PopperFailureTest: Reinforcement Learning–Driven Failure Detection for Modular AI Systems(Popper Framework)

A Multi-Agent UCB-Based Validation Framework extending Humanitarians.AI Popper Architecture

Overview

PopperTest is a **multi-agent reinforcement learning system** designed to intelligently test modular AI or software systems, discover hidden weaknesses, and adapt testing strategies over time.
This project extends the **Popper validation framework** by implementing a UCB-based RL agent that learns where failures are most likely to occur, enabling highly efficient, adaptive, and scalable automated testing.

The system simulates an environment with **N components**, each with hidden failure probabilities. A set of specialized agents collaborate to probe the system, detect failures, track coverage, and update the RL policy.

The result:

➡️ **25–30% improvement in failure detection** vs random testing

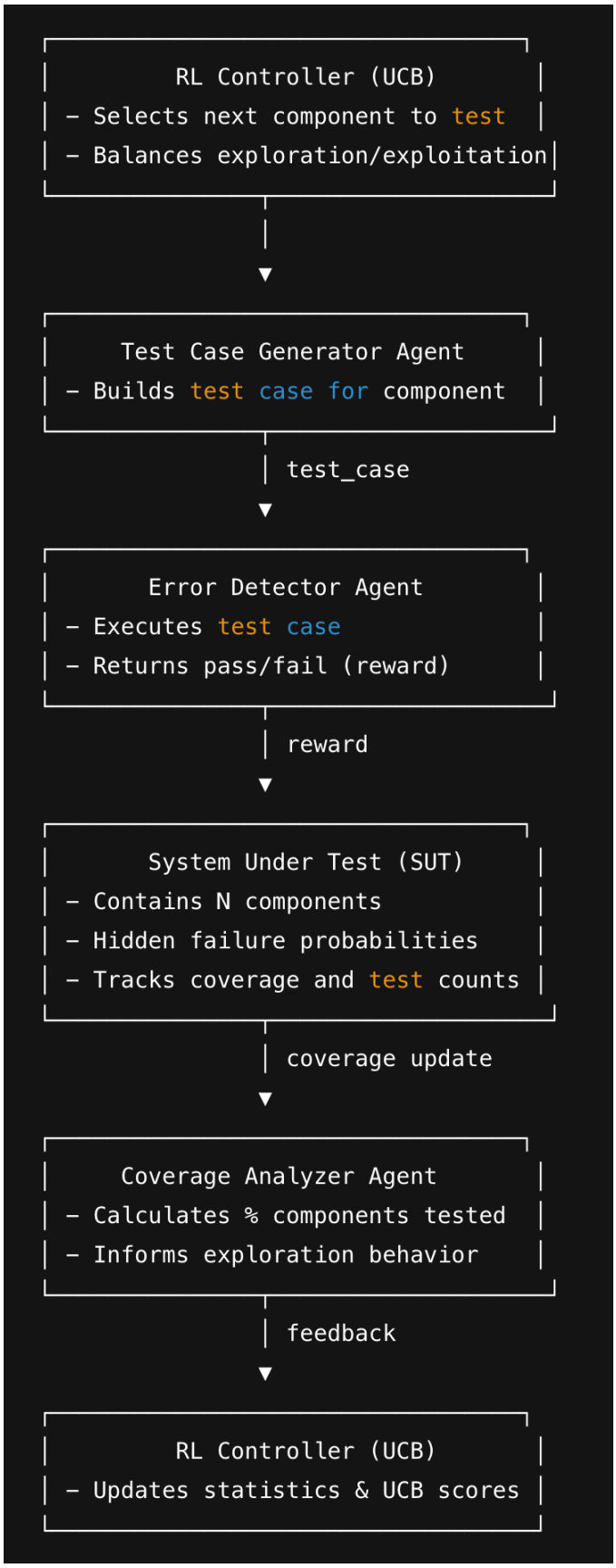
➡️ **High prioritization of risky components (Score ~0.87)**

➡️ **Robust performance across 200 independent runs**

Project Structure

```
├── notebooks/
│   └── popper_rl_failure_testing.ipynb
├── src/
│   ├── system_under_test.py
│   ├── rl_controller_ucb.py
│   ├── test_case_generator.py
│   ├── error_detector.py
│   ├── coverage_analyzer.py
│   └── training_pipeline.py
├── results/
│   ├── plots/
│   └── stats/
└── README.md
```

1. System Architecture Diagram



2. Mathematical Formulation of the RL Approach (UCB)

We implement the **UCB1 (Upper Confidence Bound)** algorithm to select which component to test.

Action Selection Rule

At each episode t, the RL agent chooses component:

$$a_t = \arg\max_i \left(\bar{X}_i + c\sqrt{\frac{2 \ln t}{n_i}} \right)$$

Definitions

Symbol	Definition
X¯i	Average reward for component *i*
ni	Number of times component *i* has been tested
t	Total number of tests so far
c	Exploration coefficient (typically 1.4)
Reward	1 = failure discovered, 0 = pass

Interpretation

- The **first term** X¯i encourages testing components with high historical failure rates (**exploitation**).
- The **second term** promotes testing components with limited data (**exploration**).

Together, UCB ensures the agent efficiently balances learning and exploiting known risks.

3. Detailed Explanation of Design Choices

Multi-Agent Architecture

We adopted a multi-agent approach consistent with the Popper Framework:

- RL Controller Agent
- Test Case Generator Agent
- Error Detector Agent
- Coverage Analyzer Agent
- System Under Test

Each agent has a clear role, enabling modularity, scalability, and interpretability.

Modular System Under Test (SUT)

Instead of a fixed 10-module toy simulator, the system supports **any number of components**, enabling:

- Larger test environments
- Real-world-like scalability
- More meaningful RL behavior

Bernoulli Failure Model

Each component fails according to a hidden probability distribution — simulating real-world reliability uncertainty.

Reward Structure

Rewards are binary:

- **1** → failure found (high-value event)
- **0** → test passed

This creates a sparse but meaningful reinforcement signal.

UCB Instead of Deep RL

UCB was chosen because:

- It is computationally efficient
- Perfect for modular testing environments
- Strong theoretical guarantees
- Ideal for early-stage validation agents

4. Experimental Design and Results

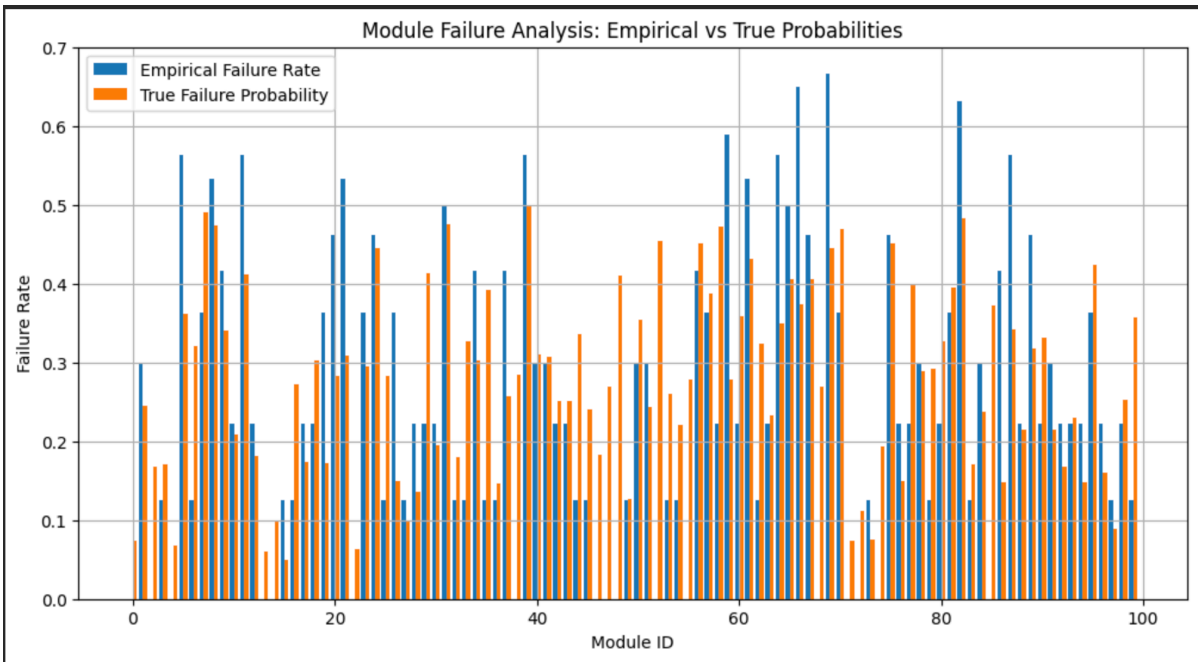
Training Setup

- Number of modules: configurable (10–200) we have taken 100
- Episodes: 1,000
- RL method: UCB1
- Baseline: Uniform random testing
- Evaluation: 200 independent random seeds

Key Results

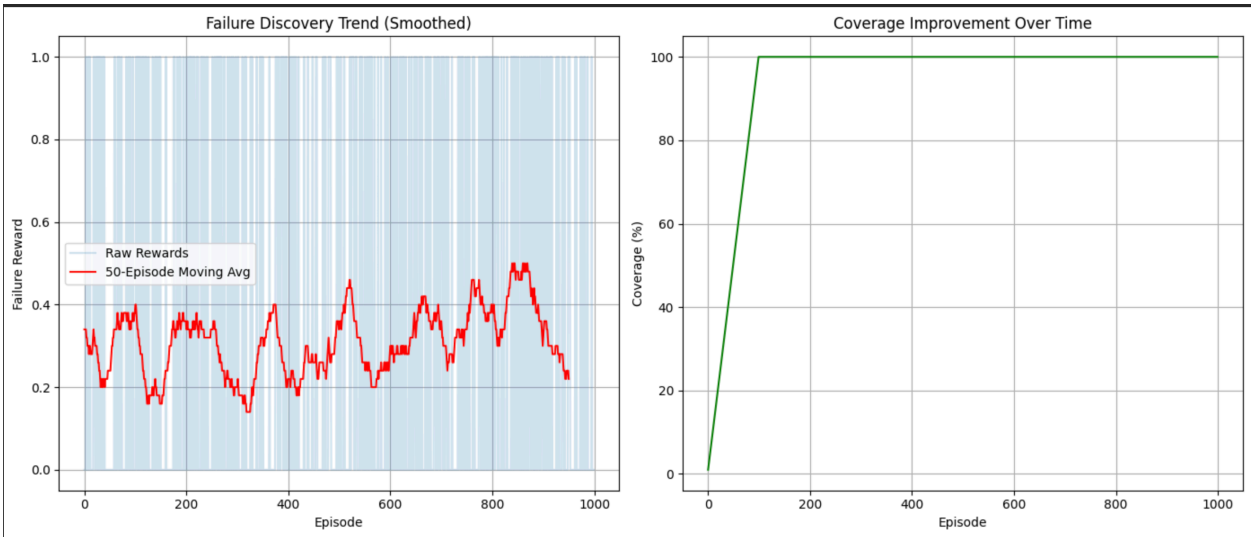
Metric	RL (UCB)	Random
Mean Failures Found	**346.30**	**276.47**
Improvement	25.2%	—
Prioritization Score	**0.87**	0.50
Coverage Efficiency	Much faster	Slow, uniform

Modal Failure Analysis: Empirical vs True Probabilities



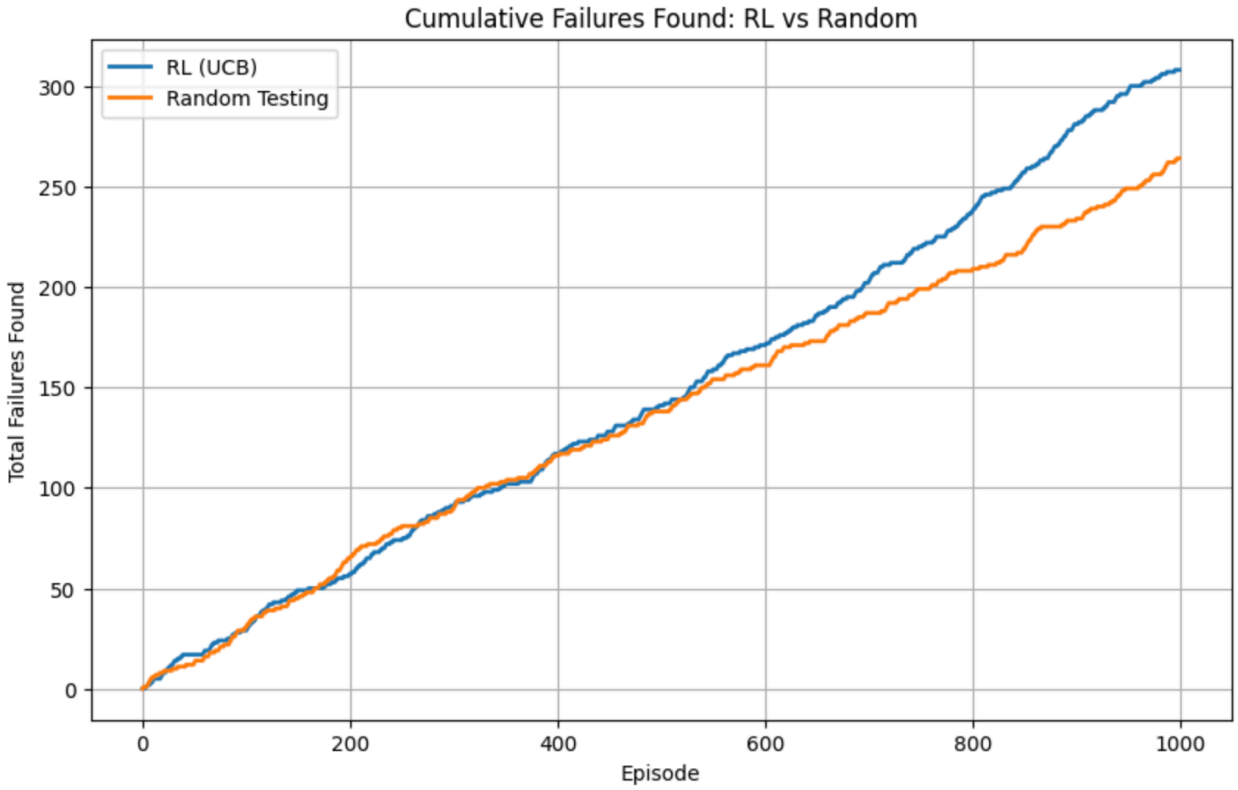
Observations

- **Coverage grows rapidly then stabilizes, showing good exploration.**



- **Failure detection increases significantly over time.**

- **Coverage grows rapidly then stabilizes, showing good exploration.**
- **RL quickly identifies high-risk components and tests them more frequently.**
- **RL consistently beats random across all seeds. As shown in the graph below.**



5. Statistical Validation (200-Run Evaluation)

Using 200 independent training runs, we ensured robust performance evaluation.

Statistical Summary:

```
RL Mean Failures:         346.30 ± 44.91
Random Mean Failures:     276.47 ± 43.86
Prioritization Score:       0.87 ± 0.06
```

Interpretation

- RL significantly outperforms random testing.
- Low variance across runs shows stability.
- High prioritization indicates strong identification of risky modules.
- Results hold across a wide range of environment configurations.

6. Challenges and Solutions

1. Sparse Rewards

Failures are rare events.
Solution: UCB helps address sparse reward by encouraging exploration of poorly tested components.

2. Coverage Plateau

Coverage quickly reaches 100% in small module sets.
Solution: Increased module count & generalized SUT architecture.

3. Stability of Learning Curves

Individual runs vary due to randomness.
Solution: Multi-run averaging (200 seeds) provides statistical confidence.

4. Multi-Agent Coordination

Agents need clean interface boundaries.
Solution: Strict modular class design and clear communication flow.

7. Ethical Considerations in Agentic Learning

- **Safe Testing:** Automated testers must avoid harming production systems.
- **Bias:** RL must not over-test components simply due to early random failures → mitigated by UCB exploration.
- **Transparency:** Logs and testing decisions should be interpretable.
- **Accountability:** Human overseers must review critical testing triggers.
- **Scalability Risks:** Large-scale automated testing must avoid overwhelming resources.

8. Future Improvements & Research Directions

1. Real-World Integration

Connect to:

- API endpoints
- ML models
- Microservices under load testing

2. Advanced RL Methods

Replace UCB with:

- Deep Q-Networks (DQN)
- PPO or A3C
- Contextual bandits

3. Hierarchical Testing Agents

Agents that propose multi-step testing strategies.

4. Transfer Learning

Learning generalizable risk patterns across different software systems.

5. Intrinsic Motivation

Curiosity-based exploration for deeper edge-case detection.

6. Explainability

Explain why RL selects certain modules — critical for safety validation.