ISSN 2093-0542




KOREAN SOCIETY FOR INTERNET INFORMATION

The 21st International Conference on Intelligent Computing (INTCOM 2026)

July 03-06, 2026, O_2 universum, Prague, Czech Republic

https://intcom.kr

Proceedings of INTCOM 2026

| Organized by |

Korean Society for Internet Information (KSII)

| Co-Organized by |

Technical University of Kosice, Slovakia

| Sponsored by |

CzechTourism

Contents

KSII The 21st International Conference on Intelligent Computing (INTCOM) 2026.

An Autonomous Multi-Layer Web Defect Verification Mechanism Using PPO-Based Agent Exploration

Haeun Lee, Junsun Chang, Janghwan Kim, Chaeyun Seo, and R. Young Chul Kim*
Department of Software and Communications Engineering,
Hongik University, Sejong, South Korea
[e-mail: haeum815@g.hongik.ac.kr, cjsun0331@hanmail.net ,{lentoconstante, chaeyun, bob}@hongik.ac.kr]
*Corresponding author: R. Young Chul Kim

Abstract

Until now, Modern Single Page Applications (SPA) and Microservice Architectures (MSA) increase web complexity. This trend causes limitations on scenario based testing. The complex connection between frontend and backend causes asynchronous failures in internal layers. Dynamic state changes also cause critical hidden defects. To solve this problem, we propose a PPO reinforcement learning system based on BrowserGym, which keeps DOM and log signals. Due on this structure, we can detect anomalies in multiple layers like UI, API, DB, server, and security in real time. Batch-based cumulative learning and a penalty strategy for non-defect states prevent exploration stagnation. These methods maximize the detection of edge cases.

Finally, this study built a dataset of 400 websites with multi-layer defects. The experiment proved excellent detection performance for runtime exceptions and logical errors. It confirmed the high potential of autonomous Quality Assurance (QA).

Keywords: Autonomous Web Testing, Deep Reinforcement Learning, Proximal Policy Optimization (PPO), Anomaly Detection, BrowserGym

1. Introduction

Modern Single Page Applications (SPA) and Microservice Architectures (MSA) increase web complexity [1][2]. This trend creates clear limitations for traditional static testing. Modern web defects occur asynchronously across multiple layers like UI, backend API, DB, server, and security. Therefore, UI-centered exploration alone cannot easily detect these multi-layer errors.

However, previous studies did not integrate multi-layer errors into reward systems effectively [3][4][5][6]. This failure causes agents to fall into exploration stagnation or miss critical edge cases. To solve this problem, this paper proposes a PPO reinforcement learning dynamic web error detection system based on BrowserGym. The proposed system observes DOM and log signals. Then, it classifies and detects errors as anomalies. It builds a versatile autonomous exploration framework through a differential reward strategy and port-based batch cumulative learning.

The remainder of this paper is organized as follows. Section 2 reviews related work including traditional web testing techniques and reinforcement learning-based exploration studies to analyze limitations. Section 3 describes the mechanism of the proposed system structure, the layer architecture, and the differential reward model. Section 4 presents the experimental environment using the error injection testbed and multi-layer defect results. Finally, Section 5 concludes the paper and suggests future research directions.

2. Related Work

As a prior work, WebExplor is a study that models web interaction as an MDP to overcome random testing limitations and learn the optimal exploration policy. The core mechanism of this research includes HTML-based state abstraction, curiosity-driven rewards, and DFA-guided exploration. **Fig. 1** shows the workflow of WebExplor, a representative reinforcement learning web testing study [3].

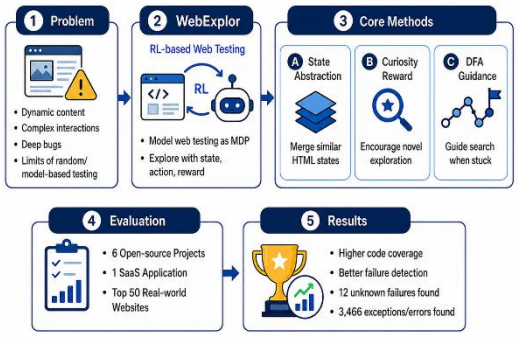

Fig. 1. WebExplor research mechanism and execution workflow

The researchers tested open-source projects and commercial SaaS environments. This system proved excellent coverage in real web environments and discovered over 3,400 errors. However, it focuses mostly on UI surface exploration expansion. Therefore, it has limitations in detecting multi-layer defects linked with backend APIs or databases. Also, it ignores infrastructure signals, which makes detecting complex asynchronous failures difficult.

3. Autonomous Intelligent Web Defect Exploration System

3.1 System Overview and Integrated Architecture

This section describes the overall structure and core mechanism of the proposed system. **Fig. 2** shows the entire architecture and data flow of the framework. The proposed framework removes reliance on fixed scenarios or predefined scripts of traditional Quality Assurance (QA). Instead, the agent interacts directly with complex dynamic web environments and explores runtime defects autonomously.

The entire process forms an organic feedback loop. It includes environment initialization, state refinement, PPO-based policy optimization, multi-layer defect classification, and differential reward calculation. This structure maximizes exploration efficiency within a large web state space. It traces undefined potential defects autonomously. Especially, the framework updates cumulative trajectory experiences from multi-port distributed defect environments into a central shared model, and it analyzes lower infrastructure metrics and upper frontend logs in real time to clarify the causal relationships of complex failures. Consequently, it expands test coverage and ensures software reliability efficiently.

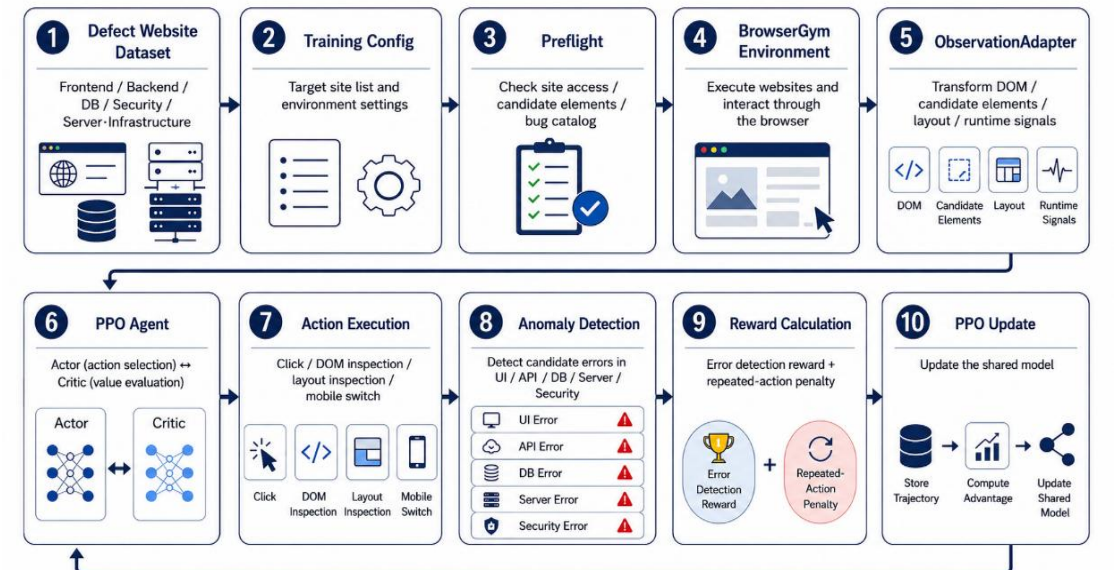

Fig. 2. System Overview and Integrated Mechanism

3.2 Pipeline-Based Autonomous Exploration and Learning Process

The proposed system operates the entire process from web environment perception to model optimization through an organic pipeline. First, the framework builds a defect website dataset with multi-layer errors and synchronizes the target environment through configuration settings. Before exploration, a preflight step checks site access and identifies interactive candidate elements. Then, the system executes the target website in a browser environment. A specialized observation adapter receives real-time runtime signals like DOM structures, layouts, and console logs. It transforms these signals into refined feature vectors without noise.

The actor-critic network of the autonomous agent receives this optimized state input to determine behavior policies. When the agent decides the optimal action, an execution module performs tasks like clicks or DOM inspections in the browser in real time. Immediately after the action, an anomaly detection module categorizes multi-layer defects across the UI, API, and database. Then, a reward calculation module applies a differential reward system. It gives positive rewards for new error detection and penalties for continuous exploration stagnation. Finally, the system stores the trajectory, computes the advantage, and updates the shared model to complete the autonomous optimization loop.

4. Error Injection Testbed for System Feasibility Verification

4.1 Pre-Verification Using Error Injection Testbed

To evaluate the feasibility of the proposed system, a dynamic testbed based on a concert ticketing system was constructed. To verify the multi-layer state perception capability of the agent in an environment with real-time DOM mutations and large-scale traffic simulations, 21 types of architecture-specific defects were injected to scale the environmental complexity up to a commercial service level.

Specifically, 11 server and infrastructure errors, including interrupt-storm, thundering-herd, and kernel-lockup, were configured in the kernel and container layers to simulate massive traffic situations, inducing abrupt changes in CPU Steal, Context Switches, Memory Pressure, and P99 Latency.

In addition, 5 frontend errors for observing visual and structural anomalies in the UI layer inside the BrowserGym environment were linked with 5 backend API errors that capture HTTP status code distortions and JSON schema mismatches during network communication, thereby establishing a complex multi-layer defect environment.

During the experimental process, a differential reward control strategy was applied to continuously assign negative rewards to exploration states that repeat normal paths without finding defects. This prevents early convergence to specific normal states or local optima and encourages the agent to actively explore uncharted state spaces, thereby helping to successfully trace rare, intermittent edge-case errors that are difficult to discover with traditional static scripts, while substantially improving overall fault coverage and defect detection reliability across multiple architectural layers simultaneously.

4.2 Experimental Results and Analysis in the Extended Framework

After prototype verification, the framework was expanded into a port-based multi-environment batch learning architecture utilizing BrowserGym. As the autonomous agent accumulates trajectory experiences from each port into a shared PPO model, the system learns optimal action sequences with a high probability of causing defects. **Fig. 3** shows the monitoring interface classifying real-time anomalies based on infrastructure metrics and browser runtime logs from a specific defect port.

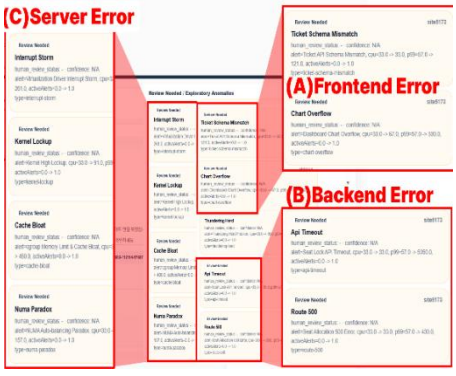

Fig. 3. Multi-layer defect classification and real-time anomaly monitoring interface

The Frontend Error area (A) of the interface classifies data mismatch exceptions between the UI and network interactions alongside chart overflow, tracking how visual layout degradation directly correlates with delayed client-side state synchronization. Concurrently, the Backend Error area (B) identifies API timeouts caused by seat lock delays and Route 500 errors from internal asynchronous verification failures, pinpointing bottlenecks within the distributed service mesh layer. The Server Error area (C) tracks lower system layer resource exhaustion, including interrupt storms triggered by massive requests, cache bloat, and the NUMA paradox, which reveals hidden hardware-level constraints. Rather than merely capturing isolated single-layer faults, this integrated multi-layer monitoring structure functions as a precise root-cause analysis tool that maps the sequential propagation of failures; it traces how lower infrastructure metric pressures cascade into backend API timeouts, eventually culminating in frontend UI deadlocks or synchronization anomalies. Finally, the system automatically records all real-time exploration histories, telemetry logs, and state transition trajectories into structured CSV files, significantly improving quality management efficiency, fault reproducibility, and post-error debugging precision.

5. Conclusion

This study proposed a PPO reinforcement learning dynamic web error detection system using BrowserGym and Playwright environments to detect runtime errors and anomalies in complex and dynamic web environments autonomously. The proposed system introduced a differential reward framework that gives high positive rewards for defect detection and negative rewards for exploration stagnation.

This mechanism successfully improved the detection probability of edge cases and logical defects. Additionally, the framework structured exploration events into five core layers: Navigate, Action, State, Error, and Network. This structure proved its feasibility as a causal analysis tool to trace defect causes and state transitions.

However, a current limitation involves some manual tasks during the initial environment configuration process. To overcome this limitation, future research will integrate Large Language Models (LLMs) to diagnose the root causes of defects based on collected structured logs and exception data. This extended system will generate intelligent quality analysis reports automatically.

This future direction will intellectualize the entire QA process from defect detection to root cause analysis. Consequently, it will advance a fully autonomous software quality verification framework with minimal human intervention.

Acknowledgment

This study is the result of a capstone design project by undergraduate students in the Department of Software Convergence at Hongik University. Additionally, this study is the result of a project by the 6th cohort trainees of the Metaverse Convergence SW Academy at Hongik University (Project Number: 202301830004).

References

[1] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang, “World of Bits: An Open-Domain Platform for Web-Based Agents,” in *Proc. of International Conference on Machine Learning (ICML)*, pp.3135-3144, 2017.

[2] Y. Lu, J. Huang, H. Liu, J. Gesi, Y. Han, S. Fu, T. Zheng, and D. Wang, “WEBSERV: A Browser-Server Environment for Efficient Training of Reinforcement Learning-based Web Agents at Scale,” *arXiv preprint arXiv:2510.16252*, 2025.

[3] Y. Zheng, Y. Liu, X. Xie, Y. Liu, L. Ma, J. Hao, and Y. Liu, “Automatic Web Testing Using Curiosity-Driven Reinforcement Learning,” *arXiv preprint arXiv:2103.06018*, 2021.

[4] A. H. Mughal, “An Autonomous RL Agent Methodology for Dynamic Web UI Testing in a BDD Framework,” *arXiv preprint arXiv:2503.08464*, 2025.

[5] Z. Gu, C. Liu, G. Wu, Y. Zhang, C. Yang, Z. Liang, W. Chen, and J. Wei, “Deep Reinforcement Learning for Automated Web GUI Testing,” *arXiv preprint arXiv:2504.19237*, 2025.

[6] D. López-Montero, J. L. Álvarez-Aldana, A. Morales-Martínez, M. Gil-López, and J. M. Auñón García, “Reinforcement Learning for Automated Cybersecurity Penetration Testing,” *arXiv preprint arXiv:2507.02969*, 2025.