Project Plan

Data Driven Innovation Challenge

Thomas Van der Molen

|  |  |
| --- | --- |
| **Project Information** | |
| Project members | Thomas van der Molen |
| Project Name | Data Driven Innovation Challenge |

Table of Contents

[Introduction 3](#_Toc150011794)

[Background 3](#_Toc150011795)

[Goal 3](#_Toc150011796)

[Feasibility 3](#_Toc150011797)

[Similar products 4](#_Toc150011798)

[The difference 4](#_Toc150011799)

[Downsides 4](#_Toc150011800)

[Malicious use 4](#_Toc150011801)

[Energy/Environment 4](#_Toc150011802)

[Unforeseen consequences 4](#_Toc150011803)

[First steps 5](#_Toc150011804)

# Introduction

For my Data Driven Innovation Challenge, I will be attempting to leverage the training process of Reinforcement learning models to perform automated integration tests for web-applications.

## Background

Integration tests, are tests that are performed on an entire application (web-application in this context). An integration test could be something like logging in as a user, this involves multiple steps with possible issues along the way, such as filling in the correct information, sending the login request to the authentication service, obtaining the authentication data and showing the logged in web-page.

These integration tests can become very cumbersome to be done by humans, tools have been created for this purpose to try and automate this process such as [Cypress](https://www.cypress.io/).

Using Cypress; automated integration tests can be written for a web-application, but this solution still has many downsides such as: The extra time to create the automated tests, the maintenance on these tests when possible interfaces or functionality changes and the fact that these tests have to be purpose built for every specific scenario for a given web-application.

All these caveats/down-sides of human and automated integration tests, cause them to not only cost a lot of time (and money) but to often not be done at all at the cost of application quality.

## Goal

The goal of this project is to utilize Reinforcement Learning to automatically test web-applications for anomalous behavior. By setting up proper RL environments and agents, the hope is that a RL agent can train and thereby test any web-application it is provided, no extra coding or time needed.

Another benefit of this approach is; an AI model will be able to not only target known weak points of an application, but can also explore new possible issues not previously encountered or even considered.

### Feasibility

The feasibility of this project can be proven by both research gathered from [Problem Analysis](https://ictresearchmethods.nl/Problem_analysis), [Literary Study](https://ictresearchmethods.nl/Literature_study), [Expert Interview](https://ictresearchmethods.nl/Expert_interview), [Available product analysis](https://ictresearchmethods.nl/Available_product_analysis) and via a proof-of-concept utilizing [Prototyping](https://ictresearchmethods.nl/Prototyping) and [Usability testing](https://ictresearchmethods.nl/Usability_testing) (amongst other methods).

For prototyping and usability testing, Open-Source projects could be established as base lines, as these will allow for full view and control over the environment, this could also include personal web-application projects.

## Similar products

* [Applitools](https://applitools.com/)
* [Testim](https://www.testim.io/)
* [Mabl](https://www.mabl.com/)
* [Perfecto](https://www.perfecto.io/)
* [Test.ai](https://test.ai/all-products)

These similar products all market themselves as automated testing platforms powered by AI. None of these however, do what I want to achieve as most of them are either glorified [GitHub copilot](https://github.com/features/copilot) for writing tests, or are based on visual testing which boils down to checking the validity of HTML (which to my understanding is done just as well by [google lighthouse](https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl=nl) without any AI and is completely Open-Source).

### The difference

My solutions, wants to offer a 0-integration automated testing, by having the AI model be set-up in a generic way that allows it to investigate any website without any prior set-up required, allowing the model to become better and smarter at testing your web-application the more it is used.

# Downsides

My solution sadly also has its downsides, that should be considered.

## Malicious use

Due to my AI model being built to identify possible anomalies within a web applications functionality, it can very easily be used maliciously to find weak points within an application that can be exploited by bad actors.

## Energy/Environment

In recent years, the concerns of how are technology has grown to wastefully consume more and more energy has become a big talking point, especially with the resource intensive block-chains and large AI models of the current day.

My solution does not achieve to train a RL model to be extremely good at finding specific bugs/anomalies but instead utilizes to training process of such a model to rapidly explore a large range of possibilities. This process is sadly very resource inefficient as most of the results gained from such a training phase are thrown away and re-done the next time a web-application is being tested.

## Unforeseen consequences

If you were to go even further and are open to some “science-fiction-y” futures, an extremely well trained and large enough AI model, might be able to identify techniques to efficiently break login systems, causing a new form of brute force login attacks to be possible that were previously not considered. (Personally, I do not think this is likely to happen within the scope of the project, or even in the near future)

# First steps

Getting such a RL model working, actually takes a lot more data exploration and engineering than you might think, as the goal is to allow an agent to explore any web-application with minimal to no integration required.

To achieve this generic system, the first step will be to create a tool to convert any web-page data into a valid RL environment (this includes parsing the HTML, and possible other networking requests when loading the webpage and converting that into a valid observation/action -space).

Secondly a reward policy will have to be created that can efficiently give rewards to the agent to steer it into a direction where it might be able to find unexpected behavior/anomalies. Figuring out an optimal reward policy that can be created and adjusted automatically based on the web-application, will take a large amount of data analysis and collection to fine-tune and filter the useful information from the poisonous/bad ones.