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

Growing Concerns about Safety:

• Numerous papers showing that *Deep Neural Networks can be easily fooled*
• *Fatal accidents* involving potential failure of object detection/classification systems in self-driving cars
Can **Formal Methods** Help?

- Formal methods = Mathematical, Algorithmic techniques for modeling, design, analysis
  - **Specification**: WHAT the system must/must not do
  - **Verification**: WHY it meets the spec. (or not)
  - **Synthesis**: HOW it meets the spec. (correct-by-construction design)

- Industry need for higher assurance → Increasing interest in Formal Methods
- A success story: Digital circuit design / EDA
- *Can we address the Design & Verification challenges of AI/ML-based CPS?*
Challenges for Verified AI

System $S$
Environment $E$
Specification $\varphi$

Does $S \parallel E$ satisfy $\varphi$?

Need to Search Very High-Dimensional Input and State Spaces

Design Correct-by-Construction?

Opportunities for Runtime Verification Community

- System Complexity and Lack of Component Specifications → Execution/Simulation often the only practical option

- Environment Complexity → Need for Runtime Monitoring and Failure-Tolerant Runtime Operation
Research Questions

• What is the Specification?

• How to Scale Up Verification?

• How to Model Environment?

• How to Assure Safe Operation at Run Time?
Outline

• Motivating Example

• Specification

• Verification: Compositionality and Abstraction

• Environment Modeling

• Runtime Assurance

• The VerifAI toolkit & Conclusion
Example: Automatic Emergency Braking System (AEBS) using Deep Learning for Perception

- **Goal:** Brake when an obstacle is near, to maintain a minimum safety distance
- **Models:** Controller, Plant, Env modeled in a software-in-the-loop simulator (Matlab/Simulink, Udacity, Webots, CARLA, ...)
- **Perception:** Object detection/classification system based on deep neural networks
  - Inception-v3, AlexNet, ... trained on ImageNet
  - more recent: squeezeDet, Yolo, ... trained on KITTI

Challenge: Formal Specification

Principle: Start at the System Level (i.e. Specify Semantic Behavior of the Overall System)
Our Approach: Start with a **System-Level Specification**

- “Verify the Deep Neural Network Object Detector”
- “Verify the System containing the Deep Neural Network”

Formally Specify the *End-to-End Behavior* of the System

**Temporal Logic:** \( G(\text{dist(ego vehicle, env object)} > \Delta) \)

Property does not mention inputs/outputs of the neural network
Do We Need to Formally Specify ML Component Requirements?

Sometimes...

1. When Component Specifications are Meaningful
   - ML-based Controllers
   - Semantic Robustness*, Input-Output Relations, Monotonicity, Fairness, etc.

2. For Compositional/Modular Analysis
   - Derive component specifications from system-level specifications

Challenge: Scalability of Verification

Principle: Compositional Simulation-Based Verification (Falsification)

Automotive system with ML-based perception (CPSML)

Learning-Based Perception

Traditional closed-loop model (CPS)

Challenge: $s_e << x$
Our Approach: Three Key Ideas

1. Reduce CPSML falsification problem to combination of CPS falsification and ML analysis by abstraction

2. Simulation-based temporal logic falsification for CPS model
   - Scalable technology already used on production automotive systems (powertrain)

3. Semantic feature space analysis of ML component
   - Derive constraints on input feature space of neural network (pixels) from semantic constraints on environment model

[Dreossi et al., NFM’17, JAR’19; Dreossi et al., CAV’18]
Simulation-based Falsification: Logical Formulas to Objective Functions

• Use Temporal Logics with Quantitative Semantics (STL, MTL, etc.)

• Example:

\[ G_{[0,\tau]}(\text{dist(vehicle, obstacle)} > \delta) \]

\[ \inf_{[0,\tau]} \ [ \text{dist(vehicle, obstacle)} - \delta ] \]

• Verification → Optimization
Need *Compositional* Falsification of CPSML

- **Challenge**: Very High Dimensionality of Input Space!
- Standard solution: Use *Compositional (Modular)* Verification

- However: *no formal spec.* for neural network!
- Compositional Verification *without* Compositional Specification?!!

(see [Seshia, UCB/EECS TR 2017])
Compositional Approach: Combine Temporal Logic CPS Falsifier with ML Analyzer

- **Abstract ML component away from M**
- **Invoke CPS Falsifier (multiple times)**
- **Overapproximate \( \overline{M} \)**
- **Underapproximate \( M \)**

**CPSML model \( M \)**

Property \( \Phi \)

**Region of Uncertainty** \( \text{ROU}(s_e, s_p, u) \)

**Full CPSML Simulation**

**Component (ML) Analysis**

**Project to ML Feature Space**

**FS(x)**

**Component-level errors (misclassifications)**

**Counterexample(s)**

**Refine**

**where ML decision matters**

[Dreossi, Donze, Seshia, NFM 2017]
Identifying Component-Level Input Constraints (ROU) for Automatic Emergency Braking System

Green $\rightarrow$ environments where system-level safety property is satisfied

ML always correct

ML always wrong

Potentially unsafe region depending on ML component (yellow)

[Dreossi et al., NFM’17, JAR’19; Dreossi et al., CAV’18]
Inception-v3 Neural Network (pre-trained on ImageNet using TensorFlow)

Result on AEBS Example

This misclassification not of concern

Misclassifications

Sample image

[Dreossi, Donze, Seshia, NFM 2017; J. Automated Reasoning 2019]
Inception-v3

Neural Network
(pre-trained on ImageNet using TensorFlow)

Result on AEBS Example

But this one is a real hazard!

[Dreossi, Donze, Seshia, NFM 2017; J. Automated Reasoning 2019]
Extending to Image Streams

Superimposition of tests on background

Blind spots

Results on squeezeDet NN and KITTI dataset for autonomous driving

[Dreossi, Ghosh, et al., ICML 2017 workshop]
Challenge: Environment Modeling

Principles: Data-Driven, Probabilistic, Introspective, Modeling

Environment Model ➔ Meaningful Sensor Data

• Need sensor data that makes *physical/semantic sense* and are *interesting and useful* for training, testing, design, and verification

• How to model environment and use it for data generation?

➢ Our approach: **Scene Improvisation**

SCENIC: Scenario Description Language

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

• Example scenario: a badly-parked car

```python
from gta import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * (10, 20) deg
Car left of (spot offset by -0.5 @ 0),
    facing badAngle relative to roadDirection
```

Images created with GTA-V

SCENIC: Scenario Description Language

*Scenic* makes it possible to specify broad scenarios with complex structure, then generate many concrete instances from them automatically:

**Platoons**

**Bumper-to-Bumper Traffic**
Use Case: Retraining with Hard Cases

*Improves accuracy on hard cases without compromising accuracy on original training set*

e.g. for car detection, one car partially occluding another:
Use Case: Debugging a Known Failure
Use Case: Debugging a Known Failure

*Scenic* makes it easy to vary a scenario along different dimensions:

- Add noise
- Change car model
- Change global position
VERIFAI: A Toolkit for the Design and Analysis of AI-Based Systems  

https://github.com/BerkeleyLearnVerify/VerifAI
Case Study for Temporal Logic Falsification with VerifAI: Navigation around an accident scenario

- Ego Car (AV)
- Broken Car
- Cones
- Lane Keeping
- Lane Change
- \( d < 15 \)
- lane change complete
Modeling Case Study in the SCENIC Language

# Pick location for blockage randomly along curb
blockageSite = OrientedPoint on curb

# Place traffic cones
spot1 = OrientedPoint left of blockageSite by (0.3, 1)
cone1 = TrafficCone at spot1,
       facing (0, 360) deg

...

# Place disabled car ahead of cones
SmallCar ahead of spot2 by (-1, 0.5) @ (4, 10),
    facing (0, 360) deg
Using Scenic to Generate Initial Scenes
Using Scenic to Generate Initial Scenes
Using Scenic to Generate Initial Scenes
Falsification
Analyzing the failure

Fix the controller:
Update model assumptions and re-design controller

Retrain the perception module:
Collect the counter-example images and retrain the network [IJCAI’18]

\( d = 30 \)

Incorrectly detected 14.5

\( v < 15 \)

Violates controller assumptions
Different Case Study: Automated Taxiing

- NN-based research prototype from Boeing
- Specification: track centerline within 1.5 m
Falsifying Run Found with VerifAl/Scenic
Challenge: Correct-by-Construction Design

Principle: Use Programming Framework for Runtime Assurance

Robotics Software Stack: Untrusted Components, Unknown Environment

(1) Obstacle Avoidance ($\phi_{obs}$): Stay a minimum distance from obstacles.

(2) Battery Safety ($\phi_{bat}$): Land safely when the battery is low.

How to provide safety guarantees?

Simplex Architecture for Run-Time Assurance of Periodic Real-Time Systems

[Lui Sha, RTSS’98]

Our key advances:
1. Automatically generating switching logic
2. Switches back from SC to AC (not just AC → SC)
3. Deployed on robotics platforms

Already used in fault-tolerant avionics systems
Runtime Assurance (RTA) Module

An RTA Module is a tuple \((M_{ac}, M_{sc}, M_{dm}, \Delta, \phi_{safe}, \phi_{safer})\)

- \(M_{ac}\) is the Advanced Controller
- \(M_{sc}\) is the Safe (verified) Controller
- \(M_{dm}\) is the Decision Module (determines switching between \(M_{ac}\) and \(M_{sc}\))
- \(\Delta\) is the sampling rate of the DM
- \(\phi_{safe}\) is the desired safety specification
- \(\phi_{safer}\) is a stronger safety specification \((\phi_{safer} \subseteq \phi_{safe})\)
  - Starting from \(\phi_{safer}\) we stay in \(\phi_{safe}\) for at least \(2\Delta\) time even with arbitrary controller

Key question: how to determine switching logic?
Our Approach: Switching Logic of DM based on Controlled Reachable Sets for Reach-Avoid Objectives

Theorem: The following is an inductive invariant:

\[
\text{Mode} = \text{SC} \land s \in \phi_{\text{safe}} \\
\lor \\
\text{Mode} = \text{AC} \land \text{Reach}(s,*,\Delta) \in \phi_{\text{safe}}
\]


Uses reach set computation toolbox from Claire Tomlin’s group.
Results: Drone Navigating in an Unknown Workspace (reach-avoid problems)
How to Generate Switching Conditions when Variables/Dynamics of Environment are Unknown?

Impossible to model all possible scenarios
Unknown variables, dynamics

Approach: *Introspect on System to Model the Environment*

**Identify:** (i) **Interface** between System & Environment – *Sensor Model*,
(ii) *(Weakest) Assumptions* needed to Guarantee Safety/Correctness

Algorithmic techniques to *generate weakest interface assumptions* at design time and *monitor them at run time* for potential violation/mitigation

[Seshia RV’19; Li, Sadigh, Sastry, Seshia; TACAS’14]
Links and Ongoing Work

• Open-Source Releases: the Verified AI toolkit [CAV 2019]
  – VerifAI: https://github.com/BerkeleyLearnVerify/VerifAI
  – Scenic: https://github.com/BerkeleyLearnVerify/Scenic
  – Operate with any simulator

• Other results/ongoing projects:
  – Bridging Simulation and Real World
    • Domain adaptation to produce “real” data from simulated data
    • Quantifying distance between simulated and real behaviors [HSCC 2019]
  – More Complex Sensors: Video + LiDAR + RADAR + ...
  – Counterexample-Guided Retraining/Data Set Design [IJCAI 2018]
**Conclusion: Towards Verified AI/ML based CPS**

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Core Principles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Environment (incl. Human) Modeling</td>
<td>Data-Driven, Introspective, Probabilistic Modeling</td>
</tr>
<tr>
<td>2. Specification</td>
<td>Start with System-Level Specification, then Component Spec (robustness, ...)</td>
</tr>
<tr>
<td>3. Learning Systems Complexity</td>
<td>Abstraction, Semantic Representation, and Explanations</td>
</tr>
<tr>
<td>5. Design for Correctness</td>
<td>Oracle-Guided Inductive Synthesis; Run-Time Assurance</td>
</tr>
</tbody>
</table>

**Exciting Times Ahead!!! Thank you!**