IJICTR
International Journal of Information & Communication Technology Research

Volume 16- Number 2 – 2024 (25 -33)

Deep Reinforcement Learning-based Exploration of Web Applications

Mohammadreza Abbasnezhad
Department of Computer Engineering
Yazd University
Yazd, Iran.
abbasnezhad.m.r@stu.yazd.ac.ir

Amir Jahangard Rafsanjani*
Department of Computer Engineering
Yazd University
Yazd, Iran.
jahangard@yazd.ac.ir

Amin Milani Fard
Department of Computer Science
New York Inst. of Technology
Vancouver, BC, Canada.
amilanif@nyit.edu

Received: 15 May 2023 – Revised: 16 November 2023 - Accepted: 13 March 2024

Abstract—**Web application (app) exploration is a crucial part of various analysis and testing techniques. However, the current methods are not able to properly explore the state space of web apps. As a result, techniques must be developed to guide the exploration in order to get acceptable functionality coverage for web apps. Reinforcement Learning (RL) is a machine learning method in which the best way to do a task is learned through trial and error, with the help of positive or negative rewards, instead of direct supervision. Deep RL is a recent expansion of RL that makes use of neural networks' learning capabilities. This feature makes Deep RL suitable for exploring the complex state space of web apps. However, current methods provide fundamental RL. In this research, we offer DeepEx, a Deep RL-based exploration strategy for systematically exploring web apps. Empirically evaluated on seven open-source web apps, DeepEx demonstrated a 17% improvement in code coverage and a 16% enhancement in navigational diversity over the state-of-the-art RL-based method. Additionally, it showed a 19% increase in structural diversity. These results confirm the superiority of Deep RL over traditional RL methods in web app exploration.**

Keywords: Deep Reinforcement Learning, Exploration, Model Generation, Web Application.

Article type: Research Article

I. INTRODUCTION

Web applications (apps) are an important part of our daily lives because they help us in many ways. A recent survey [1] found that there are over 1 billion web apps. Web apps are developed using various technologies and multiple programming languages, such as JavaScript, HTML, CSS, and PHP. In an event-driven architecture, the structure of the web apps changes in response to events, such as clicks. In other words, web apps frequently use JavaScript to modify the Document Object Model (DOM) [2] dynamically. That gives web apps different states and gives users better response and interaction.

Numerous web app analysis, understanding, and testing methods rely on automated black-box exploration of web apps [3]–[8]. Exploration of web apps can reduce the complexities associated with analyzing the complex source code of web apps. In other words, the exploration methodologies exercise the User Interface (UI) elements to explore the state space of a web app. To create a behavior model of the web

* Corresponding Author

app, events are triggered on web elements automatically, and possible UI state changes are tracked during exploration. This inferred model, represented by a State Flow Graph (SFG) [9], is subsequently utilized for a variety of purposes. The efficacy of web app analysis and testing activities is highly dependent on the efficacy of the SFG developed during web app exploration.

To adequately cover its state space [10], there are many web app exploration methodologies. The goal of generic exploration methods, like Crawljax [9], is to fully explore the state space of a web app. The state explosion problem [11] is a drawback to exhaustive exploration. In reality, the majority of web apps have such a vast state space that exhaustive exploration is impossible. In addition, generic techniques have a tendency to become mired in unimportant parts of web apps, which leads to inadequate coverage of the app's functionality because there is no feedback to direct the exploration.

Guided exploration is an alternative to generic exploration that helps alleviate the state explosion problem in web apps by deriving a partial SFG through directing exploration towards areas of interest to achieve adequate coverage of the app's functionalities. FeedEx [12] utilizes parameters to take into account many facets of the exploration. It supervises and directs the exploration at runtime using the parameters. Similarly, Keyjaxtest [13] explores a web app to derive a partial SFG by employing specific keywords that characterize particular app functionalities. Familiarity with the web app is necessary for guided techniques. For instance, if the user is not familiar with the required phrases, Keyjaxtest might not accurately explore the functionality of the web apps. When presented with a new web app, this constraint renders the guided exploration strategy ineffective. The fact that dynamic exploration approaches automatically examine new apps is also one of their main benefits [14].

Recent studies [15] on Reinforcement Learning (RL) [16] have demonstrated that it is capable of learning a policy to explore web apps. RL is a machine learning method that learns from positive or negative task rewards without a labeled training set. Therefore, it represents a method for dynamically constructing an appropriate exploration strategy based on past successes or failures. Even though RL has been used to solve the problem of exploring web apps [15], so far only the most basic type of RL, tabular RL, has been used to explore web apps. Tabular RL maintains a table of state-action values. Deep neural networks replaced tabular methods with Deep learning methods, in which the action-value function is learned from a neural network's past good and bad attempts. When the state space is huge (such as when there are many events and states within a web app), deep RL has proven to be much superior to tabular RL [17], [18]. In this way, we say that the state space of web apps is a great place to use Deep RL instead of tabular RL for successful exploration.

DeepEx (**Deep** Reinforcement Learning-based **Ex**ploration of Web Apps), the first Deep RL solution for automated web app exploration, is presented in this article. DeepEx uses a Deep neural network to figure out the best way to explore by looking at what has already been tried. Due to the use of a Deep neural network, the system is both highly scalable and capable of managing the complex functionalities of web apps. DeepEx was used to evaluate a benchmark of seven different web apps. In the benchmark, DeepEx's performance was compared to that of the other web app exploration techniques like QExplore [15] and FeedEx [12]. The experimental findings supported the claim that Deep RL beats tabular RL in the exploration of web apps, with deepEx obtaining better code coverage with more navigational and structural diversity.

The following is a summary of this paper's contributions:

- The first exploration strategy built on Deep RL that we suggest is called DeepEx.
- We give an empirical assessment of the proposed method. Our approach outperforms existing ones, according to the results.

The rest of this paper is structured as follows: Introductions to web app exploration and Deep RL are provided in Section 2. Section 3 reviews related work. In Section 4, we will discuss our exploration strategy, which is based on Deep RL. The fifth section provides an empirical evaluation of our proposed methodology for seven web apps. The paper concludes in Section 6, which also provides ideas for additional research.

II. BACKGROUND

This part gives background information on web app exploration in order to make a model of how it works. In addition to this, it explains the fundamental principles of Deep RL, which are necessary to comprehend the rest of the work.

A. Behavioral Model

Web apps that offer better user interaction are now widely available [1] thanks to the development of web and browser technology. In order to alter the UI in reaction to runtime events, web apps modify the DOM [2]. During runtime, these incremental modifications lead to dynamically produced states. As a result, DOM that is created dynamically can serve as a representation of a UI state, and a state transition can be described as a change in DOM. To model these UI state transitions in a web app, the following SFG [9] is defined, where nodes represent the dynamic DOM states of the web app and edges represent the event-based transitions between them:

State-flow graph SFG for a web app is a labeled directed graph with the notation $\langle r, V, E \rangle$:

- r is the root node and represents the original state of the web app after it has been fully loaded into the browser.
- V is a collection of vertices that represent the states. Each $v \in V$ represents a runtime DOM state in the web app.
- E, we refer to the set of directed, labeled edges between vertices as events. Each $(v_1, v_2)_e \in E$ indicates a change between two nodes v_1, v_2 if and only if the event e in v_1 leads to v_2.
- SFG may contain multi-edges and be cyclic.

By creating events, such as clicking on DOM components, users can communicate with a web app. A DOM element is referred to as a clickable element if it has an attached event listener or if it is clickable in general, like element $< a >$. The actions (such as clicking on clickables) can activate the associated event handler functionality and ultimately change the state of the web app. Therefore, by investigating event-driven DOM transitions in web apps, the SFG can be automatically derived.

Since an SFG captures dynamic UI states and event-based transitions between them, it is assumed to be the web app's behavioral model. Event sequences in the SFG typically exercise a web app's functionality. Therefore, the SFG has a wide variety of uses in web app analysis, comprehension, and testing. For instance, testers can automatically generate test cases by extracting event sequences from the SFG. Generally speaking, web app exploration can help with activities such as invariant-based testing [3], cross-browser compatibility testing [4], mutation testing [5], automated test case generation [6], model-based testing [7], test dependency analysis [8].

B. Deep RL

A model-free RL technique called Q-learning [19] aims to learn a policy for any Markov decision process by identifying the best possible policy, π, to maximize the expected cumulative reward for a series of actions. Q-learning is based on trial-and-error learning, in which an agent interacts with the environment and assigns Q values, which are approximated values, to each state-action pair.

As depicted in Fig. 1, the agent interacts iteratively with the environment. Assuming S and A are the sets of all states and actions, at each iteration t, the agent selects and executes an action $a_t \in A$ based on the current state $s_t \in S$. s_t and a_t represents the state and action at time t, respectively. After performing the action, the agent can observe a new state $s_{t+1} \in S$. In the meantime, an instant reward $r_t = R(s_t, a_t)$ is received. This is the immediate reward for doing action a_t in state s_t. The agent will then use the Bellman equation [20] to update the Q value, as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (r_t + \gamma * \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (1)$$

α is a learning rate between 0 and 1 and γ is a discount factor between 0 and 1 in this equation. After being learned, these Q values can determine the optimal behavior in each state by selecting the action $a_t = \arg\max_{a_t} Q(s_t, a_t)$.

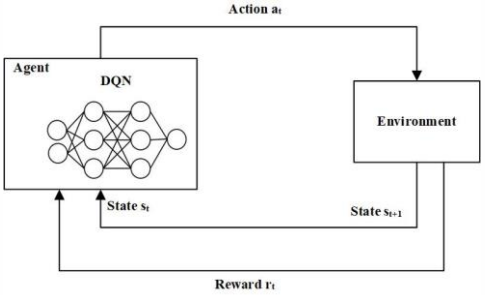

Figure 1. Deep RL overview

Deep Q-Networks (DQN) are used to scale traditional Q-learning to larger state and action spaces [17], [18]. $Q(s_t, a_t)$ are stored and visited in a Q-table for traditional Q-learning. It can only manage state and action spaces with low dimensions. As shown in Fig. 1, DQN is a multi-layered neural network that outputs Q values for each action a_t in a given state s_t, i.e., $Q(s_t, a_t)$. DQN can scale more complicated state and action spaces because a neural network can input and output high-dimensional state and action spaces. In contrast to a Q-table, a neural network can generalize Q values to previously unobserved states. It employs the following loss function [17], [18] to modify the neural network in order to reduce the error:

$$loss = \left(r_t + \gamma * \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right)^2 \quad (2)$$

In other words, the neural network is trained to predict the value of Q as follows, given the input (s_t, a_t):

$$Q(s_t, a_t) = r_t + \gamma * \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \quad (3)$$

In a training sample, therefore, the input is (s_t, a_t) and the output is the corresponding Q value, which can be calculated as $r_t + \gamma * \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$.

III. RELATED WORK

In the research that has been done on the topic, various strategies for improving the efficiency of exploring web apps have been offered. We will briefly go through the current state-of-the-art solutions and how their shortcomings call for a new way of exploring web apps.

The state space of web apps is automatically explored, and an SFG is produced using generic exploration algorithms like Crawljax [9], which was proposed by Mesbah et al. In order to thoroughly explore the state space of web apps, Crawljax uses generic exploration algorithms such as breadth-first search, depth-first search, or random search. Due to the state explosion problem [11], however, such general exploration algorithms cannot fully explore the state space of web apps in a finite amount of time. Another disadvantage of Crawljax is that it can become stuck in unimportant areas of web apps, leading to insufficient functionality coverage. The primary reason for this is that Crawljax's method lacks feedback to guide the exploration.

There have been guided exploration approaches that employ heuristic strategies to direct the exploration of web apps towards areas of interest based on predetermined objectives. In actuality, guided approaches explore a web app by limiting the scope of exploration in order to derive an incomplete model with adequate functionality coverage, as opposed to a complete model. Milani Fard and Mesbah [12] came up with an exploration method called FeedEx that uses heuristics, such as code coverage, navigational diversity, and structural diversity, to lead the exploration. KeyjaxTest, suggested by Qi et al. [13], uses keywords of defined functionality to direct exploration towards attaining good coverage of them,

making it a keyword-guided technique for exploring the state space of a web app. KeyjaxTest guides the exploration to find states and transitions that are important to the defined functionalities by figuring out how similar the text in the states and the given keywords are.

Without a prior understanding of the web app, guided approaches are ineffective for exploring the state space. For instance, FeedEx's efficacy depends on the weights used to combine the parameters, which can differ between apps and are challenging to predict before exploration. Similarly, prior to exploration, KeyajxTest demands not only knowledge of probabilistic weights but also knowledge of the various functionalities accessible and their relevant keywords. To put it another way, KeyjaxTest needs the web app's desired functionalities described using keywords. It is difficult to predict those keywords if the user is unfamiliar with the web app. However, one of the primary expectations from dynamic exploration approaches is that they explore new apps automatically [14].

Liu et al.'s GUIDE [14] is a guided approach that allows the user to provide directives (such as stopping the exploration of particular states) incrementally. The user gives GUIDE more and more instructions over time to explore more and more states and functionalities. More states are expected to be examined when more directives are employed. As a result, this solution needs the use of a human agent and manual effort, making it unsuitable for web apps.

Similarly, research has investigated the use of RL for web app exploration. QExplore [15] proposed by Sherin et al. utilizes RL, allowing it to anticipate and develop the behavioral model incrementally while interacting with the web app. QExplore employs Q-learning [19], a model-free RL method, based on curiosity reward to accomplish exploration. Similarly, WebExplor [21] by Zheng et al. is another existing work that is most relevant to both the RL and web domain since it uses RL to generate test cases incrementally while interacting with a web app. Both QExplore and WebExplor, in contrast to our study, are based on the most basic type of RL, tabular RL. In contrast, DeepEx learns the action-value function based on Deep RL during its interaction with the web app. To the best of our knowledge, DeepEx is the first Deep RL-based approach that explore web apps and outperforms state-of-the-art methods in terms of effectiveness.

IV. Proposed Approach

This section covers DeepEx (**Deep** Reinforcement Learning-based **Ex**ploration of Web Applications**)**, our proposed Deep RL-based approach to exploring web apps. In Fig. 2, we can see the main building blocks of the proposed approach, which are Browser, DOM Analyzer, DQN, Action Selector, Calculator, and Memory.

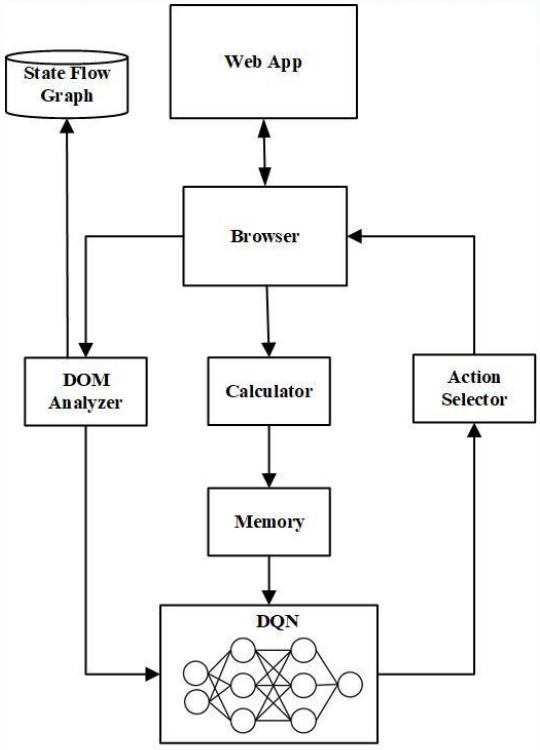

Figure 2. Fig. 2: Proposed approach Overview

It is the responsibility of Browser to provide a common interface for communicating with the web app. It has access to runtime DOMs and the JavaScript engine. Additionally, Browser executes the actions in web app states. DOM Analyzer parses the DOM tree and extracts the state and actions associated with it. The current state and actions are converted into an input by DOM Analyzer, which is subsequently sent into the DQN. DQN receives the web app's state and its actions. DQN uses a model of a neural network to figure out Q values for actions which it then sends to Action Selector. The next action to execute is selected by Action Selector based on an Epsilon-Greedy policy [16]. The browser executes the selected action. The web app enters a new state. DOM Analyzer monitors performed actions and resulted states to construct the SFG incrementally as an output. Using equation (3), Calculator computes the transition reward and obtains the Q value. The transition is stored in Memory along with the state, action, and Q value. DQN learns from a sampling batch of transitions in Memory to update its weights. DQN would learn poorly if it merely used sequential samples of experience from the environment because of their correlation [17], [18]. This correlation is broken by sampling Memory at random.

A. Problem Formulation

To use Deep RL, we must first convert the web app exploration problem to the conventional mathematical formalization of RL. The web app exploration problem can be formalized formally as a Markov decision process, which can be demonstrated by a 4-tuple, $\langle S, A, P, R \rangle$. These are described below.

1) ***S***: *States*

Our approach is black-box because it does not access the App Under Exploration (AUE) source code. It only uses the UI of AUE. DeepEx extracts the DOM from the web app's current UI. DeepEx analyzes the DOM to find clickable elements in the current state. State s_t is represented by $(c_1, c_2, \dots, c_n)$, where c_i are the clickable elements in s_t. Each c_i is an index that indicates the element's position in the DOM tree's pre-order traversal.

Fig. 3 illustrates the partial DOM trees of two pages, DOM 1 and DOM 2, as an example. They are both made up of elements $\langle body\rangle, \langle div\rangle, \langle p\rangle$ and $\langle a\rangle$. The elements that can be clicked (only elements $\langle a\rangle$) are highlighted.

DOM 1's pre-order traversal is $(\langle body\rangle, \langle div\rangle, \langle a\rangle, \langle a\rangle, \langle div\rangle, \langle a\rangle, \langle p\rangle)$, and DOM 2's is $(\langle body\rangle, \langle div\rangle, \langle p\rangle, \langle p\rangle, \langle div\rangle, \langle a\rangle, \langle a\rangle)$. For simplicity, we replace elements that can be clicked with 1 and those that cannot with 0. They are transformed into $(0,0,1,1,0,1,0)$ and $(0,0,0,0,0,1,1)$. To acquire s_1 and s_2, the respective states of DOM 1 and DOM 2, we must take into account the positions of clickable elements, i.e., the positions of number 1. As a result, $s_1 = (2,3,5)$ and $s_2 = (5,6)$.

2) ***A***: *Actions*

Clickable elements indicate actions. In other words, clickables and click events in web apps are formulated as actions in the Markov decision process. Actions are represented by the index of the clickables in the relevant state, which is similar to states. In s_1, for example, there are three actions marked by $A_1 = (2,3,5)$. In the same way, $A_2 = (5,6)$ shows that s_2 has two actions. In this paper, we don't make a difference between actions and events, because they are the same. In web apps, clickables suffice to complete the majority of tasks.

3) ***P***: *Transition Function*

The transition function indicates the state the web app will enter once an event occurs. We have no control over it; the AUE decides what it is.

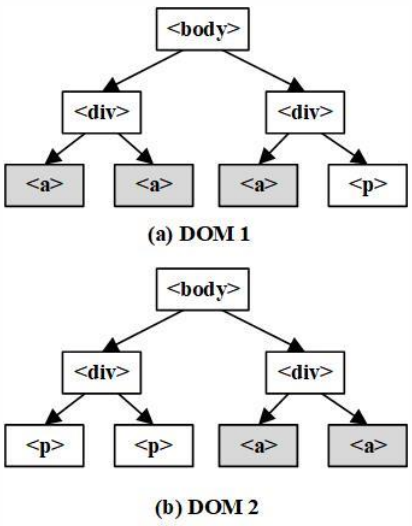

Figure 3. Partial DOM trees

4) ***R***: *Reward*

When DeepEx executes an event, it receives a reward. We present a mechanism for determining the reward that complements our exploration approach. The reward function gives a bigger reward to actions that change the state a lot. This is a heuristic way to understand which actions lead to new functionalities. The intuition is to provide greater rewards for actions that can result in multiple new actions. In fact, a state with more new clickables is more likely to result in additional states and functionalities. The reward function is defined by the following equation:

$$r_t = R(s_t, a_t, s_{t+1}) = \frac{|s_{t+1} - s_t|}{|s_{t+1}|} \quad (4)$$

The reward function, given two states, s_t and s_{t+1}, estimates the degree of change from s_t to s_{t+1} by comparing and detecting the number of clickables in s_{t+1} that were not present in s_t, which is described as $|s_{t+1} - s_t|$. The ratio $\frac{|s_{t+1} - s_t|}{|s_{t+1}|}$, where $|s_{t+1}|$ is the number of clickables in s_{t+1}, defines the relative change. This reward function takes into account the actions that are introduced in s_{t+1} but are absent in s_t.

As an illustration, given $s_1 = (2,3,5)$ and $s_2 = (5,6)$, as previously defined, $\frac{|(6)|}{|(5,6)|} = \frac{1}{2} = 0.5$ is the reward of the transition from s_1 to s_2. In fact, the clickable (6) is not in s_1, but in s_2, and there are only two clickables in s_2: $(5,6)$.

B. Algorithm

Algorithm 1 details the DeepEx approach for Deep RL-based exploration. It takes the AUE, the exploration time budget, and the maximum number of actions per episode as input. In fact, we need to turn the exploration problem into an RL task that is broken up into several episodes. A series of actions is referred to as an episode. In other words, each episode consists of multiple steps or iterations in which an action is conducted. DeepEx outputs state flow graph SFG. A memory is used to store samples from previous iterations, each of which comprises the state, action, and related Q value. DeepEx begins by initializing memory M (line 1) and the state flow graph SFG (line 2). Now, exploration starts and goes on until the time limit is met (lines 3–20). DeepEx restarts the web app and navigates to the homepage (line 4). Line 5 returns the AUE's initial state. Line 6 adds the initial sate to the SFG. In each episode, we limit the number of steps that can be taken (lines 7–20). The default setting for the episode length in DeepEx is 25, but it can be changed. Each episode starts with an action in the initial state.

Algorithm 1 DeepEx for Deep RL based exploration

Require: App under exploration AUE, the time budget for exploration, the length of each episode
Ensure: State flow graph SFG

1: $M \leftarrow \emptyset$
2: $SFG \leftarrow \emptyset$
3: **while** time budget not completed **do**
4: reset(AUT)
5: $s_t \leftarrow$ getState(AUT)
6: $addInitialState(SFG, s_t)$
7: **while** the episode not completed **do**
8: **if** A random number $\in [0, 1] < \epsilon$ **then**
9: $a_t \leftarrow$ getRandomAction(s_t)
10: **else**
11: $a_t \leftarrow$ getBestAction(s_t, DQN)
12: execute(a_t, AUT)
13: $s_{t+1} \leftarrow$ getState(AUT)
14: $r_t \leftarrow$ getReward(s_t, s_{t+1})
15: $Q(s_t, a_t) = r_t + \gamma \times \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ where $\gamma = 0.9$
16: $batch \leftarrow$ getMiniBatch(M) $\cup \{(s_t, a_t, Q(s_t, a_t))\}$
17: $DQN \leftarrow$ updateModel($batch, DQN$)
18: $M \leftarrow M \cup \{(s_t, a_t, Q(s_t, a_t))\}$
19: $Update(SFG, s_t, a_t, s_{t+1})$
20: $s_t \leftarrow s_{t+1}$
return SFG

Epsilon-Greedy is the policy that DeepEx employs (lines 8–11). It decides based on a predefined threshold ε in the interval $[0, 1]$ to determine whether it will explore new actions or exploit its existing knowledge. In fact, it chooses the action with the highest Q value based on the DQN (exploitation, Line 11) with a chance of $1 - \varepsilon$ and a random action (exploration, Line 9) with a chance of ε. Randomness is required for an agent to discover the optimal strategy [22]. We want the DeepEx to explore as various states as possible at the start of the testing in order to explore new actions more; thus, a high value of ε should be used. DeepEx is therefore expected to follow the Q values in order to exploit its knowledge, so a smaller value of ε is expected. By default, DeepEx starts with 1 to enable maximum exploration, then decreases its value uniformly during the first 30 episodes until a final minimum value of 0.2 transforming its behavior towards exploitation.

DeepEx executes the selected action (line 12). In fact, DeepEx generates the SFG by executing the appropriate action in the current state of AUE. DeepEx retrieves the new state (line 13) and computes the reward (line 14) using equation (4) based on s_t, s_{t+1}. DeepEx uses equation (3) to calculate the Q value of action a_t in s_t with parameters s_t, s_{t+1}, a_t, and r_t (line 15). The discount factor, γ, balances how important the immediate reward is compared to future actions, and a number of 0.9 maximizes the reward earned over the whole episode, not just the immediate reward.

DeepEx employs a set of random training samples, including both the current sample and historical samples (line 16), to train the neural network (line 17). Each sample takes (s_t, a_t) as input and has $Q(s_t, a_t)$ as output. The current transition is saved in memory M, which stores historical samples from previous iterations (line 18). The SFG is updated by the new transition $(s_t, s_{t+1})_{a_t} \in E$ (line 19). The prior state is then updated to continue exploration (line 20).

V. EVALUATION

In this part, we show experiments to test how well our Deep RL method for exploring web apps works. In other words, the effectiveness of DeepEx in the exploration of web apps in comparison to other methods that are considered to be state-of-the-art in web app exploration is our primary research question.

A. Metrics

Code coverage, navigational diversity, and structural diversity are three metrics that we use to evaluate the success of our method. According to [12], [13], [15], it is thought that these metrics represent the features that an SFG ought to have in order to cover many parts of the behavior of a web app in an efficient manner.

1) Code coverage

When evaluating the effectiveness of exploration, code coverage is a useful metric to consider. One goal of exploring web apps is to run enough code to adequately cover the app's functionalities. Code coverage is a metric that determines the proportion of an app's lines of code that are successfully run while the app is being explored. Code coverage is also a reliable indicator of test robustness [23]. Similar to [12], [13], [15], each web app was instrumented to acquire code coverage.

2) Navigational diversity

A web app's navigation structure enables users to move around it in different ways. Each navigational path offers a varying level of functionality. A behavioral model should adequately encompass the web app's navigational structure. The model should cover web app navigational branches to do that. The position of the leaf nodes in the graph is an indication of the diversity of its event paths (that is, paths from the index node to the leaves). In order to measure the navigational diversity of an SFG, we measure the average pair-wise navigational diversity of leaf nodes (states without any outgoing edges). This is similar to [12], [13], [15]. Common and uncommon events in SFG routes determine their diversity.

3) Structural diversity

A webpage's DOM structure serves as the primary interface for user interaction. Different DOMs offer varying levels of functionality. Because of this, directing the exploration toward a variety of DOM states can lead to improved coverage of the web app. In order to accurately represent this structural diversity, an SFG should include the various DOM structures of the web app. In the same way as [12], [13], [15], we use the average pair-wise structural diversity of DOM states in the derived SFG to measure the structural diversity. The normalized DOM tree edit distance can be used to define state DOM diversity. Similar to [12], [13], [15], we employ the tree edit distance between two ordered labeled trees, which was proposed [24] and implemented [25] as the minimum cost of a series of edit operations that converts one tree into another. The operations consist of deleting a node and connecting its children to the parent, inserting a node between a node and its children, and renaming a node.

B. Setup

DeepEx was implemented in Python on top of QExplore [15] to assess its effectiveness. QExplore supports the RL approach. We changed the RL strategy by replacing it with our Deep RL algorithm. To interact with the web app, Selenium [26] was utilized. The DQN was built and executed using Keras [27]. DQN uses a 3-layer fully connected neural network, and Adam to

optimize the model with a learning rate of 0.001. Two state-of-the-art strategies were chosen for a comparison study. These include QExplore [15] and FeedEx [12]. It is worth noting that WebExplor [21] focuses on generating test cases and uses tabular RL as a foundation. Our research focuses on web app exploration using Deep RL, which has never been done previously. Despite having some similarities with QExplore, WebExplor is not a behavioral model generator. So, it is not possible to compare DeepEx directly to WebExplor. Extending our deep RL technique to develop test cases for web apps and comparing its effectiveness with that of WebExplor is an interesting piece of future work that can be done.

Similar to [15], we utilize an existing open-source data generator for elements requiring input data (e.g., text fields) based on the context of the elements. In fact, we employ Mocker Data Generator [28], which offers input data for web app input fields. DeepEx also gives you the option to manually enter data for some inputs, such as login and password.

We chose seven open-source web apps to evaluate. These web apps perform various tasks and belong to several categories. A web app called Voting [29] enables communities and groups to vote online. E-commerce Site [30] is a marketplace where users may purchase and sell products to one another. A web app called Hostel [31] handles important operations connected to running a hostel. NodeBB [32] is an online forum. Keystone [33] is a content management system. TimeOff [34] is an employee absence management system. Petclinic [35] is a web app used to manage a veterinary clinic. These web apps were selected for their varying levels of complexity and user interaction patterns, providing a robust testbed for our Deep RL testing approach. Each web app features distinct navigational structures and user interfaces. By choosing this web apps, we aimed to demonstrate the versatility and adaptability of our approach across different web app architectures. The varied nature of these web apps significantly contributed to assessing the approach's performance.

Each approach was tested on each subject web app. We gave each strategy the same 100-minute time limit. In addition, we repeated each experiment three times and calculated the average of all the results to confirm the general trend. The experiments were conducted on a PC running Windows 10, with a processor of an Intel Core i7-13700K 3.40 GHz and memory RAM 31.7 GB. It is noteworthy that the required parameters of the approaches QExplore and FeedEx were set according to the recommended defaults in their papers.

C. Results

The effectiveness of DeepEx, QExplore, and FeedEx in terms of code coverage, navigational diversity, and structural diversity are compared in Table 1. The table details the average values obtained from three iterations of each of the three methodologies within the time constraint of ninety minutes. In this case, the highest values are bold.

It is clear from looking at the code coverage column that DeepEx outperforms both QExplore and FeedEx. It shows an improvement of 17% and 43% when compared to them, respectively. DeepEx obtained greater code coverage than the other two approaches in each of the web apps under consideration.

The results show that DeepEx outperformed QExplore and FeedEx in terms of navigational diversity across all of the investigated web apps. DeepEx improved navigational diversity by 16% compared to QExplore. Similarly, DeepEx improved navigational diversity by 45% when compared to FeedEx.

According to the structural diversity column, it is clear that the SFG obtained through DeepEx has more structural diversity in all subject web apps than those received through QExplore and FeedEx. DeepEx scored better than QExplore and FeedEx when it came to capturing structural diversity in its SFG, with an improvement of 19% and 49%, respectively.

When comparing DeepEx's improvements in code coverage, navigational diversity, and structural diversity to those of RL and heuristic-based approaches, it is clear that the Deep RL-based methodology is more effective in the exploration of web apps. In other words, one of the primary reasons for the improved results obtained by DeepEx is that it systematically explores the web app by directing exploration toward more effective actions and gaining access to various states based on the learning capabilities of Deep RL.

Web apps, known for their sophistication and diverse user interactions, often pose challenges in exploration. In our investigation of the exploration capabilities of DeepEx and two alternative approaches, we delved into the complexities arising from intricate user interactions. Notably, web app exploration involves identifying patterns—specific sequences of actions required to transition between states. This adds a layer of difficulty as certain functionalities are only revealed through precise sequences of actions.

For instance, in the Petclinic subject web app , a crucial functionality, such as "Adding new visit", necessitates a specific sequence: clicking "Find Owners", typing the owner's name, clicking "Find Owner", clicking "Add New Visit", filling details, and clicking "Add Visit". Interestingly, none of the other approaches could detect this functionality. DeepEx, leveraging Deep RL guidance, successfully identified such action sequences not only in Petclinic but also in other web apps. It's crucial to note that any interruption in the process results in redirection to another page, impacting exploration efficacy. Deep RL's ability to efficiently execute these sequences sets it apart.

Our findings underscore the superiority of the Deep RL algorithm over RL and heuristic-based approaches. DeepEx, powered by Deep RL, outperformed other methods in replicating human behaviors. This was particularly evident in generating action sequences without distractions from prior states or ineffective actions in high-dimensional spaces. The learning capabilities of the DQN used in Deep RL facilitated the efficient production of these behaviors—a feat more challenging for the RL algorithm with its limited adaptation capabilities. Our paper shows Deep RL's effectiveness in learning exploration strategies contributes to its superior performance in uncovering intricate functionalities within web apps.

TABLE I. EFFECTIVENESS RESULTS FOR COMPARISON

App	Code coverage			Navigational diversity			Structural diversity		
	DeepEx	*QExplore*	*FeedEx*	*DeepEx*	*QExplore*	*FeedEx*	*DeepEx*	*QExplore*	*FeedEx*
Voting	**76.47**	63.50	49.14	**0.86**	0.74	0.53	**0.65**	0.53	0.42
E-commerce Site	**86.37**	74.27	66.40	**0.84**	0.71	0.59	**0.70**	0.62	0.47
Hostel	**72.87**	60.39	49.41	**0.79**	0.68	0.56	**0.73**	0.64	0.50
NodeBB	**66.70**	54.65	43.67	**0.72**	0.63	0.49	**0.59**	0.49	0.41
Keystone	**56.62**	48.59	40.55	**0.63**	0.61	0.53	**0.68**	0.57	0.47
TimeOff	**78.62**	69.59	59.55	**0.71**	0.59	0.51	**0.58**	0.47	0.39
Petclinic	**64.67**	56.54	42.52	**0.66**	0.54	0.39	**0.53**	0.43	0.34
Average	**71.76**	61.08	50.18	**0.74**	0.64	0.51	**0.64**	0.54	0.43

VI. CONCLUSIONS

In this paper, we have proposed DeepEx, an approach based on Deep RL for the exploration of web apps. A Deep Q-network agent is employed in this approach to explore and model the web app through trial and error. Instead of relying on heuristic principles to find the appropriate action to discover new states, DeepEx can learn how to explore web apps on its own. We have tested DeepEx on seven publicly available web apps and found that it achieves better results than the current methods for web app exploration in terms of code coverage, navigational diversity, and structural diversity.

In future work, we plan to extend the testing of our methodology to a wider array of web apps. It is important to note that the use of apps from various categories, as well as the consistency of the results in this study, indicate that our Deep RL-based technique has some potential usefulness. Despite this, expanding our subject web apps will allow for a more comprehensive evaluation to further validate the effectiveness of DeepEx.

It is crucial to remember that our basic and intuitive definition of states and reward function produced encouraging results, indicating its usefulness. However, we aim to enhance DeepEx by refining the state space and exploring different reward function strategies within Deep Q-networks. These enhancements are directed towards developing more complex state definitions and reward functions, particularly to improve adaptability to a more diverse range of web apps.

Furthermore, we will investigate the efficacy of testing and analyzing methods for web apps using the behavioral model created by DeepEx. This includes examining the efficacy of test suites for web app regression testing derived from the corresponding state flow graph.

REFERENCES

[1] "January 2023 Web Server Survey | Netcraft News." https://news.netcraft.com/archives/2023/01/27/january-2023-web-server-survey.html (accessed Apr. 05, 2023).

[2] "What is the Document Object Model?" https://www.w3.org/TR/WD-DOM/introduction.html (accessed Jan. 05, 2023).

[3] A. Mesbah, A. van Deursen, and D. Roest, "Invariant-Based Automatic Testing of Modern Web Applications," *IEEE Trans. Softw. Eng.*, vol. 38, no. 1, pp. 35–53, Jan. 2012, doi: 10.1109/TSE.2011.28.

[4] A. Mesbah and M. R. Prasad, "Automated Cross-Browser Compatibility Testing," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 561–570, doi: 10.1145/1985793.1985870.

[5] S. Mirshokraie, A. Mesbah, and K. Pattabiraman, "Guided Mutation Testing for JavaScript Web Applications," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 429–444, May 2015, doi: 10.1109/TSE.2014.2371458.

[6] A. Milani Fard, M. Mirzaaghaei, and A. Mesbah, "Leveraging Existing Tests in Automated Test Generation for Web Applications," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, 2014, pp. 67–78, doi: 10.1145/2642937.2642991.

[7] M. Biagiola, A. Stocco, F. Ricca, and P. Tonella, "Diversity-based Web Test Generation," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 142–153, doi: 10.1145/3338906.3338970.

[8] M. Biagiola, A. Stocco, F. Ricca, and P. Tonella, "Dependency-Aware Web Test Generation," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, Oct. 2020, pp. 175–185, doi: 10.1109/ICST46399.2020.00027.

[9] A. Mesbah, A. van Deursen, and S. Lenselink, "Crawling Ajax-Based Web Applications through Dynamic Analysis of User Interface State Changes," *ACM Trans. Web*, vol. 6, no. 1, Mar. 2012, doi: 10.1145/2109205.2109208.

[10] M. Mirzaaghaei and A. Mesbah, "DOM-Based Test Adequacy Criteria for Web Applications," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 71–81, doi: 10.1145/2610384.2610406.

[11] A. van Deursen, A. Mesbah, and A. Nederlof, "Crawl-based analysis of web applications: Prospects and challenges," *Sci. Comput. Program.*, vol. 97, pp. 173–180, 2015, doi: https://doi.org/10.1016/j.scico.2014.09.005.

[12] A. M. Fard and A. Mesbah, "Feedback-directed exploration of web applications to derive test models," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, Nov. 2013, pp. 278–287, doi: 10.1109/ISSRE.2013.6698880.

[13] X.-F. Qi, Y.-L. Hua, P. Wang, and Z.-Y. Wang, "Leveraging keyword-guided exploration to build test models for web applications," *Inf. Softw. Technol.*, vol. 111, pp. 110–119, 2019, doi: https://doi.org/10.1016/j.infsof.2019.03.016.

[14] C.-H. Liu, W.-K. Chen, and C.-C. Sun, "GUIDE: an interactive and incremental approach for crawling Web applications," *J. Supercomput.*, Mar. 2018, doi: 10.1007/s11227-018-2335-4.

[15] S. Sherin, A. Muqeet, M. U. Khan, and M. Z. Iqbal, "QExplore: An exploration strategy for dynamic web applications using guided search," *J. Syst. Softw.*, p. 111512, 2022, doi: https://doi.org/10.1016/j.jss.2022.111512.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[17] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5, 2013, [Online]. Available: http://arxiv.org/abs/1312.5602.

[18] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017, doi: 10.1109/MSP.2017.2743240.

[19] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, 1992, doi: 10.1007/BF00992698.

[20] R. Bellman, "On the Theory of Dynamic Programming," *Proc. Natl. Acad. Sci.*, vol. 38, no. 8, pp. 716–719, 1952, doi: 10.1073/pnas.38.8.716.

[21] Y. Zheng *et al.*, "Automatic Web Testing Using Curiosity-Driven Reinforcement Learning," in *Proceedings of the 43rd International Conference on Software Engineering*, 2021, pp. 423–435, doi: 10.1109/ICSE43902.2021.00048.

[22] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for Q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2016, pp. 1–8, doi: 10.1109/SSCI.2016.7849366.

[23] R. Gopinath, C. Jensen, and A. Groce, "Code Coverage for Suite Evaluation by Developers," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 72–82, doi: 10.1145/2568225.2568278.

[24] K.-C. Tai, "The Tree-to-Tree Correction Problem," *J. ACM*, vol. 26, no. 3, pp. 422–433, Jul. 1979, doi: 10.1145/322139.322143.

[25] M. Pawlik and N. Augsten, "RTED: A Robust Algorithm for the Tree Edit Distance," *Proc. VLDB Endow.*, vol. 5, no. 4, pp. 334–345, Dec. 2011, doi: 10.14778/2095686.2095692.

[26] "Selenium." https://www.selenium.dev/ (accessed Jan. 11, 2022).

[27] "Keras: Deep Learning for humans." https://keras.io/ (accessed Feb. 28, 2022).

[28] "mocker-data-generator: A simplified way to generate masive mock data based on a schema." https://github.com/danibram/mocker-data-generator (accessed Aug. 12, 2022).

[29] "Voting System." https://code-projects.org/voting-system-in-php-with-source-code/ (accessed Dec. 11, 2022).

[30] "E-commerce Site." https://code-projects.org/e-commerce-site-in-php-with-source-code/ (accessed Nov. 15, 2022).

[31] "Hostel Management System." https://code-projects.org/hostel-management-site-using-php-source-code/ (accessed Oct. 14, 2022).

[32] "NodeBB: Node.js based forum software built for the modern web." https://github.com/NodeBB/NodeBB (accessed Jul. 14, 2022).

[33] "keystone: The most powerful headless CMS." https://github.com/keystonejs/keystone (accessed Jun. 14, 2022).

[34] "timeoff: Simple yet powerful absence management software." https://github.com/timeoff-management/timeoff-management-application (accessed Apr. 16, 2022).

[35] "petclinic: Angular version of the Spring Petclinic Application." https://github.com/spring-petclinic/spring-petclinic-angular (accessed May 14, 2022).

Mohammadreza Abbasnezhad obtained his B.Sc. in Software Engineering from Vali-e-Asr University of Rafsanjan in 2014 and his M.Sc. in Software Engineering from Yazd University in 2017. He is currently pursuing his Ph.D. degree in Software Engineering at Yazd University. His research interests include AI-driven Software Engineering, Software Testing and Analysis.

Amir Jahangard-Rafsanjani received his B.Sc. degree in Software Engineering from Shahid Beheshti University, Tehran, Iran, in 2003, and his M.Sc. degree in Software Engineering from Sharif University of Technology, Tehran, Iran, in 2005. He also received his Ph.D. degree in Software Engineering from Sharif University of Technology in 2014. He is currently an Assistant Professor in the Department of Computer Engineering at Yazd University. His research interests include Database, Data and Text Mining, and Software Testing.

Amin Milani Fard recieved his Ph.D. degree in Software Engineering from the University of British Columbia, Canada, in 2017, and his M.Sc. degree in Computer Science from Simon Fraser University in 2010. Additionally, he received his B.Sc. degree in Computer Engineering from Ferdowsi University of Mashhad, Iran, in 2008. Currently, he serves as an Assistant Professor of Computer Science at the New York Institute of Technology's Vancouver campus in Canada and is a visiting faculty member in Management Information Systems at Simon Fraser University, Canada. His research focuses on Software Engineering and Analysis, Data Security and Privacy, Artificial Intelligence and Machine Learning.