Scenic:
An Open-Source Probabilistic Programming System for Data Generation and Safety in AI-Based Autonomy

Daniel J. Fremont **Edward Kim** **Sanjit A. Seshia**

UC Santa Cruz UC Berkeley

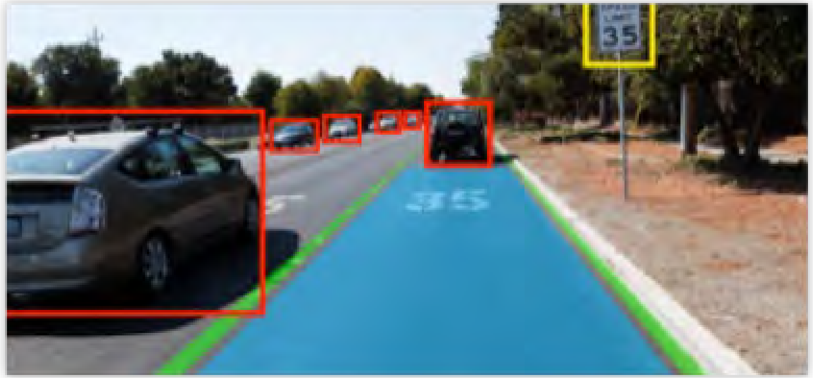
Thanks to our many Scenic Team Members and Contributors
https://docs.scenic-lang.org/en/latest/credits.html

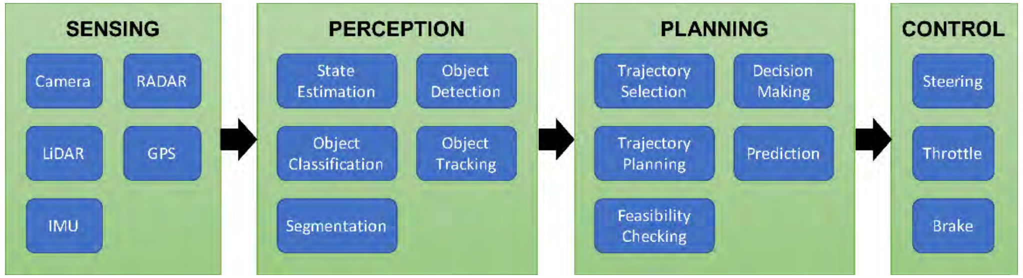
https://scenic-lang.org

CVPR Tutorial
June 17, 2024

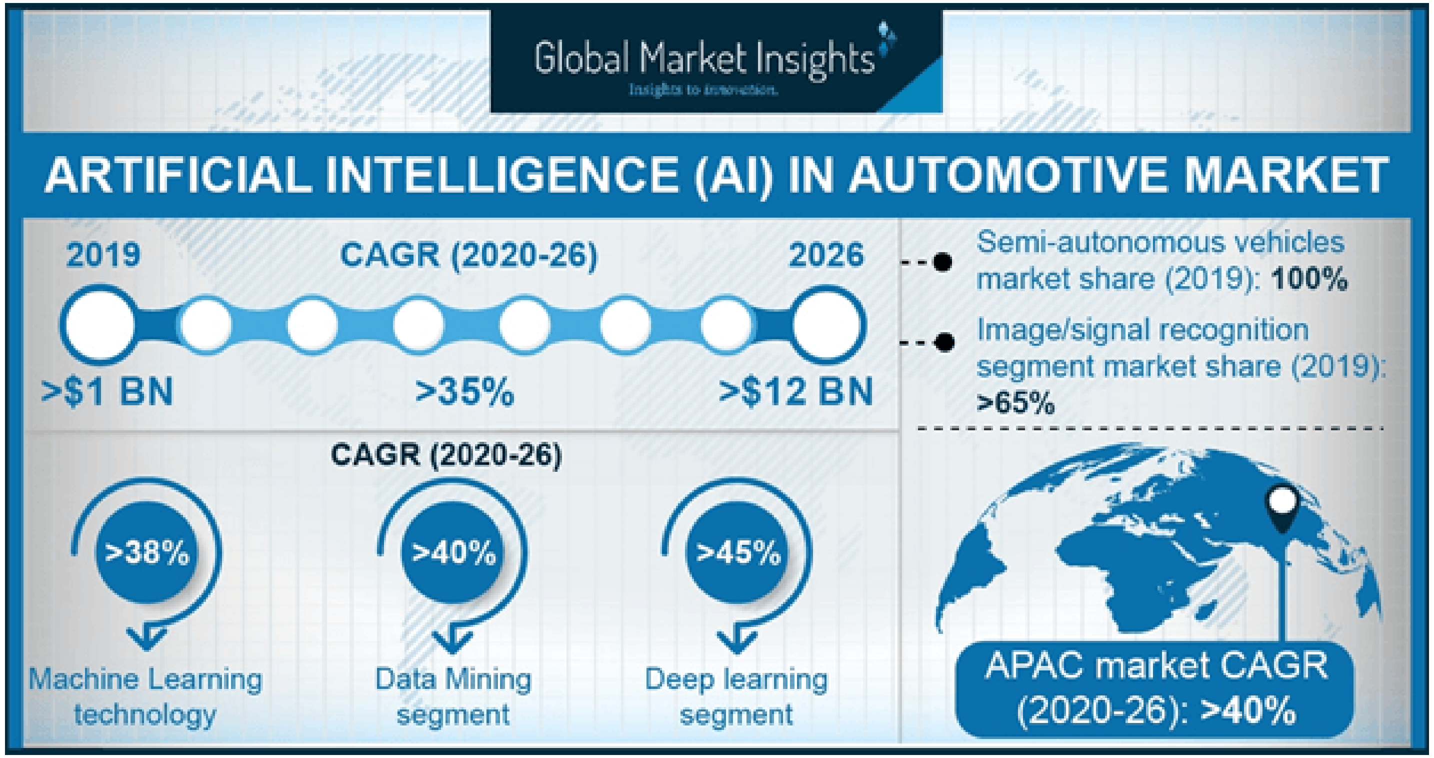
https://scenic-lang.org/cvpr24/

Growing Use of Machine Learning/Artificial Intelligence in Safety-Critical Autonomous Cyber-Physical Systems





Source: Waymo



Source: gminsights.com

Lack of Safety, Dependability, Robustness a Major Obstacle

The Guardian — US edition — For 200 years

2015

Self-driving cars: from 2020 you will become a permanent backseat driver

Driverless cars will revolutionise motoring, claim the manufacturers. But is the greatest danger that they will be too safe?

Tim Adams
@TimAdamsWrites
Sun 13 Sep 2015 05.05 EDT

THE WALL STREET JOURNAL.

2021

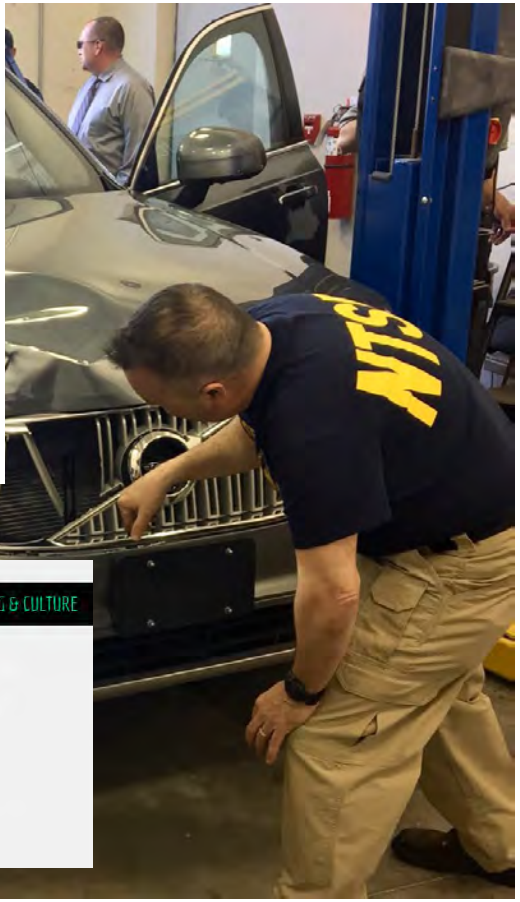
Self-Driving Cars Could Be Decades Away, No Matter What Elon Musk Said

Experts aren't sure when, if ever, we'll have truly autonomous vehicles that can drive anywhere without help. First, AI will need to get a lot smarter.

By Christopher Mims Follow
Jun. 5, 2021 12:00 am ET

Tesla driver dies in first fatal autonomous car crash in US

Hands-off driving faces tough questions
Beck Diefenbach/Reuters



ars TECHNICA — BIZ & IT | TECH | SCIENCE | POLICY | CARS | GAMING & CULTURE

DRIVERLESS CAR SAFETY —

Report: Software bug led to death in Uber's self-driving crash

Sensors detected Elaine Herzberg, but software reportedly decided to ignore her.

Cruise's Driverless Taxi Service in San Francisco Is Suspended

Oct. 24, 2023

The New York Times

The *Verified AI* Research Agenda

Create a Design Flow for AI-based Autonomy supported by Theory, Techniques, and Tools ensuring Safety, Dependability, and Robustness

S. A. Seshia, D. Sadigh, S. S. Sastry. *Towards Verified Artificial Intelligence*.
July 2016. https://arxiv.org/abs/1606.08514. Revision in Communications of the ACM, July 2022.

Formal Methods: A Key Enabler for Design of Safe AI

Precise, Programmatic **Environment/System Modeling**

φ Mathematical **Specification of Requirements and Metrics**

Scalable Algorithms for **Verification, Synthesis, Testing, Debugging**

Methodologies for **Provably-Robust System Design**

Berkeley DeepDrive

DARPA

iCyPhy

NSF

VehiCal

http://vehical.org

S. A. Seshia, D. Sadigh, S. S. Sastry. *Towards Verified Artificial Intelligence*. July 2016. https://arxiv.org/abs/1606.08514. Revised version in Communications of the ACM, July 2022.

Scenic

High-Level, Probabilistic Programming Language for World/Environment Modeling

VerifAI

Requirements Specification + Algorithms for Design, Verification, Testing, Debugging

GitHub **Open-Source Tools**

https://github.com/BerkeleyLearnVerify/Scenic
https://github.com/BerkeleyLearnVerify/VerifAI

for

Industry

Improve assurance of industrial AI systems

Academia

Leverage tools in research & education

Government/ Regulators

Evaluate the safety of AI-based autonomy

Share Scenarios and Metrics **Community** Develop Corpus of Tools and Data

Tutorial Outline

- Overview of Scenic and Its Applications
  - Scenic, VerifAI, and Two Industrial Case Studies
- Introduction to the Scenic 3.0 Language
  - Major Language Features with Examples and Hands-On Coding

Coffee Break

- Applications of Scenic
  - Systematically test and debug perception, behavior prediction, and planning components or the full autopilot stack in simulation
  - Generate sensor data (e.g. RGB, LiDAR) and labels (e.g. segmentation, 3D bounding boxes) using Scenic, and perform sim-to-real validation
- Ongoing and Future Directions
  - Extended Reality, Large Language Models, and more…

SCENIC: Environment Modeling and Data Generation

- *Scenic* is a probabilistic programming language defining *distributions over scenes/scenarios*
- *Use cases:* data generation, test generation, verification, debugging, design exploration, etc.

```
model scenic.domains.driving.model

ego = new Car

spot = new OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
parkedCar = new Car left of spot by 0.5,
            facing badAngle relative to roadDirection
```

```
model scenic.simulators.carla.model

behavior PullIntoRoad():
        while (distance from self to ego) > 15:
                wait
        do FollowLaneBehavior(lane=ego.lane)

ego = new Car with behavior EgoBehavior()

spot = new OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
parkedCar = new Car left of spot by 0.5,
                    facing badAngle relative to roadDirection,
                    with behavior PullIntoRoad()
```

Example: Badly-parked car

Image created with GTA-V

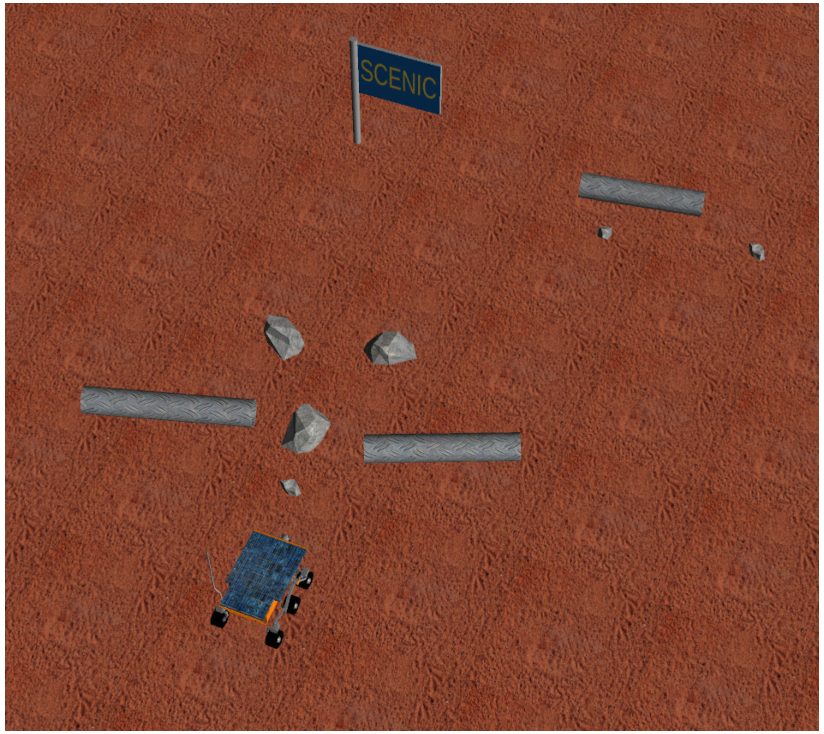
Video created with CARLA

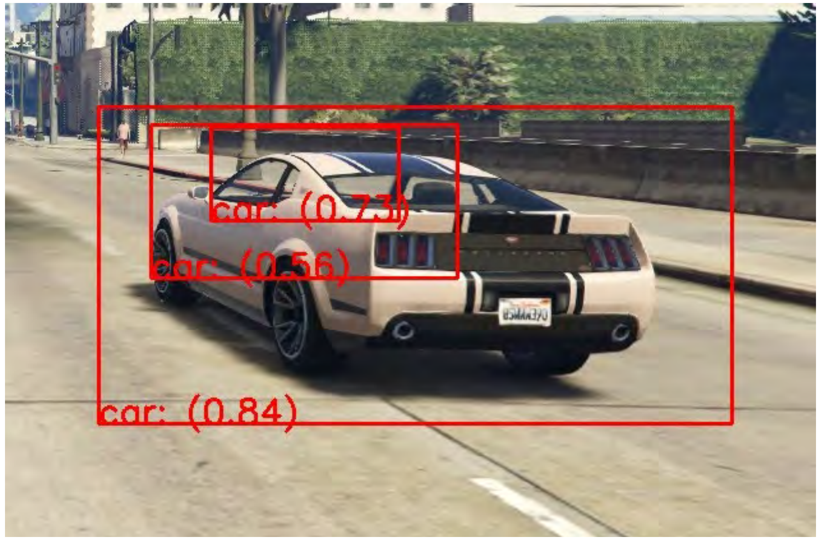
[D. Fremont et al., "Scenic: A Language for Scenario Specification and Scene Generation", TR 2018, PLDI 2019, Extended version in Machine Learning journal. https://arxiv.org/abs/2010.06580.]

Some Applications of Scenic

[see PLDI'19, MLJ'22 papers]

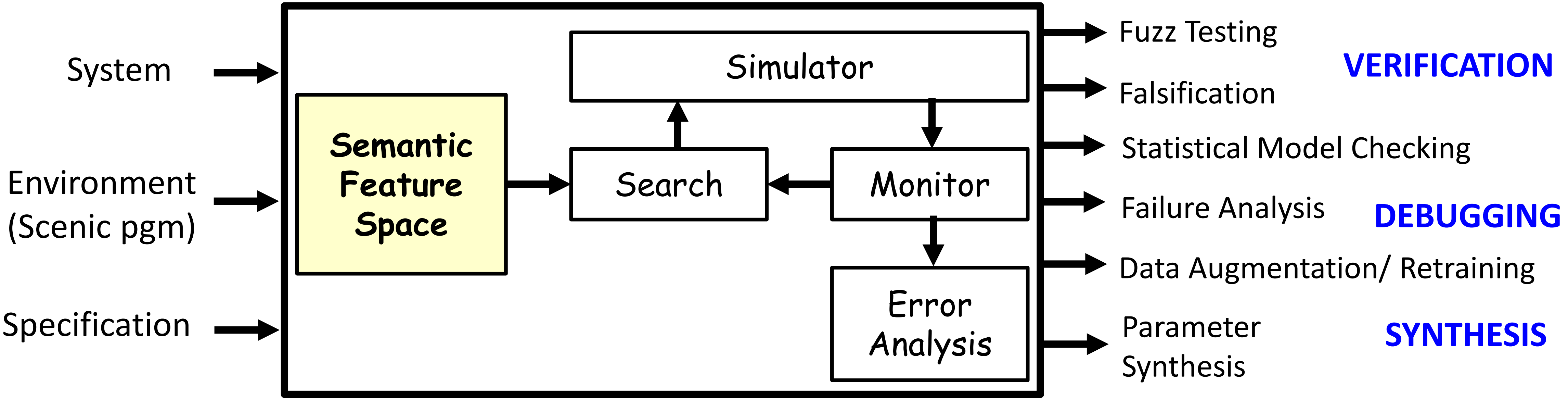
- Modeling, testing, verification
- Exploring system performance
  – Generating specialized test sets
- Debugging a known failure
  – Generalizing in different directions
- Designing more effective training sets
  – Training on hard cases
- Design space exploration





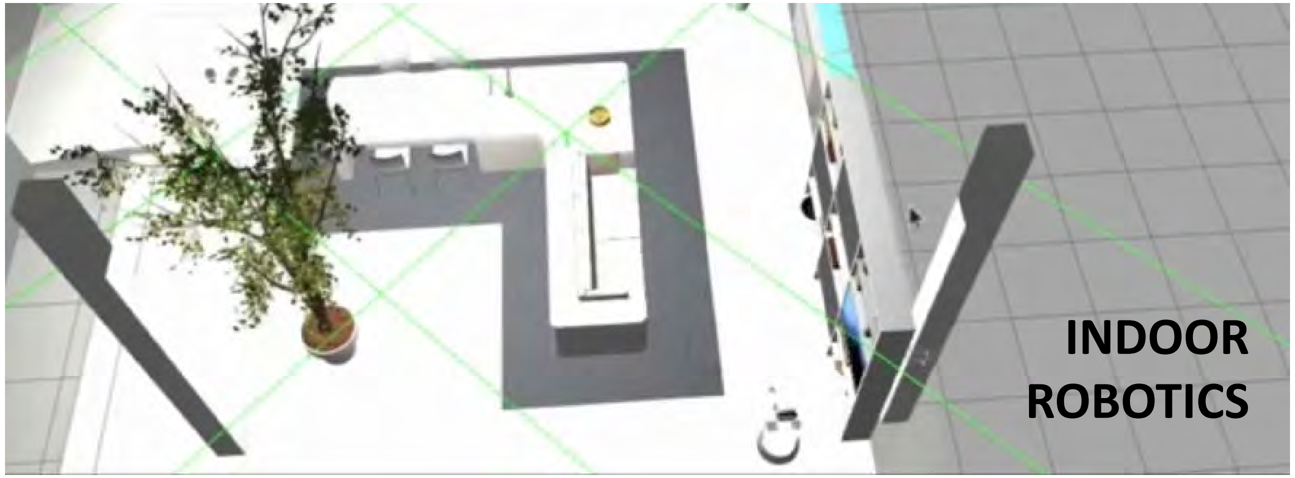
VERIFAI: A Toolkit for the Design and Analysis of AI-CPS

[Dreossi et al. CAV 2019, https://github.com/BerkeleyLearnVerify/VerifAI]



Many Application Domains

AUTONOMOUS VEHICLES



INDOOR ROBOTICS

MULTI-AGENT LEARNING SYSTEMS

AEROSPACE SYSTEMS

AUGMENTED REALITY

A Full Design Iteration: Autonomous Airplane Taxiing

Modeling → Verification → Synthesis/Training → Run-Time Assurance

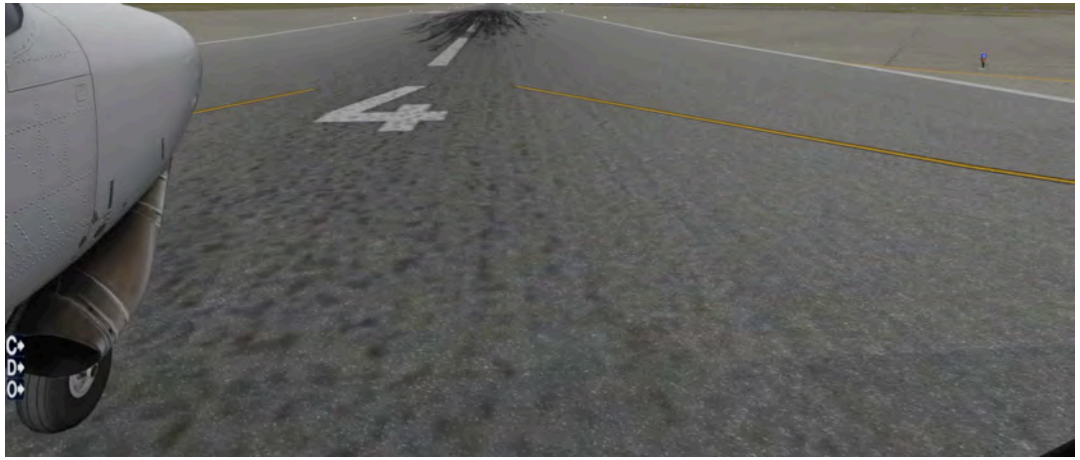
Assured Autonomy

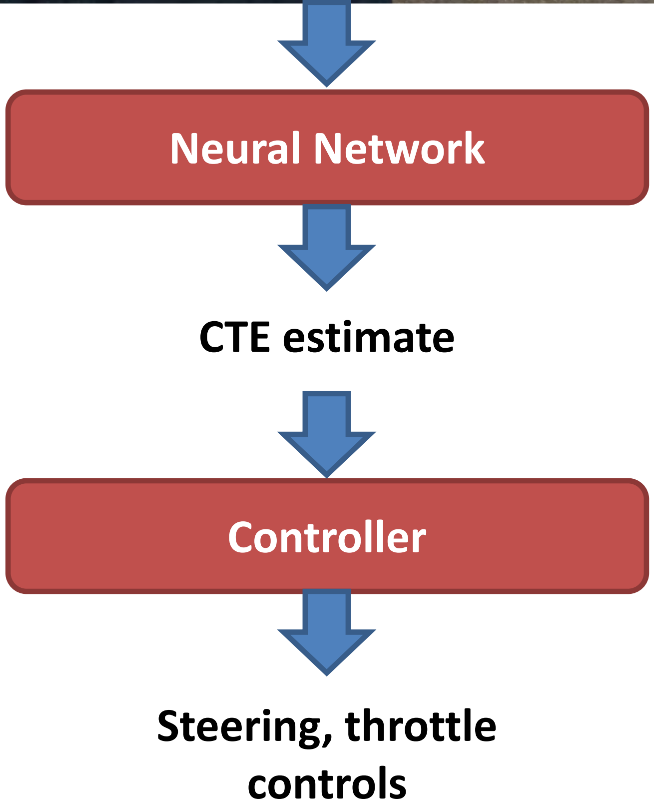
Collaboration with:

D. Fremont et al. *Formal Analysis and Redesign of a Neural Network-Based Aircraft Taxiing System with VerifAI.* In 32nd International Conference on Computer Aided Verification (CAV), July 2020.
H. Torfah et al. *Learning Monitorable Operational Design Domains for Assured Autonomy*. In Proceedings of the International Symposium on Automated Technology for Verification and Analysis (ATVA), October 2022.

TaxiNet: Deep Learning for Autonomous Taxiing

- Experimental autonomous aircraft taxiing system developed by Boeing
- Neural network uses camera image to estimate the *cross-track error*
  - CTE = distance from centerline
- Specification: plane must track centerline to within 1.5 meters





A Full Design Iteration using Scenic & VerifAI

- **Modeling** runway scenarios in SCENIC
- **Specifying** the safety requirement
- **Falsifying** the system, finding scenarios when it violates its safety specification
- **Debugging** to find distinct failures and their root causes
- **Retraining** the neural network to eliminate failures and improve performance
- **Runtime assurance** to predict and handle unsafe situations at run time

Modeling with the Scenic Language

```
model scenic.simulators.xplane.model

# Time of day: from 6am to 6pm. (+8 to get GMT, as used by X-Plane)
param zulu_time = (Range(6, 18) + 8) * 3600 # in seconds

# Rain: 1/3 of the time.
# Clouds: rain requires types 3-5; otherwise, 0-5.
clouds_and_rain = Options({
        tuple([Uniform(0, 1, 2, 3, 4, 5), 0]): 2,
        tuple([Uniform(3, 4, 5), (0.25, 1)]): 1
})

param cloud_type    = clouds_and_rain[0]
param rain_percent = clouds_and_rain[1]

# Plane: up to 8m left/right, 2000 m down the runway, 30 deg left/right
ego = new Plane at Range(-8, 8) @ Range(0, 2000),
                facing Range(-30, 30) deg
```

Semantic features: time, clouds, rain, position/ orientation of plane on the runway

Falsification: Algorithmic Search for Unsafe Behaviors

1. Specify safety condition as temporal logic assertion

$$\varphi_{\text{eventually}} = \Diamond_{[0,10]}\Box(\text{CTE} \leq 1.5)$$

2. Transform assertion into cost function

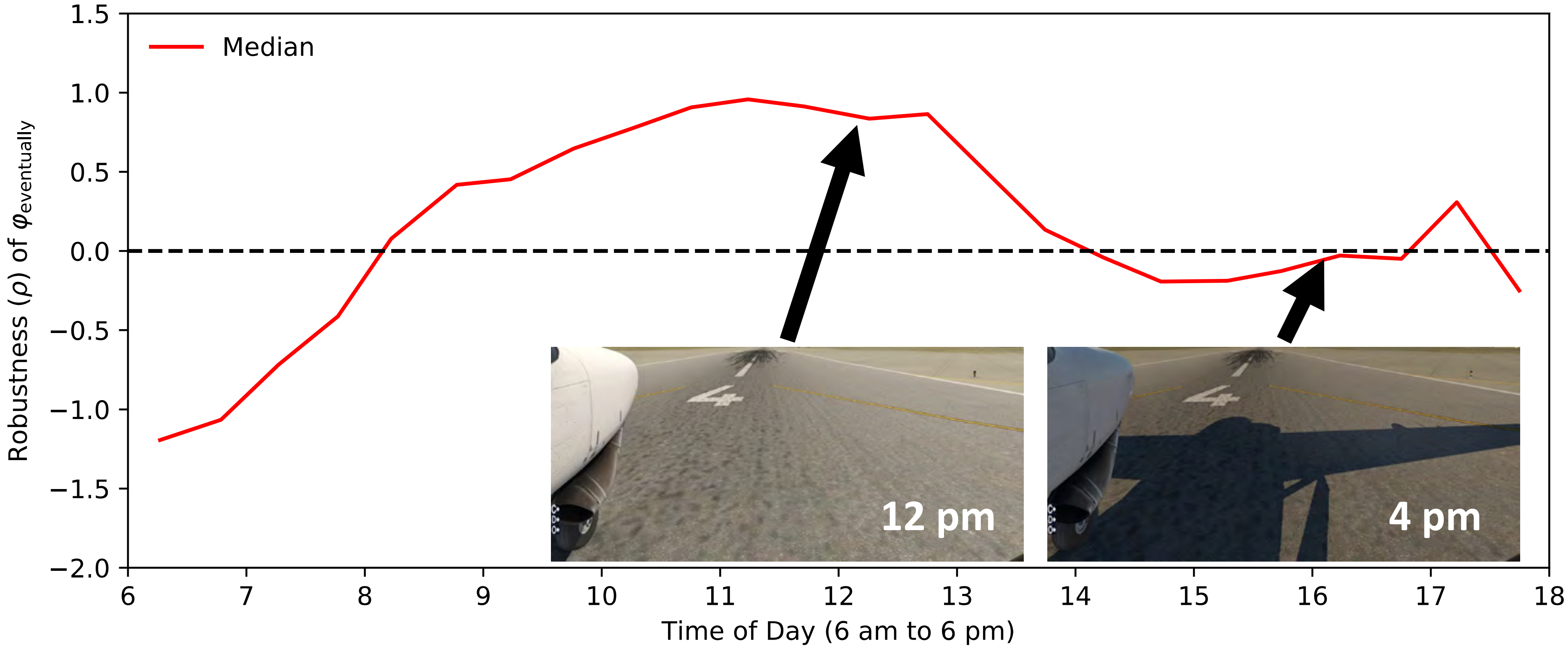
$$\rho_{\text{eventually}} = \sup_{t\in[0,10]} \inf_{[t,\infty]} (1.5 - \text{CTE}(t))$$

3. Find safety violation by minimizing cost function
   - Cost function < 0 → Safety violation

- Falsification: out of ~4,000 auto-generated simulations
  - **45%** violated $\varphi_{\text{eventually}}$
  - **9%** left runway entirely

What Went Wrong? → Debugging & Root Causing

- Falsification found several types of failures, e.g. sensitivity to time

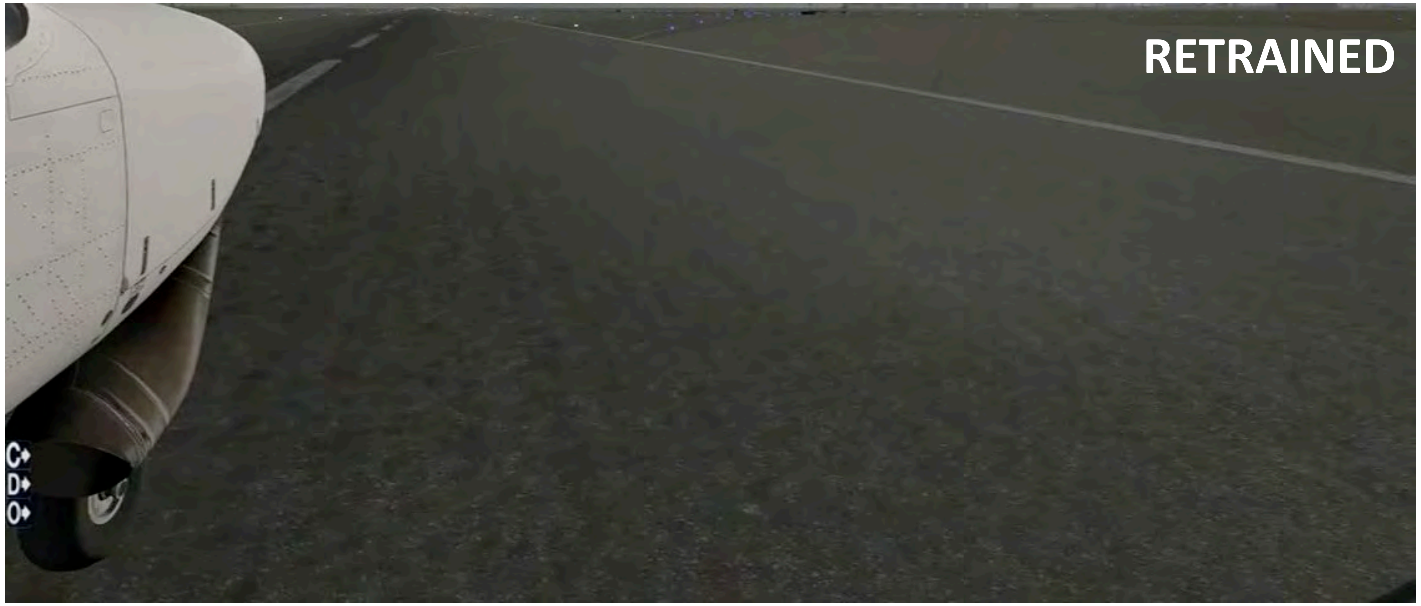


- Follow-up experiments confirmed root cause is the plane's shadow

Scenic-Guided Retraining

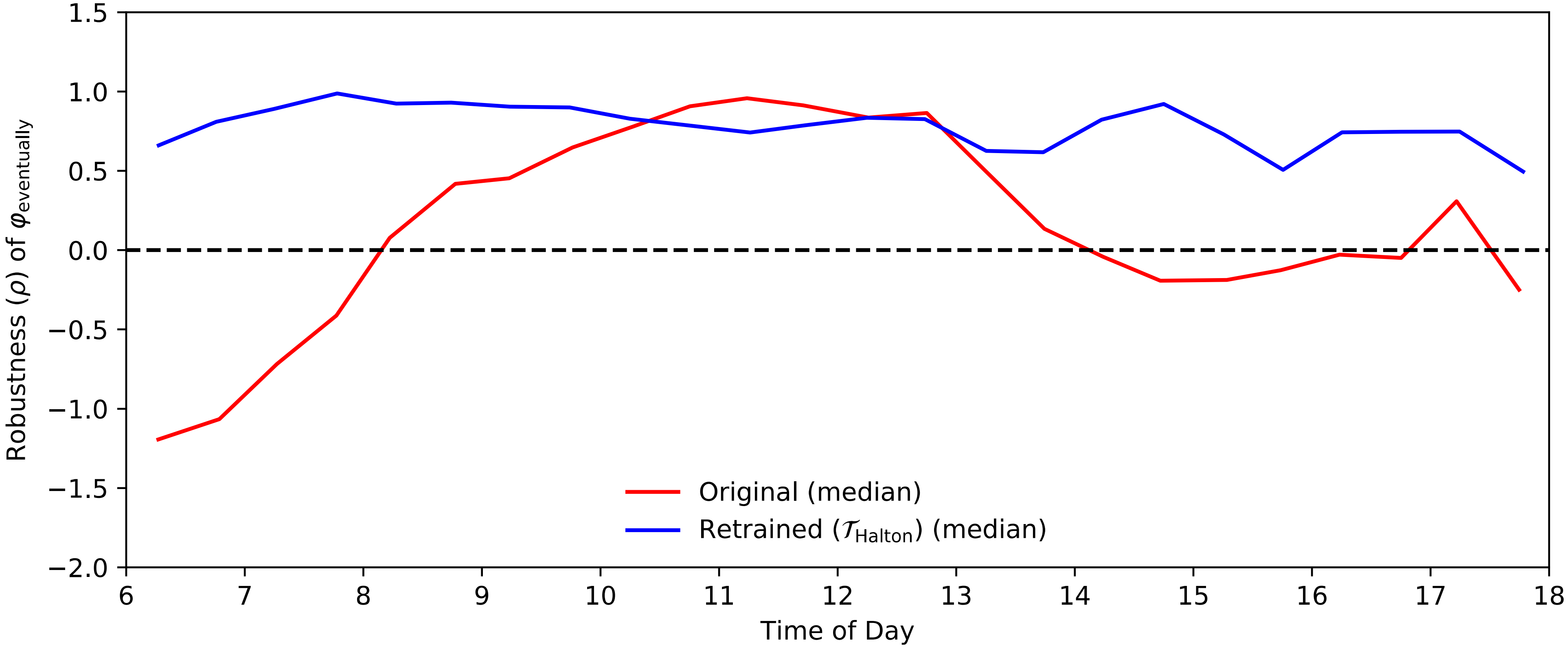
- Use VerifAI to generate a new training set (same size as original)

- Obtained much better performance
  - **17%** violated $\varphi_{\text{eventually}}$ (vs. 45%)
  - **0.6%** left runway entirely (vs. 9%)



Retraining

- Eliminated dependence on time of day



Robust Operation: Runtime Monitoring and Failure Mitigation

- Automatically extract environment assumptions from Scenic model
- Use the Simplex fault-tolerant architecture with detection of potentially unsafe scenarios

From Simulation to Real-World Testing

D. Fremont et al. *Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World.* In 23rd IEEE International Conference on Intelligent Transportation Systems (ITSC), September 2020.

From Simulation to Real-World Testing: Key Questions

#1 **Safety violations in simulation**: Do they transfer to the **real world**? How well?

#2 **Scenario testability**: Can we use **formally guided simulation** to effectively design **real-world tests**?

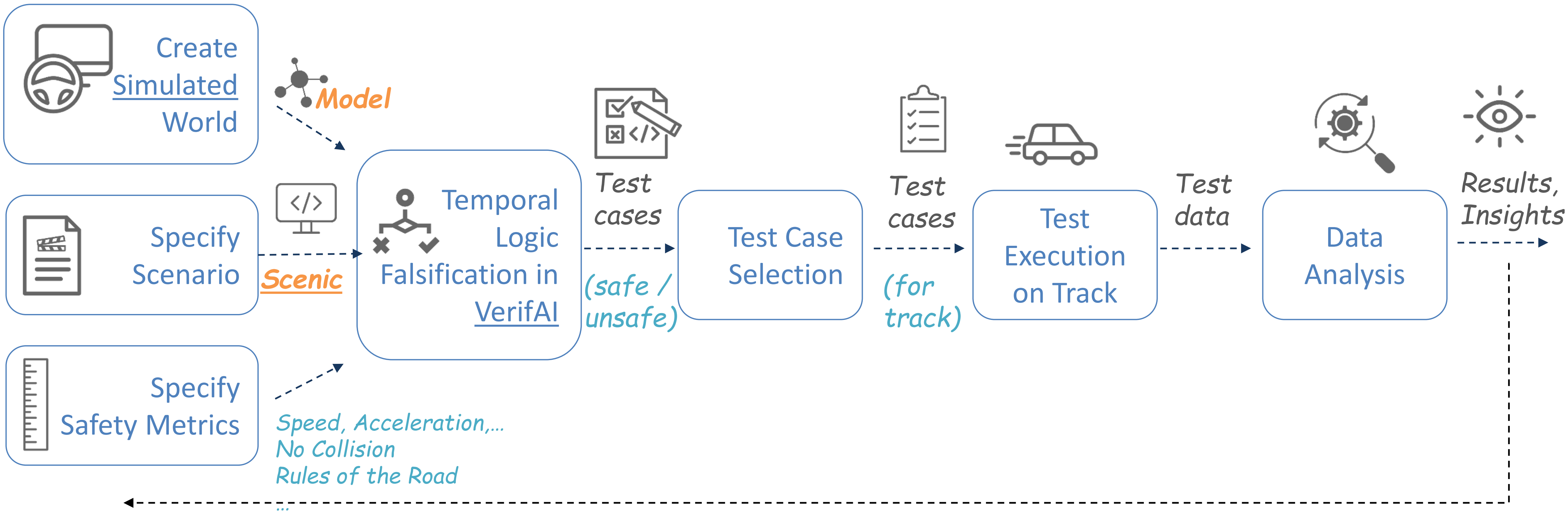
First use of formal methods for scenario-based testing of AI-based autonomy in both simulation and real world

Fremont, Kim, Pant, Seshia, Acharya, Bruso, Wells, Lemke, Lu, Mehta, ***"Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World"***, Arxiv e-prints, https://arxiv.org/abs/2003.07739 ***[ITSC 2020]***

GOMENTUM STATION

Owned and Operated by AAA NCNU

Formal Scenario-Based Testing (with Scenic and VerifAI)



Source: Fremont et al., *"Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World"*, Intelligent Transportation Systems Conference (ITSC), September 2020. https://arxiv.org/abs/2003.07739

Scenario Overview: Focus on Vulnerable Road Users (VRUs)

+53%

Pedestrian fatalities: 53% increase in the last decade (2009-2019)
2019: ~6500 (estimated)

17%

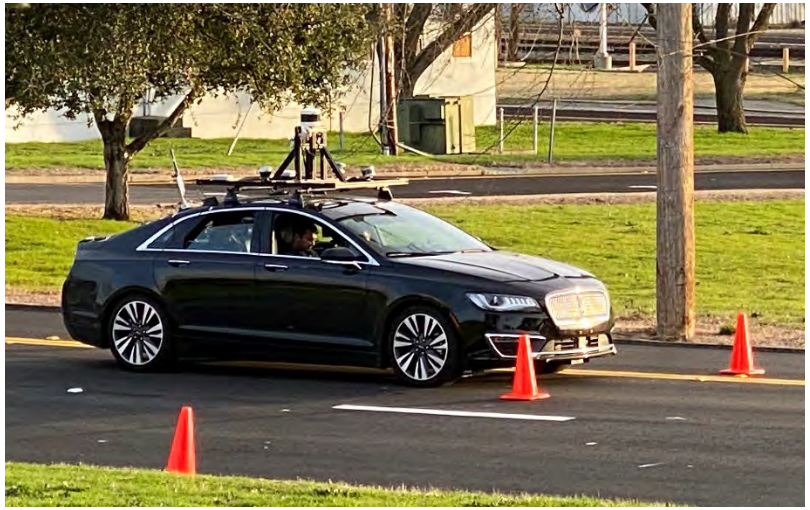
Of all traffic fatalities, 17% are **Pedestrians**

67%

Fatalities at **night** (low-light, limited vision environment)

Source:
GHSA: https://www.thecarconnection.com/news/1127308_pedestrian-deaths-reach-30-year-high-in-2019
IIHS: https://www.iihs.org/topics/pedestrians-and-bicyclists

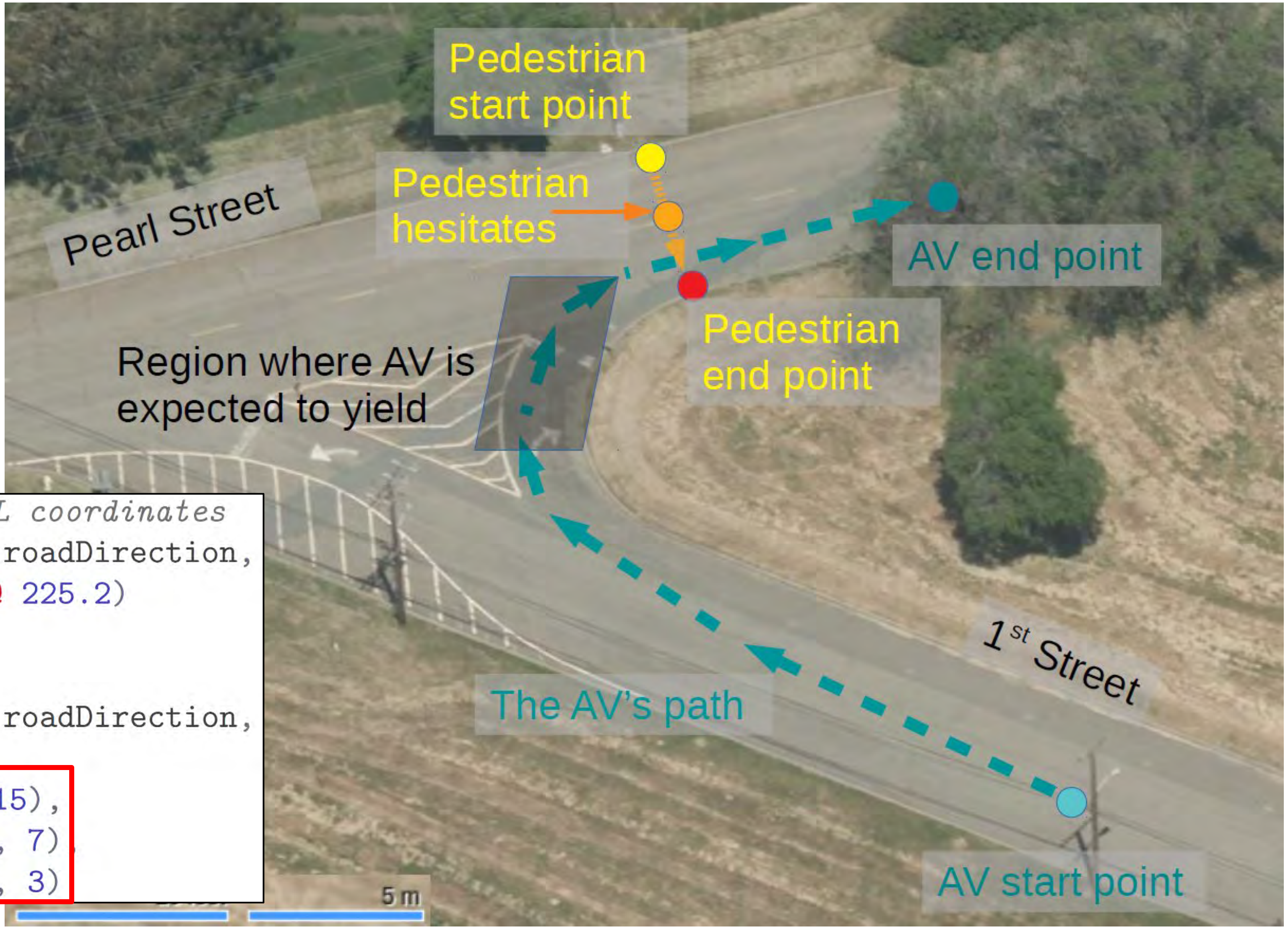
Example Scenario: AV making right turn, pedestrian crossing



Lincoln MKZ running Apollo 3.5

```
ego = new Autopilot at 38.6 @ 183.9, # SVL coordinates
                 facing 10 deg relative to roadDirection,
                 with behavior DriveTo(40 @ 225.2)

ped = new Pedestrian at 19.78 @ 225.68,
                 facing 90 deg relative to roadDirection,
                 with behavior Hesitate()
                 with startDelay Range(7, 15),
                 with walkDistance Range(4, 7)
                 with hesitateTime Range(1, 3)
```

Snippet of Scenic program



Results: Falsification and Test Selection

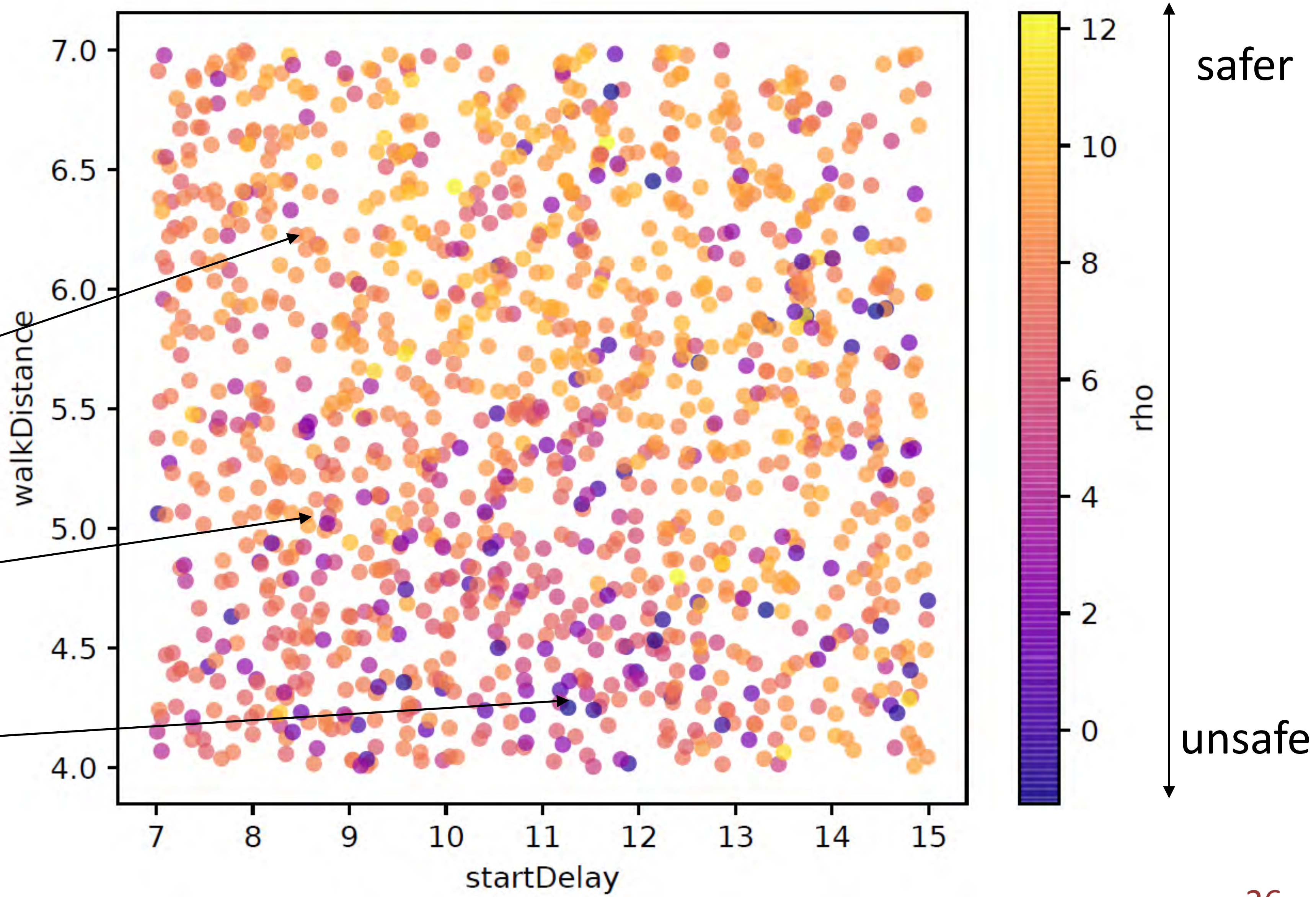
1294 simulations explored
2% violated safety property

Total 7 test cases selected

S2: robustly safe

M2: marginally safe

F2: collision



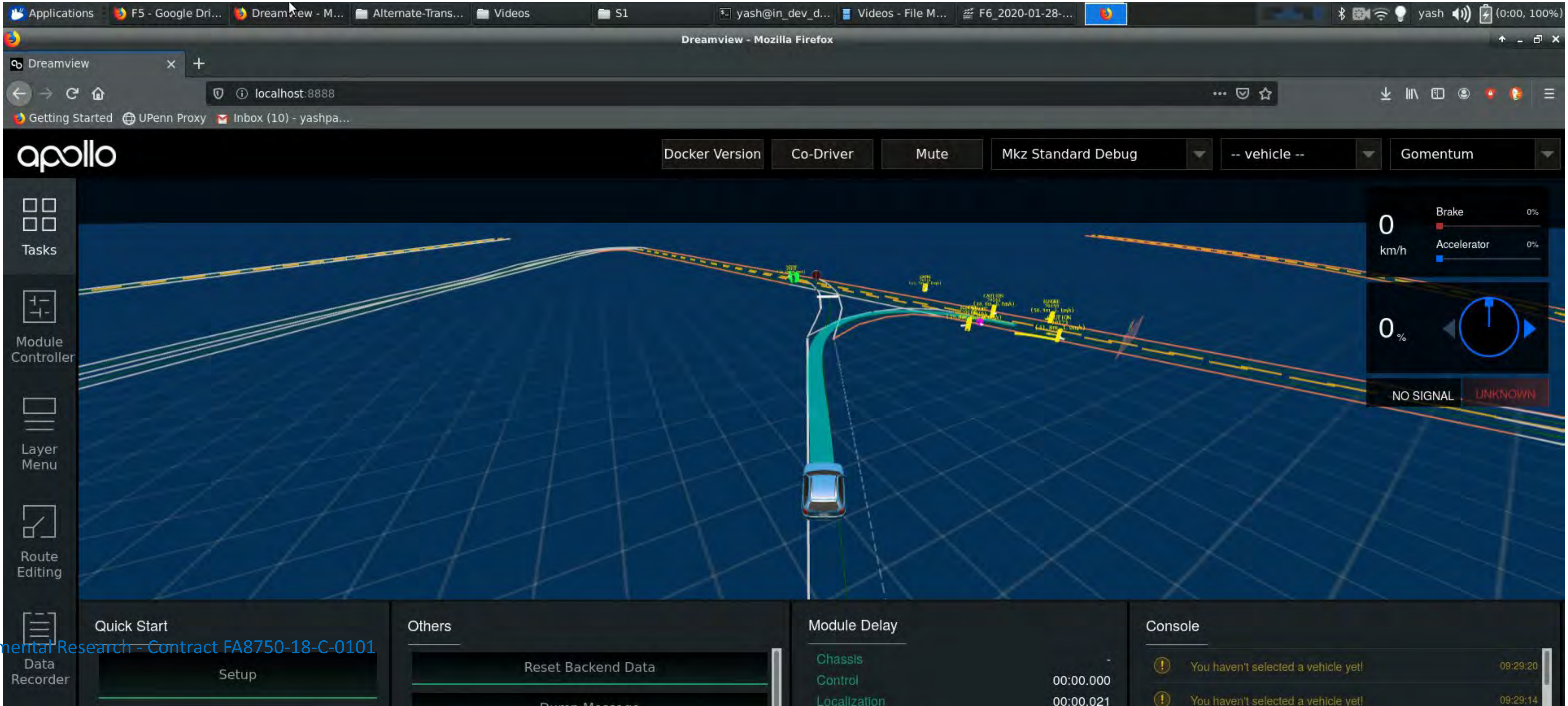
Results: Does Safety in Simulation → Safety on the Road?

Unsafe in simulation → unsafe on the road: **62.5% (incl. collision)**

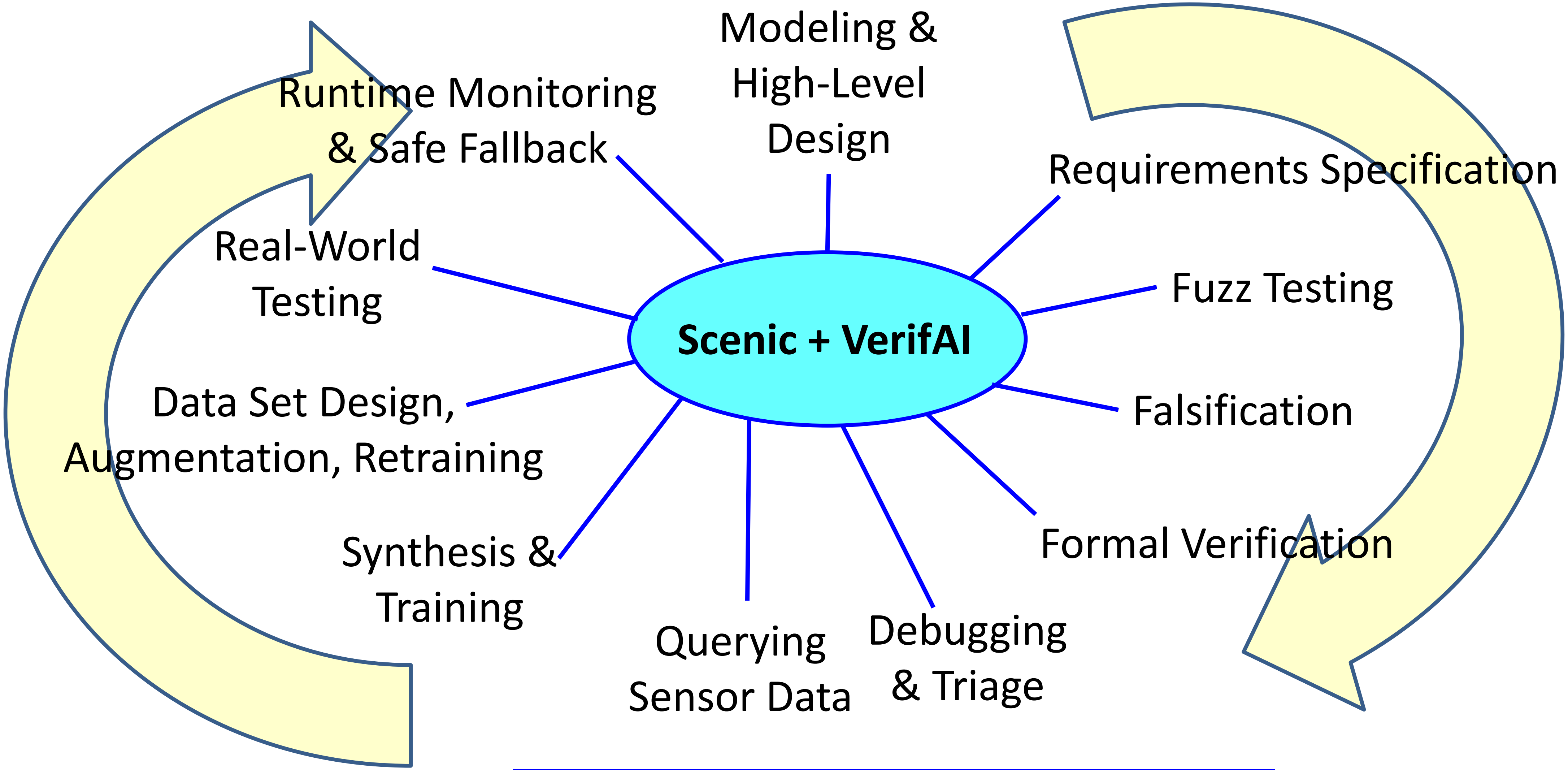
Safe in simulation → safe on the road: **93.5% (no collision)**

Results: Why did the AV Fail?

Perception Failure: Apollo 3.5 lost track of the pedestrian several times



Ecosystem of Design Tools for AI-based Autonomy



Supporting the Full Design Cycle

Scenic for Multi-Agent Strategy Games and Extended Reality

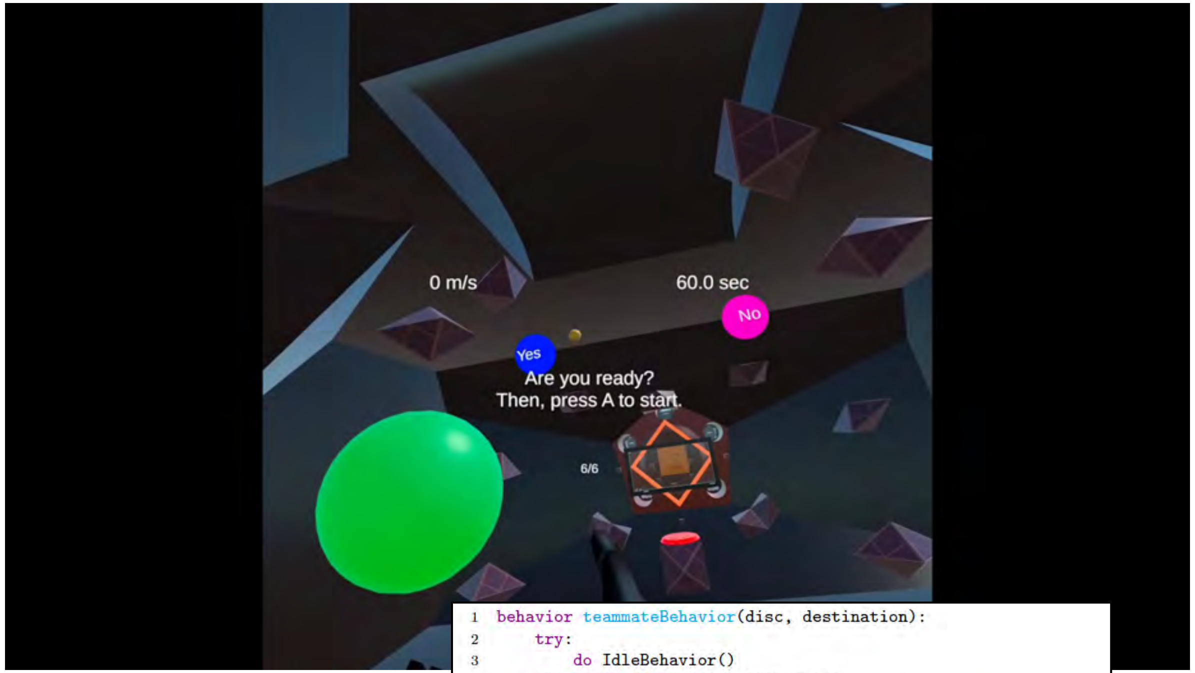
Programmatic Training of Reinforcement Learning Agents


```
behavior rightRMBehavior(destination_point, ego):
    try:
        do Uniform(HighPassTo(ego), LowPassTo(ego))
        do MoveToPosition(destination_point, sprint=True)
        new_dest_point = Point on rightRMAttackRegion
        do egoBehavior(new_dest_point)
    interrupt when opponentTeamHasBallPossession(self):
        do FollowObject(ball, sprint=True)

LeftGK
LeftRB
LeftLM on LeftMidRegion

RightGK
ego_destinationPoint = Point on egoAttackRegion
rightRM_destinationPoint = Point on rightRMAttackRegion

ego = RightAM on egoInitialRegion, with behavior egoBehavior(ego_destinationPoint)
rightRM = RightRM with behavior rightRMBehavior(rightRM_destinationPoint, ego)
ball = Ball ahead of rightRM by Range(2,3)
```

Google Research Football (soccer) simulator

A. Azad, E. Kim, et al. “Programmatic Modeling and Generation of Real-time Strategic Soccer Environments for Reinforcement Learning,” AAAI 2022.

Training Human eSports Players in Virtual Reality (VR)




```
behavior teammateBehavior(disc, destination):
    try:
        do IdleBehavior()
    interrupt when disc.heldByHuman:
        take MoveToAction(destination)
    interrupt when (distance from self to disc) < 0.5:
        take GrabDiscAction(True)

behavior disableThrustBrake():
    take disableThrustAction(), disableBrakeAction()

ego = HumanPlayer at (0, 0, 0), facing toward goal,
        with behavior disableThrustBrake()

GreenSphere ahead of ego

disc = Disc offset by (Range(-2, 2), Range(-2, 2), Range(-2, 2))

tm = BluePlayer behind ego by Range(3, 5),
        with behavior teammateBehavior(disc, goal)
```

Modeled on Meta/Oculus EchoArena Virtual Reality Game

… on to Part 2!

A Hands-On Introduction to the Scenic 3.0 Language

Introduction to Writing Scenic Scenarios

Daniel J. Fremont

University of California, Santa Cruz

The plan

- 9:30: defining static scenarios in Scenic
- 9:40: hands-on time with Colab notebook 1

- 10:05: defining dynamic scenarios in Scenic
- 10:15: hands-on time with Colab notebook 2

- 10:40: more Scenic features
- 10:45: coffee break

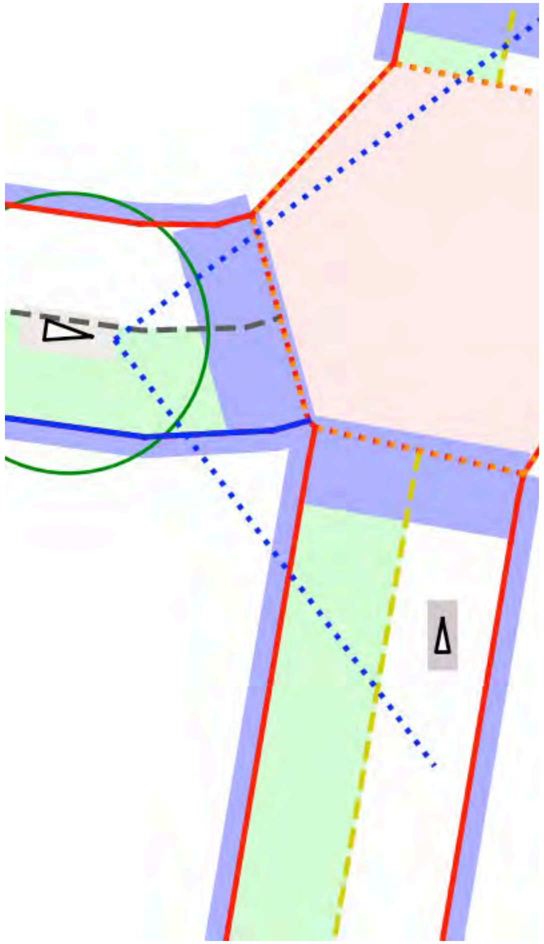
The history of Scenic

- We’ll be working with Scenic 3.0, the latest version. A bit of history:
- Scenic 1.0 (tech. report 2018; PLDI 2019)
  - Static 2D scenarios; interfaces to GTA V and Webots
- Scenic 2.0 (*Machine Learning* Journal 2022)
  - Dynamic 2D scenarios
  - Library for driving scenarios; interfaces to CARLA and LGSVL
- Scenic 3.0 (CAV 2023)
  - 3D scenarios; precise modeling of shapes & occlusion
  - Temporal logic requirements

WRITING STATIC SCENARIOS

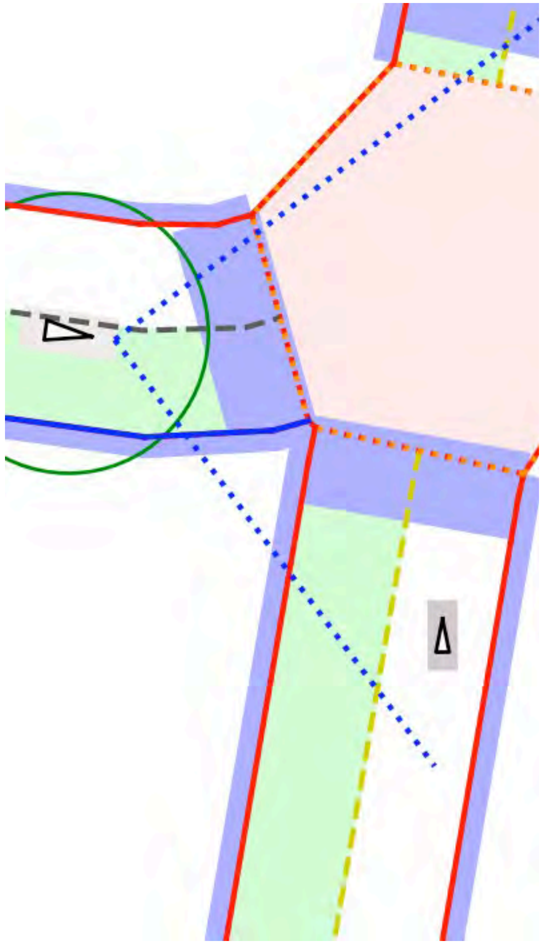
Example: a Badly-Parked Car

Example: a Badly-Parked Car



```
model scenic.domains.driving.model  # defines Car, etc.
```

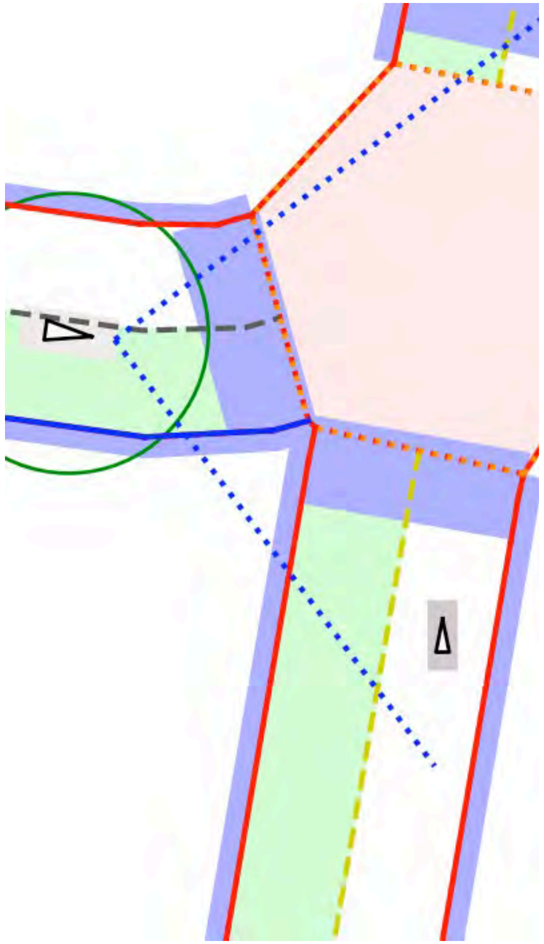
Example: a Badly-Parked Car



```
model scenic.domains.driving.model  # defines Car, etc.

ego = new Car
```

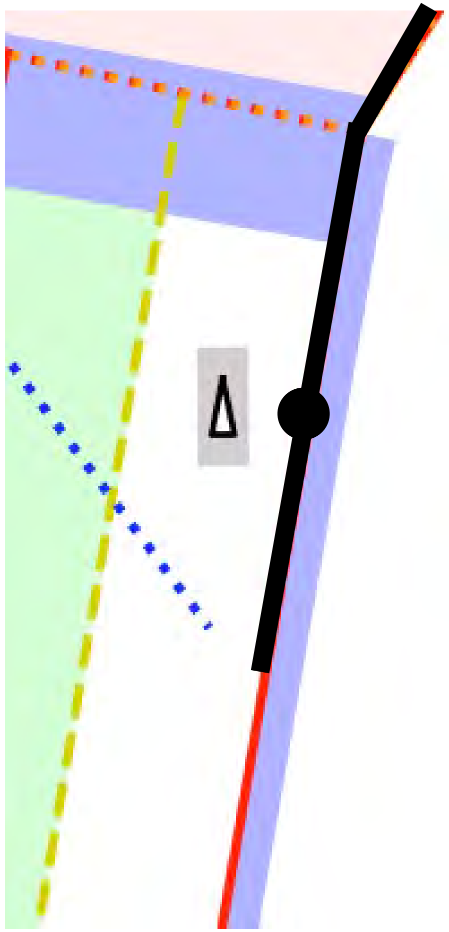
use default values for position, orientation, and other properties

Example: a Badly-Parked Car



```
model scenic.domains.driving.model  # defines Car, etc.

ego = new Car

spot = new OrientedPoint on visible curb
```

Example: a Badly-Parked Car



```
model scenic.domains.driving.model  # defines Car, etc.

ego = new Car

spot = new OrientedPoint on visible curb
```

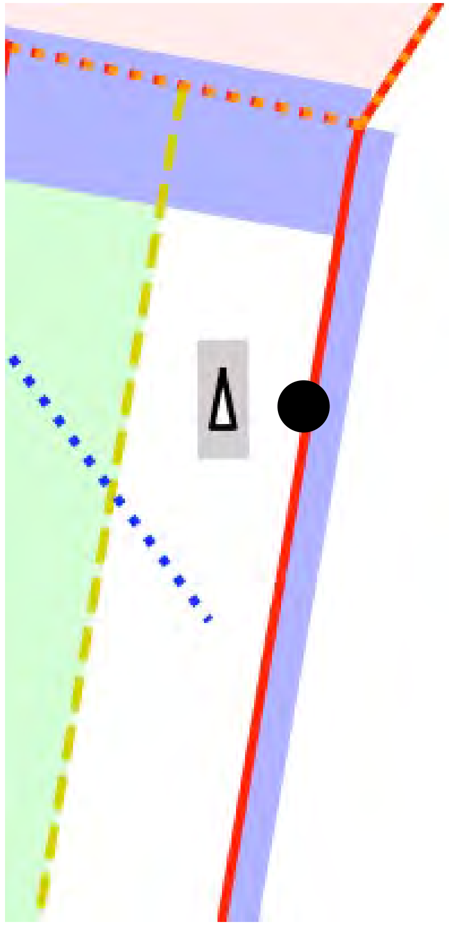
region

class

specifier

function

Example: a Badly-Parked Car

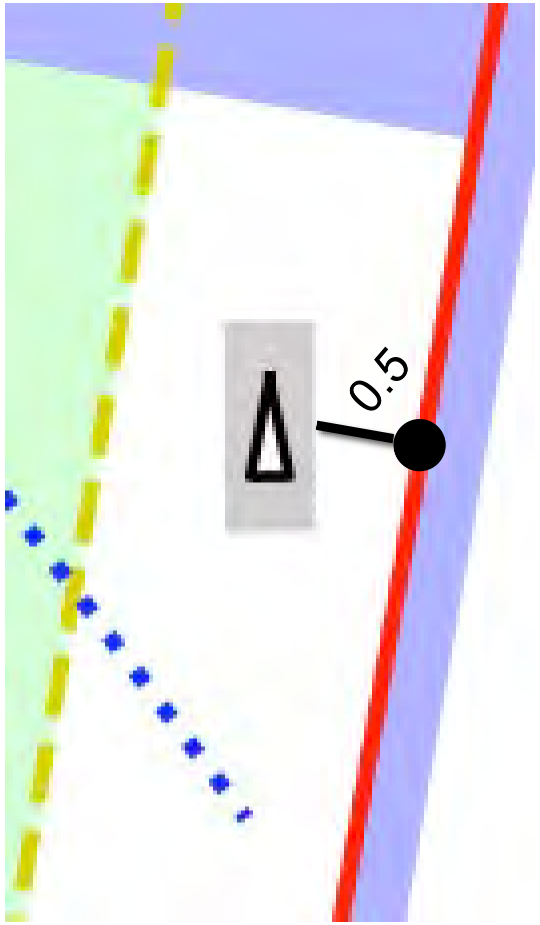


```
model scenic.domains.driving.model  # defines Car, etc.

ego = new Car

spot = new OrientedPoint on visible curb
badAngle = Uniform(1, -1) * Range(10, 20) deg
```

uniform choice of one of these values

uniform distribution over this interval

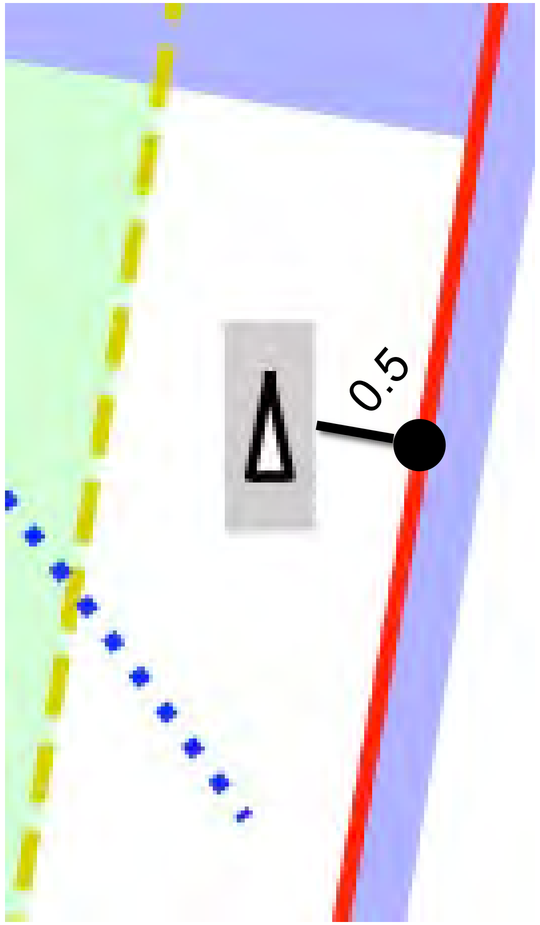
Example: a Badly-Parked Car




```
model scenic.domains.driving.model  # defines Car, etc.

ego = new Car

spot = new OrientedPoint on visible curb
badAngle = Uniform(1, -1) * Range(10, 20) deg
new Car left of spot by 0.5,
```

specifier depending
on car's width

Example: a Badly-Parked Car

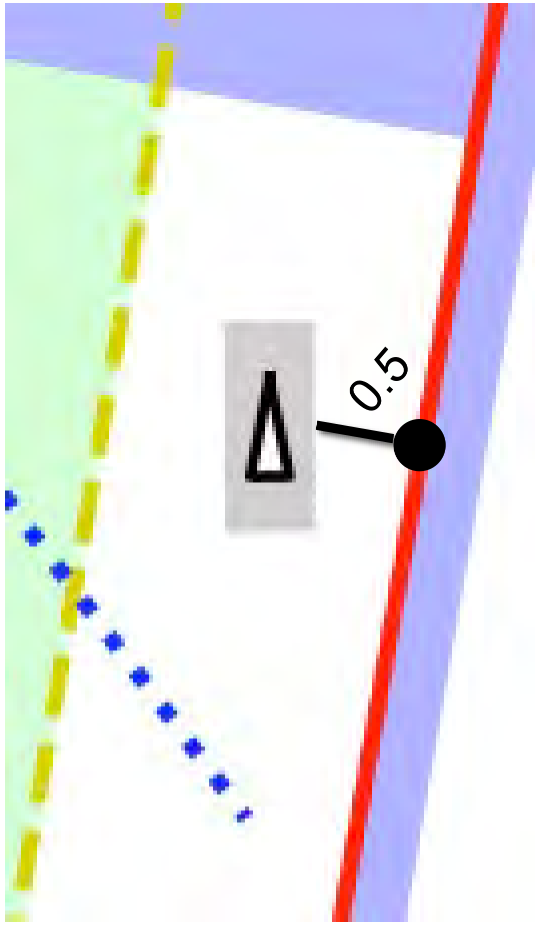



```
model scenic.domains.driving.model  # defines Car, etc.

ego = new Car

spot = new OrientedPoint on visible curb
badAngle = Uniform(1, -1) * Range(10, 20) deg
new Car left of spot by 0.5,
        facing badAngle relative to roadDirection
```

operator depending on car's position

vector field

Example: a Badly-Parked Car




```
model scenic.domains.driving.model  # defines Car, etc.

ego = new Car

spot = new OrientedPoint on visible curb
badAngle = Uniform(1, -1) * Range(10, 20) deg
new Car left of spot by 0.5,
        facing badAngle relative to roadDirection
```

Implicit hard constraints: objects don’t overlap, cars are contained in the road, ego can see the other car.

Example: a Badly-Parked Car

Hands-on Scenic: Static Scenarios

- Work through some examples in our Colab notebook

- Feel free to try anything out; we're here to help!

Static Scenic Cheat Sheet

```
# Built-in classes with their main properties
new Point  # position, visibleDistance
new OrientedPoint  # adds orientation, viewAngles
new Object  # adds shape, width, length, height, etc.

# Defining an object with specifiers
new MyObjClass with prop value,  # set any property
    at (x, y, z),
    above positionOrObject by distance,
    left of ... by ...,
    offset by vector,  # position relative to ego
    in region,  # place center randomly in region
    contained in region,  # whole object fits in region
    on surface,
    following vectorField from start for distance,
    visible,  # likewise "not visible"
    visible from obj,
    facing (yaw, pitch, roll),
    facing toward position,  # likewise "away from"
    facing directly toward position
```

```
# Probability distributions
Range(low, high)  # uniform on interval
Uniform(option1, option2, option3)
Uniform(*listOfOptions)
Discrete({opt1: wt1, opt2: wt2})
Normal(mean, stdDev)

# Requirements
require condition
require[probability] condition

# Operators
angle deg  # write an angle in degrees
distance from X to Y
obj can see positionOrObject
positionOrObject in region
visible region  # likewise "not visible"

# Class definitions
class MyClass:  # superclass is Object
    prop: defaultValue  # can be random
```

WRITING DYNAMIC SCENARIOS

Going Beyond Initial Conditions

- Scenic can also describe *dynamic agents* which take actions over time, reacting to a changing environment

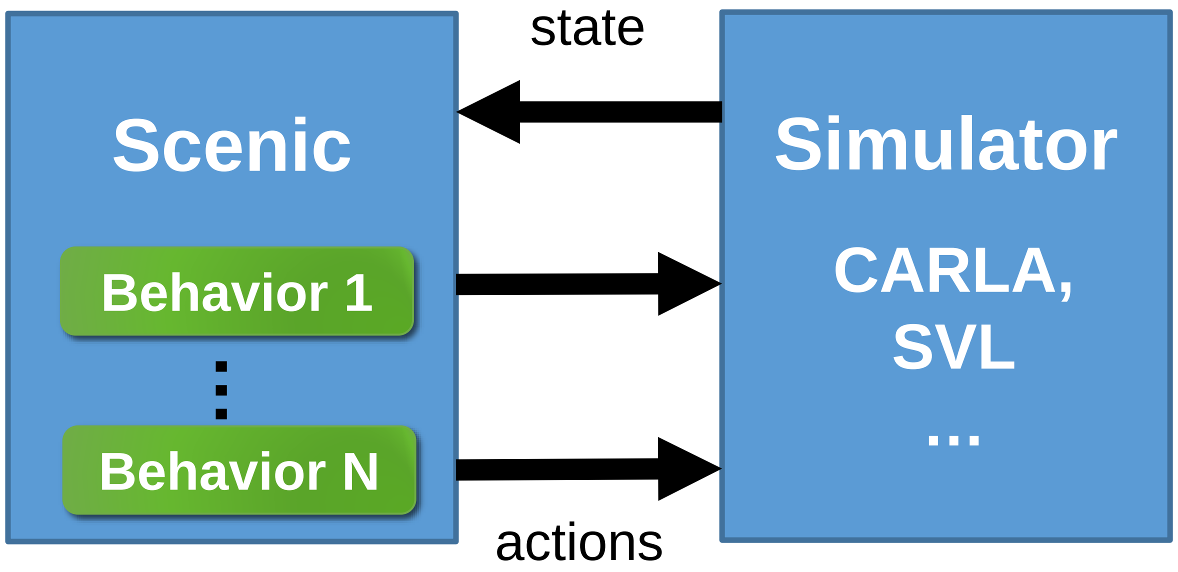
- Example: ”a badly-parked car, which suddenly pulls into the road as the ego car approaches”

- The dynamic actions of the car are specified by giving it a *behavior*

```
parkedCar = new Car left of spot by 0.5,
                facing badAngle relative to roadDirection,
                with behavior PullIntoRoad
```

Behaviors and Actions

- Behaviors are functions running in parallel with the simulation, issuing *actions* at each time step
  - e.g. for AVs: set throttle, set steering angle, turn on turn signal
  - Provided by a Scenic library for the driving domain
  - Abstract away details of simulator interface
- Behaviors can access the state of the simulation and make choices accordingly




```
behavior FollowLaneBehavior(lane):
    while True:
        throttle, steering = ...
        take (SetThrottleAction(throttle),
              SetSteerAction(steering))
```

More Advanced Temporal Constructs

- *Interrupts* allow adding special cases to behaviors without modifying their code

```
behavior FollowLeadCar(safety_distance=10):
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when distance to other < safety_distance:
        do CollisionAvoidance()
```

- *Temporal requirements* and *monitors* allow enforcing constraints during simulation

```
require always taxi in lane
require eventually ego can see pedestrian
```

A Worked Example

- OAS Voyage Scenario 2-2-XX-CF-STR-CAR:02

- Lead car periodically stops and starts; ego car must brake to avoid collision

```
behavior FollowLeadCar(safety_distance=10):
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when distance to other < safety_distance:
        do CollisionAvoidance()

behavior StopsAndStarts():
    stop_delay = Range(3, 6)
    last_stop = 0
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when simulation().currentTime - last_stop > stop_delay:
        do FullBraking() for 5 seconds
        last_stop = simulation().currentTime

ego = new Car with behavior FollowLeadCar(safety_distance=10)
other = new Car ahead of ego by 10,
                with behavior StopsAndStarts

require (new Point ahead of ego by 100) in road

terminate when ego._lane is None
```

A Worked Example: CARLA

Hands-on Scenic: Dynamic Scenarios

- We’ll use Scenic’s built-in simple Newtonian physics simulator

- As before, feel free to improvise!

- We can run some of your scenarios in CARLA at the end

Dynamic Scenic Cheat Sheet

```
new Object with behavior MyBehavior(args)

behavior MyBehavior(arg=defaultValue):
    take Action1(), Action2()  # take actions this step
    wait

    do OtherBehavior()  # run another behavior
    do Other() for 5 seconds
    do Other() until condition

    try:
        do A()  # do this...
    interrupt when condition:  # until this happens
        do B()  # then run this and resume A

    if condition:  # as in Python; random conditions OK
        ...
    while condition:
        ...
```

```
require condition  # at start
require always condition
require eventually condition

terminate after 15 seconds
terminate when condition
```

MORE SCENIC FEATURES

Composing Scenarios

- Scenic allows scenarios to be defined modularly and combined into more complex scenarios

- Parallel, sequential, and more complex forms of composition

```
import StopAndStart, BadlyParkedCar

scenario StopStartWithParkedCar():
    compose:
        do StopAndStart(), BadlyParkedCar()


scenario StopStartThenParkedCar():
    compose:
        do StopAndStart()
        do BadlyParkedCar()


scenario StopStartThenParkedCar():
    compose:
        try:
            do StopAndStart()
        interrupt when ...:
            do BadlyParkedCar()
```

Orchestrating Simulations

- Recording data from simulations

```
record ego can see pedestrian as "ped_visible"
record final distance to goal as "end_dist"
```

- Saving and replaying scenes and simulations

```
scenario = scenic.scenarioFromFile('examples/gta/parkedCar.scenic', mode2D=True)
scene, _ = scenario.generate()
data = scenario.sceneToBytes(scene)
```

- Integration with *VerifAI* for falsification, optimization, etc.

Resources for Learning More

- Documentation: docs.scenic-lang.org
  - Installation instructions for all major platforms
  - Full versions of today's tutorials, plus others
  - Syntax Guide and detailed Language Reference
  - Python API
  - How to interface Scenic to a new simulator
- Community forum: forum.scenic-lang.org

Main website:

scenic-lang.org

Applications of Scenic (Part I)

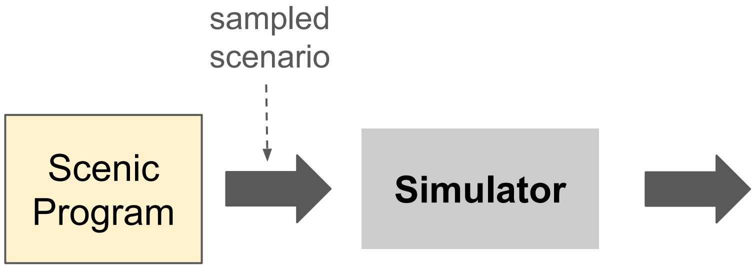
Edward Kim

EECS, UC Berkeley

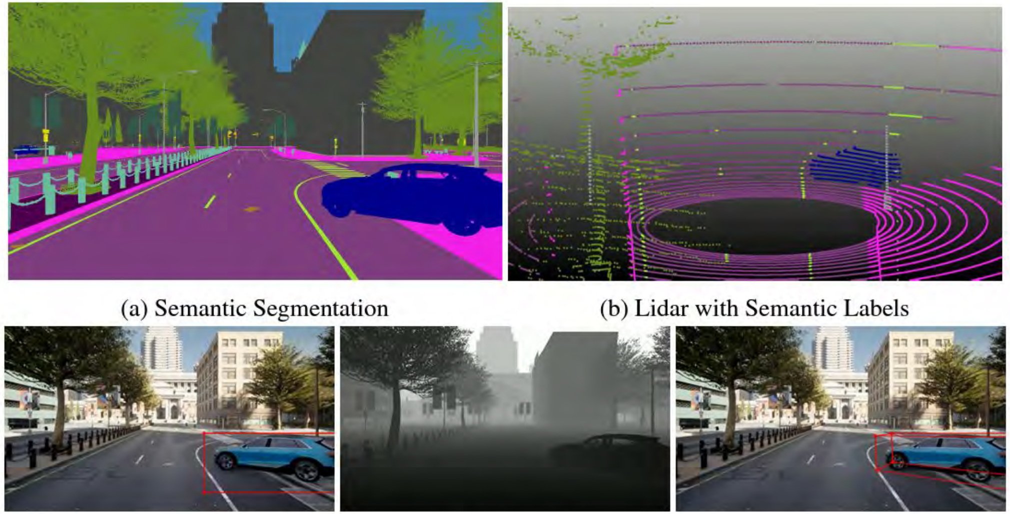
Outline

1. Sensor Data Collection
2. Testing Perception, Behavior Prediction, and Planners with VerifAI
3. Interfacing Scenic to Your Simulator of Choice

Sensor Data Collection



A Set of Sensor Data + GT Labels



(a) Semantic Segmentation (b) Lidar with Semantic Labels

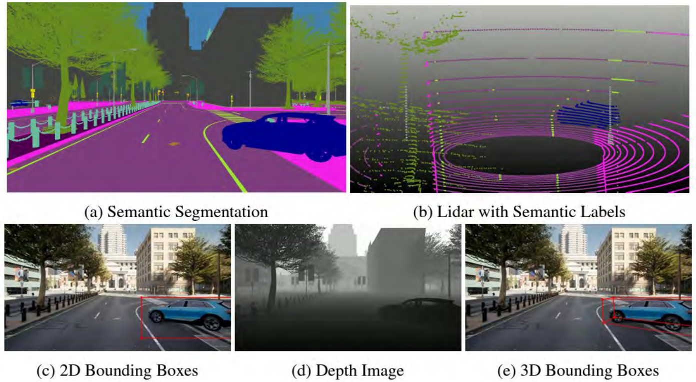
(c) 2D Bounding Boxes (d) Depth Image (e) 3D Bounding Boxes

Sensor Data Generation

```
model scenic.simulators.carla.model

# Sample a lane at random
lane = Uniform(*network.lanes)

spot = new OrientedPoint on lane.centerline

car_model = "vehicle.tesla.model3"

# Spawn car on that spot with logging autopilot behavior and
# - SSSensor : a semantic segmentation sensor
# - RGBSensor: an RGB Camera
ego = new Car at spot,
    with blueprint car_model,
    with behavior AutopilotBehavior(),
    with sensors {"front_rgb": RGBSensor(offset=(1.6, 0, 1.7)),
                  "front_ss": SSSensor(offset=(1.6, 0, 1.7))}

other = new Car offset by 0 @ Range(10, 30),
    with behavior AutopilotBehavior()

record ego.observations["front_rgb"] as "front_rgb" every 0.1 sec
record ego.observations["front_ss"] as "front_ss" every 0.1 sec
```

Testing Perception, Behavior Prediction, and Planners

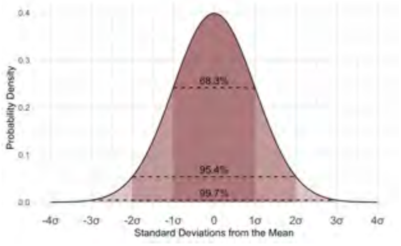
A Set of Sensor Data + GT Labels

Scenic Program → sampled scenario → **Simulator** → 

Test → **ML Component**

Defines a Distribution of Scenarios

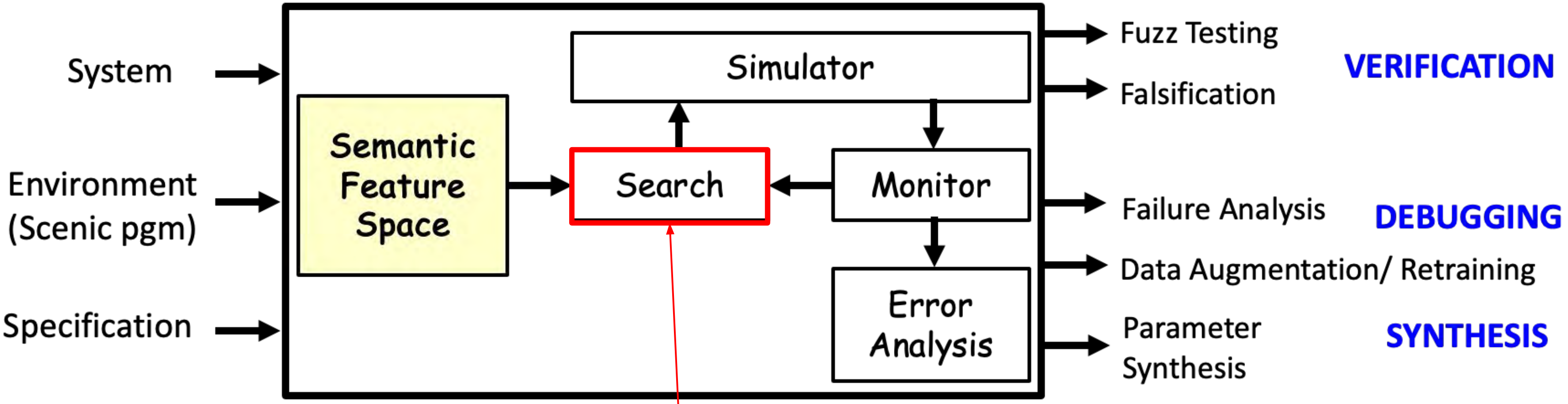
Challenge:



How to Search/Sample Scenarios from the Distribution?

- **Exploration**
- **Exploitation**

VERIFAI: A Toolkit for the Design and Analysis of AI-Based Systems



Supports **Different Search Strategies**: **[You Can Add Your Own Samplers Too]**
Bayesian Optimization, Multi-Arm Bandit, Cross Entropy, Simulated Annealing, Halton, Random

[Dreossi et al. CAV 2019, https://github.com/BerkeleyLearnVerify/VerifAI]

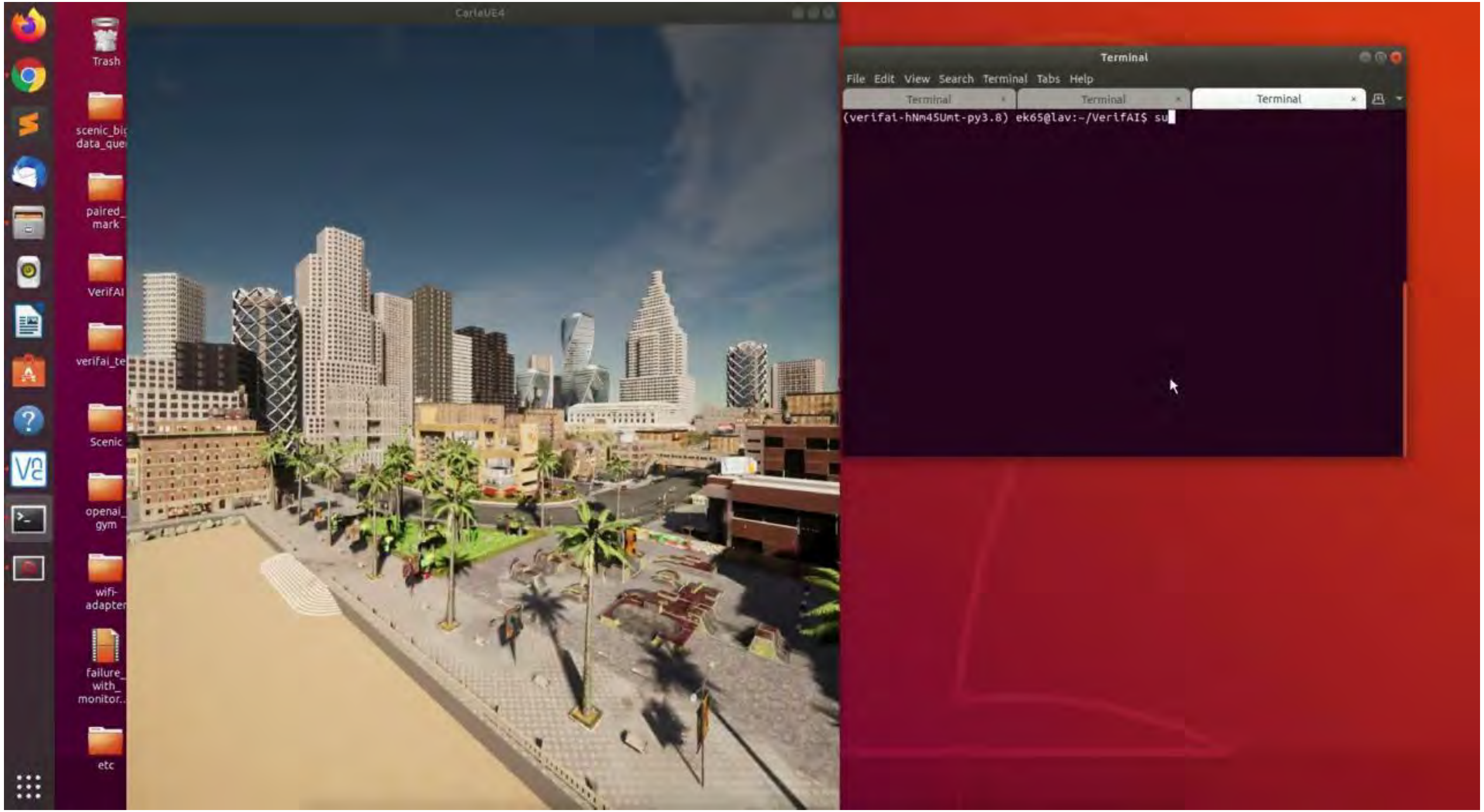
Testing with Scenic & VerifAI

```
model scenic.simulators.carla.model
import pytorch

# import the machine learning model
ML_model = pytorch.load(...)

# Sample a lane at random
lane = Uniform(*network.lanes)

spot = new OrientedPoint on lane.centerline

car_model = "vehicle.tesla.model3"

# Spawn car on that spot with logging autopilot behavior and
# - SSSensor : a semantic segmentation sensor
# - RGBSensor: an RGB Camera
ego = new Car at spot,
    with blueprint car_model,
    with behavior AutopilotBehavior(),
    with sensors {"front_rgb": RGBSensor(offset=(1.6, 0, 1.7)),
                  "front_ss": SSSensor(offset=(1.6, 0, 1.7))}

other = new Car offset by 0 @ Range(10, 30),
    with behavior AutopilotBehavior()

record ego.observations["front_rgb"] as "front_rgb" every 0.1 sec
record ego.observations["front_ss"] as "front_ss" every 0.1 sec
record ML_model(ego.observations["front_rgb"],
        ego.observations["front_ss"]) as "ml_performance" every 0.1 sec
```

Testing with Scenic & VerifAI

```python
from verifai.monitor import specification_monitor
# The specification must assume specification_monitor class
class evaluation_metric(specification_monitor):
    def __init__(self):
        def specification(simulation_results):
            time_series_IOU = simulation_results['ml_performance']
            mean_IOU = compute_Mean_IOU(time_series_IOU)
            return mean_IOU
        super().__init__(specification)


from dotmap import DotMap
falsifier_params = DotMap(
        n_iters=1000,    # Number of simulations to run (or None for no limit)
        max_time=20,  # Time limit in seconds, if any
        fal_thres=0.5,    # Monitor return value below which a sample is considered a violation
        save_error_table=True,    # Record samples that violated the monitor/specification
        save_safe_table=False,  # Don't record samples that satisfied the monitor/specification
        sampler_params=None    # optional DotMap of sampler-specific parameters
)
```

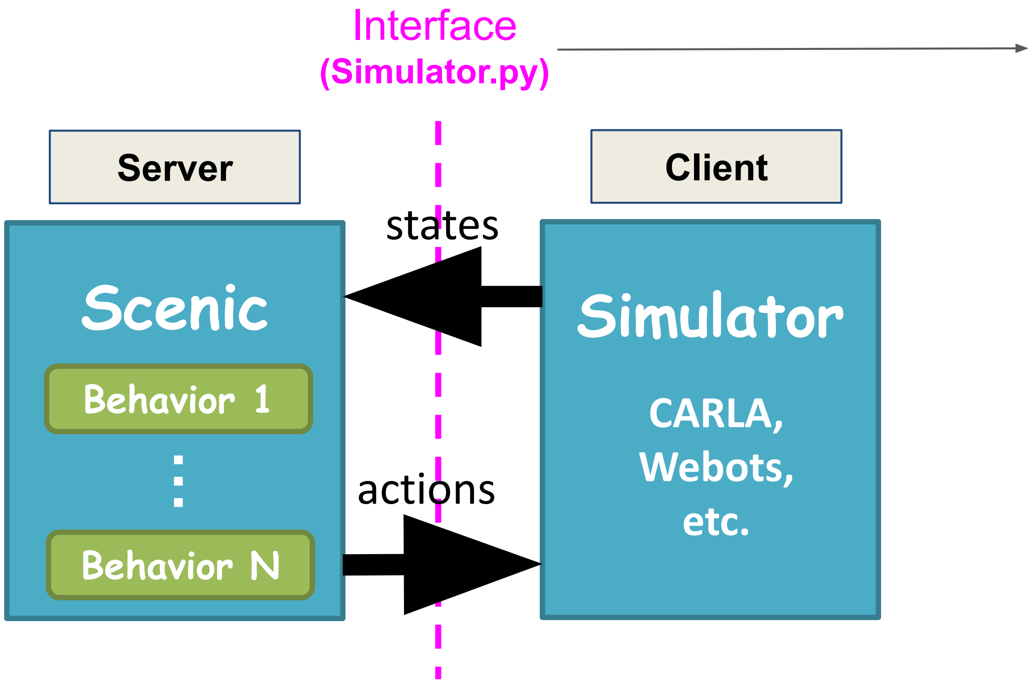
Demo



Trash
VerifAI
Scenic
etc
Terminal
File Edit View Search Terminal Tabs Help
(verifai-hNm45Umt-py3.8) ek65@lav:~/VerifAI$ su

Q & A

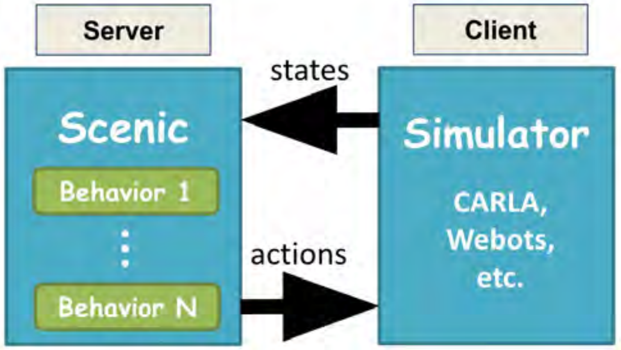
Interfacing Scenic to Your Simulator of Choice



Roles of the Interface

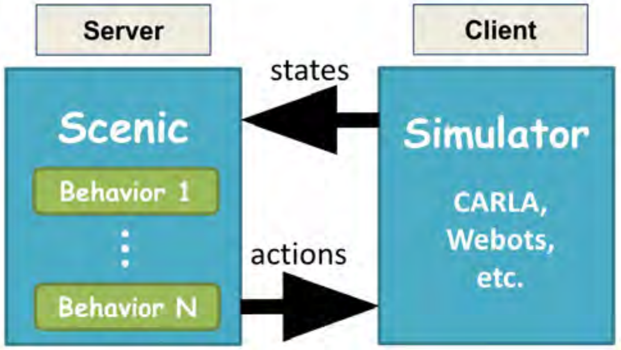
- Initiate Client/Server Communication
- Instantiate the Map, Agents / Objects
- Control the actions of each agent
- Simulate the actions for 1 timestep
- Update the world state at each timestep

Interfacing Scenic to Your Simulator of Choice



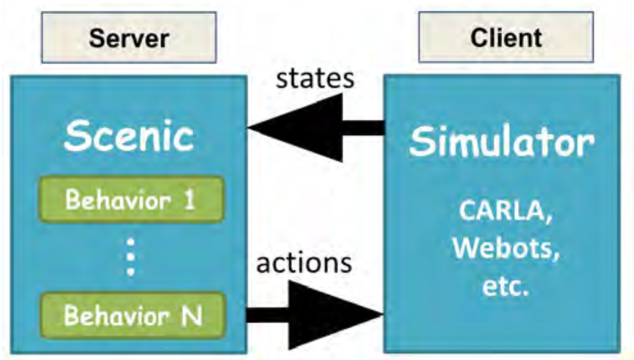

```python
class TemplateSimulator(Simulator):
    """
    Implementation of `Simulator` in scenic.core.simulator.
    At each simulation, Simulator creates a Simulation class
    The Simulation class then runs the actual simulation.
    """

    def __init__(self):
        """
        Initialize server <> client relation between Scenic and the simulator
        """

        super().__init__()
```

Interfacing Scenic to Your Simulator of Choice



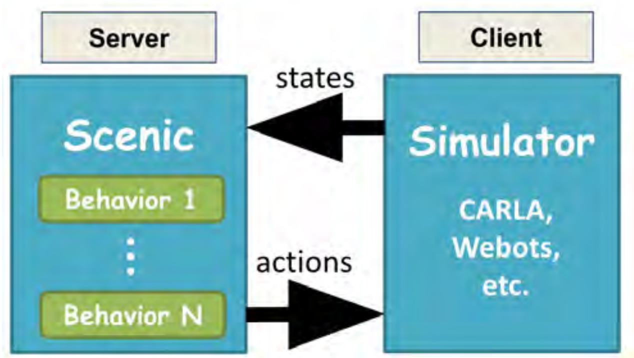

```
def createObjectInSimulator(self, obj):
    """
    Arg
    Object obj: a Scenic obj/agent

    Spawns a single object/agent in the simulator
    with the desired parameters (position, orientation, color, etc.)
```

Interfacing Scenic to Your Simulator of Choice



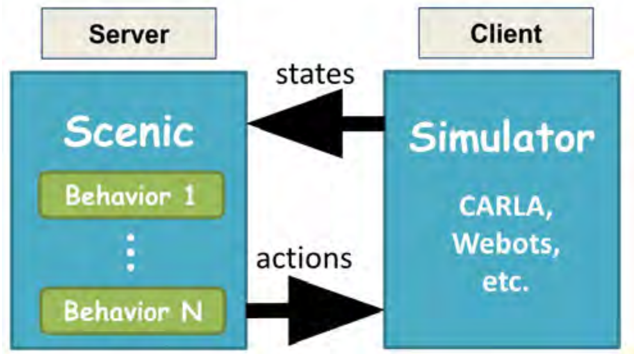

```
def step(self):
    """
    This function unfreezes the simulation world and
    advances the physics by 1 timestep.
    When this function is called, all the Scenic Actions will be executed and
    rendered in simulation.
```

Interfacing Scenic to Your Simulator of Choice



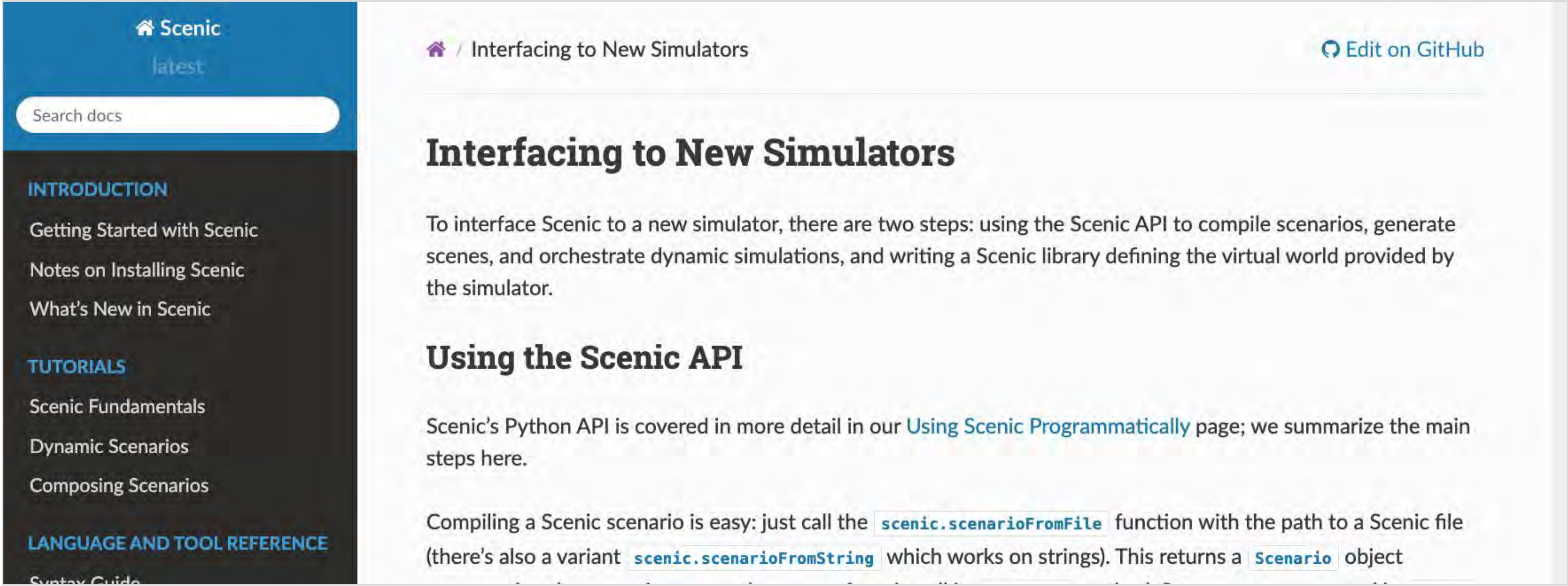

```
def getProperties(self, obj, properties):
    """
    Args:
    obj (Object): Scenic object in question.
    properties (set): Set of names of properties to read from the simulator.
    It is safe to destructively iterate through the set if you want.

    Returns:
    values (dict): A dictionary that with the property names as keys
    that returns the current value of the property
```

Interfacing Scenic to Your Simulator of Choice




```
def executeActions(self, allActions):
    """
    Args:
    allActions: a :obj:`~collections.defaultdict` mapping each agent to a tuple
                of actions, with the default value being an empty tuple. The order of
                agents in the dict should be respected in case the order of actions matters.

    Iterates through all the Scenic Actions to be carried out for all the agents.
    For each Scenic Action, this function calls the applyTo() method of the Action's class
    as defined in actions.py.
```

Interfacing Scenic to Your Simulator of Choice



Scenic
latest
Search docs

INTRODUCTION
Getting Started with Scenic
Notes on Installing Scenic
What's New in Scenic

TUTORIALS
Scenic Fundamentals
Dynamic Scenarios
Composing Scenarios

LANGUAGE AND TOOL REFERENCE

/ Interfacing to New Simulators

Edit on GitHub

Interfacing to New Simulators

To interface Scenic to a new simulator, there are two steps: using the Scenic API to compile scenarios, generate scenes, and orchestrate dynamic simulations, and writing a Scenic library defining the virtual world provided by the simulator.

Using the Scenic API

Scenic's Python API is covered in more detail in our Using Scenic Programmatically page; we summarize the main steps here.

Compiling a Scenic scenario is easy: just call the `scenic.scenarioFromFile` function with the path to a Scenic file (there's also a variant `scenic.scenarioFromString` which works on strings). This returns a `Scenario` object

https://scenic-lang.readthedocs.io/en/latest/new_simulator.html

Summary

1. Flexible Sensor Data Generation using Scenic
2. Testing Perception, Behavior Prediction, and Planners with Scenic & VerifAI
3. Scenic is Simulator-Agnostic & Can be Interfaced to a Simulator of Choice

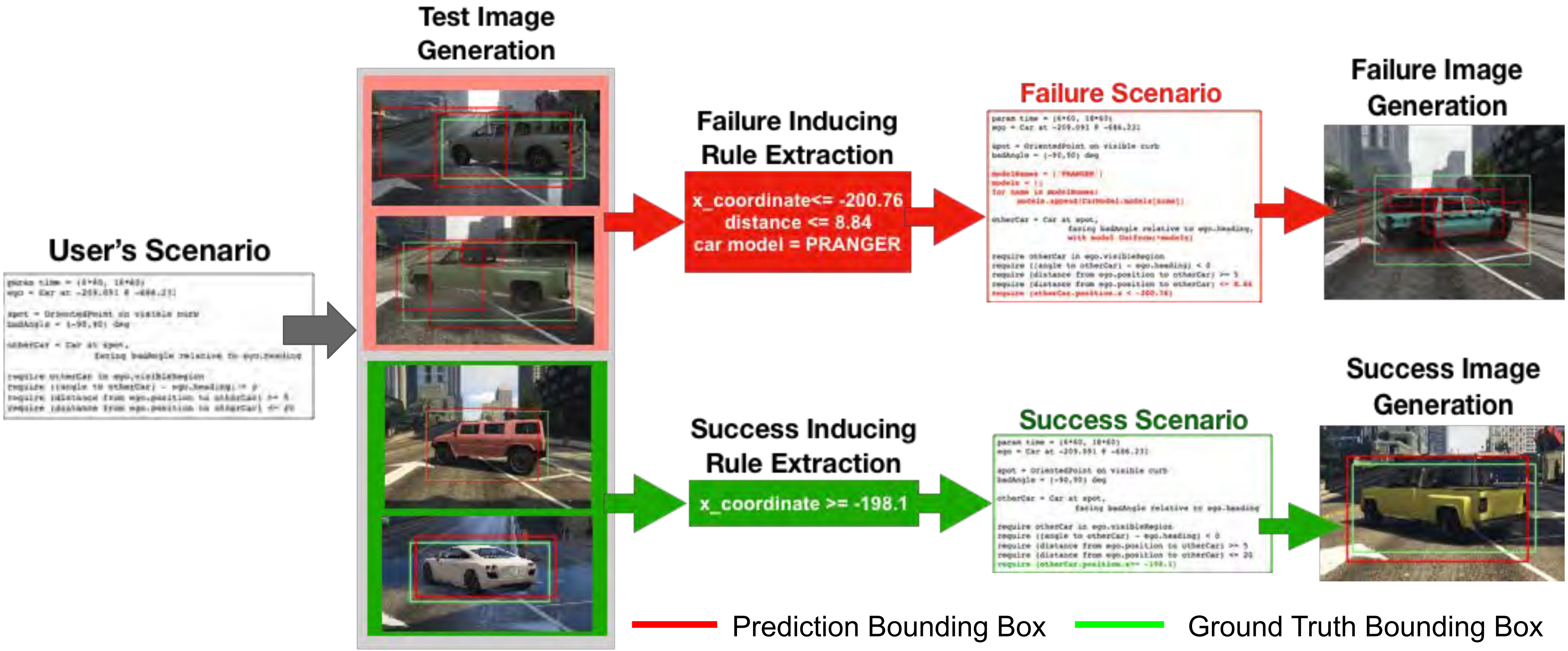
Applications of Scenic (Part II)

Edward Kim
EECS, UC Berkeley

Outline

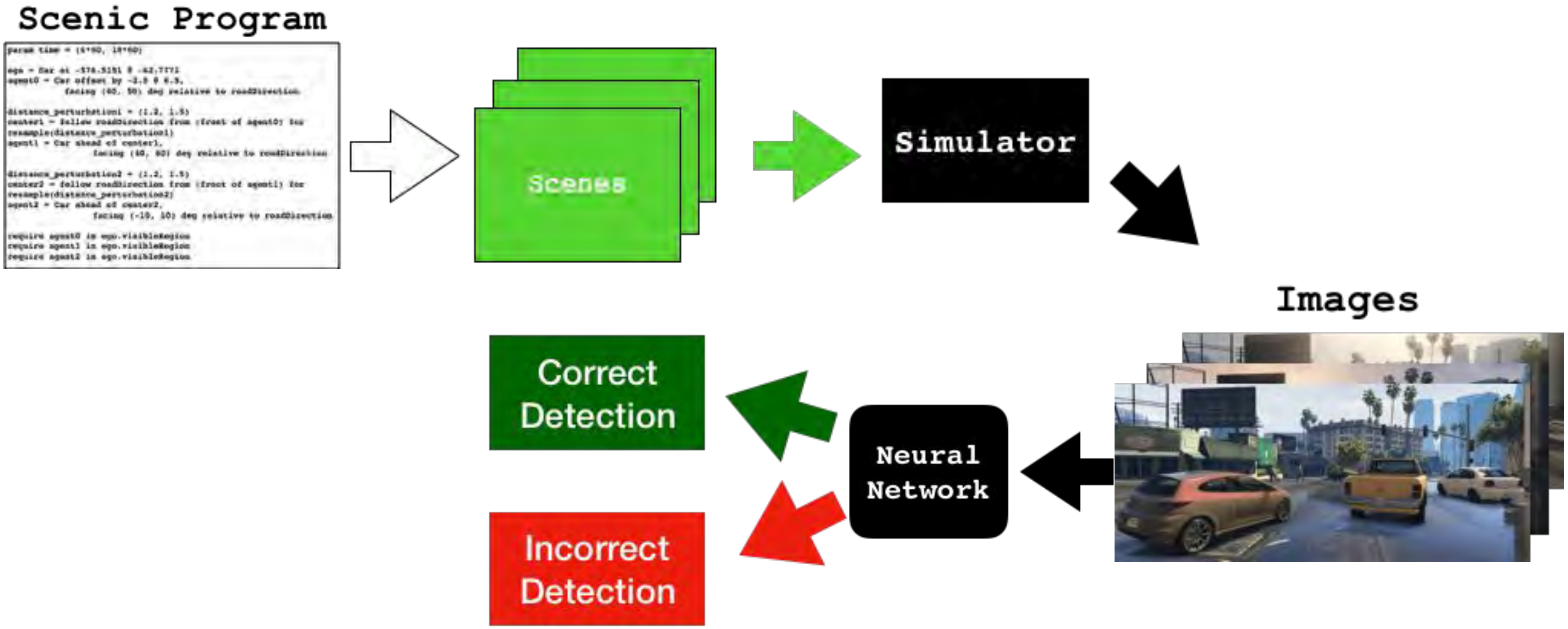
1. Debugging ML Models
2. Sim-to-Real Validation
3. Sensor Data Exploration
4. Extended Reality

Debugging ML Models



Edward Kim, Divya Gopinath, Corina Pasareanu, Sanjit Seshia, "A Programmatic and Semantic Approach to Explaining and Debugging Neural Network Based Object Detectors," CVPR 2020

Data Generation Pipeline



Data Collection

Scenes **Labels**

<e1_1, e1_2, .., e1_n> → correct
<e2_1, e2_2, .., e2_n> → correct
<e3_1, e3_2, .., e3_n> → incorrect
<e4_1, e4_2, .., e4_n> → correct
. .
. .
. .
. .
. .
<em_1, em_2, .., em_n> → correct

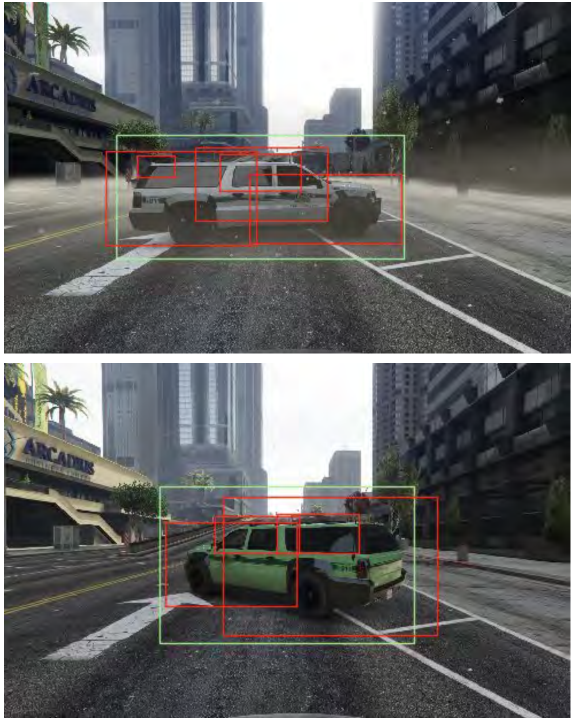
e.g. <weather='snowy,' car_model='cybertruck,' …. >

Classification Problem

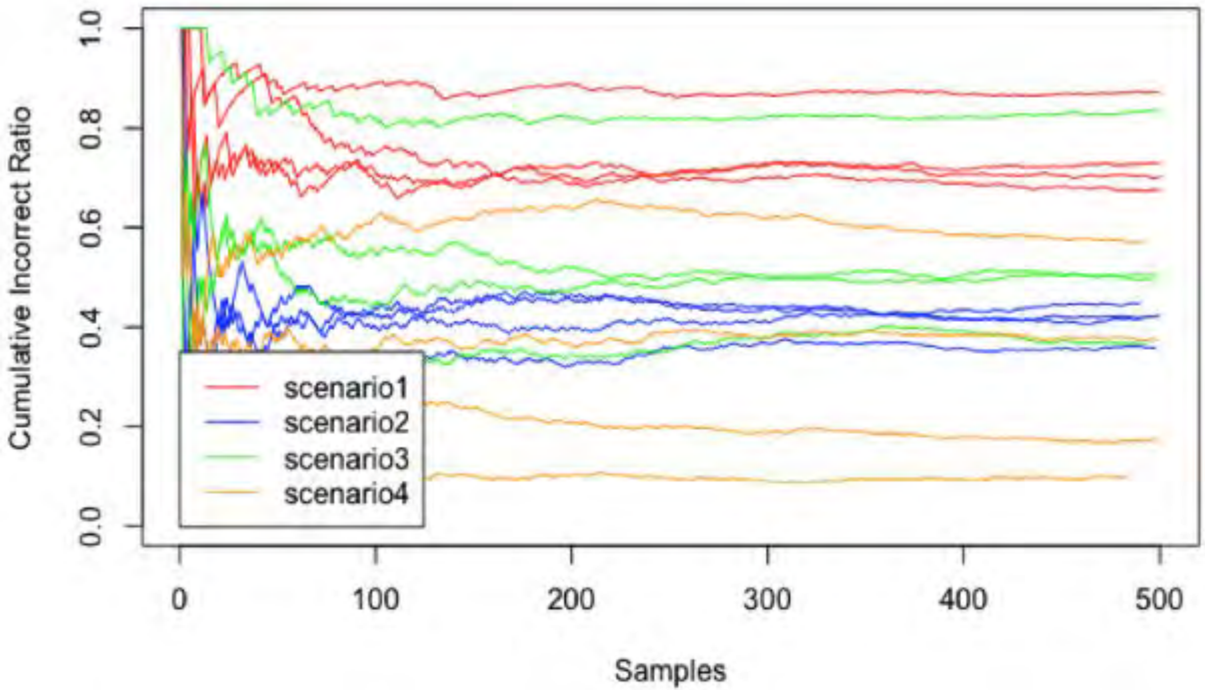
Summary on Worst Failure Inducing Rules

Scenario # (Baseline→Rule Precision)	Rules
Scenario1 (34.7% → 87.2%)	**x coordinate** $\leqslant$ -200.76 $\wedge$ **distance** $\leqslant$ 8.84 $\wedge$ **car model** = PRANGER
Scenario 2 (27.7% → 44.9%)	**hour** $\geqslant$ 7.5 $\wedge$ **weather** = all except Neutral $\wedge$ **car0 distance from ego** < 11.3
Scenario 3 (38.3% → 83.4%)	**weather** = neutral $\wedge$ **agent0 heading =** $\leqslant$ 218.08 deg $\wedge$ **hour** $\leqslant$ 8.00 $\wedge$ **car2 red color** $\leqslant$ 95.00
Scenario 4 (10.4% → 57.3%)	**car0 model** = PATRIOT $\wedge$ **car1 model** = NINEF $\wedge$ **car2 model** = BALLER $\wedge$ 92.25 < **car0 green color** $\leqslant$ 158 $\wedge$ **car0 blue color** $\leqslant$ 84.25 $\wedge$ 178.00 < **car2 red color** $\leqslant$ 224

From Failure Scenic Program



Based on Debugging, Generate Failure Inducing Data



Outline

1. Debugging ML Models
2. Sim-to-Real Validation
3. Sensor Data Exploration
4. Extended Reality

Sim-to-Real Validation

Do system / component failures identified in simulation *actually occur* in reality?

Potential Causes of Discrepancy

1. Sensor Data
2. Dynamics Models
3. Agent Behaviors

⋮

Sim-to-Real Validation

Daniel J Fremont, Edward Kim, Yash Vardhan Pant, Sanjit A Seshia, Atul Acharya, Xantha Bruso, Paul Wells, Steve Lemke, Qiang Lu, Shalin Mehta, “Formal scenario-based testing of autonomous vehicles: From simulation to the real world,” International Conference on Intelligent Transportation Systems (ITSC), 2020

Experiment Results

Collision

Near-Collision

Unsafe with Larger Margin

Effectiveness of the Methodology to Design Test Cases:

Unsafe Tests in Simulation → unsafe in real-world: 62.5%
Marginally safe in simulation → safe in real world: 90%
Robustly safe in simulation → safe in real world: 100 %

Test Run	Minimum TTC	Minimum Distance	ρ
F1 Simulation	–	2.23	-0.27
F1 Run1	2.10	2.06	-0.44
F1 Run2	1.27	2.24	-0.26
F1 Run3	2.97	4.02	1.52
F1 Run4	5.05	6.19	3.69
F2 Simulation	–	1.91	-0.59
F2 Run1	0.94	2.44	-0.06
F2 Run2	2.70	3.24	0.74
F2 Run3	1.20	1.58	-0.92
F2 Run4	1.05	2.24	-0.26
M1 Simulation	–	4.05	1.55
M1 Run1	6.07	7.20	4.70
M1 Run2	7.16	7.89	5.39
M2 Simulation	–	4.78	2.28
M2 Run1	3.24	3.40	0.90
M2 Run2	6.16	8.01	5.51
M2 Run3	9.10	14.38	11.88
M2 Run4	6.80	8.05	5.55
M2 Run5	7.69	8.48	5.98
M3 Simulation	–	5.85	3.35
M3 Run1	0.75	1.94	-0.56
M3 Run2	6.00	6.36	3.86
M3 Run3	4.27	5.73	3.23
S1 Simulation	–	5.45	2.95
S1 Run1	1.32	2.79	0.29
S1 Run2	9.72	8.50	6.00
S1 Run3	9.35	7.85	5.35
S2 Simulation	–	5.95	3.45
S2 Run1	3.13	6.36	3.86
S2 Run2	8.66	9.00	6.50

Outline

1. Debugging ML Models
2. Sim-to-Real Validation
3. Sensor Data Exploration
4. Extended Reality

In this era of AI, terabytes of sensor data are being collected and labelled.

The size of data matters, but also contents matters

e.g. what if a self-driving car's training dataset **does not** contain any unprotected left turn scenario?

How can we algorithmically explore and understand the dataset?

Part 3: **Sensor Data Retrieval**

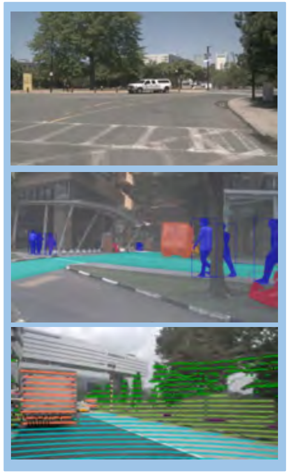
Encode Failures as a Scenario Program

```
param time = (6*60, 18*60)
ego = Car at -209.091 @ -686.231

spot = OrientedPoint on visible curb
badAngle = (-90,90) deg

otherCar = Car at spot,
               facing badAngle relative to ego.heading

require otherCar in ego.visibleRegion
require ((angle to otherCar) - ego.heading) < 0
require (distance from ego.position to otherCar) >= 5
require (distance from ego.position to otherCar) <= 20
```

A Given Labelled Real World Dataset

Proposed Query Algorithm

Matching Subset of Real World Sensor Data to the Scenario Description



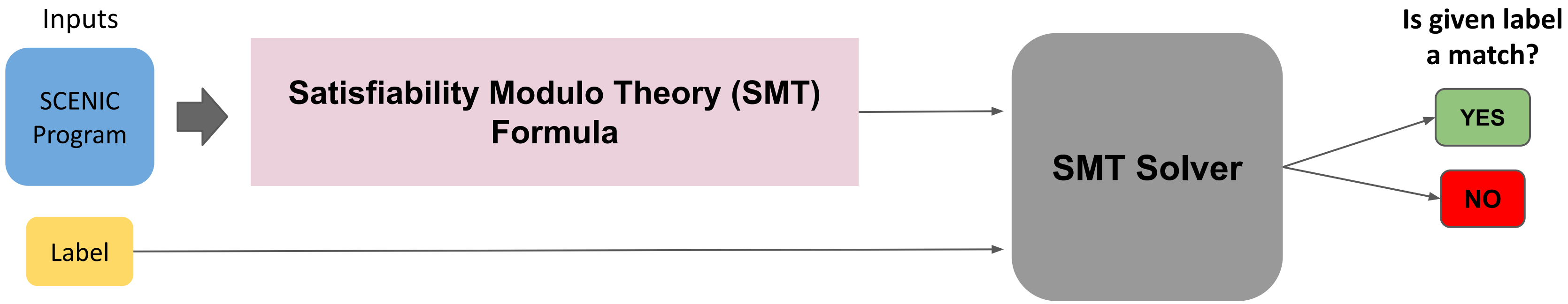
Edward Kim, Jay Shenoy, Sebastian Junges, Daniel J Fremont, Alberto Sangiovanni-Vincentelli, Sanjit A Seshia, “Querying Labelled Data with Scenario Programs for Sim-to-Real Validation,” International Conference on Cyber Physical Systems (ICCPS), 2022

Query Problem

Real Dataset Label $\overset{?}{\in}$ SCENIC Program (= distribution of labels)

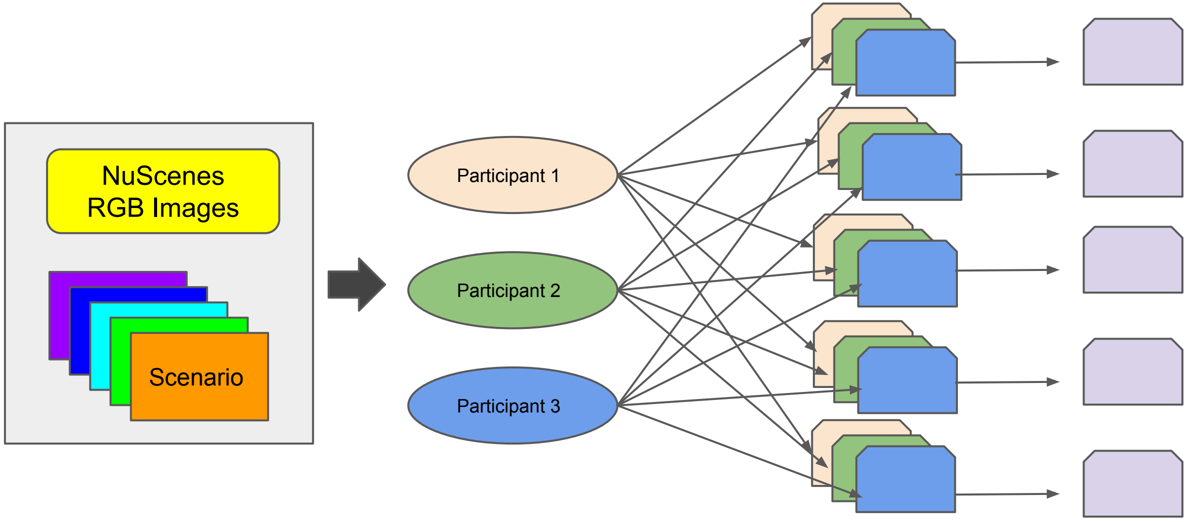
Time, positions, headings, etc.

Methodology



Experiment

1) Can we query for interesting and realistic scenarios?
2) Do the outputs of the algorithm correspond to the intuitive notion of scenario matching?



Experiment

Scenario #	1	2	3	4	5
Matching images (humans)	42	5	0	2	0
Matching images (our algorithm)	58	7	2	2	0

Table 1: For several scenarios, the number of images identified by 3 human subjects (unanimously) and our algorithm.

Edward Kim, Jay Shenoy, Sebastian Junges, Daniel J Fremont, Alberto Sangiovanni-Vincentelli, Sanjit A Seshia, “Querying Labelled Data with Scenario Programs for Sim-to-Real Validation,” International Conference on Cyber Physical Systems (ICCPS), 2022

Experiment 1 Results

Scenario #	1	2	3	4	5
Matching images (humans)	42	5	0	2	0
Matching images (our algorithm)	58	7	2	2	0

Table 1: For several scenarios, the number of images identified by 3 human subjects (unanimously) and our algorithm.

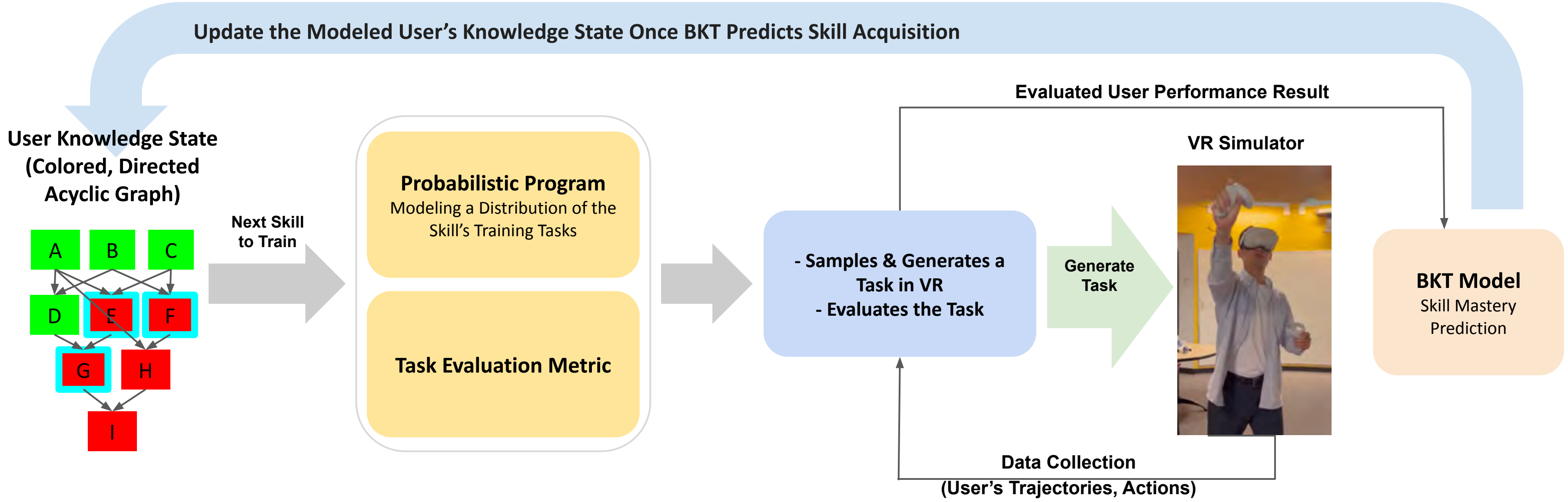
Error in the label

Limitation: the query accuracy hinges on the labels' accuracy

Outline

1. Debugging ML Models
2. Sim-to-Real Validation
3. Sensor Data Exploration
4. Extended Reality

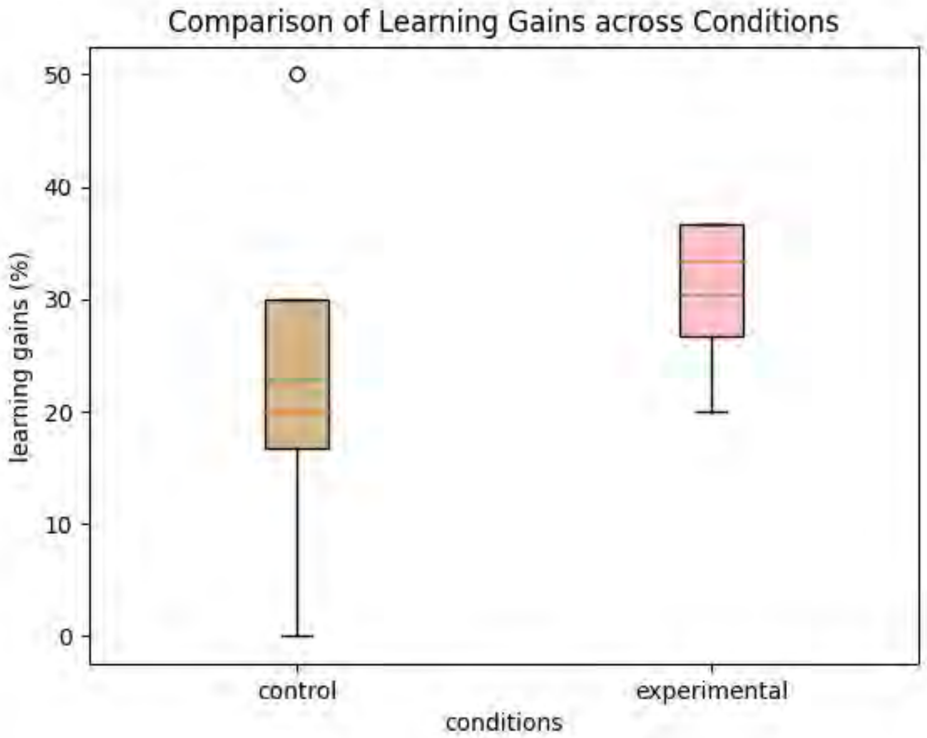
Training Motor Skills for Humans in XR



Edward Kim, Zachary Pardos, Bjoern Hartmann, Sanjit Seshia, "A Principled Intelligent Occupational Training of Psychomotor Skills in Virtual Reality," UC Berkeley EECS Technical Report No. UCB/EECS-2023-17

Example: Passing to a Dynamic Player

Experiment Result



Edward Kim, Zachary Pardos, Bjoern Hartmann, Sanjit Seshia, “A Principled Intelligent Occupational Training of Psychomotor Skills in Virtual Reality,” UC Berkeley EECS Technical Report No. UCB/EECS-2023-17

Summary

We covered different applications of Scenic

1. Testing & Debugging ML Models
2. Test Case Generation for Track Testing for Sim-to-Real Validation
3. Sensor Data Exploration using Scenic as a Query Language
4. Training Humans for Motor Skills in Extended Reality

Summary of Tutorial Topics

- Introduction to Scenic and VerifAI
  - Two Industrial Case Studies
- Hands-On Introduction to the Scenic 3.0 Language
- Applications of Scenic
  - Synthetic Data Generation
  - Testing and Falsification
  - Interfacing to Simulators
  - Debugging ML Models
  - Sim-to-Real Validation
  - Querying Data with Scenic
  - Training in Extended Reality

Scenic and VerifAI: Summary of Features and Use Cases

- Classes, Objects, 3D Geometry, and Distributions
- Local Coordinate Systems
- Readable, Flexible Specifiers
- Declarative Hard & Soft Constraints
- Externally-Controllable Parameters
- Agent Actions and Behaviors, Interrupts, Termination
- Monitors, Temporal Constraints
- Logging Simulation Data
- Scenario Composition

…

- Synthetic Data Generation
- Test Generation, Fuzz Testing
- *Requirements Specification*
- Falsification, *Statistical Model Checking*
- Debugging and Triage
- Data Augmentation
- *Goal-Directed Parameter Synthesis*
- *Run-Time Monitor Generation*
- Sim-to-Real Validation
- *Training Reinforcement Learning Agents*
- Training People in Extended Reality

…*<your use case here!>*

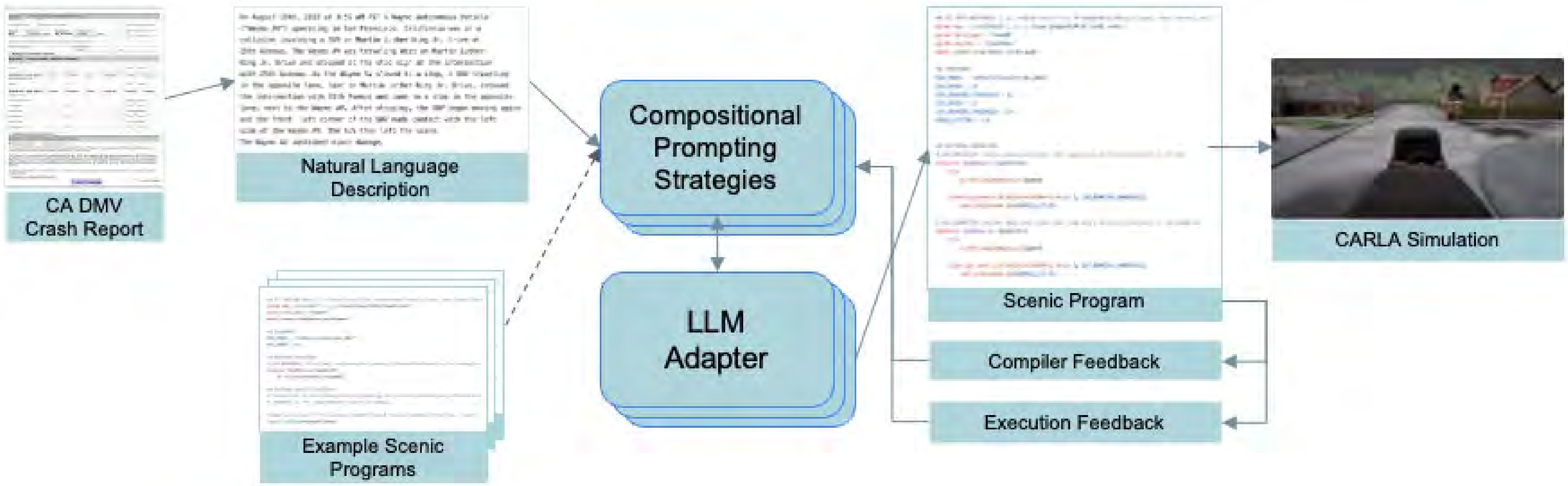
New: Generating Scenic Programs from Natural Language

We generate Scenic programs from natural language descriptions of Autonomous Vehicle crash reports

ScenicNL: Compound AI System

Input:
Over 500 California DMV Crash Reports of Autonomous Vehicles for the last 5 years

Output:
A dataset of classified scenarios and corresponding Scenic programs for a subset of those scenarios

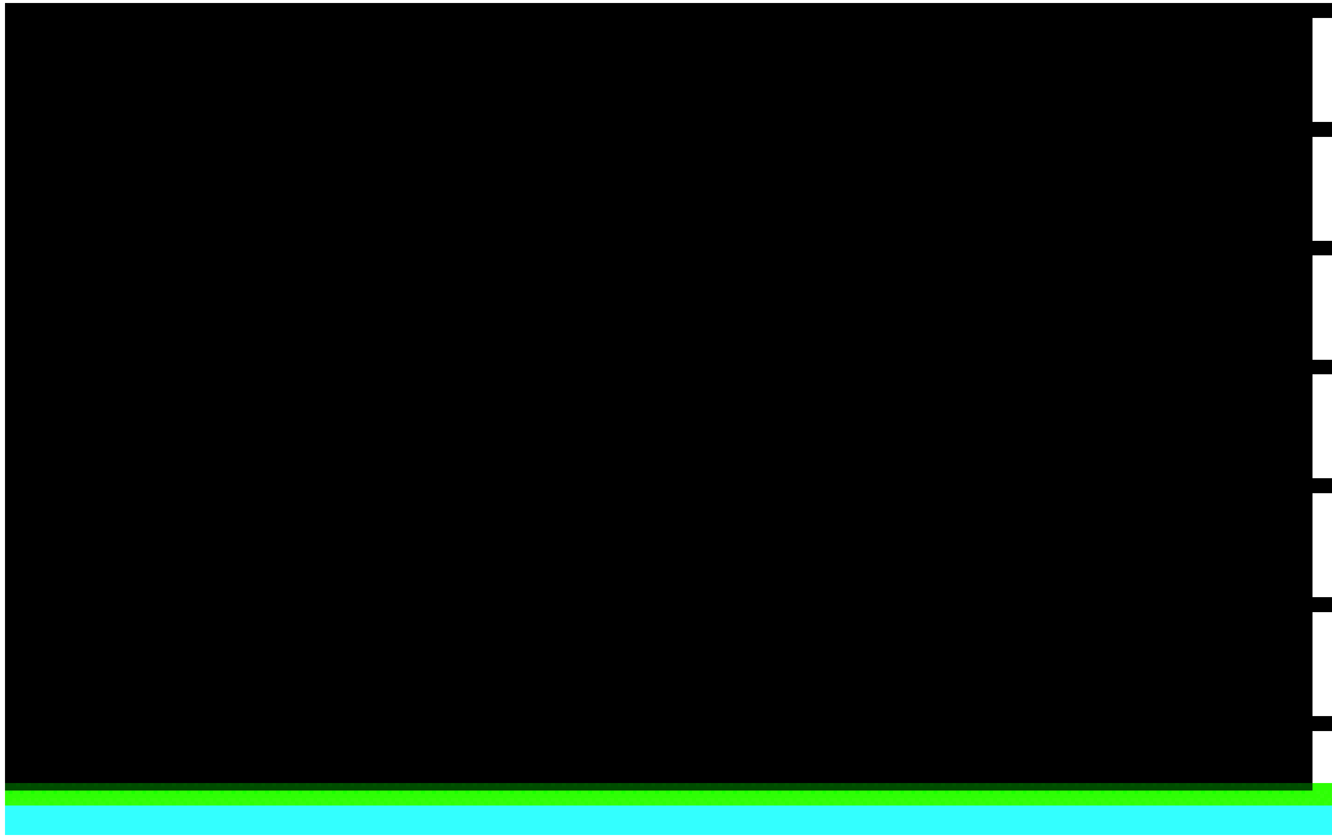


K. Elmaaroufi, D. Shankar, A. Cismaru, A. Sangiovanni-Vincentelli, M. Zaharia, and S. A. Seshia, "Generating Probabilistic Scenario Programs from Natural Language," April 2024.

Generating Scenic Programs from Natural Language

```
# SCENARIO DESCRIPTION
# A Cruise autonomous vehicle ("Cruise AV"), operating in autonomous mode, was traveling eastbound on Clay Street at the intersection
# with Kearny Street when the Cruise AV slowed down. The driver of the Cruise AV disengaged from autonomous mode and, shortly
# thereafter, a bicyclist proceeding straight on northbound Kearny Street made contact with the right rear fender of the Cruise AV, damaging
# its fender. The bicyclist left the scene without exchanging information. There were no injuries and police were not called.
```

```
param map = localPath('../../../assets/maps/CARLA/Town01.xodr')
param carla_map = 'Town01'
model scenic.simulators.carla.model
EGO_MODEL = "vehicle.lincoln.mkz_2017"
EGO_SPEED = 1
EGO_BRAKING_THRESHOLD = .1
BICYCLE_SPEED = 10
BRAKE_ACTION = 1.0
behavior EgoBehavior(speed=10):
    try:
        do FollowLaneBehavior(speed)
    interrupt when withinDistanceToAnyObjs(self, EGO_BRAKING_THRESHOLD):
        #take SetBrakeAction(BRAKE_ACTION)
        do AutopilotBehavior()
behavior BicycleBehavior(speed=5):
    do FollowLaneBehavior(speed)
intersection = Uniform(*network.intersections)
egoLane = Uniform(*intersection.incomingLanes)
bicycleLane = egoLane # Uniform(*intersection.incomingLanes)
egoSpawn = new OrientedPoint on egoLane.centerline
ego = new Car at egoSpawn,
    with blueprint EGO_MODEL,
    with behavior EgoBehavior(EGO_SPEED)
bicycleSpawn = new OrientedPoint on bicycleLane.centerline
bicycle = new Bicycle at bicycleSpawn,
    with behavior BicycleBehavior(BICYCLE_SPEED)
```



All Scenic code and CARLA renderings generated by scenicNL system

Join the Scenic Open-Source Project!

- This Tutorial: https://scenic-lang.org/cvpr24/
- Detailed Documentation: https://docs.scenic-lang.org
- Community Forum: https://forum.scenic-lang.org
- GitHub: https://github.com/BerkeleyLearnVerify/Scenic/
- August 26-28: Scenic Workshop & “Bootcamp” at UC Santa Cruz

Thank you!

https://scenic-lang.org

Thanks to our many Scenic Team Members and Contributors
https://docs.scenic-lang.org/en/latest/credits.html

S. A. Seshia, D. Sadigh, S. S. Sastry. *Towards Verified Artificial Intelligence*. Communications of the ACM, July 2022.