ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Self-Healing Test Automation Frameworks Using Reinforcement Learning for Full-Stack Test Automation

Hariprasad Sivaraman

USA

ABSTRACT

In dynamic environments such as e-commerce, test automation frameworks often struggle as applications are continuously changing. In this paper a test automation framework is being proposed with self-healing ability that leverages Reinforcement Learning (RL) to automatically adapt test cases to new versions of interface, Application Programming Interfaces (APIs) and the database schema. With the integration of RL, the framework operates with minimal human intervention, makes it easier to generalize the results and lowers the costs related to maintaining the tests. Through the framework, examples inspired from common e-commerce use-cases are assessed and demonstrate the potential of RL to inject additional resilience & adaptive behavior in full-stack test automation.

***Corresponding author**

Hariprasad Sivaraman, USA.

Received: June 10, 2022; **Accepted:** June 17, 2022; **Published:** June 30, 2022

Keywords: Self-Healing Framework, Reinforcement Learning, Full-Stack Test Automation, Autonomous Testing, Machine Learning, E-Commerce Testing

Introduction

The evolution of e-commerce platforms has hastened the requirement of advanced, robust, and versatile test automation frameworks that can accommodate changes from the underlying technology stack such as UI, API or backend. This over-dependence on static identifiers and hard-coded elements makes traditional test automation approaches susceptible to breakage when user interfaces, API endpoints, or backend schemas change. For example, if a user changes the checkout page structure or change an API endpoint, it may break the automated tests causing false negatives and costing lot of effort to maintain these tests.

In such scenarios there is a requirement of an intelligent, self-healing test automation framework that works on autopilot and quickly adapts to any changes in the application. The journey of reinforcement learning, a subfield of machine learning aimed at maximizing cumulative rewards to optimize decisions in an environment, is an interesting one and offers a decent solution. With its flexibility, RL is applied for full stack test automation where this framework can automatically adapt for any changes in UI, APIs or database schemas and thus bringing in high level of adaptability and low maintenance.

Problem Statement

Classic test automation is prone to brittle tests that need to be constantly updated with every application component update. In e-commerce contexts, this challenge is exacerbated by the need for frequent change driven by the high demand for user interaction, and the need for rapid response to changing business requirements. Here are some examples of issues you might encounter:

- **UI Updates:** A static locator is likely to become useless when the UI gets updated frequently (e.g. changing button Ids or adding new fields).
- **API Endpoint Changes:** These changes in URL, parameters or response structure break API-based tests, especially tests related to transaction, check-out flow and inventory updates.
- **Database Schema Changes:** If the database schema has been changed (add a field, remove a column, change validation) then tests that validate data from the backend will fail.

This framework embeds RL where tests can adapt themselves with changes made at the application side and helps in increasing stability over UI, API and backend layers by solving these problems.

Solution: Reinforcement Learning in Self-Healing Frameworks

RL models are based on learning the best actions to take in an environment by maximizing the sum of rewards over time. In the introductory example of test automation, RL agent gets trained to dynamically modify test cases by observing states of application and choosing appropriate corrective actions to ensure test reliability.

Self-Healing Framework Using RL Components

State Representation

From this perspective, the state will be a full view of the testing environment and its response to the test case being executed. Each state represents the following information:

- **UI Elements:** Holds the configuration of the DOM elements used in a test case (like button ID, class names, XPaths, data attributes)
- **API Endpoints and Payload:** Details of API endpoint URLs,

parameter structure, headers, and response structure.
- **Database Schema:** Represents the table schemas and how the data is expected to be saved for being verified at the backend.
- **Execution Context:** Maintains the current execution step in the test case – If its a load state for a checkout page, or transaction state, or database query result.

The state here is specifically a vector encoding of the current state of the DOM (Document Object Model) structure, API schema, and the state of the database schemas. By representing it in this format, the RL model can treat every state as a single entity and learn to generalize on the UI, API, and the backend layers.

Action Space
The actions that the RL agent can take are the possible changes it can implement on self-heal test failures. Those actions tackle the most common failures areas in full-stack testing:

UI Self-Healing Actions
- **Locator Modification:** Change CSS selector or XPath based on adjacent elements or data attributes.
- **Element Retry:** Retry locating an element with a small delay, handy for asynchronous UI loads.
- **e Element Replacement:** Replace with an alternate element when a similar element is detected (Multiple "Add to Cart" buttons.)

API Self-Healing Actions
- **Payload Change:** Change payload keys or values based on pattern seen in successful requests.
- **Retry with Fallback Parameters:** If it does not work, the agent will attempt a fallback parameter seen to work before.
- **Endpoint Update:** Adjust API endpoint or parameters based on common changes (e.g., modifying /api/v1/checkout to / api/v2/checkout)

Self-healing Actions of Database
- **Schema Adaptation Change:** SQL queries in case new fields are added or some fields are eliminated from the database.
- **Conditional Validation:** Update the backend validation to include/exclude fields based on the latest schema.
- **Fallback Strategy:** Fallback Query Retry with Default Field Rather than querying all the columns apply where clause to exclude rows with non-NULL columns

Given the current state and its training to maximize the success of a test passing, the agent selects an action.

Reward Mechanism
In the prior step, each action can be evaluated by providing a reward system that tells the RL agent if the action is success or not, e.g.

Positive Reward
- Due to self-healing of a test case and making it pass.
- To reduce the time of executing tests (e.g., Attempts should be minimal).

Negative Reward:
- To commit an undesired act or overtry
- Number of attempts to self-heal after which the test fails.
- Neutral Reward:
- Does nothing beneficial but keeps the test pipeline with no substantial advance or regression.

By receiving rewards, the agent learns to avoid actions that lead to test failures and execution time, thus leading it to take the best actions in future runs.

Reinforcement Learning Algorithm
Deep Q-Learning (DQN), which approximates the Q-value for each state-action pair using deep neural networks, is a widely used algorithm for this type of problem. The Q-values indicate expected total rewards for taking a specific action in a specific state in this framework.

Architecture of Deep Q-Network (DQN)
- **Encoder (Input Layer):** Encodes the current state vector (UI, API, and Database details)
- **Hidden Layers:** One or more dense layers to model the non-linear relationships between the surface, the API changes, and the changes in the backend schema, respectively.
- **Output Layer:** Q-values for each action, letting the agent select the action with the best expected reward.

Training the DQN Agent
- **Experience Replay:** The agent keeps a memory of state-action-reward-next-state tuples. At training time, it samples random batches from this memory for increased generalization.
- **Target Network:** a second network, which is updated to the weights of the first, reduces correlations which stabilizes learning.
- **Exploration vs. Exploitation:** The agent explores random actions initially to learn good self-healing paths. Eventually it becomes exploitative and picks actions which had a higher reward in the past.

Training Example
For e.g., take a checkout test case in an e commerce site where a dynamic change happened with the ID of the Place Order button to cause the test case to fail.
- **First State:** The agent now sees that the "Place Order" button is not present.
- **Action:** It attempts to locate the similar type of buttons using a new locator strategy.
- **Reward:** The action restores the test flow (positive reward) or goes back to penalty.
- **Learn:** The agent modifies its Q-values, more likely to select locator modifications for the same set of UI changes in the future execution.

Uses and Benefits of Self-Healing Frameworks
The ability of a self-healing test automation framework to solve these new-age challenges — powered by RL — can translate to significant benefits for businesses at scale, especially for large and complex e-commerce environments. Beyond core advantages such as lower maintenance and higher stability, there are a few unique features that come in with niRL based approach for automation of tests:
- **Dynamic Change Resistance:** E-commerce platforms continuously change UI/UX, API, and schema. RL, on the other hand, the framework self-adjusts to these changes, continues to adapt to promise the same test coverage as the app layer changes.
- **Learning / Adaptive:** The RL-based framework improves every time it carries out the test vs the scripted automation that keeps running the same script for the same test. The agent fine-tunes its approach as it faces different situations, improving at solving breakdowns without need for help.

- **Lower Manual Effort and Error:** In automatically addressing common points of failure, the framework reduces manual effort to update test cases by developers and testers thereby allowing them to concentrate their time and efforts on strategic test design and higher-level validation activities.
- **Release Cycle Disruption:** Repeated maintenance of tests takes up time and finances. The result is lesser dependency, low maintenance costs, quick testing timelines, and ultimately quicker product deployments and lesser operational expenditure as compared to rule-based frameworks (enabled with RL).
- **Improved Accuracy in Complicated Use Cases:** E-commerce systems commonly incorporate complex, multi-step workflows such as the checkout process and inventory management. In these situations, an RL-based framework that minimizes false negatives, increases test robustness by learning optimal correction strategies, and therefore enhances the precision of these tests can be advantageous.

Practical Example: Applying RL in E-Commerce Full-Stack Test Case

For instance, a typical scenario is a checkout process in an e-commerce test environment, where you have multiple interactions between layers such as UI, API and Database interacting with each other and a RL-based self-healing framework running to detect and heal errors on the fly. The RL agent must adapt to the changes independently, as each layer is a challenge on its own.

Scenario: The Checkout Workflow

A normal checkout workflow on an e-commerce site consists of the below steps:

- **Product Selection and Cart Addition:** The user picks the goods and adds the same to the shopping cart.
- **Review and Checkout:** The user sees their cart, changes the quantities if needed, and presses the Proceed to Checkout button.
- **User Authentication:** The user must be authenticated (login) if he/she is not logged in to continue
- **Order processing:** The checkout API handles the order—totals, taxes, and discounts.
- **Order Confirmation:** This involves an update in the database which assures the order and records data in the backend.

In this process, the RL agent interacts with each layer which can change/fail that can break traditional automation scripts.

Layer-Specific Adaptations Using Reinforcement Learning

UI Layer - Dynamic Locator Handling

In the UI layer, dynamic changes to element locators, such as the IDs or classes of buttons and
text fields, are common. For example, imagine that the ID for the "Place Order" button was changed after a user interface styling update.

- **State representation:** DOM structure that agent is focused on including attributes of button, other elements that are neighboring it, and the class name to capture
- **Reward:** The RL agent tries different locators. In case primary locator fails for "Place Order" button, it falls back to XPath relative to a stable parent element or use CSS selectors based on nearby text labels.
- **Reward System:** Positive reward when the button is clicked, thus allowing the test to move forward. In this case, the agent will receive a negative reward, and it will be punished for trying too many times.
- By interacting with the world many times, the agent learns successful locator

API Layer - Endpoint and Payload Adaptation

One way the backend can affect the API layer is by changing the endpoint structure or payload. For instance, if the order processing endpoint is changed from /api/v1/checkout to /api/v2/checkout, or if the payload schema includes new fields.

- State""" Representation""" State""" """ Info""""""" State-action\n State input\n""" Example""" State""" Representation""" The agent holds information about the endpoint to connect to, url, request headers, and other useful payload structure
- **Behavior:** To accommodate for the failed API call, RL agent acts by changing endpoint URL or payload keys. A good example would be to find out if a parameter is missing, like discountCode, and have the agent inject it because previous requests were successful with it.
- **Reward Mechanism:** The API call returns successfully, which means that these parameter changes match the backend requirements, and thus we are given a positive reward. In case of fails in retries or too many modifications it applies negative awards.

Through its actions, the agent updates the API interactions, allowing it to operate independently to respond to backend changes with minimal human intervention, learning over time what changes lead to successful API calls.

Database Layer - Schema Adjustment for Data Validation

Example 1: There are changes in schema fields or table structure in the database layer which affects the validation of the test. Such as a user can add a new column — order Discount in the orders table to store discount applied at XML checkout.

- **States Representation:** The agent learns the schema of the database, which includes the table layouts, the column names and the data types which are used for testing validations.
- **Example of Changing Queries:** The RL agent updates queries by verifying the existence of the order Discount field. If so, it modifies the SQL query to use the new column in data validations. If a given test doesn't require that column then it can simply choose to ignore it, which is the flexibility validation needs.
- **Reward Model:** A positive reward is given when the query is validated successfully based on the adjusted schema. Spamming/retrying or omitting information will incur penalties. This allows the agent to automatically reconfigure to the schema at hand and accurately validate backend data throughout the tests.

Conclusion and Future Directions

This paper proposed a self-healing test automation framework driven by reinforcement learning, specifically for complex full-stack environments like e-commerce platforms. With a framework that automatically adapts to changes in UI, API and other backend components, this ensures minimum manual test maintenance, which means, lesser-cost and faster testing, along with increased reliability of software.

Future Directions

Despite success using RL for this purpose, opportunistic exploration may not be the only way for further improving our framework, as future work can explore:

- **Multi-Agent Reinforcement Learning:** Using multiple agents for different application layers (i.e., UI, API and database) might allow more specialized learning with respect to different application layers since each agent would maximize its resilience at that specific layer.
- **Improved Reward Shaping:** The reward structure can be fine-tuned for incentivizing the stability along complex workflows leading to improved stability and accuracy for multi-step test cases.
- **Transfer Learning:** Transfer Learning can be utilized to speed up learning for the RL agent by transferring knowledge from one domain (e.g., user checkout flows) to updated, but similar workflows within the same application.
- **Real-Time Monitoring and Adaptation:** By integrating the RL agent with different real-time monitoring tools, the agent can detect application changes in real-time and adapt its testing plan, accordingly, thereby improving test success ratio in CI/CD pipelines.

Such a self-healing test automation framework powered with RL is an ideal example of resilient, efficient and adaptive test automation which can cater to the needs of fast, changing nature of e-commerce platforms and similar kinds of high-dynamic software environments [1-13].

References

1. Sutton RS, Barto AG (2018) Reinforcement Learning: An Introduction. MIT Press https://ieeexplore.ieee.org/document/712192.
2. Wang Y, Kong X, Bai J, Zhang J, Chen H (2021) Self-healing Test Automation for GUI Testing of Web Applications. IEEE Transactions on Software Engineering 47: 1153-1172.
3. Li Y, Duan Y, Chen X, Schulman J (2017) Self-Healing Automation Framework Using Machine Learning. IEEE Transactions on Software Engineering 43: 342-354.
4. Mao H, Alizadeh M, Menache I, Kandula S (2016) Resource Management with Deep Reinforcement Learning. Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets-XV) 50-56.
5. Machado MC, Bellemare MG, Bowling M (2018) Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. Journal of Artificial Intelligence Research 61: 523-562.
6. Nascimento M, Cunha T, Silveira R (2020) An Adaptive Test Automation Framework Using Reinforcement Learning. Proceedings of the International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 171-175.
7. Testa G, Angius D, Orsenigo C (2019) A Framework for Self-Healing Software Using Deep Learning Techniques. Future Generation Computer Systems 100: 1025-1041.
8. Yosinski J, Clune J, Bengio Y, Lipson H (2014) How Transferable Are Features in Deep Neural Networks? Advances in Neural Information Processing Systems (NeurIPS) 3320-3328.
9. Finn C, Abbeel P, Levine S (2017) Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. Proceedings of the 34th International Conference on Machine Learning (ICML) 1126-1135.
10. Murdoch WJ, Singh C, Kumbier K, Abbasi-Asl R, Yu B (2019) Interpretable Machine Learning: Definitions, Methods, and Applications.
11. Dietterich TG (2000) An Overview of Hierarchical Reinforcement Learning. Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics.
12. Huang W, Boehm B (2017) Exploring Self-Healing in Software Systems.
13. Liang X, Mendelson J (2019) Deep Reinforcement Learning for Continuous Control in Software Engineering Applications.