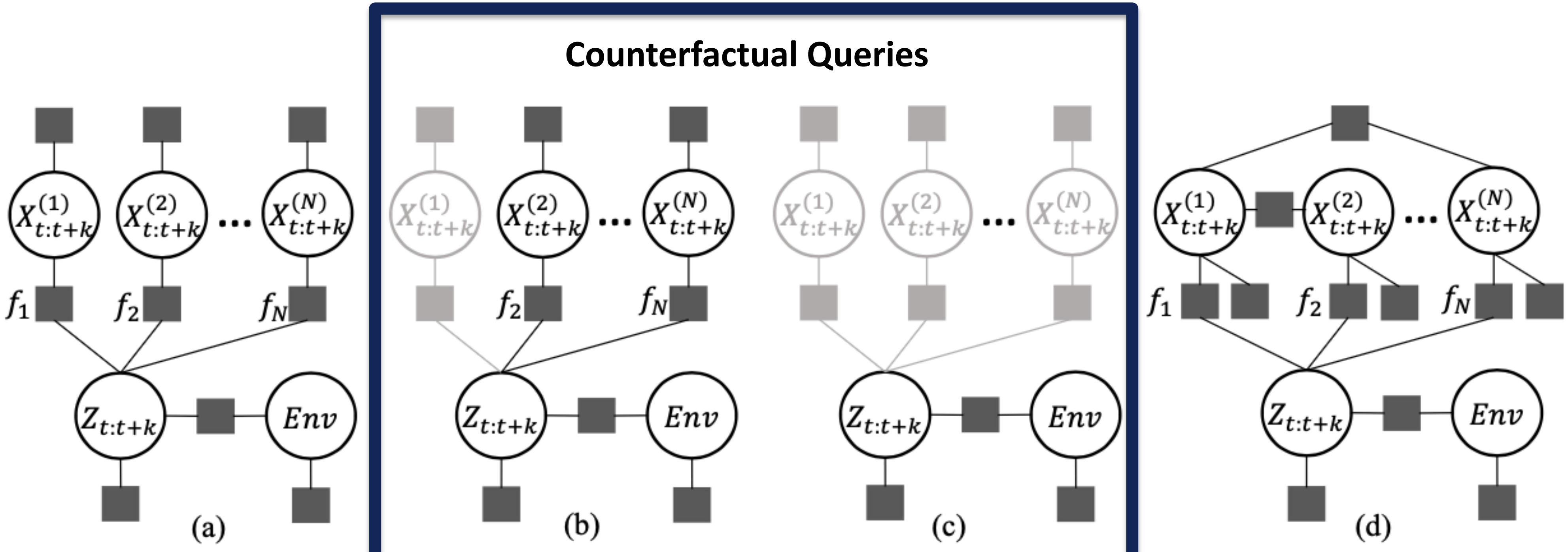
Probability distribution of drivable future trajectories before and after deletion of actor i

Risk under uncertainty in future trajectories:

$$\rho^{i}_{t,k} = P^{\mathbb{Z}^{/i}_{t,k}} - P^{\mathbb{Z}_{t,k}}$$

KL divergence or sampling or means

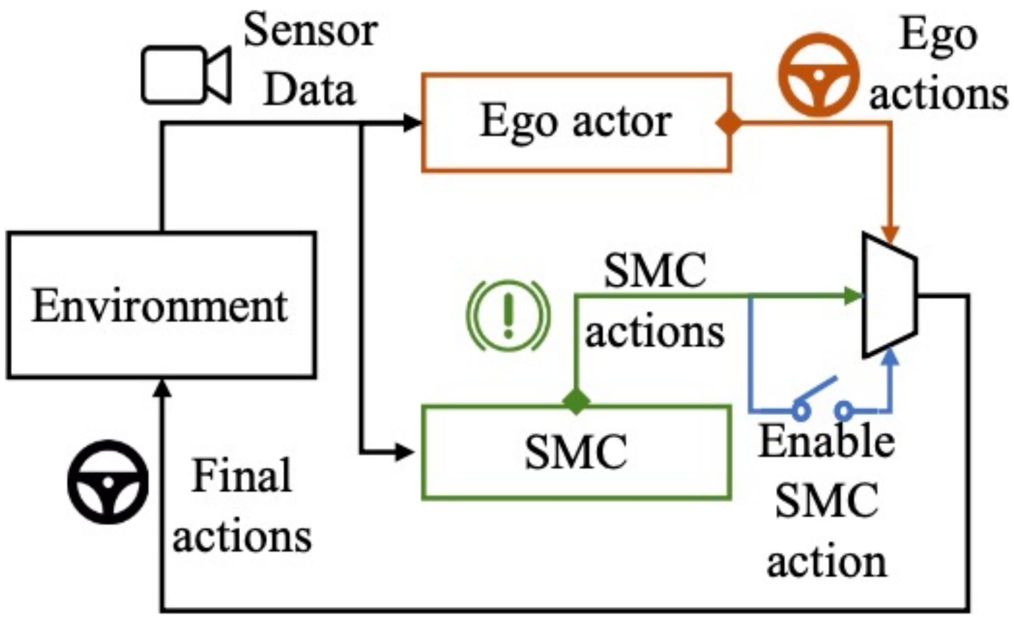
Formalizing Threat Posed by an Actor



Counterfactual Queries

Figure 2: The factor graph representation of the counterfactual query. $X^{(i)}_{t:t+k}$ represents the trajectories of actor i from time t to time $t + k$, $Z_{t:t+k}$ is the set of ego actor's safe future trajectories, and Env represents the static environment. Circle are nodes representing an actor's distribution and squares are factor functions. Factor Graph (a) All actor presence, (b) One actor removed, (c) All actors removed, and (d) Actor-actor influence included.

Proactive Threat Mitigation



Design of Safety Hazard Mitigation Controller (SMC)

Formulated as Partially Observable Markov Decision Process (PoMDP)

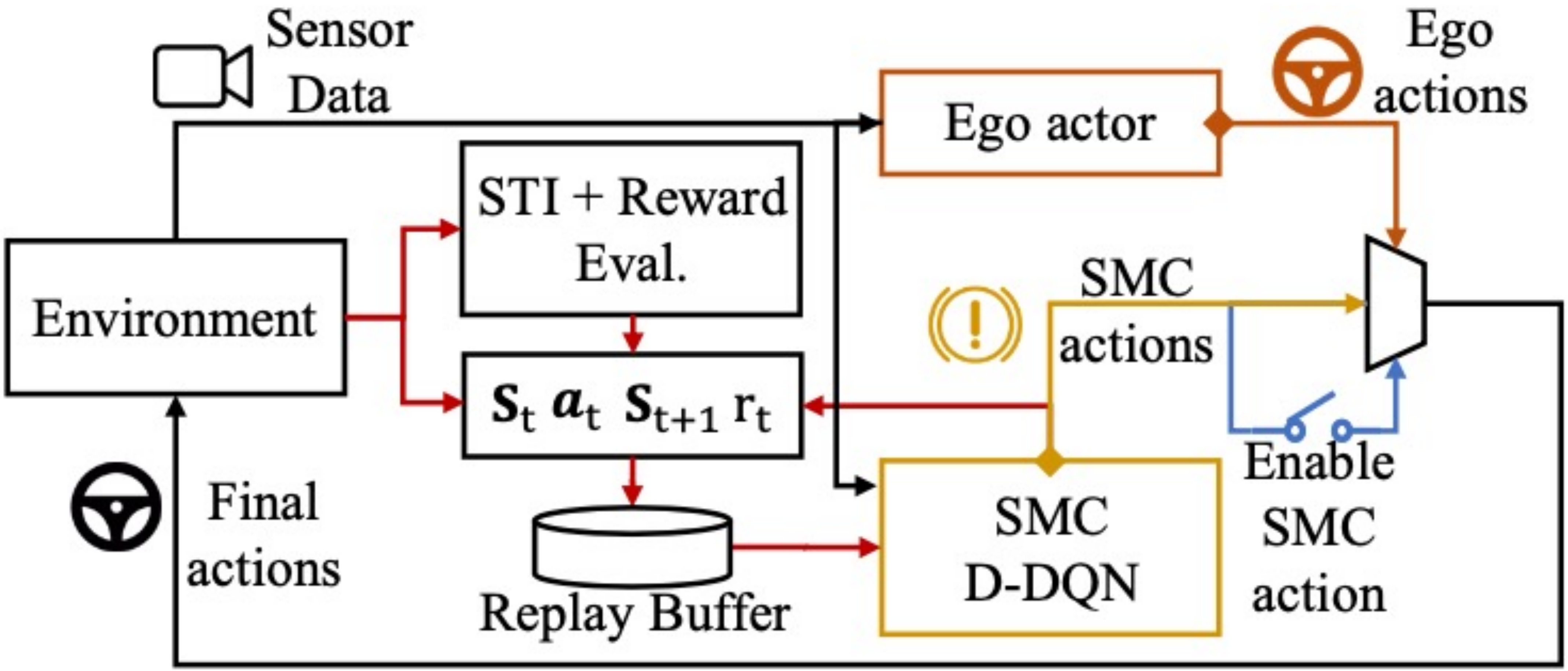
Ego actor's policy $\phi \sim \Phi$, where Φ is the policy distribution.

$\mathcal{U}$ be the overall autonomous system, and *M* be the SMC
a_t be the action taken at time *t*
$\boldsymbol{S_t}$ be the observed system state
SMC policy $\psi \sim \Psi$, where Ψ is the mitigation policy distribution

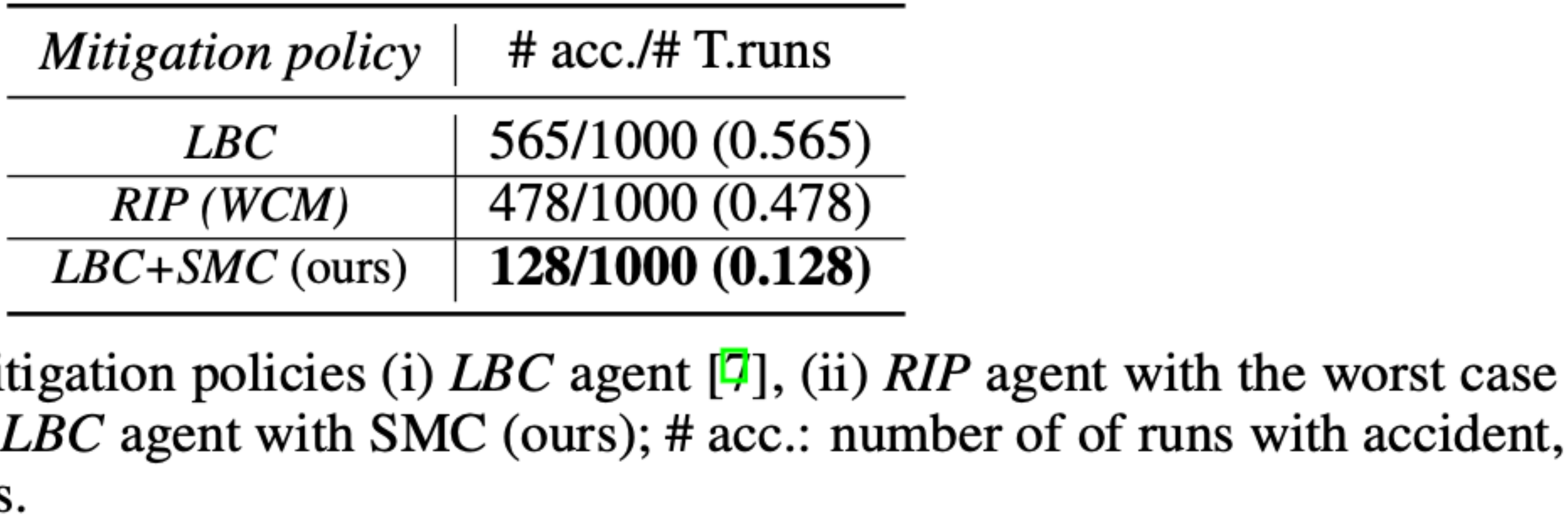
$$a_t = \mathcal{U}(\mathbf{S}_t; \phi, \psi) = f_{actagg}(\mathcal{A}(\mathbf{S}_t; \phi), \mathcal{M}(\mathbf{S}_t; \psi)) = f_{actagg}(a_t^{smc}, a_t^{ego})$$

$$\boxed{r_t = \alpha_0(1 - STI) + \alpha_1 r_{pc} + \alpha_2 p_{am} + \alpha_3 r_{comfort} + \ldots + \alpha_n r_{nth\_factor}}$$

Training SMC using Reinforcement Learning



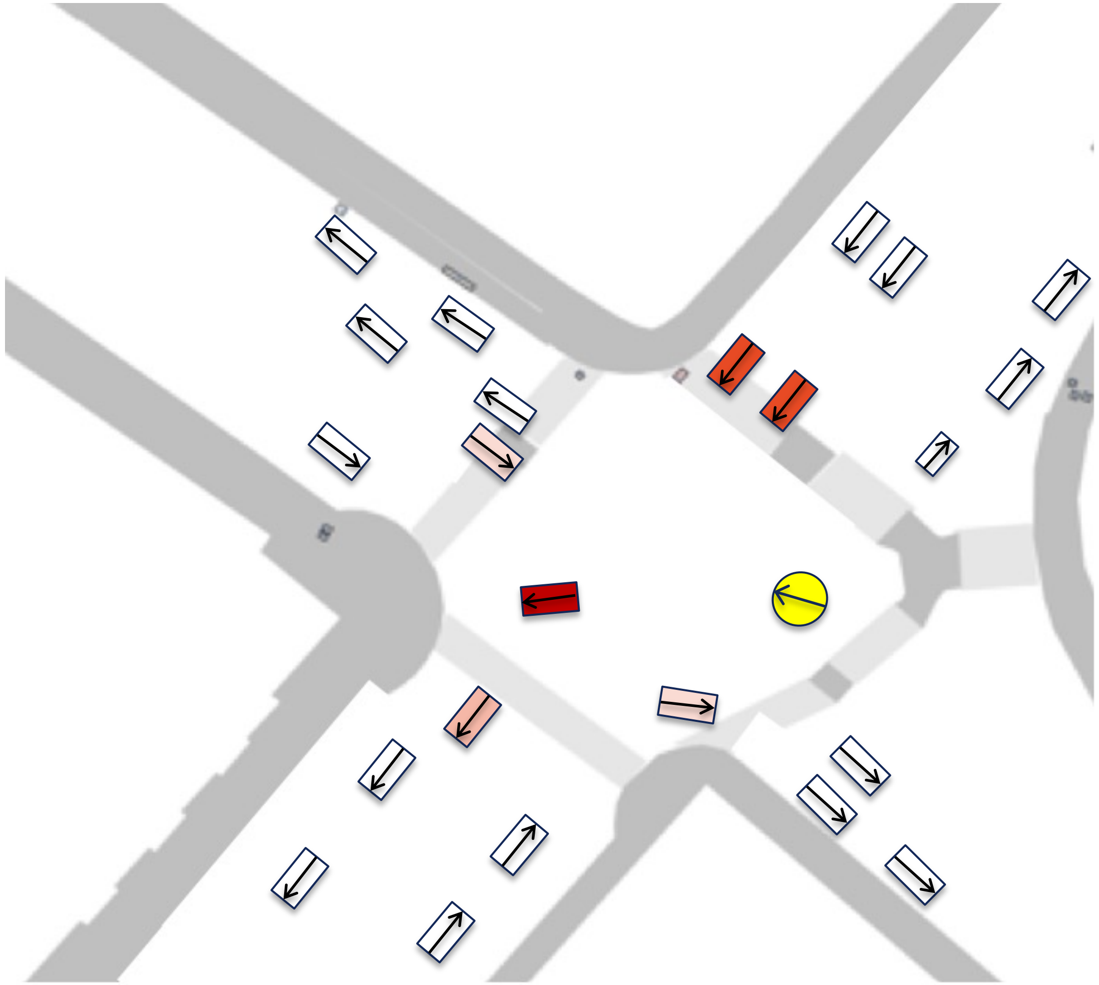
Proactive Threat Mitigation



Mitigation policy	# acc./# T.runs
LBC	565/1000 (0.565)
RIP (WCM)	478/1000 (0.478)
LBC+SMC (ours)	**128/1000 (0.128)**

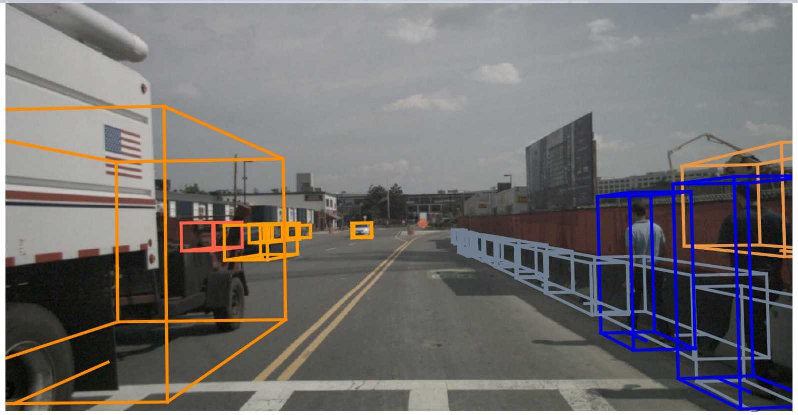
Table 2: Comparison of mitigation policies (i) *LBC* agent [7], (ii) *RIP* agent with the worst case model (WCM) [9], and (iii) *LBC* agent with SMC (ours); # acc.: number of of runs with accident, #T.runs: total number of runs.

Curating Evaluation Dataset: Characterizing Threat in Existing Datasets



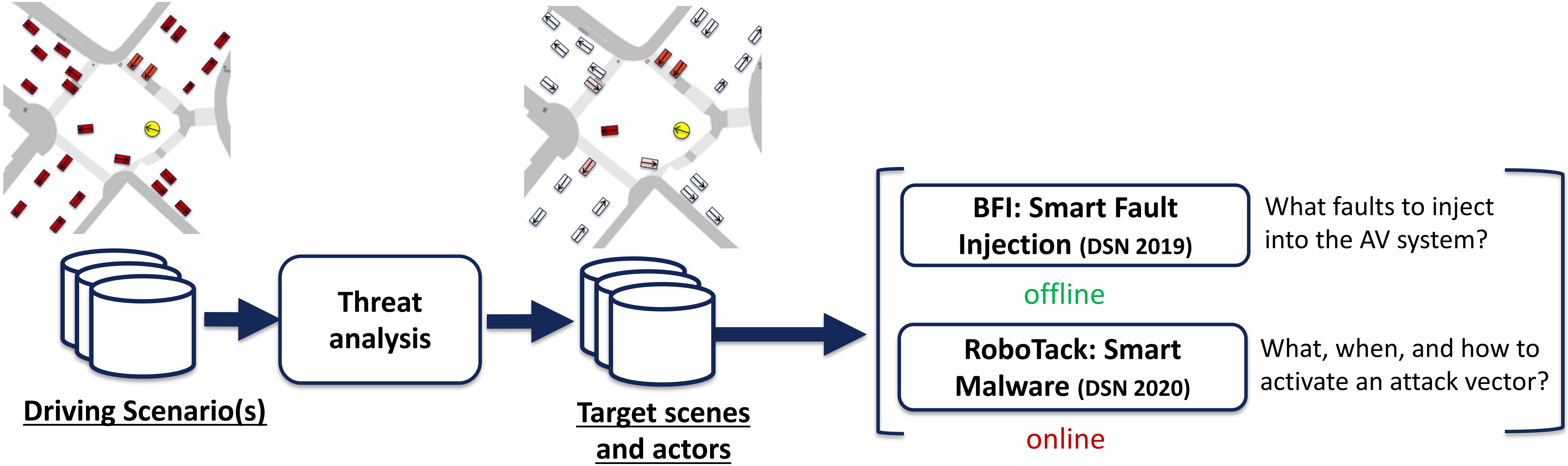
Intensity of red indicates the level of risk

- **Evaluated on Argoverse dataset (1000 driving hrs)**
- 50th, 75th, 90th, and 99th percentiles of the actor STI values are 0.0, 0.0, 0.0, and 0.4, respectively



Extracted driving scenario from nuScenes

Accelerated Assessment



Driving Scenario(s)

Threat analysis

Target scenes and actors

BFI: Smart Fault Injection **(DSN 2019)**

What faults to inject into the AV system?

offline

RoboTack: Smart Malware **(DSN 2020)**

What, when, and how to activate an attack vector?

online

24x speedup for BFI

40% more accurate in the case of RoboTack

Conclusion

- AVs have significantly higher #vulnerabilities than Non-AVs
- Vulnerabilities can be used for attack (perhaps not as trivial to launch as we think) --- multiple weaknesses needs to be leverage for complete automation
- Assessment of safety hazards can prevent many of these attacks in deployment
- Attacker and mitigator learn from each other which is an ever-evolving game (need for game-theoretic models)