Assessing Robustness and Resilience of AI: 
The ALC Project

Gabor Karsai, Vanderbilt University
with contributions by Taylor Johnson, Xenofon Koutsoukos
Supported by DARPA under
Assurance of Learning-Enabled Cyber-Physical Systems
“The proposed research effort will address the ... technical areas with overall goal of delivering an integrated design tool suite and reusable operation support components for constructing autonomous CPS including Learning Enabled Components (LECs). Our vision is to ... create a new design flow that extends from design-time to operation time, re-interprets the traditional assurance argumentation to become a dynamic, operational concept. Our ultimate goal is to establish a fusion of model- and component-based methods with data-driven methods.”

**Model-driven design flow**

[Diagram of Model-driven design flow]

**Model-driven design flow with LEC-s**

[Diagram of Model-driven design flow with LEC-s]
Project activities

**Thrusts:**
- **Verification**: formal and/or coverage-driven verification of safety and liveness properties of components, subsystems, and systems, at design-time and at run-time, to provide evidence for assurance arguments
- **Assurance**: construction and continuous evaluation of logical arguments that demonstrate the *truth* or *strength* of a safety claim based on available evidence
- **Toolchain**: design-time and run-time software tools to implement and support the above, for real systems

**Learning (component adaptation)**
- Design-time: in design tools, while the system is not operational
- Run-time: in the running system, on-the-fly
- Mixed – learning from operational, ‘overnight’
Verification Technology

Example-1: Robustness Assessment
Example-2: Run-time Verification

Prof. Taylor Johnson and team
**LEC Verification: Reachability Analysis of Feedforward/Convolutional Neural Networks**

- Given a NN $F$ & an input set $\mathcal{X}$, the **output reachable set** of $F$ is $\mathcal{Y} = \{y \mid y = F(x), x \in \mathcal{X}\}$

- Computationally: Given a NN $F$, a convex initial set of inputs $I$ represented as a polytope $\text{poly}(\mathcal{X})$, compute the output set $\mathcal{Y} = F(I)$ of the network

  \[ I = \text{poly}(\mathcal{X}) \]
  \[ I = \{ x \mid Ax \leq B, x \in \mathbb{R}^n \} \]

- Layer-by-Layer Propagation of Polytopes

\[ Y = F(I) = ? \]
Is VGG16/19 robust to FGSM attacks for $a \leq 2 \times 10^{-8}$?

Disturbed images = Original image + $a \times$ Noise

VGG Classifiers: ~93% accuracy in top-5 classification on ImageNet
VGG16: 16 layers, 138M parameters
VGG19: 19 layers, 144M parameters
Classify images into 1000 classes, e.g., car, horse, bell pepper, ...

Layers of interest
- Convolutional
- Average pooling
- Max pooling
- Fully connected
- ReLU

CNN Classification Robustness Analysis: ImageStars

- **ImageStar**

  \[ \Theta = \{ x | x = c + \sum_{i=1}^{m} \alpha_i v_i, P(\alpha) \} \]

  \[ c \in \mathbb{R}^{h \times w \times nc} \] is the center image

- \[ V = \{ v_1, v_2, \ldots, v_m \}, v_i \in \mathbb{R}^{h \times w \times nc} \] is a set of basis images

- \[ P(\alpha) \triangleq C\alpha \leq d, \] is a predicate

- \[ \alpha = [\alpha_1, \alpha_2, \ldots, \alpha_m]^T, \] is predicate variable

- Extension of Star Sets [Tran et al, FM’19]

- Represent infinite sets of multi-channels images
Is VGG16/19 robust to FGSM attacks for $a \leq 2 \times 10^{-8}$?

Reachable set computation time: **518** seconds

Verifying Robustness Time: **56** seconds

Number of ImageStars in the output reachable set: **8**

Total Verification Time: **574** seconds ($\approx 10$ minutes)

Number of cores: **1**

Robust? **Yes**
Closed-Loop CPS with LECs: Verification Flow and Tools

- Plant models: **hybrid automata**, or networks thereof, represented in HyST/SpaceEx/CIF formats
  - Hybrid automaton: **finite state machine** + set of real-valued variables that evolve continuously over intervals of real time according to **ordinary differential equations (ODEs)**
  - **Hybrid** behaviors: discrete transitions and continuous trajectories over real time
  - Plant dynamics: linear, nonlinear, hybrid, continuous-time, discrete-time, …

- LEC and cyber models: feedforward **neural networks**, represented in **ONNX** format (compatible with Keras, Tensorflow, Matlab, etc.)

- Specifications: primarily **safety properties** for now, some **reachability properties**

- Verification: composed LEC and plant analysis: autonomous closed-loop CPS
  - **Bounded model checking**: $k$ control periods, alternating reachability analysis of controller and plant

---

https://github.com/verivital/hyst  
https://github.com/verivital/nnv  
https://cps-vo.org/group/hyst  
https://github.com/verivital/nnmt
Runtime (Online) Verification of Autonomous Systems with Real-Time Reachability

- For controller LECs **online monitoring at runtime** is essential
- How can we provide formal and provable guarantees of system-level behaviors, such as safety, **online at runtime**?
  - Key idea: abstract LEC behaviors (see other approaches on out of distribution detection, etc.) and simply **observe the influence of their behavior on plant/system-level at runtime**
  - Necessary technology: **online reachability analysis** of plant models, ideally with worst-case execution time (WCET) guarantees for implementation in embedded hardware
  - Builds on **real-time reachability** of linear/nonlinear ordinary differential equations (ODEs) and hybrid automata with WCET guarantees, implemented as an **anytime** algorithm [FORTE’19, TECS’16, RTSS’14]
  - Based on **mixed face lifting reachability** [Dang and Maler, HSCC’98 & HSCC’19 Test of Time Award Winner], using hyperrectangles (intervals) as state-space representation

[Johnson et al, “Real-Time Reachability for Verified Simplex Design”, TECS’16]
[Bak et al, “Real-Time Reachability for Verified Simplex Design”, RTSS’14]

http://www.verivital.com/rtreach/
Complex controller: can do **anything**, be output from LECs, etc., abstracted to just produce control inputs \((u)\) for the plant

Assumptions: analytical (linear or nonlinear ordinary differential equation [ODE]) plant model available, and controller input remains fixed over finite time horizon

**Supervisory control** via **Simplex architecture**

Check these control inputs on closed-loop for a finite time horizon using **reachability analysis with real-time (WCET) guarantees**, if there’s a problem, fall back to safety strategy

Real-time reachability algorithm implementation is cross-platform C (x86, ARM, AVR, etc.) with no dynamic memory allocation, recursion, or library dependencies:

[https://github.com/verivital/rtreach](https://github.com/verivital/rtreach)

Safety Verification with Reachability

- **Safe** if intersection of overapproximation of reachable states with unsafe states is empty (*soundness*)

---

- If safe, then red trajectory reaching an unsafe state **cannot exist**
- All trajectories contained in reachable states
End-to-end (E2E) controller: takes images and produces steering control inputs.

Classification-based control: determining steering angle (straight, weak left, weak right, etc.) with fixed speed.

Reachable sets visualized below right: if intersection with obstacles occurs, use fallback safety controller.

Plant model: nonlinear ODEs (bicycle, Ackermann steering).
Assurance Monitoring Technology

Example-1: Detecting distribution shifts
Example-2: Detecting adversarial attacks

Prof. Xenofon Koutsoukos and team
Assurance Monitoring: Can we trust the output of the LEC?

Assurance Monitoring Based on Inductive Conformal Prediction

- Characterize how close the LEC behavior is to a model that represents the expected safe behavior obtained during the training phase.
- Compute measures of confidence associated with predictions from LECs.
- Nonconformity measure is used to evaluate the degree to which a new example disagrees from a set of examples.
- Confidence is computed based on how different is a test example compared to a set of calibration examples.
Inductive Conformal Prediction (ICP)

1. Split the training set into
   - The proper **training** set
   - The **calibration** set
2. Use the proper training set to train the neural network
3. For each example in the calibration set:
   - Supply the input to the trained neural network to obtain the prediction
   - Calculate the nonconformity scores
   - Sort the calibration examples using descending order of the **nonconformity scores** in the set $A$
4. For each new example, compute the fraction of examples that are equally or more nonconforming ($p$-values)
5. Compute a **predictor** with a given confidence based on the $p$-values

**Nonconformity measure:**
A function that measures the disagreement between the actual label and the prediction using the neural network

---

Anomaly Detection

- The nonconformity measure can be used to evaluate the degree to which a new example disagrees from a set of examples.
- For test examples, we compute the fraction of nonconformity scores for the calibration data that are larger than the nonconformity score of the test input (empirical $p$-value).
- If the empirical $p$-value < $\varepsilon$ the example is classified as a conformal anomaly.
- There are at least three explanations for a conformal anomaly:
  - A rare or previously unseen example from the same probability distribution as the training set.
  - A true novelty not generated from the same probability distribution as the training set.
  - The training examples are not IID.

In CPS, examples arrive one by one and after observing each new example, we would like to quantify the degree to which the examples disagree with the training data.

If the examples are IID, the inductive conformal anomaly detection algorithm produces $p$-values that are independent and uniformly distributed in $[0, 1]$. Out-of-distribution detection can be performed by testing the hypothesis that $p$-values that are independent and uniformly distributed in $[0, 1]$ – or not.
Exchangeability Martingales

- Given the sequence of $p$-values, a martingale is calculated as a function of the $p$-values
  - Power martingale
    \[ M_n^\varepsilon = \prod_{i=1}^{n} \varepsilon p_i^{\varepsilon-1} \]
  - Simple mixture martingale
    \[ M_n = \int_0^1 M_n^\varepsilon d\varepsilon \]

- The value of the martingale reflects the strength of evidence against the exchangeability assumption, i.e. that the examples are generated from the same probability distribution independently
- Such a martingale will grow only if there are many small $p$-values in the sequence
- If the generated $p$-values concentrate in any other part of the unit interval, we cannot expect the martingale to grow

---

Nonconformity Measure (NCM)

- Computing the NCM using $k$-nearest neighbors requires storing the training data which may be infeasible for autonomous CPS $\Rightarrow$ Reduce the memory/time requirements
- Train an appropriate neural network architecture which can be used to compute efficiently the NCM

1. Autoencoders
   - Use the reconstruction error as the NCM
   - Based on current experiments, the method is not robust

2. Variational autoencoders
   - Use the generative model to sample multiple IID examples for the input of the current time step
   - Use the reconstruction error (probability) as the NCM

3. Deep One-Class Classification
   - Deep Support Vector Description (SVDD)
Assurance Monitoring
Distribution Shift Detection Example

Precipitation parameter [0,100] is the full range. Precipitation [0,20] is in distribution.
Assurance Monitoring: Distribution Shift Detection in adversarial scenarios

Shift detected
ALC Toolchain
Tool architecture - coverage
The model driven toolchain supports training, verification and design-time assurance of learning enabled components. Toolchain helps with developing safety assurance cases for the system using collected evidence. Complete provenance tracking of Experimental runs and data collection is supported.
ALC Toolchain Concepts

- **Modeling**
  - System Architecture / SysML

- **LEC Construction**
  - Data collection
  - Training
  - Evaluation

- **Testing -- Verification/Validation/Assurance**
ALC Workflow

- MDE with support for LEC development + Assurance

Diagram:
1. System Modeling
   - Message Library
   - Component Library
   - Architecture Models
   - Assembly Models
   - Iterate Design

2. LEC Construction
   - Data Generation
     - Design Experiment
     - Configure Simulation
   - Supervised Learning
     - Configure Training
     - Test Set Evaluation
     - Adjust Parameters
   - Reinforcement Learning
     - Design Experiment
     - Design Reward Function
     - Configure Simulation
     - Configure Learning Algorithm

3. Verification & Validation
   - Verification Tools
   - Assurance Cases

Workflow/Orchestration
- Job
- Job
- Job
Models in the ALC Toolchain

Model Systems using:
- Component blocks (hardware/software)
- Messages/datatypes for software
- System architecture

Construct Experiments consisting of:
- Data collection
- LEC training
- Assurance Monitor construction

Verification, Validation, and Assurance via:
- Formal Verification (Design-time)
- LEC testing
- Assurance argument construction

Workflow automation:
- Create/Execute sequences of operations

Datasets to:
- Manage all data produced by Experiments and Workflows
- Track data provenance
- Perform automated analysis/evaluation of data
System Modeling

Data Models, Messages

Components:
Hardware, Software/LEC

Systems: Components/ Subsystems; Parameters,...

World models: Scenarios, Environments, Parameters
System architecture
SysML block diagrams

Subset of SysML Blocks, IBD to model all blocks, implementation alternatives for flexibility

Software Components And Assemblies
Block Library
Assurance Monitors
Dynamic Assurance Process

Controllers (LEC)

Physical Components And Assemblies

LEC & Conventional Alternatives
LEC Construction

1. Data Collection

2. Training LEC + A/M

3. Testing

Select Configuration

Parameter | Values
--- | ---
pipe_roll | 0, 3.14159
pipe_name | 15_bend_pipe, 30_bend_pipe
LEC Construction: 1. Data Collection

- **Assembly model** selects a specific implementation variant of a system architecture
- **Mission**, **Environment**, and **Execution parameters** set up the experiment scenario
- **Campaigns** across parameters and configurations related to system and environments
- Tool generates configuration for running the simulation, capture results + metadata for all trials

Dockerized ALC-toolchain services for portability

Remote job: launching of dockers, management of results.
LEC Construction:

2. Training

- Neural Net model and parameters specified in “LEC Model”

- “Training Data” links to data generated from previous experiments

- Training job is dispatched to worker machines (typically with GPUs)

- Results and metadata are saved from the training sessions
2. Training: Assurance Monitor

Deploy trained LEC, build assurance monitor.
LEC Construction:
3. Evaluation: Testing/Verification

- Trained Neural Net can be tested in the simulator with another experiment model.
- Performance metrics are recorded for LEC evaluation, e.g.:
  - Distance from ideal path
  - Pipe within camera field of view

Analysis in Jupyter Notebook
Also, “single step” the process for debugging

Training Model Data Managed on GitLab

Results in file store + git, cross-linked for data provenance
Workflow models are for the specification and execution of job graphs
- Each workflow job specifies execution of one or more activity models
- Data dependencies between jobs are handled automatically

Workflow supports
- Loops – For (parallel), while/ do-while (sequential)
- Transforms - Filter / Join (subset or aggregation of results)
- Branch – execution path based on user-specified condition

Example workflow to train and optimize a LEC
Tool automation
Support for Data Provenance

- All artifacts – generated during data collection, training, evaluation
- Recorded for each execution:
  - Parameter settings
  - LEC(s) Models (Deployed/Initial)
  - Data used in training, validation and evaluation
- Allows re-execution of any step/workflow
- Track the evolution of data/LECs/Assurance
- Maintain traceability links at each stage to:
  - Data used in training LECs
  - Initial trained model used in training LECs
  - LECs used in generating data sets/Assurance
System Assurance Case: GSN

- Top-level goals correspond to high level safety claims
- Leaf goals correspond to claims which can be directly supported by evidence/solutions
- Evaluation metrics from LEC experiments can be used as evidence for leaf goals
- User Defined Combination Logic (E.g. M-of-N, etc.)

Example GSN Model for UUV

Cross-Referencing Components, Datasets, for Context/Evidence
Summary
ALC Project

- **Verification:**
  - Design-time: reachability + robustness of AI/LEC components
  - Run-time: safety given in the given situation

- **Assurance monitoring:**
  - Detect distribution shift
  - Assess confidence/credibility in the output

- **Toolchain:**
  - Automation for evaluating LECs
  - Modeling for assurance arguments (with evidence)

Publications:
[https://alc.isis.vanderbilt.edu/redmine/projects/alc-project-public/documents](https://alc.isis.vanderbilt.edu/redmine/projects/alc-project-public/documents)

Portal for AA program tools:
[https://assured-autonomy.org/](https://assured-autonomy.org/)
Credits

- Ted Bapty
- Dimitrios Boursinos
- Feiyang Cai
- Abhishek Dubey
- Charles Hartsell
- Taylor Johnson
- Xenofon Koutsoukos
- Jiani Li
- Nagabhushan Mahadevan
- Diego Manzanas Lopez
- Mary Metelko
- Patrick Musau
- Harmon Nine
- Shreyas Ramakrishnan
- Joel Rosenfeld
- Janos Sztipanovits
- Hoang-Dung Tran
- Ayana Wild
- Weiming Xiang
- Xiaodong Yang