VISUAL ID

Description to images made easy

MCI project Testing plan
Text to Perpetrator: visualising suspect description

Team number: 15

1. Introduction

1.1. Project Overview

Our project is a web-based tool for classifying the description that is related to human appearances in text and then visualising these textual descriptions into images. We start with generating ground truth data that is used for training machine learning models. Follow by developing a machine learning model that is using the state-of-the-art language model, Roberta. The image generation is done by using 3D models of the corresponding categories such as Tops and Bottoms. Since there are ways to describe a person, our project shall capture the semantic meaning of this text and produce a relatively accurate image.

1.2. Scope of the test plan document

This document will include all of the various testing scenarios as well as the process required to complete them. The testing can be divided into 2 parts, the performance of the machine learning model and production-level testing that real-life users may trigger given that the initial version of the Web App is complete. The document will also provide examples to demonstrate the testing and execution strategy. At the same time, the plan will outline the section's responsibilities in relation to the testing scenario. Finally, the document will detail the steps involved in carrying out such testing plans.

2. Test strategy

2.1. The scope of the test

The objective of the test is to verify that the software developed works according to the user requirements. Functionality, performance, and extensibility are the three criteria that the test would focus on. The functionality will be evaluated through unit tests which are used to test the key individual functions. The performance will be determined in the integration test, we will test the accuracy and the running time. Finally, we will test the extensibility-based evaluation on the process of training, testing, and deployment when a new dataset is available.

2.2. Testing report

In the test report, all of the tests will be recorded in a test result recording table, along with the test case and a brief explanation of each test (see Appendix, Table 1).

2.3. Test assumptions

- All parts of the program are fully functional with no visible external disruption.

- The end-user knows how to operate the program.
- The end-user has access to modern web browsers (Chrome, Firefox, and Safari)
- Testing user has the testing software installed and knows how to operate
- The Testing user has access to the Web App and knows how to operate the program.
- The Machine Learning Model is trained on a Macbook Pro M1 2022

3. Test execution

3.1. Machine Learning Model Performance testing

As per the assumption, for the machine learning model, we check the model's performance, which includes the time of running the model and the accuracy of the prediction (see Appendix, Table 2). We pay attention to overfitting and the preprocessing function.

3.2. Integration testing

As per the assumption, the program has 3 parts that live separately on the cloud: Machine Learning Model API, Database and Web App (see Appendix, Table 3). Therefore, we need to make sure that all programs are properly connected.

3.3. Stress testing

As per the assumption, Stress testing is for checking how the program would do in a production environment that may have a large number of users. Through stress testing, it can highlight any inefficiency in the program back-end that requires optimization. The planned stress test scenarios are 10/50/100/500/1000/5000 users are generating sentences at once (see Appendix, Table 4).

3.4. Fallback testing

As per the assumption, fallback testing is for checking if the program is able to handle various errors. The program has 3 parts that live separately on the cloud: Machine Learning Model API, Database and Web App. Therefore, in case one part fails, the Web App should be able to display or handle the error correction in various situations (see Appendix, Table 5).

4. Testing plan schedule

Testing the smaller part first then going to the bigger one.

Early June	Middle of June	End of June
Initiate Integration testing, Fallback testing and record results • Prepare test report documents • Write testing code for each testing execution • Follow the Situation layout in the example	**Initiate Performance testing, Stress testing, record results and refine the product based on the previous results** • Follow the Situation layout in the example • Rewrite the testing code if needed • Re-initiate Test Execution and record results • Refine the testing plan if needed	**Finalise the system based on the previous result.** • Start focusing on fixing the issue based on the result (if applicable) • Re-organise all test documents

5. Risks

5.1. Delays

If an error is found during testing, it could delay the entire project for an unforeseeable number of days. A possible solution is to evaluate the error. If it is a critical error, it needs fixing asap, else it can go into the backlog.

5.2. Bias Testing

During testing, there is a chance that the testing sample is working in favour of the program and not exploring all possible scenarios. A possible solution is to create a separate "Validation" data sample for testing purposes.

5.3. Functions changes while the testing is being written

During the development of the project, if the person who is writing the test and the person who is responsible for developing are not communicating with each other on their progress, the test might not fit with the current state of the program. A possible solution is to only allow the test to be written after one development phase is finished. Each development phase is separated by time, every Monday of each week.

Appendix

Table 1 test result recording table example

Situation	Number of Testing Unit	Expected output	Current output
.			
Test type: **Date:**		**Responsible:** **Reviewer:**	

Table 2 Machine Learning Model Performance testing Example

Situation	Number of Testing Unit	Expected output	Current output
Use the testing software to start evaluating the model's performance. Input the Number of Testing Units accordingly.	1000 simple data points	With over 90% accuracy and the time to train the model is less than 1 day.	The test performance shows different accuracies which are all around 99%, and the time to train the model is approximately 2 hours.
	2000 simple data points		
	1000 advanced data points		
	2000 advanced data points		
Test type: Performance Testing **Date:** 19th May 2022		**Responsible:** Kelvin Ting, Colin Zhu **Reviewer:** Colin Zhu	

Table 3 Integration testing Example

Situation	Scenario	Expected output	Current output
1. Open the web app visual-id.webflow.io 2. Right-click and Inspect the page. 3. Go to the Network tab and follow the scenario. 4. Go back to the Web App 5. Type in **"Male described as caucasian, wearing a grey hoodie,**	With 3G Network Throttled.	1. Captured detail on the left sidebar is correctly captured. 2. The Image Generation correctly generates the corresponding output from across all categories.	The image generation is slow and the UI is unresponsive.
	With No Cache Enabled.		Nothing significant happened.
	On Mobile Screen.		Nothing significant happened.
	On Tablet Screen.		Nothing significant happened.
	None of the above and on a Desktop Screen.		Nothing significant happened.

black pants, sneakers, red cap” as the sentence. 6. Click on Generate.			
Test type: Integration Testing **Date:** 19th May 2022		**Responsible:** Kelvin Ting, Alan Pham **Reviewer:** Colin Zhu	

Table 4 Stress testing Example

Situation	Number of Testing Unit	Expected output	Current output
Use the testing software to start the test. Input the Number of Testing Units accordingly.	10 users	The program shows a loading screen to ensure the user that the system is working. If errors are not caused by integration, display the error. If it is integration’s fault, retry. The Waiting time should not be more than 30 seconds.	No information is showing on the screen.
	50		
	100		
	500		
	1000		
	5000		
Test type: FallBack Testing **Date:** 19th May 2022		**Responsible:** Alan Pham **Reviewer:** Colin Zhu	

Table 5 Fallback testing Example

Initial Setup	Scenarios	Expected output	Current output
1. Open the web app visual-id.webflow.io 2. Right-click and Inspect the page. 3. Go to the Network tab and follow the scenario. 4. Go back to the Web App.	Type “@” and click on Generate	Error Message saying “input text error.”	No information is showing on the screen, and the program continues to work.
	Type “💇🏻” and click on Generate		
	Type “中文” and click on Generate		
	Type “けいおん” and click on Generate		
	Type “Male described as caucasian, wearing a grey pot, black underwear, skating shoes, and red headphones” and click on Generate.	Output on the left sidebar displays “Cannot find the item to generate.”	The output displays the correct text but is not able to generate the image.
	Type “Male described as caucasian, wearing a grey pot, black underwear, skating shoes, and red headphones.” 1. Disable the Internet. 2. Disable Javascript.	1. Error Message saying “No Internet Connection Found.” 2. Error Message saying “No Javascript Enabled.”	No information is showing on the screen, and the program continues to work.
	Resize the browser window to below 320px wide.	Error Message saying “Browser screen is too small”	No information is showing on the screen, and the program continues to work.
Test type: FallBack Testing **Date:** 19th May 2022		**Responsible:** Alan Pham **Reviewer:** Colin Zhu	