

# AI Verification Tools

Adrian Agogino

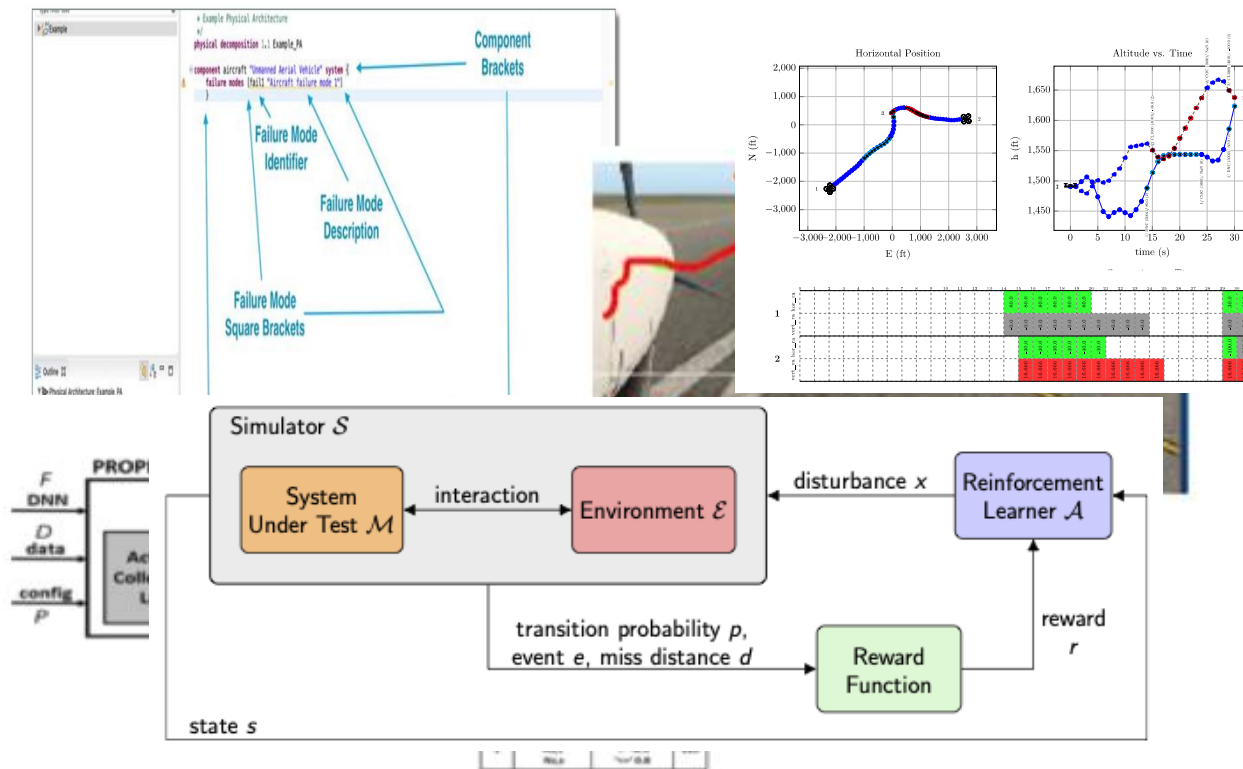
Robust Software Engineering

Code TI

Ames Research Center

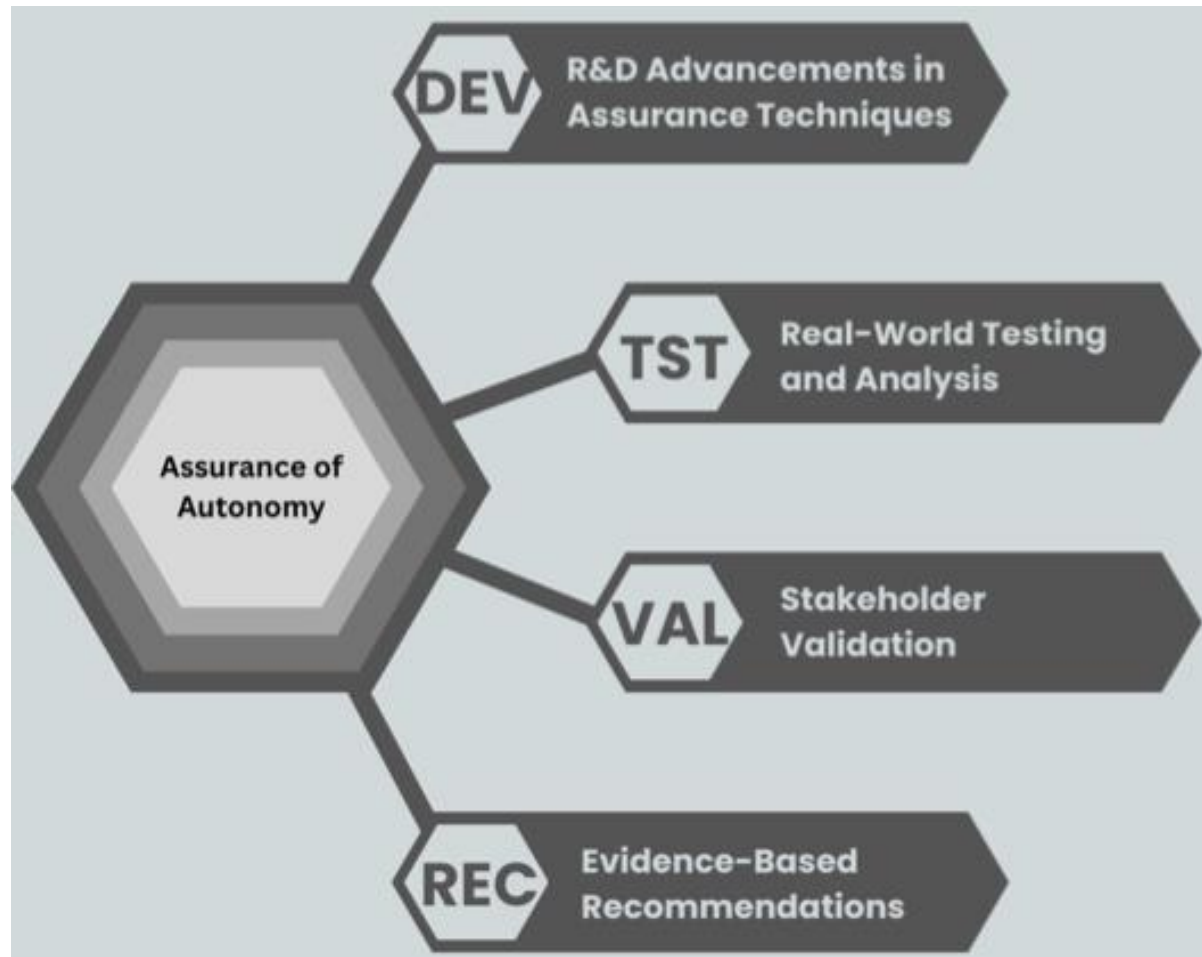
# Robust Software Engineering Group

- Leads: Guillaume Brat, Sandy Lozito



- Software testing
- Software verification and validation
- Machine learning
- Autonomy
- Multiagent systems

# Complex, Autonomous Systems Assurance



## Stakeholders



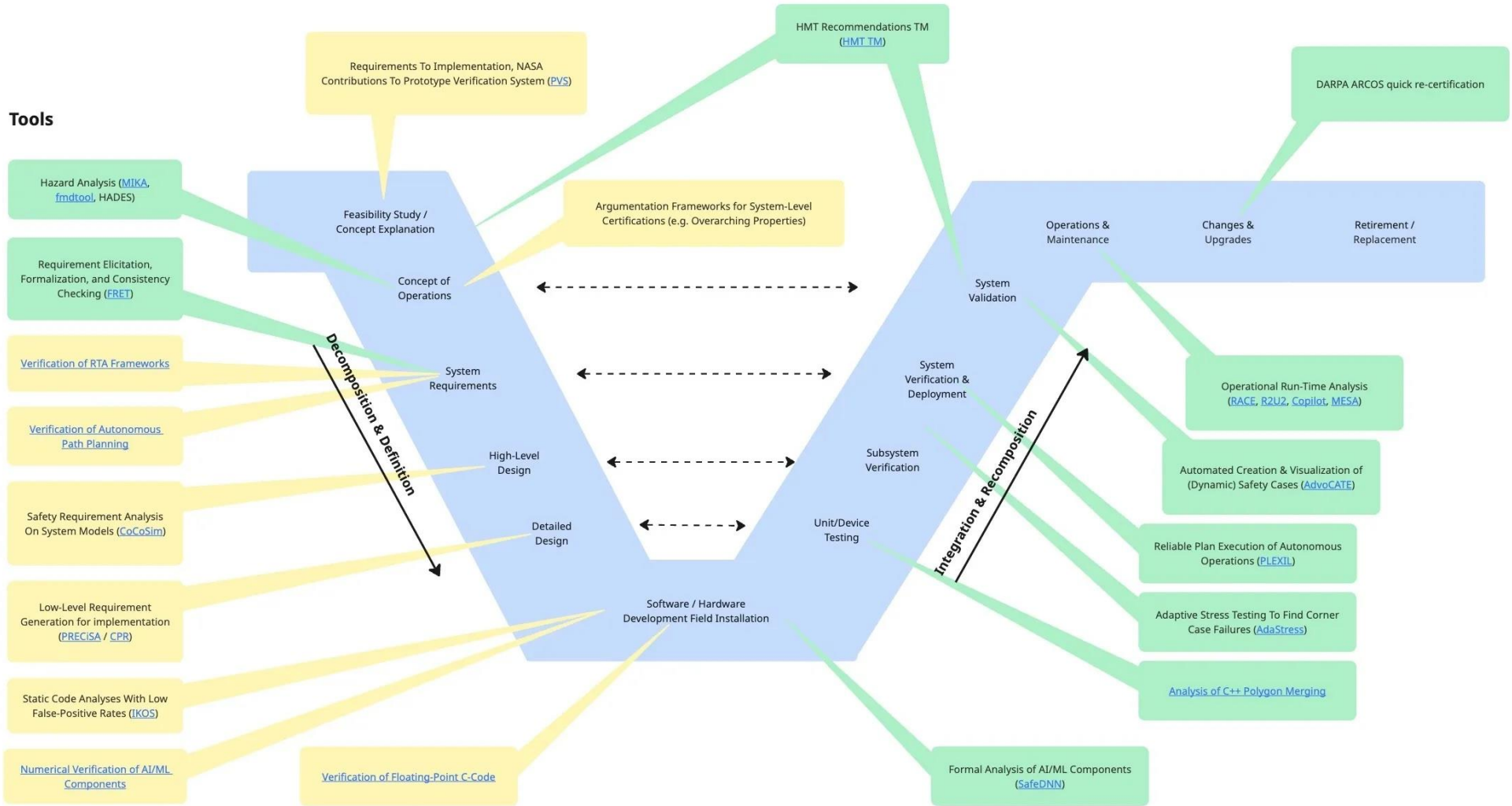
# AI and Complexity

- AI algorithms tend to be large and complex
  - Hundreds of thousands of neuron for small image processor
  - Millions of neurons for large image processor
  - Billions to trillion neurons for large language models
- What we are asking from AI is complex
  - Perception
  - Understanding world model
  - Language understanding
  - End-to-end control

# Need Tools for AI Lifecycle

- Architecture of complex systems
- Requirements for complex systems
- Explainability of complex systems
- Testing and verification of complex systems

# AI Tools for Design Lifecycle



# FRET: Creating Formal Requirements

- FRET allows natural language to be made into formal requirements

The screenshot displays the 'Create Requirement' interface in the FRET tool. It features a 'Create Requirement' header with a dropdown menu. Below this is a table with columns for 'Requirement ID', 'Parent Requirement ID', and 'Project Default'. A 'Rationale and Comments' section is also present. The 'Requirement Description' section includes a guide on how to use the sentence structure, with a visual aid showing bubbles for 'SCOPE', 'CONDITIONS', 'COMPONENT\*', 'SHALL\*', 'TIMING', and 'RESPONSES\*'. The main text area shows a requirement template: 'component shall always satisfy if (input\_state & condition) then output\_state'. A dropdown menu is open over the 'condition' field, showing options: 'predicate' (Predicate is described by name), '! predicate' (Predicate should not hold), and 'predicate1 & predicate2' (Conjunction). The 'SEMANTICS' label is visible to the right. At the bottom, there are 'CANCEL' and 'CREATE' buttons.

Assistant    **TEMPLATES**    GLOSSARY

Template  
Change State

Choose a predefined template

This template describes how the state of a finite-state-machine component changes. It describes the input state and some conditions based on which the change must occur. The corresponding output state must reflect the required change. The input and output states have a pre-post- relationship

Examples:

```
FSM_Autopilot shall always satisfy if (  
state = ap_standby_state & ! standby & ! apfail ) then  
STATE = ap_transition_state
```

predicate  
Predicate is described by name

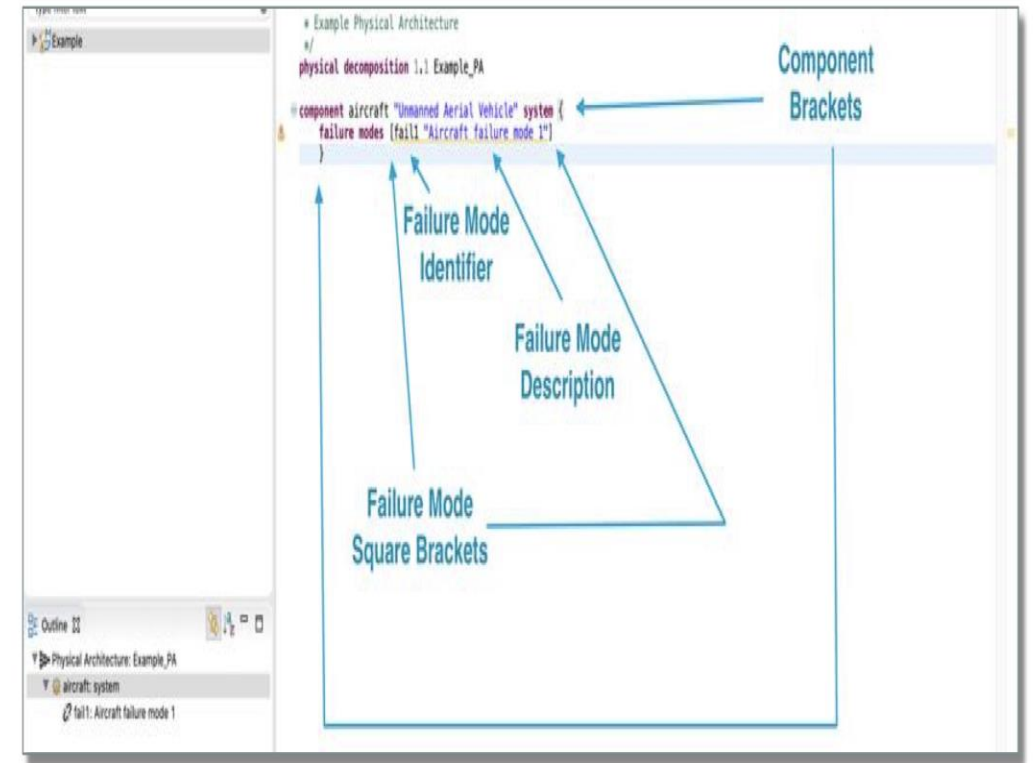
! predicate  
Predicate should not hold

predicate1 & predicate2  
Conjunction

CANCEL    CREATE

# AdvoCATE

- Safety risk management
- hazard analysis
- linking of hazards to requirement logs
- risk modeling using bow tie diagrams
- structured arguments
- evidence logs



# Fault Model Design Tools (FMD)

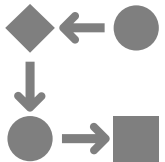
- Simulation-based (rather than document-based) hazard analysis process



- Flexible Modelling Paradigm



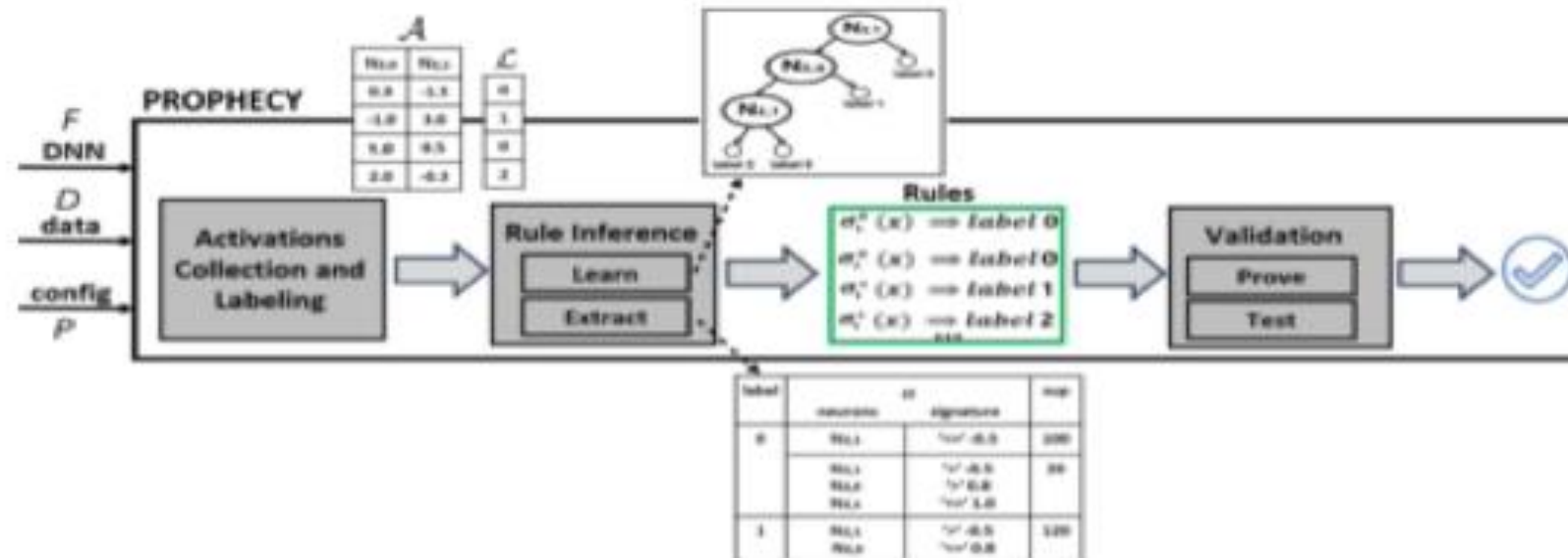
- Powerful Simulation Techniques



- Efficient Analysis Process

# SafeDNN: Understanding and Verifying Neural Networks

- Extract rules from neural network
- Verify properties from these rules
  - Formal proofs, Runtime monitoring, Specification-driven Testing



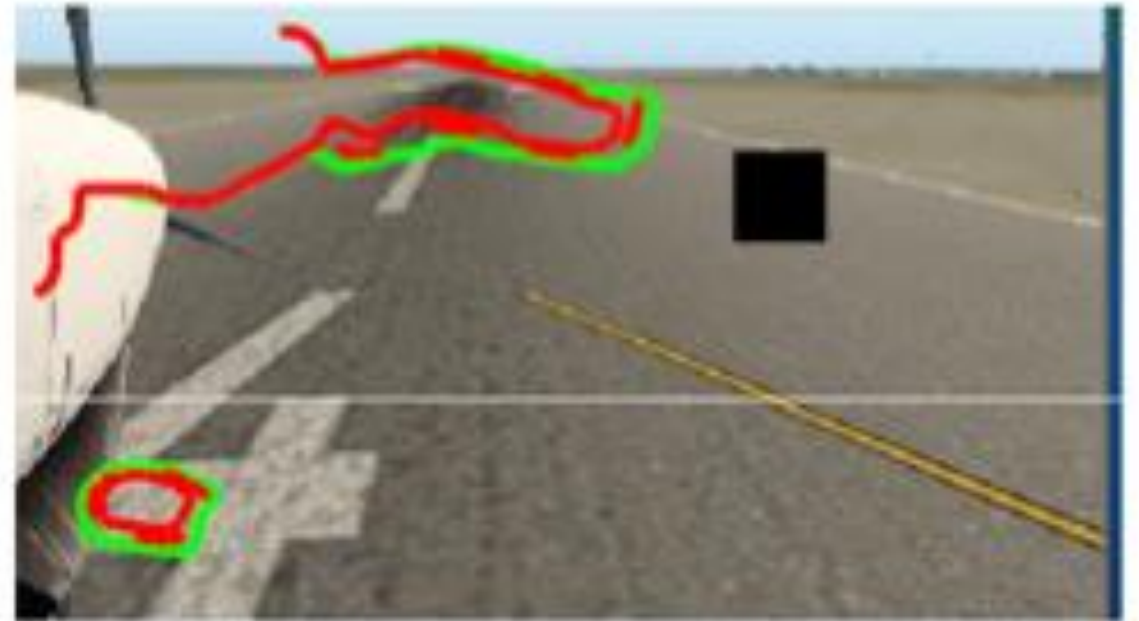
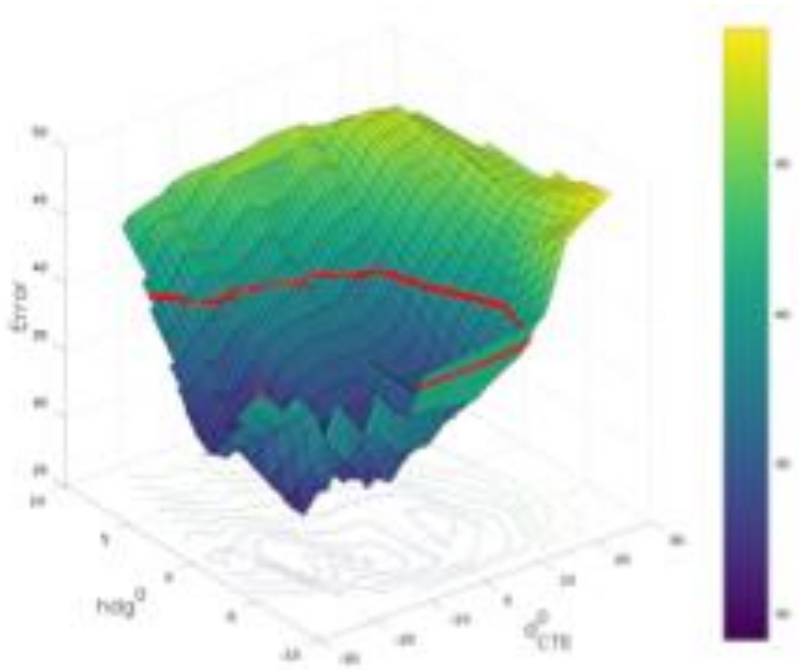
# Closed Loop AI Analysis

- Can define requirements for outer loop system even if perception component is complex
- Enables
  - Probabilistic analysis,
  - Worst-case analysis,
  - Scenario-based Compositional Verification



# SysAI

- Find safety envelopes
- Find regions of failure

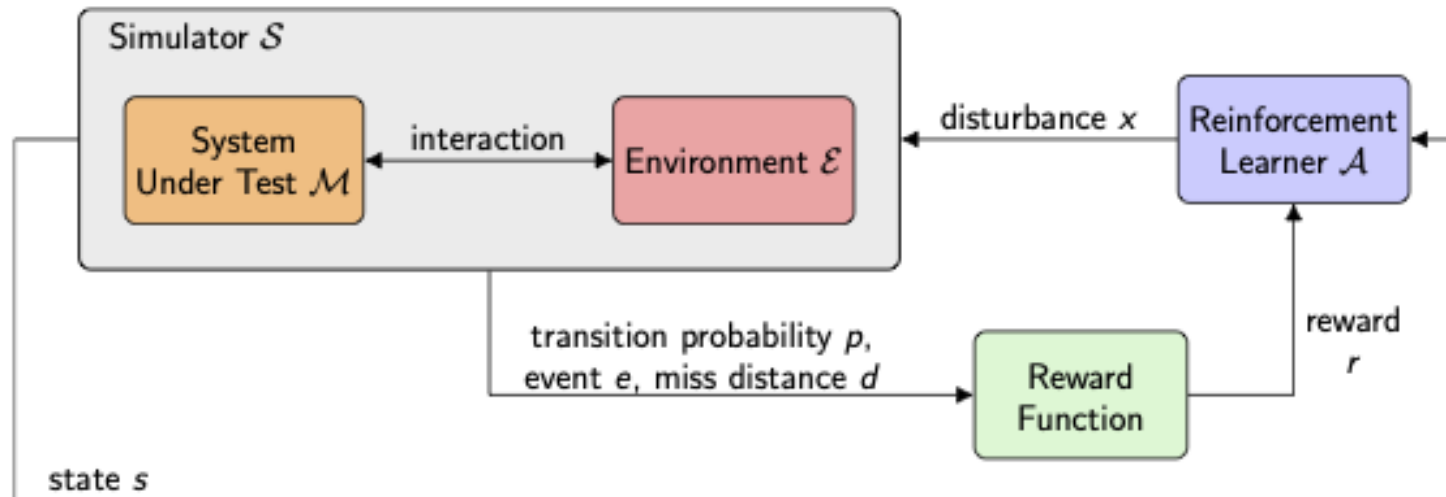


# AdaStress

- Many systems too complex to verify formally
- Many systems have too large of input space to perform Monte Carlo (random) testing
- AdaStress learns to test regions that are most likely to fail to find failures even when input space is large
  - Finds failures with few samples
  - Can be used on blackbox systems such as large neural networks

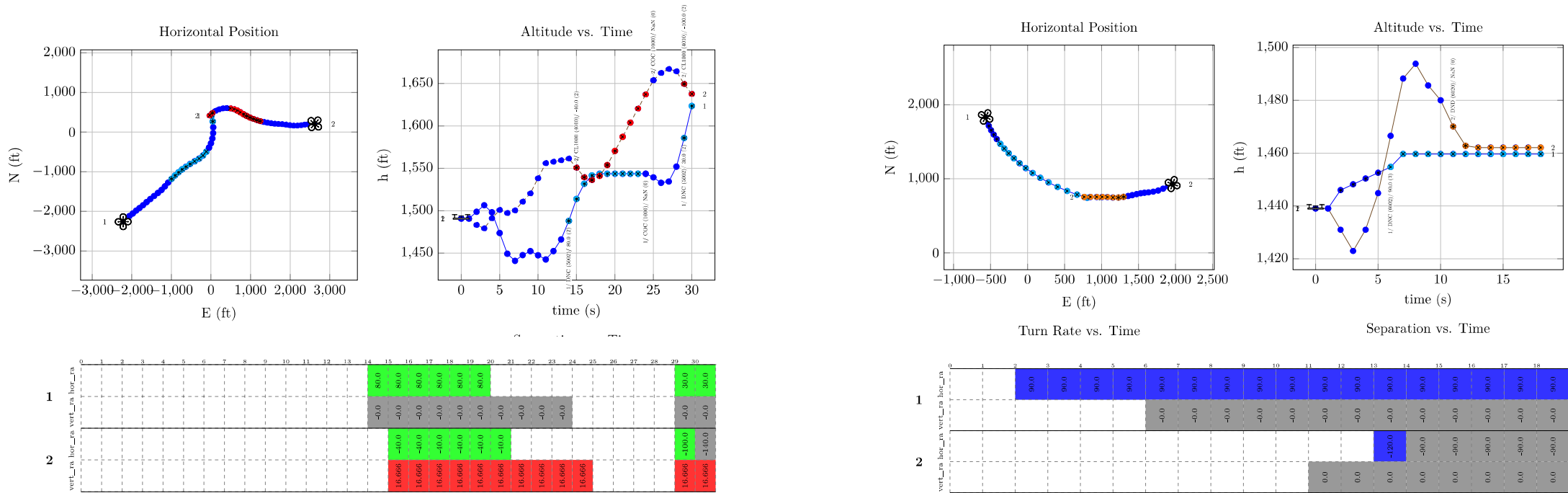
# AdaStress

- Perform adaptive stress testing with:
  1. Reinforcement Learning
  2. Likelihoods
  3. Blackbox Manipulation (optional)



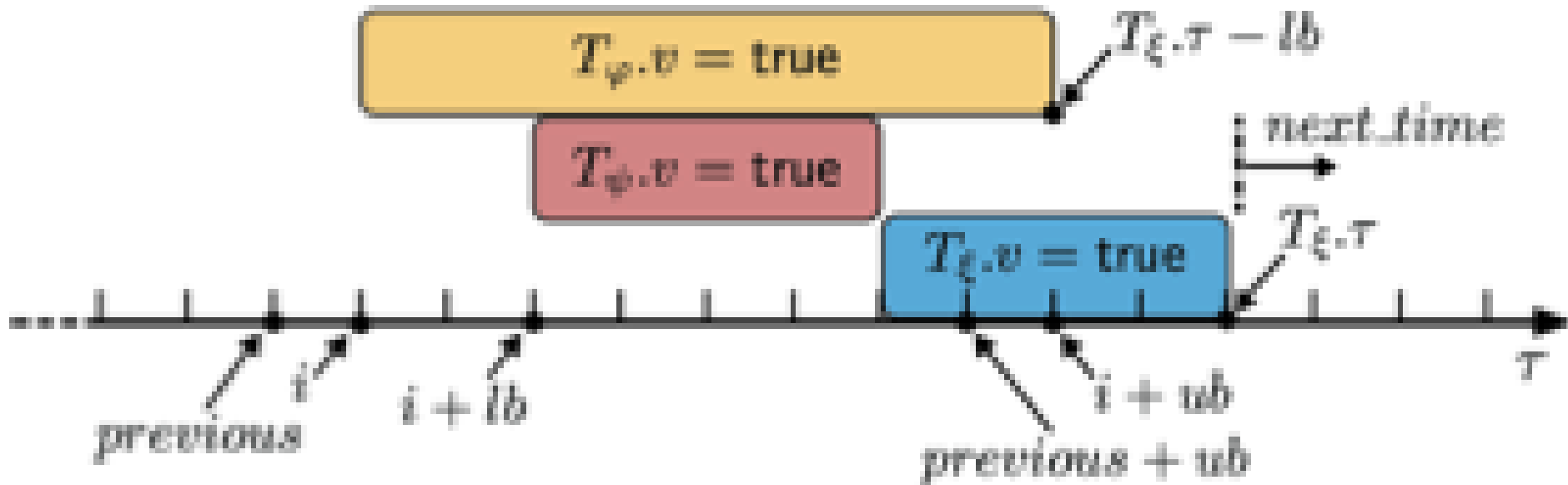
# AdaStress Case Study Case Study: xSu

- AdaStress can find previous unknown failure conditions



# R2U2: Realizable, Responsive, Unobtrusive Unit

- Stream based runtime monitoring
  - Use temporal logic



# Example of Tool Workflow in “V” Lifecycle

