Reinforcement Learning Foundations and Curiosity-Driven Exploration for Web Application Testing

A Study of RL Basics and Analysis of WebRLED

Rujula Rahate

January 05, 2026

Contents

1 Reinforcement Learning Basics

1.1 Agent–Environment Interaction

In reinforcement learning (RL), an agent interacts with an environment over discrete time steps. The most important conceptual distinction is that while the agent has complete control over the actions it chooses to take, it has no direct control over how the environment responds. The environment evolves according to its own internal dynamics, which may be deterministic or stochastic, simple or highly complex, and often unknown to the agent.

Learning in RL occurs entirely through interaction. The agent is never explicitly told which actions are correct or incorrect. Instead, it must infer good behavior by observing how the environment changes in response to its actions and by interpreting the feedback signal provided in the form of rewards.

At each time step, the agent observes the current state of the environment, which represents everything the agent is aware of at that moment. Based on this observation and its current policy, the agent selects an action. The environment then transitions to a new state according to its internal transition rules, and a scalar reward is returned. This reward serves as a quantitative measure of how beneficial or harmful the chosen action was in that context. This loop of observation, action, transition, and reward forms the fundamental feedback mechanism of reinforcement learning and drives the learning process over time.

1.2 Markov Decision Processes (MDPs)

The standard mathematical framework used to describe reinforcement learning problems is the Markov Decision Process (MDP). An MDP formalizes sequential decision-making in situations where outcomes are uncertain and rewards may be delayed.

An MDP is defined by a set of possible states that describe the environment, a set of actions available to the agent, a transition function that specifies how the environment moves from one state to another given an action, a reward function that assigns a numerical value to state–action pairs, and a discount factor that determines how much future rewards are valued relative to immediate ones.

The interaction between the agent and the environment generates a trajectory of the form:

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

This sequence captures the complete experience of the agent over time and serves as the data from which learning occurs.

1.2.1 Limitations of the Markov Assumption

A defining characteristic of an MDP is the Markov property, which assumes that the future depends only on the current state and action and not on the entire history. While this assumption simplifies both modeling and learning, it is often violated in real-world systems.

In many practical scenarios, the state representation available to the agent may be incomplete or lossy, meaning that important historical or contextual information is not captured. Additionally, hidden variables or latent system states may influence transitions in ways that are not observable. Web applications are a strong example of this limitation, as two pages may appear structurally identical but behave very differently due to hidden backend logic, session variables, or JavaScript state. As a result, designing a meaningful and robust state representation becomes a critical challenge.

1.3 Rewards and Returns

In reinforcement learning, the reward received at each time step is represented as a single scalar value. Even when a system has multiple performance objectives, such as coverage, efficiency, safety, or novelty, these objectives must ultimately be compressed into a single numerical signal. This makes reward design one of the most delicate and important aspects of reinforcement learning, as poorly designed rewards can easily lead to unintended or undesirable behavior.

To evaluate long-term performance rather than just immediate outcomes, the notion of return is introduced. The return is defined as the discounted sum of future rewards:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

The discount factor γ reflects uncertainty in future predictions and ensures that rewards received further in the future have less influence on current decisions. Intuitively, immediate outcomes are trusted more than distant ones, since the future is harder to predict accurately.

1.4 Policy Function

A policy defines the behavior of the agent by specifying how actions are chosen in each state. Formally, a policy $\pi(a \mid s)$ represents the probability of taking action a when the agent is in state s.

Rather than being deterministic, policies in reinforcement learning are typically stochastic. This means that even in the same state, the agent may choose different actions on different occasions. This randomness is not accidental; it plays a crucial role in exploration by allowing the agent to try multiple strategies and avoid prematurely committing to suboptimal behavior.

The ultimate goal of reinforcement learning is to find an optimal policy π^* that maximizes the expected return over time.

1.5 Value Functions

Value functions provide a way to evaluate how good different states and actions are under a given policy. The state-value function $V^\pi(s)$ represents the expected return obtained by starting in state s and following policy π thereafter. Similarly, the action-value function $Q^\pi(s, a)$ represents the expected return obtained by taking action a in state s and then following policy π.

The optimal value functions, denoted $V^*(s)$ and $Q^*(s, a)$, correspond to the maximum achievable return when acting optimally. A key conceptual insight is that there exists a cyclic dependency between policies and value functions. The policy relies on the value function to decide which actions are preferable, while the value function depends on the policy to evaluate expected outcomes. Most reinforcement learning algorithms can be understood as methods that iteratively refine this interdependent relationship.

1.6 Exploration vs Exploitation

One of the central challenges in reinforcement learning is balancing exploration and exploitation. Exploration refers to the agent's tendency to try different actions in order to gather information about the environment, while exploitation refers to choosing actions that are already known to yield high rewards.

Early in training, the agent has little knowledge about the environment, so exploration is emphasized. As learning progresses and value estimates become more reliable, the agent gradually shifts toward exploitation, increasingly favoring actions that are believed to be optimal. Managing this tradeoff effectively is essential for avoiding local optima and achieving good long-term performance.

1.7 Monte Carlo Methods

Monte Carlo methods estimate value functions by averaging the returns observed over complete episodes. In this approach, no updates are made until an episode has fully terminated, at which point the observed return is used to update the value estimates for all states or state–action pairs encountered during the episode.

While Monte Carlo methods are conceptually simple and produce unbiased estimates, they suffer from high variance and poor sample efficiency, especially in environments with long episodes. Additionally, because rewards are only evaluated after an episode ends, it becomes difficult to assign credit to individual actions taken earlier in the sequence.

1.8 Temporal Difference Learning

Temporal Difference (TD) learning addresses the limitations of Monte Carlo methods by updating value estimates incrementally at each time step. Rather than waiting until the end of an episode, TD methods use current reward information together with an estimate of future value to update the value function:

$$V(s_t) \leftarrow V(s_t) + \alpha \left(r_t + \gamma V(s_{t+1}) - V(s_t) \right)$$

This approach significantly improves learning efficiency and directly addresses the credit assignment problem by assigning partial responsibility to actions as soon as their consequences begin to unfold.

1.9 Deep Q-Networks (DQN)

Deep Q-Networks extend traditional Q-learning by using neural networks to approximate the action-value function. This allows reinforcement learning to scale to environments with high-dimensional and continuous state spaces, such as images or complex sensor inputs.

However, DQN still requires the action space to be discrete. This limitation arises because value-based methods must evaluate the value of all possible actions in order to select the best one, which is not feasible when the action space is continuous.

1.10 Policy Gradient Methods

Policy gradient methods take a fundamentally different approach by learning the policy directly. Instead of estimating value functions and deriving a policy from them, these methods optimize a performance objective that measures how well the policy performs.

The policy parameters are updated in the direction that increases expected reward, according to the gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E} \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot \text{Advantage}(s_t, a_t) \right]$$

This formulation captures the intuitive idea that actions leading to good outcomes should become more likely, while actions leading to poor outcomes should become less likely. Policy gradient methods naturally support both discrete and continuous action spaces.

2 Research Paper Analysis

2.1 Tester's Perspective

To understand the WebRLED paper effectively, it is helpful to abandon reinforcement learning terminology initially and instead think like a software tester. From this perspective, the agent is simply an automated tester interacting with a web application.

At any moment, the tester sees exactly one web page and can interact with elements present on that page. Each interaction corresponds to a user action such as clicking, typing, or selecting an option. When mapped to reinforcement learning, the state corresponds to a semantic representation of the current page, the action corresponds to interacting with a single DOM element, and the reward measures how novel the resulting page is.

Importantly, the objective is not to complete a predefined task or reach a specific goal state. Instead, the goal is to explore as much of the application as possible and uncover previously unseen pages, behaviors, and potential defects.

2.2 Challenges in Applying RL to Web GUIs

2.2.1 Action Space Misalignment

Traditional reinforcement learning assumes that the same action space is available in every state and that actions have consistent meanings across states. Web applications violate this assumption completely. Each page has a different number of actionable elements, and there is no stable semantic alignment between actions on different pages. As a result, mapping states directly to action indices becomes unstable and difficult to generalize across pages.

2.2.2 Action Recognition Difficulty

Another major challenge arises from the difficulty of identifying actionable elements on web pages. In modern web applications, any DOM element can be made interactive through JavaScript, regardless of its HTML tag. Relying solely on heuristic rules based on tags such as `<button>` or `<input>` causes many valid actions to be missed. When actions are missed, entire sections of the application remain unexplored, severely limiting test coverage.

2.3 WebRLED Overview

WebRLED addresses these challenges by fundamentally rethinking how actions, states, and rewards are represented. Rather than learning values for individual actions, the agent learns which regions of a page are promising for exploration. At the same time, actions are discovered dynamically using a learned action discriminator, and rewards are driven by curiosity rather than task completion.

2.4 State Representation

Raw HTML is a poor representation for reinforcement learning because it is variable in length, noisy, and highly sensitive to irrelevant changes such as timestamps or user-specific content. WebRLED simplifies the DOM by removing scripts, styles, duplicated subtrees, and masking textual content. This abstraction ensures that the learned state representation focuses on the structural layout of the page rather than superficial details, enabling robust detection of truly novel states.

2.5 Action Recognition

Action recognition in WebRLED is performed using a combination of heuristic rules and a neural action discriminator. The discriminator learns from experience by observing whether interactions with elements lead to state changes. Training occurs both online during exploration and offline at the end of episodes, allowing the model to continuously refine its understanding of what constitutes an actionable element.

2.6 Grid-Based Action Value Learning

To address action misalignment, each page is divided into an $N \times N$ grid, and the Deep Q-Network predicts values for grid cells instead of individual actions. Action values are computed by aggregating the values of nearby cells, allowing the agent to generalize across pages by learning which areas of the interface are typically associated with novel states.

2.7 Reward Modeling

Using only a global novelty-based reward leads to reward collapse over time, as fewer genuinely new states remain to be discovered. WebRLED overcomes this issue by combining episodic novelty, which encourages exploration within a single episode, with global novelty, which discourages repeatedly visiting already explored states. Global novelty is measured using an autoencoder, where reconstruction error serves as an indicator of how unfamiliar a state is.

3 Implementation Plan for Our Project

3.1 System Pipeline

Our implementation closely follows the WebRLED architecture, but more importantly, each stage of the pipeline exists to solve a specific failure mode of applying reinforcement learning to web applications.

The pipeline begins with HTML preprocessing and state embedding, because raw HTML is not a usable input for reinforcement learning. HTML documents are variable in length, extremely noisy, and contain large amounts of information that are irrelevant for decision making, such as scripts, styles, duplicated DOM subtrees, and dynamic text content. If these elements are not removed, the agent will incorrectly treat functionally identical pages as novel states, leading to false exploration signals. By simplifying the DOM and converting it into a fixed-length semantic embedding, the agent receives a stable and comparable representation of the current page that preserves structure while ignoring superficial differences.

Once the state representation is obtained, the next stage is action detection. Unlike classical RL environments where the action space is fixed and known, web applications present a different action space in every state. Furthermore, modern web applications allow any DOM element to be interactive through JavaScript, making it impossible to rely solely on HTML tags. To address this, the system uses a combination of heuristic rules and a learned action discriminator. The heuristic rules provide an initial set of obvious actions, while the discriminator continuously learns from interaction feedback to identify additional actionable elements. This ensures that the agent does not remain blind to hidden interactions and can progressively expand its action space as exploration proceeds.

After identifying potential actions, the pipeline moves to exploration guidance using grid-based Deep Q-Learning. Instead of learning a value for each individual action, which would suffer from action misalignment across pages, the page is divided into a fixed grid and the DQN learns values for grid cells. This design allows the agent to learn spatial exploration patterns such as which regions of the page tend to lead to new states. By decoupling value learning from specific DOM elements, the agent can generalize knowledge across pages with similar layouts, even when the exact actions differ.

Finally, reward computation is handled using a curiosity-driven model that combines episodic and global novelty. Episodic novelty ensures that within a single episode, the agent is encouraged to continuously move toward new states rather than oscillating between a small set of pages. Global novelty ensures that the agent does not repeatedly explore the same states across episodes, preventing reward collapse over long training runs. Together, these reward signals provide consistent guidance throughout training, even when the number of genuinely new pages becomes limited.

3.2 Learning Loop

The learning loop describes how all components of the system interact during actual execution, and understanding this loop is essential for implementing and debugging the system.

At each interaction step, the agent first observes the current web page and converts it into a semantic embedding using the state representation module. This embedding serves as the agent's perception of the environment and is used as input to both the value-learning and reward models.

Next, the system identifies actionable elements on the page. This is done by combining actions detected through heuristic rules with actions predicted by the trained action discriminator. Importantly, this step is dynamic: the set of available actions can change from state to state and can grow over time as the discriminator improves.

Once the set of candidate actions is determined, the grid-based DQN predicts values for each grid cell of the page. These cell values represent how promising different regions of the interface are in terms of leading to novel states. Since actions are associated with specific screen locations, the system then upsamples cell values to compute action values, assigning higher value to actions located in high-value regions.

An action is then selected using an ϵ-greedy strategy. With probability ϵ, the agent explores by choosing a random action, ensuring continued discovery of new behaviors. With probability $1-\epsilon$, the agent exploits its learned knowledge by selecting the action with the highest predicted value.

After the selected action is executed in the environment, the agent observes the resulting page and computes a novelty-based reward. The episodic novelty component measures how different the new state is from states visited earlier in the same episode, while the global novelty component measures how unfamiliar the state is across the entire training history using the autoencoder's reconstruction error.

Finally, the agent performs a learning update. The DQN is updated using the observed reward and the estimated value of the next state, and auxiliary models such as the action discriminator and global novelty model are also updated. This completes one iteration of the learning loop, which is repeated until training converges or exploration coverage saturates.

3.3 Expected Outcomes

By following this architecture and learning loop, the system is expected to achieve several practical benefits that are difficult to obtain using traditional testing approaches.

First, the agent is able to efficiently explore large and complex web applications without requiring any prior knowledge of the application structure or test cases. The grid-based value learning allows the agent to focus exploration on promising regions of the interface rather than wasting time on unproductive interactions.

Second, the combination of dynamic action discovery and curiosity-driven rewards enables the agent to discover hidden states and interactions that are often missed by rule-based or random testing tools. This includes pages that are only reachable through non-obvious interactions or sequences of actions.

Finally, by systematically maximizing state coverage, the approach significantly improves bug detection and behavioral coverage. Since many defects only appear in rarely visited states, the ability to reach deep and unusual parts of the application increases the likelihood of uncovering subtle bugs and inconsistencies.

4 Conclusion

This report presents a detailed understanding of reinforcement learning fundamentals and demonstrates how these concepts can be adapted to the challenging domain of automated web application testing. Rather than directly applying standard RL algorithms, the WebRLED approach rethinks core components such as states, actions, and rewards to better align with the structure and dynamics of web interfaces.

By adopting a tester's perspective, designing robust state abstractions, learning actions dynamically, and guiding exploration using curiosity-driven rewards, the system overcomes many of the limitations that prevent traditional reinforcement learning methods from scaling to real-world web applications. As a result, WebRLED illustrates how reinforcement learning can move beyond controlled benchmarks and become a practical tool for large-scale, real-world software testing.