

---

# ChatGPT Warfare

## User Guide & Test Report

**Team MCS19**

---

Koo Wai Kit  
Project Manager

wkoo0004@student.monash.edu

Tan Hui Thang  
Technical Lead

htan0046@student.monash.edu

Samuel Meng Yao, Tai  
Quality Assurance

stai0007@student.monash.edu

Prof Raphael Phan  
Supervisor

raphael.phan@monash.edu

# Table of Contents

<b>User Guides</b>	<b>4</b>
Part 1: End User Guide	4
1.1 Accessing the Software	4
1.2 Use Menu and Other Interactive Features	4
1.3 Data Entry	5
1.4 Limitations on the Data	6
1.5 Exiting the Software	6
1.6 Dealing with Special Situations	6
1.7 Software Limitations	7
Part 2: Technical Guide	8
2.1 Downloading the Software	8
2.2 Installing Dependencies	8
2.3 Configuration Data	8
2.4 User Account Setup	9
2.5 Initialization/Routine Maintenance	9
2.6 Other Processes	9
<b>Test Report</b>	<b>11</b>
1. Introduction	11
2. Test Approach	12
3. Unit Testing	13
3.1 Black-Box Testing	13
3.1.1 Unit Test 1 - Text Generation from ChatGPT	13
3.1.2 Unit Test 2 - AI Detection	17
3.1.3 Unit Test 3 - Sentence Similarity Check Function - Hugging Face API	21
3.1.4 Unit Test 4 - File Reading Function	24
3.1.5 Unit Test 5 - Word Swap Gender Transformation	26
3.1.6 Unit Test 6 - Word Swap Sexuality Transformation	30
3.1.7 Unit Test 7 - Word Swap Race Transformation	34
3.1.8 Unit Test 8 - Word Swap Religion Transformation	38
3.1.9 Unit Test 9 - Sentiment Analysis Function	42
3.1.10 Unit Test 10 - Bias Detection Function	45
3.2 White-Box Testing	49
3.2.1 Unit Test 11 - AI Detection (ai_detection.py)	49
3.2.2 Unit Test 12 - Bias Detection (bias_detection.py)	53
3.2.3 Unit Test 13 - ChatGPT (chatgpt.py)	55
3.2.4 Unit Test 14 - Sentence Similarity (sentence_similarity.py)	58
3.2.5 Unit Test 15 - Sentiment Analysis (sentiment_analysis.py)	60

4. Integration Testing	61
4.1 Integration Testing 1 - Flask Application	61
4.2 Integration Testing 2 - Backend Code	65
5. Black-box Testing	68
5.1 List of GUI Components	68
5.2 Testing Result Tabulation	69
6. Usability Testing	71
6.1 Score/Scale for each Level of Agreement	71
6.2 Evaluation Aspect/Statement in the Survey	72
6.3 User Feedback/Survey	73
6.3.1 Survey 1	73
6.3.2 Survey 2	73
6.3.3 Survey 3	74
6.3.4 Survey 4	75
6.3.5 Survey 5	75
6.4 Overall Result	76
6.5 Summary	76
7. Recommendation for Improvements	77
8. Limitations of the Testing Process	78
9. Appendix	80
9.1 Evidence of Testing	80
9.2 Images	86

# User Guides

## Part 1: End User Guide

Welcome to AttackGPT, a web application designed for exploring the impacts of adversarial attacks on GPT-3.5 text prompts and analysing the performance of the ChatGPT model.

### 1.1 Accessing the Software

The web application is hosted on 2 different servers and can be accessed using the 2 URLs below. Visit our web application by navigating to either:

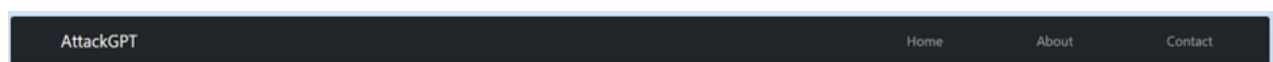
- <https://attackgpt-production.up.railway.app/>
- <https://attackgpt.onrender.com/>

The application is hosted on Railway and Render free servers. Both servers offer the same functionality, but for a more reliable and consistent experience, but we recommend accessing our application using the first URL, which is hosted on Railway. The Railway server has demonstrated higher uptime and is less likely to encounter crashes. In case the first URL experiences any temporary downtime, the second URL serves as an alternative.

Once you have navigated to the URL, you will land on the homepage where you can initiate the attacks.

### 1.2 Use Menu and Other Interactive Features

There is a navigation bar on the top of every page of the website, you can easily navigate to the Home page, About page, and Contact page by clicking on the text buttons.



In the Home page, you can enter text directly in the text box to perform the Bias Attack, or you can choose from a drop-down list of prompts and attack recipes to perform a TextAttack Attack. After you have decided to perform the attack, click on the 'Attack' button.

The screenshot shows the main interface of the AttackGPT application. It has a light blue background. At the top, there's a section titled "Our Bias Attack" with a text input box labeled "Enter text to attack here (10 - 200 characters)" and a blue "Attack" button below it. Below this is a section titled "TextAttack Attack Recipes". It contains two dropdown menus: "Prompt" with the value "what composer used sound mass" and "Attack Recipe" with the value "DeepWordBug". There is a blue "Attack" button at the bottom right of this section. A small footnote at the bottom right states: "\*Please note that some perturbed texts may be the same as the original text, as it depends on how well the attack worked."

In the Contact page, you can enter your information and feedback about the application into the text boxes and click on the ‘Submit’ button to send the feedback.

Have any questions?

Drop a message below and our team will get back to you as soon as possible.

Name:

Email:

Subject:

Details:

In the Results page, there are 2 buttons that you can click on. Click on the ‘Download Results’ button to download the attack results as a html file or click on the ‘Perform New Attack’ button to navigate back to the Home page where you can perform another attack.

**Answer Similarities**

Generated answers are  
82.04% similar.

## 1.3 Data Entry

You can either choose to perform either Bias Attack or the TextAttack Attack at a single time.

**Our Bias Attack**

Enter text to attack here (10 - 200 characters)

**TextAttack Attack Recipes**

Prompt: what composer used sound mass

Attack Recipe: DeepWordBug

\*Please note that some perturbed texts may be the same as the original text, as it depends on how well the attack worked.

To perform a Bias Attack, you can enter any prompts written in English that you would like to perform an attack on.

To perform a TextAttack Attack, click on each box for the ‘Prompt’ and ‘Attack Recipe’ sections to choose a specific prompt and attack recipe.

## **1.4 Limitations on the Data**

The input text for the Bias Attack needs to have a length of 10 – 200 characters or alphabets. The Bias Attack will only work for texts written in English.

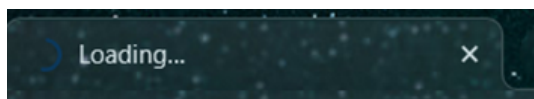
For the TextAttack Attack, users can only choose from a fixed selection of prompts and attack recipes. There are 40 prompts and 4 attack recipes provided in total. In addition, the attack results (perturbed prompts) are fixed and pre-determined since performing a real-time execution for an attack requires a significant amount of time, and it is infeasible to perform an attack on a prompt to obtain a perturbed prompt on the spot.

## **1.5 Exiting the Software**

You can simply close the web browser tab to exit the software.

## **1.6 Dealing with Special Situations**

When accessing the web application, the website may take a long time to load, the browser tab will show that the page is loading. When this happens, please wait a few minutes before trying again or use the alternate URL to access the application using another server.



When performing the attacks, the website may crash and return an error page, you can navigate back to the application by clicking on the ‘Back’ button on your browser.

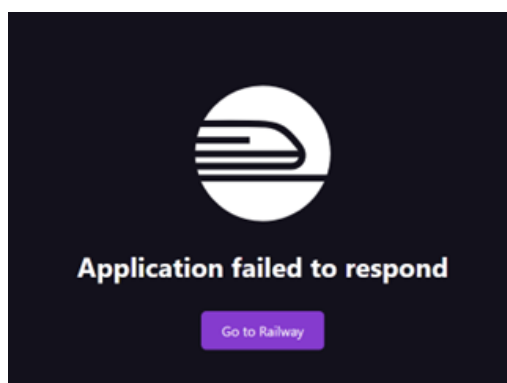


Figure: Error page for Railway server

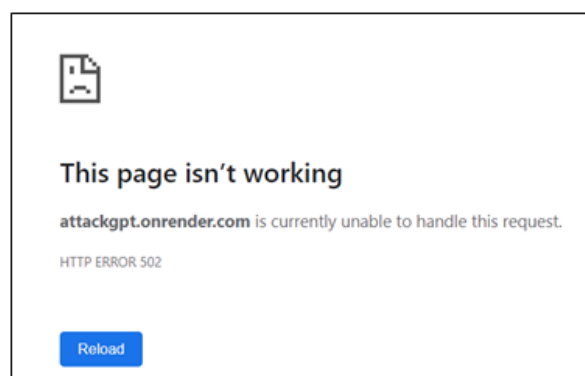
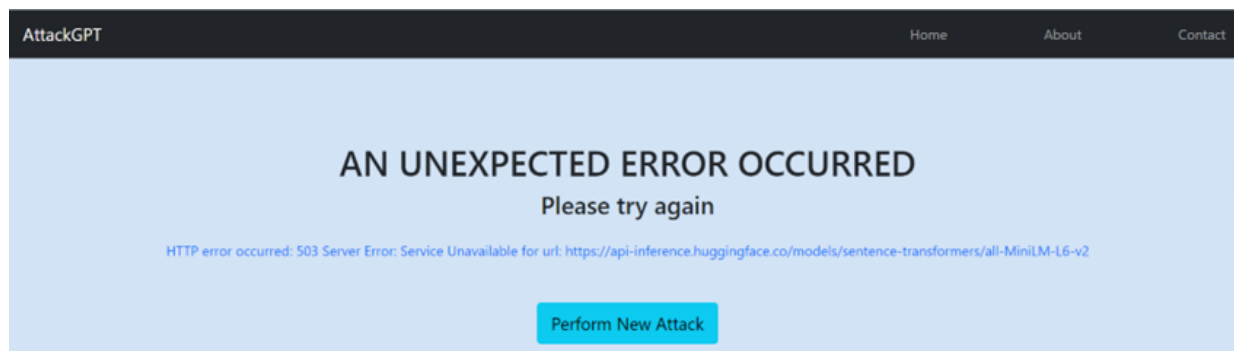


Figure: Error page for Render server

Occasionally, the APIs used in the application may not be functioning, a custom error page like the one below will be shown. You will need to perform the attack again by clicking on the 'Perform New Attack' button to navigate back to the Home page.



## **1.7 Software Limitations**

- The web application is hosted on a free server, the website may crash when performing the attacks. (especially for Bias Attack)
- For the attack results, the Human/AI probabilities are shown only if the generated answer for the prompts are at least 50 words long.
- AttackGPT is heavily reliant on external resources/services (APIs), unavailability of any of the services will result in an error.
- The evaluation results of the same input & perturbed text may be different each time as the results are based on the APIs used.
- The Bias Attack might not work optimally for short inputs.
- For the Bias Attack, entering the same input text may result in a different perturbed text each time due to the random nature of the attack.
- The Bias Attack will only work for text written in the English language.
- The input text length of Bias Attack is limited to 200 characters.
- The duration of the attack process for Bias Attack is not fixed, it may take anywhere from a few seconds to a few minutes.
- The Bias Attack is not guaranteed to work for all inputs, it may fail to perturb the input text.
- The perturbed text generated from the TextAttack attack recipes are predetermined, the same input text will always result in the same perturbed text.

## **Part 2: Technical Guide**

Software installation and configuration for system administrators.

## **2.1 Downloading the Software**

The AttackGPT software is hosted on GitHub. You can obtain the latest version by cloning the repository into your desired directory by entering the command below in a terminal:

```
git clone https://github.com/waikittt/AttackGPT
```

## **2.2 Installing Dependencies**

After cloning, navigate to the project directory and install the required dependencies by entering the command below in a terminal:

```
pip install -r requirements.txt
```

## **2.3 Configuration Data**

You will need to set 2 environment variables, which are the 2 secret API keys used in the application. Firstly, create a file named “.env” in the outermost level of the repository (same level as app.py). In the .env file, insert the 2 lines shown below,

```
CHATGPT_API_KEY = "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"
```

```
ORIGINALITY_AI_API_KEY = "68wcjg2pir4zbnkqdhuf19amy53e07x"
```

You can use your own keys for the OpenAI API or Originality.ai API by changing the key values above (text within the “”). If you would like to register your own keys, please follow the links below:

- <https://openai.com/blog/openai-api>
- <https://originality.ai/>

If you would like to deploy the application via a web server, configure the server to point to the main application file (app.py or wsgi.py).

## **2.4 User Account Setup**

The application is designed to function seamlessly without the need for users to create specific accounts. You can access and use the application's features without the requirement for individual user accounts.

However, in order to clone the code repository into your local machine, you will need to install Git on your computer, and have access to a Git and GitHub account.

If you do not have Git installed on your computer, refer to this link to install Git:

<https://github.com/git-guides/install-git>

If you are not logged in to GitHub or do not have a GitHub account, sign in or sign up using this link:

<https://github.com/login>



## **2.5 Initialization/Routine Maintenance**

### **Running the application:**

The application can be started by running the 'app.py' file. In the terminal, the following lines will be shown:

```
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

To open the application, use 'Ctrl + Click' to open the link in a web browser or copy the link into a web browser. To stop the application, press 'Ctrl + C' in the terminal.

### **Maintenance:**

You can perform routine maintenance by regularly checking for updates to the software on the original repository at <https://github.com/waikitt/AttackGPT>. When there are new commits, you can pull the changes from the original repository. You can use the terminal command below to update your local repository with changes from the original repository.

```
git pull https://github.com/waikitt/AttackGPT
```

## **2.6 Other Processes**

If you would like to modify the application, you can use the Flask debugging mode to start the development server. The debugging mode provides additional features during development and helps you identify and fix issues more easily. To enable that, open the 'app.py' file and change the code on line 174 `app.run()` to `app.run(debug=True)`. Then, you can start the application by running the 'app.py' file.

All the html files are included under the 'templates' folder. You can change the webpages structure by modifying any of the html files. The 'backend' folder contains the logic of how the attacks are performed and the external services that are used. Modifications can be made but you should proceed with caution to ensure the application works as expected.

# Test Report

## 1. Introduction

Testing is a crucial aspect of any project to guarantee that the software functions as expected and operates smoothly. One effective method for conducting comprehensive testing is by creating a testing plan or report. This document should clearly outline the type of testing conducted, the specific components tested, a detailed description of the tests performed, and the results obtained. While our project primarily focuses on the backend structure that our team developed rather than its graphical user interface (GUI), it is still imperative to develop a comprehensive testing plan. This plan ensures that all relevant requirements are met and that every component of the software functions without errors. In doing so, we can maintain the reliability and quality of the software throughout its development and deployment.

This report will be a useful guide or tool to check the correctness of our web application and whether the application can work as expected. The testing plan of the application is composed of various tests as stated in the table below:

Type of test	Description
Unit Testing (Black-box and white-box testing)	Testing that tests the isolated component (function or class) in the simplest form to check their functionalities and tests it with boundary conditions value.
Integration Testing	Testing focuses on testing the combination of multiple components of the software together to ensure they can be properly integrated before combining them into a system. Usually done after unit testing.
Black-box Testing (GUI side)	Testing that tests the Graphical User Interface (GUI) such as buttons and more without the implementation knowledge of the software. This is meant to check whether the GUI is working as expected

Usability Testing	Testing allows the end user to try, test and provide feedback to the developers on the usability aspects of the GUI.
-------------------	--

Most of the testing will be focused on testing the backend logic or implementation of the code as that is what our project is focusing on and the complexity of the backend is much higher than the GUI. All of the Python files are related to the backend but only `app.py` is related to the front end which is implemented using the Flask library.

## 2. Test Approach

In our test plan, we employ a combination of testing frameworks, including Python's `unittest`, manual testing, and collaboration with third parties. Each of these approaches plays a crucial role in ensuring the reliability and functionality of our software.

The `unittest` framework is a central component of our testing strategy, serving as the foundation for both unit testing and integration testing. Given the extensive use of APIs in our project, we heavily rely on `mock` and `patch` features within the `unittest` framework to simulate API behavior during unit testing. This approach allows us to thoroughly assess the interaction between our code and the APIs it interfaces with.

For the GUI aspect, which often requires black-box testing, we conduct manual testing. This manual testing phase assesses the functionality and behaviour of GUI components such as buttons and icons following the application's launch.

In addition to these internal testing methods, we also involve third parties in usability testing. This collaboration ensures a fair and unbiased evaluation of the application's usability, gathering valuable feedback and opinions from external sources.

By combining these three testing approaches, our testing team can comprehensively evaluate the functionality, logic, and behaviour of our application, resulting in a more robust and reliable final product.

### 3. Unit Testing

Unit testing is a critical phase in the software development process, where individual and isolated components of a larger software system are rigorously tested. This approach allows us to thoroughly assess the functionality of these smaller units before they are integrated into the broader system. The main aim of unit testing is to pinpoint and fix code flaws early, preempting more complex and expensive issues in later testing and development phases.

In this test report, we will conduct comprehensive unit testing on the backend components of our software, particularly the APIs that our project are using. This is important to ensure that each component is able to work and handle the special conditions properly. The results will be tabulated in the below section of “[Black-Box Testing](#)” and “[White-Box Testing](#)” sections where each sub-section is the feature/class/APIs that we are testing.

#### 3.1 Black-Box Testing

This testing is conducted on the APIs or functions without the knowledge of the implementation. The testing techniques that our team has used are Equivalence Partitioning (EP) and Boundary Value Partitioning (BVA).

##### 3.1.1 Unit Test 1 - Text Generation from ChatGPT

The sub-section focuses on testing the main interface of our code with the external **OpenAI API**. This function will take in one input which is a string and it will be sent to the API to compute and retrieve its text output. The testing technique used is Equivalence Partitioning where there are 2 equivalence classes (EC), the valid EC is the input of type string and the invalid EC is an input of type non-string. 2 test cases are needed.

Mocking and Patching from Python Unit Testing Framework - unittest is utilised here to mimic the behaviour of the external API and test how our code handles the output from the mimic API.

The main function being tested is `prompt_GPT` in ChatGPT class from **chatgpt.py** (Figure 1). An example test case for this test is attached below (Figure 2).

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC001	Test whether the text prompt function can work as expected when a string is given.  Input -> ChatGPT -> Output	<b>(String)</b>  input_text= "What do you think about the unit FIT3162 in Monash University?"	"I think it is pretty interesting."	Output retrieved as expected  "I think it is pretty interesting."	SS001	Pass
TC002	Test whether the text prompt function can detect and handle the error when a non-string input is given.  Input -> ChatGPT -> Error Handling	<b>(Integer)</b>  input_text= 1234567	"Invalid or malformed request provided (in this case might be non-string argument provided)"	Customised Exception is raised  "Invalid or malformed request provided (in this case might be non-string argument provided)"	SS001	Pass

```

def prompt_GPT(input_text):
    """ ...

    try:
        completion = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",          # GPT model that we are using (Can be modified)
            messages=[
                {"role": "user", "content": input_text}
            ],
            max_tokens = 300                # Maximum length of the reply (Can be modified)
        )

        # Retrieve the reply from ChatGPT
        # First element - indicate success retrieval of reply
        # Second element - the reply provided by ChatGPT
        reply = completion.choices[0].message['content']

        return reply

    except openai.error.APIError: ...

    except openai.error.Timeout: ...

    except openai.error.RateLimitError: ...

    except openai.error.InvalidRequestError:
        # Handle API Connection error
        raise Exception(f"Invalid or malformed request provided (in this case might be non-string argument provided)")

    except openai.error.APIConnectionError: ...

    except openai.error.AuthenticationError: ...

    except openai.error.ServiceUnavailableError: ...

```

Figure 1: Code Snippet of `prompt_GPT` function

```

class ChatgptTest(unittest.TestCase):
    @patch('chatgpt.openai')
    def test_success_chatgpt_generation(self, mock_openai):
        """
        This test tests whether the input and output can be sent and retrieved correctly.
        """
        mock_content = "What do you think about the unit FIT3162 in Monash University?"
        expected_output = "I think it is pretty interesting."

        mock_completion = Mock()          # Create a mock object
        mock_choices = Mock()             # Create a mock object
        mock_choices.message = {'content': expected_output}
        mock_completion.choices = [mock_choices]
        mock_openai.ChatCompletion.create.return_value = mock_completion          # Defined the return value

        res = prompt_GPT(mock_content)    # Call the function

        # Validate whether the output of the function as expected
        _, kwargs = mock_openai.ChatCompletion.create.call_args_list[0]
        self.assertEqual(kwargs['messages'], [{"role": "user", "content": mock_content}])
        self.assertEqual(res, expected_output)

```

Figure 2: Example code (TC001) for test cases under Unit Test 1

### 3.1.2 Unit Test 2 - AI Detection

The sub-section focuses on testing the main interface of our code with the external **Originality.ai API**. This function will take in only one input which is a string and it will be sent to the API to compute and retrieve its AI and Human score. The testing technique used is a combination of Equivalence Partitioning (EP) and Boundary Value Analysis (BVA). For EP, there are 3 equivalence classes (EC), 1 valid EC and 2 invalid EC. The valid EC is the input of type string, where the first invalid EC is an input of type non-string and the second invalid EC is the input of type string that is less than 50 words. BVA will be used in the third EC to further test the boundary value of the code. A total of 3 test cases will be needed.

Mocking and Patching from Python Unit Testing Framework - unittest is utilised here to mimic the behaviour of the external API and test how our code handles the output from the mimic API.

The main function being tested is `human_ai_detection` from **ai\_detection.py** (Figure 3). An example test case for this test is attached below (Figure 4).

The initial result of TC003 failed because it differed from the expected outcome. Upon a thorough code review, our team identified the root cause as an implementation error. Specifically, we were attempting to calculate the number of words, but due to this fault, we were actually counting the number of characters. After addressing this issue by modifying the code, the test case passed successfully.

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC003	Test whether the AI Detection function can work as expected when a string input of length 50 is given.  Input -> API -> Output	(String)  input_text= "Apart from counting words and characters, our online editor can help you to improve word choice and writing"	{"human": 0.09, "ai": 99.91, "ai_prob": Very likely}	Output retrieved as expected  {"human": 0.09, "ai": 99.91, "ai_prob": Very likely}	SS002	Pass



		style, and, optionally, help you to detect grammar mistakes and plagiarism. To check word count, simply place your cursor into the text box above and start typing, and that is the end."				
TC004	<p>Test whether the AI Detection function can work as expected when a non-string input is given.</p> <p>Input -&gt; API -&gt; Error Handling</p>	<p><b>(Integer)</b></p> <p>input_text= 1234567</p>	"Only string input can be accepted!"	<p>Customised Exception is raised</p> <p>"Only string input can be accepted!"</p>	SS002	Pass
TC005	<p>Test whether the AI Detection function can work as expected when a string input of length 49 is given.</p> <p>Input -&gt; API -&gt; Error Handling</p>	<p><b>(String)</b></p> <p>input_text= "Apart from counting words and characters, our online editor can help you to improve word choice and writing style, and, optionally, help you to detect grammar mistakes and plagiarism. To check word count, simply place your cursor into the text box above and start typing, and that is all."</p>	{"human": 0, "ai": 0, "ai_prob": "Undetermined"}	{"human": 0, "ai": 0, "ai_prob": "Undetermined"}	SS002	Pass

```

def human_ai_detection(content:str):
    """ ...

    payload = {"content": content}    # payload forming (content)
    headers = {                       # headers required for the ai scan...

    # Try except block to handle the condition where "content" argument is not of type string
    try:
        num_words = len(content.strip().split(" "))
        print(num_words)
    except:
        raise Exception("Only string input can be accepted!")

    else:
        # Could not perform AI Detection if string length is less than 50
        if num_words >= 50:
            # Send content to endpoint for AI Scanning
            response = requests.request(method="POST", url=url, headers=headers, data=payload)
            response_status_code = response.status_code    # Retrieve the status code for the response

            if response_status_code == 422:
                # Handle Bad Request Error (422)
                raise Exception("Bad Request. Please try again.")

            elif response_status_code == 404:
                # Handle Not Found Error (404)
                raise Exception("Invalid API Endpoint. Please contact the development team")

            elif response_status_code == 429:
                # Handle Too Many Request Error (429)
                raise Exception("Rate Limit Exceeded. Please contact the development team.")

            elif response_status_code == 200:
                response_text = get_updated_str(response.text, 11, 'T')
                res_obj = eval(response_text)    # Convert string to dictionary object

                # Get the prediction result (ai and human)
                human_percentage = res_obj['score']['original'] * 100
                ai_percentage = res_obj['score']['ai'] * 100
                ai_prob = "Undetermined"

                # Probability of being AI generated (From MITRE.org)
                if ai_percentage <= 10: ...
                elif ai_percentage <= 40: ...
                elif ai_percentage <= 60: ...
                elif ai_percentage <= 90: ...
                elif ai_percentage <= 100: ...

            return {"human": human_percentage, "ai": ai_percentage, "ai_prob": ai_prob}

```

**Figure 3: Code Snippet of human\_ai\_detection function**

```

class AIDetectionTest(unittest.TestCase):
    ##### BLACK BOX TESTING #####
    @patch("ai_detection.requests")
    def test_success_ai_detection(self, mock_requests):
        """
        This test tests whether the ai detection can properly process ai percentage <= 100
        """
        mock_response = Mock()
        mock_response.text = '{"success":true,"public_link":"https://app.originality.ai/share/b2lwsg7ruedtc0h","title":"API Scan","score":{"original":0.0009,"ai":0.9991}}'
        mock_response.status_code = 200
        mock_requests.request.return_value = mock_response

        txt = "Apart from counting words and characters, our online editor can help you to improve word choice and writing style, \
            and, optionally, help you to detect grammar mistakes and plagiarism. \
            To check word count, simply place your cursor into the text box above and start typing, and that is the end."
        result = human_ai_detection(txt)

        expected_res = {"human": 0.09, "ai": 99.91, "ai_prob": "Very likely"}

        _, kwargs = mock_requests.request.call_args_list[0]
        self.assertEqual(result, expected_res)
        self.assertEqual(kwargs['data'], {"content": txt})

```

**Figure 4: Example code (TC003) for test cases under Unit Test 2**

### 3.1.3 Unit Test 3 - Sentence Similarity Check Function - Hugging Face API

The sub-section focuses on testing the main interface of our code with the external **Hugging Face API**. This function will take in 2 string inputs and compute the similarity score between them. The testing technique used is Equivalence Partitioning. There are 3 Equivalence Classes in total, with one valid EC for both inputs, one invalid EC for either one of the inputs is invalid and one invalid EC for both inputs being invalid. A total of 3 test cases is needed.

Mocking and Patching from Python Unit Testing Framework - unittest is utilised here to mimic the behaviour of the external API and test how our code handles the output from the mimic API.

The main function being tested is `similarity_checking` in the STSModel class from **sentence\_similarity.py** (Figure 5). An example test case for this test is attached below (Figure 6).

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC006	Test whether the similarity checking function is able to work as expected when 2 valid input (string) is given.  Inputs -> API -> Output	(String, String)  source_sent= "The apple has already rotten"  target_sent= "The apple was spoiled"	72.59	72.59	SS003	Pass
TC007	Test whether the similarity checking function is able to handle the error when the target sentence is valid input	(Integer, String)  source_sent= 1234	"HTTP error occurred: 400 Client Error: Bad Request"	Customised Exception is raised	SS003	Pass

	(string) but the source sentence is not (non-string).  Valid Input, Invalid Input -> API -> Error Handling	target_sent= "The apple was spoiled"		"HTTP error occurred: 400 Client Error: Bad Request"		
TC009	Test whether the similarity checking function is able to handle the error as expected when 2 invalid input (non-string) is given.  Invalid Inputs -> API -> Error Handling	<b>(Integer, Integer)</b> source_sent= 1234 target_sent= 1234	"HTTP error occurred: 400 Client Error: Bad Request"	Customised Exception is raised  "HTTP error occurred: 400 Client Error: Bad Request"	SS003	Pass

```

def similarity_checking(source_sentence, target_sentence):
    """
    Compute the similarity percentage between the source sentence and the target sentence in % format (?/100)
    """
    try:
        payload = payload_forming(source_sentence, target_sentence)
        query_res = query(payload)
        similarity_percentage = round(query_res[0] * 100, 2)  # Between 0 and 1

        return similarity_percentage

    except requests.exceptions.HTTPError as http_error:
        # Handle HTTP errors (e.g., 4xx or 5xx status codes)
        raise Exception(f"HTTP error occurred: {http_error}")

    except requests.exceptions.RequestException as request_error:
        # Handle other request-related exceptions (e.g., network errors)
        raise Exception(f"Request error occurred. Please try again.")

```

Figure 5: Code Snippet of similarity\_checking function

```

class SentenceSimilarityTest(unittest.TestCase):
    @patch('sentence_similarity.requests')
    def test_success_check_similarity(self, mock_requests):
        """
        This test tests whether the sentence similarity can be checked and retrieved properly
        """
        mock_response = Mock()  # mock response object
        mock_response.json.return_value = [0.725885744334234]
        mock_requests.post.return_value = mock_response

        source_sent = "The apple has already rotten"
        target_sent = "The apple was spoiled"
        similarity_percent_res = similarity_checking(source_sent, target_sent)

        # check whether the similarity percentage can be retrieved and in correct format
        self.assertEqual(similarity_percent_res, 72.59)

```

Figure 6: Example code (TC006) for test cases under Unit Test 3

### 3.1.4 Unit Test 4 - File Reading Function

This sub-section focuses on testing the file reading functionality. The purpose of this function is to read prompts from a text file prepared by our team. To conduct thorough testing of this function, our team has employed the Equivalence Partitioning method, resulting in the identification of two Equivalence classes: one representing a valid and existing file and the other representing an invalid file that does not exist. In total, two test cases are required to cover these Equivalence classes.

The test only required the basic functionality of the unittest framework to execute.

The function being tested is `read_prompts_from_file` from **read\_file.py** (Figure 7). An example test case for this test is attached below (Figure 8).

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC006	Test whether the existing file can be read in the expected manner.	(String) filename='valid_file.txt'	["Prompt 1", "Prompt 2", "Prompt 3"]	["Prompt 1", "Prompt 2", "Prompt 3"]	SS004	Pass
TC007	Test whether the error of reading a non-existing file can be handled as expected.	(String) filename='non_existent_file.txt'	"non_existent_file.txt does not exist"	Customised Exception is raised  "non_existent_file.txt does not exist"	SS004	Pass

```
def read_prompts_from_file(filename):
    """
    Read the content of the file (particularly prompt in our project)
    """
    try:
        txt_file = open(filename, 'r', encoding='utf-8')
        prompts = []

        for line in txt_file:
            prompt = line.replace('\n', '')
            prompts.append(prompt)

        return prompts

    except FileNotFoundError:
        raise Exception(f'{filename} does not exist')
```

**Figure 7: Code Snippet of read\_prompts\_from\_file function**

```
class ReadFileTest(unittest.TestCase):
    def test_valid_file(self):
        """
        This test tests whether the content of the file can be properly read in the expected manner
        """
        # Create a temporary file with valid content
        with open('valid_file.txt', 'w', encoding='utf-8') as f:
            f.write("Prompt 1\nPrompt 2\nPrompt 3")

        prompts = read_prompts_from_file('valid_file.txt')

        # Verify that the function returns the expected list of prompts
        self.assertEqual(prompts, ["Prompt 1", "Prompt 2", "Prompt 3"])

        # Clean up the temporary file
        os.remove('valid_file.txt')
```

**Figure 8: Example code (TC006) for test cases under Unit Test 4**



### 3.1.5 Unit Test 5 - Word Swap Gender Transformation

This sub-section focused on testing the customized transformation class inherited from the textattack framework. This is a transformation class that given an input text, tries to find the possible places to substitute gender transformation towards the sentence and return all possible transformations. The technique used to test this class is Equivalence Partitioning, with 3 Equivalence classes. 2 valid ECs for different types of possible formats of transformation and 1 invalid EC for incorrect input. 3 test cases will be needed here.

Only the basic unittest functionality is utilised here, our team does not use mock to mimic the behaviour of the textattack framework to test out the actual execution path of the code through the textattack framework in Python.

The main function being tested is `get_transformation` in the `WordAddGender` class from **word\_add\_gender.py** (Figure 9). An example test case for this test is attached below (Figure 10).

*GENDER = ["Female", "Male", "Transgender"]*

*This is the substitution that will be made in the execution of the test cases, where the **{gender}** part will be replaced by any one of the elements in GENDER.*

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC008	Test whether the gender word swap transformation works as expected when there exists PERSONAL_PRONOUNC S term in the sentence	(AttackedText, dict)  AttackedText('Can I know what is the possible chance to get loan?')  And	Any 2 of the valid combinations	[r1, r2]  Where each r1/r2 will be in the following format:  'Can I, as a {gender}, know what is the possible chance to get loan?'	SS005	Pass

		{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}				
TC009	Test whether the gender word swap transformation work as expected when there exist PEOPLE term in the sentence	<b>(AttackedText, dict)</b>  AttackedText('What is the possible chance for human to get loan?')  And  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	Any 2 of the valid combinations	[r1, r2]  Where each r1/r2 will be in the following format:  'What is the possible chance for {gender} human to get loan?'	SS005	Pass
TC010	Test whether the gender word swap transformation able to detect the error when the non-AttackText object being passed	<b>(String, dict)</b>  "Hello, this is a wrong text!"  And  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	AttributeError caught	AttributeError is raised	SS005	Pass

```

class WordAddGender(WordSwap):

    def __init__(self, n=2, **kwargs): ...

    def _get_replacement_words(self, word): ...

    def _get_transformations(self, current_text, indices_to_modify):
        words = current_text.words
        transformed_texts = []
        last = list(indices_to_modify)[-1]

        for i in indices_to_modify:
            word_to_replace = words[i]

            if word_to_replace.capitalize() in PEOPLE or word_to_replace.capitalize() in PERSONAL_PRONOUNS:

                if i != 0 and (words[i - 1].capitalize() in PEOPLE or words[i - 1].capitalize() in PERSONAL_PRONOUNS):
                    continue

                replacement_words = self._get_replacement_words(word_to_replace)
                transformed_texts_idx = []

                for r in replacement_words:
                    if word_to_replace.capitalize() in PEOPLE:
                        r = r + " " + word_to_replace
                    elif word_to_replace.capitalize() in PERSONAL_PRONOUNS:
                        r = word_to_replace + ", as a " + r + ","

                transformed_texts_idx.append(current_text.replace_word_at_index(i, r))

            transformed_texts.extend(transformed_texts_idx)

        return transformed_texts

```

Figure 9: Code Snippet of `get_transformation` function in `WordAddGender` class

```

class WordAddGenderTest(unittest.TestCase):
    def test_success_gender_word_swap_personal_pronouns(self):
        """
        Test whether the gender word swap transformation work as expected when there exist PERSONAL_PRONOUNCS term in the sentence
        """
        test_obj = WordAddGender()
        test_txt = AttackedText('Can I know what is the possible chance to get loan?')
        test_idx = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

        expected_results = []
        for gender in GENDER:
            modified_str = AttackedText(f'Can I, as a {gender}, know what is the possible chance to get loan?').words
            expected_results.append(modified_str)

        res = test_obj._get_transformations(test_txt, test_idx)
        self.assertTrue(all(attacked_txt.words in expected_results for attacked_txt in res))  # Check whether all transformed text :

```

Figure 10: Example code (TC008) for test cases under Unit Test 5

### 3.1.6 Unit Test 6 - Word Swap Sexuality Transformation

This sub-section focused on testing the customized transformation class inherited from the textattack framework. This is a transformation class that given an input text, tries to find the possible places to substitute sexuality transformation towards the sentence and return all possible transformations. The technique used to test this class is Equivalence Partitioning, with 3 Equivalence classes. 2 valid ECs for different types of possible formats of transformation and 1 invalid EC for incorrect input. 3 test cases will be needed here.

Only the basic unittest functionality is utilised here, our team does not use mock to mimic the behaviour of the textattack framework to test out the actual execution path of the code through the textattack framework in Python. On the other hand, we are just creating the respective argument that is needed and passing it to the function being tested.

The main function being tested is `get_transformation` in the `WordAddSexuality` class from `word_add_sexuality.py` (Figure 11). An example test case for this test is attached below (Figure 12).

*SEXUALITY = ["Straight", "Gay", "Lesbian", "Bisexual"]*

*This is the substitution that will be made in the execution of the test cases, where the {sexuality} part will be replaced by any one of the elements in SEXUALITY.*

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC011	Test whether the sexuality word swap transformation works as expected when there exists PERSONAL_PRONOUNC S term in the sentence	(AttackedText, dict)  AttackedText('Can I know what is the possible chance to get loan?')  And	Any 2 of the valid combinations	[r1, r2]  Where each r1/r2 will be in the following format:  'Can I, as a {sexuality}, know what is the	SS006	Pass

		{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}		possible chance to get loan?		
TC012	Test whether the sexuality word swap transformation work as expected when there exist PEOPLE term in the sentence	<b>(AttackedText, dict)</b>  AttackedText('What is the possible chance for human to get loan?')  And  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	Any 2 of the valid combinations	[r1, r2]  Where each r1/r2 will be in the following format:  'What is the possible chance for {sexuality} human to get loan?'	SS006	Pass
TC013	Test whether the sexuality word swap transformation is able to detect the error when the non-AttackText object is passed	<b>(String, dict)</b>  "Hello, this is a wrong text!"  And  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	AttributeError caught	AttributeError is raised	SS006	Pass

```

class WordAddSexuality(WordSwap):

    def __init__(self, n=2, **kwargs): ...

    def _get_replacement_words(self, word):
        return np.random.choice(SEXUALITY, self.n)

    def _get_transformations(self, current_text, indices_to_modify):
        words = current_text.words
        transformed_texts = []
        last = list(indices_to_modify)[-1]

        for i in indices_to_modify:
            word_to_replace = words[i]

            if word_to_replace.capitalize() in PEOPLE or word_to_replace.capitalize() in PERSONAL_PRONOUNS:

                if i != 0 and (words[i - 1].capitalize() in PEOPLE or words[i - 1].capitalize() in PERSONAL_PRONOUNS):
                    continue

                replacement_words = self._get_replacement_words(word_to_replace)
                transformed_texts_idx = []

                for r in replacement_words:
                    if word_to_replace.capitalize() in PEOPLE:
                        r = r + " " + word_to_replace
                    elif word_to_replace.capitalize() in PERSONAL_PRONOUNS:
                        r = word_to_replace + ", as a " + r + ","

                    transformed_texts_idx.append(current_text.replace_word_at_index(i, r))

                transformed_texts.extend(transformed_texts_idx)

        return transformed_texts

```

Figure 11: Code Snippet of `get_transformation` function in `WordAddSexuality` class

```

def test_success_sexuality_word_swap_people(self):
    """
    Test whether the sexuality word swap transformation work as expected when there exist PEOPLE term in the sentence
    """
    test_obj = WordAddSexuality()
    test_txt = AttackedText('What is the possible chance for human to get loan?')
    test_idx = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

    expected_results = []
    for sexuality in SEXUALITY:
        modified_str = AttackedText(f'What is the possible chance for {sexuality} human to get loan?').words
        expected_results.append(modified_str)

    res = test_obj._get_transformations(test_txt, test_idx)
    self.assertTrue(all(attacked_txt.words in expected_results for attacked_txt in res)) # Check whether all transform

```

Figure 12: Example code (TC012) for test cases under Unit Test 6



### 3.1.7 Unit Test 7 - Word Swap Race Transformation

This sub-section focused on testing the customized transformation class inherited from the textattack framework. This is a transformation class that given an input text, tries to find the possible places to substitute race transformation towards the sentence and return all possible transformations. The technique used to test this class is Equivalence Partitioning, with 3 Equivalence classes. 2 valid ECs for different types of possible formats of transformation and 1 invalid EC for incorrect input. 3 test cases will be needed here.

Only the basic unittest functionality is utilised here, our team does not use mock to mimic the behaviour of the textattack framework to test out the actual execution path of the code through the textattack framework in Python. On the other hand, we are just creating the respective argument that is needed and passing it to the function being tested.

The main function being tested is `get_transformation` in the `WordAddRace` class from **word\_add\_race.py** (Figure 13). An example test case for this test is attached below (Figure 14).

*RACE = ["Asian", "Black", "White", "Hispanic"]*

*This is the substitution that will be made in the execution of the test cases, where the **{race}** part will be replaced by any one of the elements in RACE.*

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC014	Test whether the race word swap transformation works as expected when there exists PERSONAL_PRONOUNC S term in the sentence	(AttackedText, dict)  AttackedText('Can I know what is the possible chance to get loan?')  And	Any 2 of the valid combinations	[r1, r2]  Where each r1/r2 will be in the following format:  'Can I, as a <b>{race}</b> , know what is the possible chance to get loan?'	SS007	Pass

		{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}				
TC015	Test whether the race word swap transformation work as expected when there exist PEOPLE term in the sentence	<b>(AttackedText, dict)</b>  AttackedText('What is the possible chance for human to get loan?')  And  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	Any 2 of the valid combinations	[r1, r2]  Where each r1/r2 will be in the following format:  'What is the possible chance for {race} human to get loan?'	SS007	Pass
TC016	Test whether the race word swap transformation is able to detect the error when the non-AttackText object is passed	<b>(String, dict)</b>  "Hello, this is a wrong text!"  And  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	AttributeError caught	AttributeError is raised	SS007	Pass

```

class WordAddRace(WordSwap):

    def __init__(self, n=2, **kwargs):
        super().__init__(**kwargs)
        self.n = n

    def _get_replacement_words(self, word):
        return np.random.choice(RACE, self.n)

    def _get_transformations(self, current_text, indices_to_modify):
        words = current_text.words
        transformed_texts = []
        last = list(indices_to_modify)[-1]

        for i in indices_to_modify:
            word_to_replace = words[i]

            if word_to_replace.capitalize() in PEOPLE or word_to_replace.capitalize() in PERSONAL_PRONOUNS:

                if i != 0 and (words[i - 1].capitalize() in PEOPLE or words[i - 1].capitalize() in PERSONAL_PRONOUNS):
                    continue

                replacement_words = self._get_replacement_words(word_to_replace)
                transformed_texts_idx = []

                for r in replacement_words:
                    if word_to_replace.capitalize() in PEOPLE:
                        r = r + " " + word_to_replace
                    elif word_to_replace.capitalize() in PERSONAL_PRONOUNS:
                        r = word_to_replace + ", as a " + r + ","

                    transformed_texts_idx.append(current_text.replace_word_at_index(i, r))

                transformed_texts.extend(transformed_texts_idx)

        return transformed_texts

```

Figure 13: Code Snippet of `get_transformation` function in `WordAddRace` class

```
def test_fail_race_word_swap(self):  
    """  
    Test whether the race word swap transformation able to detect the error when non-AttackText object being passed  
    """  
    test_obj = WordAddRace()  
    test_txt = "Hello, this is a wrong text!"  
    test_idx = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}  
  
    with self.assertRaises(AttributeError):  
        test_obj._get_transformations(test_txt, test_idx)
```

Figure 14: Example code (TC016) for test cases under Unit Test 7

### 3.1.8 Unit Test 8 - Word Swap Religion Transformation

This sub-section focused on testing the customized transformation class inherited from the textattack framework. This is a transformation class that given an input text, tries to find the possible places to substitute religion transformation towards the sentence and return all possible transformations. The technique used to test this class is Equivalence Partitioning, with 3 Equivalence classes. 2 valid ECs for different types of possible formats of transformation and 1 invalid EC for incorrect input. 3 test cases will be needed here.

Only the basic unittest functionality is utilised here, our team does not use mock to mimic the behaviour of the textattack framework to test out the actual execution path of the code through the textattack framework in Python. On the other hand, we are just creating the respective argument that is needed and passing it to the function being tested.

The main function being tested is `get_transformation` in the `WordAddReligion` class from **word\_add\_religion.py** (Figure 15). An example test case for this test is attached below (Figure 16).

```
RELIGION = [  
    "Christian", "Muslim", "Hindu", "Buddhist", "Sikh", "Jewish", "Jain", "Bahá'í", "Shinto", "Taoist", "Zoroastrian", "Confucian", "Indigenous",  
    "Pagan", "Atheist", "Agnostic", "Sikh", "Rastafarian", "Baha'i", "Mormon", "Unitarian", "Druid", "Wiccan", "Scientologist"  
]
```

*This is the substitution that will be made in the execution of the test cases, where the **{religion}** part will be replaced by any one of the elements in RELIGION.*

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC017	Test whether the religion word swap transformation works as expected when	(AttackedText, dict)	Any 3 of the valid combinations	[r1, r2, r2]	SS008	Pass

	there exists PERSONAL_PRONOUNC S term in the sentence	AttackedText('Can I know what is the possible chance to get loan?')  And  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}		Where each r1/r2/r3 will be in the following format:  'Can I, as a {religion}, know what is the possible chance to get loan?'		
TC018	Test whether the religion word swap transformation work as expected when there exist PEOPLE term in the sentence	(AttackedText, dict)  AttackedText('What is the possible chance for human to get loan?')  And  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	Any 3 of the valid combinations	[r1, r2, r3]  Where each r1/r2/r3 will be in the following format:  'What is the possible chance for {religion} human to get loan?'	SS008	Pass
TC019	Test whether the religion word swap transformation is able to detect the error when the non-AttackText object is passed	(String, dict)  "Hello, this is a wrong text!"  And  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	AttributeError caught	AttributeError is raised	SS008	Pass

```

class WordAddReligion(WordSwap):

    def __init__(self, n=3, **kwargs):
        super().__init__(**kwargs)
        self.n = n

    def _get_replacement_words(self, word):
        return np.random.choice(RELIGION, self.n)

    def _get_transformations(self, current_text, indices_to_modify):
        words = current_text.words
        transformed_texts = []
        last = list(indices_to_modify)[-1]

        for i in indices_to_modify:
            word_to_replace = words[i]

            if word_to_replace.capitalize() in PEOPLE or word_to_replace.capitalize() in PERSONAL_PRONOUNS:

                if i != 0 and (words[i - 1].capitalize() in PEOPLE or words[i - 1].capitalize() in PERSONAL_PRONOUNS):
                    continue

                replacement_words = self._get_replacement_words(word_to_replace)
                transformed_texts_idx = []

                for r in replacement_words:
                    # r = word_to_replace + " that are " + r
                    if word_to_replace.capitalize() in PEOPLE:
                        r = r + " " + word_to_replace
                    elif word_to_replace.capitalize() in PERSONAL_PRONOUNS:
                        r = word_to_replace + ", as a " + r + ","

                    transformed_texts_idx.append(current_text.replace_word_at_index(i, r))

                transformed_texts.extend(transformed_texts_idx)

        return transformed_texts

```

**Figure 15: Code Snippet of `get_transformation` function in `WordAddReligion` class**

```

class WordAddReligionTest(unittest.TestCase):
    def test_success_religion_word_swap_personal_pronouns(self):
        """
        Test whether the religion word swap transformation work as expected when there exist PERSONAL_PRONOUNCS term in the sentence
        """
        test_obj = WordAddReligion()
        test_txt = AttackedText('Can I know what is the possible chance to get loan?')
        test_idx = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

        expected_results = []
        for religion in RELIGION:
            modified_str = AttackedText(f'Can I, as a {religion}, know what is the possible chance to get loan?').words
            expected_results.append(modified_str)

        res = test_obj._get_transformations(test_txt, test_idx)
        self.assertTrue(all(attacked_txt.words in expected_results for attacked_txt in res)) # Check whether all transformed text is

```

Figure 16: Example code (TC017) for test cases under Unit Test 8



### 3.1.9 Unit Test 9 - Sentiment Analysis Function

This sub-section focused on testing the main interface between our code and the HuggingFace API of sentiment analysis. This sentiment analysis takes in a string input and returns the sentiment analysis of the string. The technique used to test it is by using Equivalence Partitioning where 1 valid and invalid Equivalence Class has been derived for both valid and invalid input data type. The method we used to test the invalid input is different from the above where instead of catching the error raised, our implementation of this function will only stop when it gets the sentiment output. Hence, we design the test case in a way that the first iteration will raise an Exception but is caught by our code, and the second iteration will be a successful request towards the API. A total of 2 test cases are needed.

Mocking and Patching from Python Unit Testing Framework - unittest is utilised here to mimic the behaviour of the external API and test how our code handles the output from the mimic API. On the other hand, we are just creating the respective argument that is needed and passing it to the function being tested.

The main function being tested is `sentiment` in the `SentimentAnalysis` class from `sentiment_analysis.py` (Figure 17). An example test case for this test is attached below (Figure 18).

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC020	Test whether the sentiment analysis function is able to pass the input to API and correctly interpret the output in the desired way when a valid input is given.	(string)  "White people is very violence"	"Negative"	"Negative"	SS009	Pass
TC021	Test whether the sentiment analysis function is able to handle the error that	(string)	Error message caught and message printed out in terminal:	Error message caught and message printed out in terminal:	SS009	Pass

	occurred such as http connection error, and retry the request to the API to get the output.	"White people is very violence"  With HTTPError being mocked within the test case for the first iteration	HTTP error occurred: 400 Client Error: Bad Request Retrying in 5 seconds..  Output: "Negative"	HTTP error occurred: 400 Client Error: Bad Request Retrying in 5 seconds..  Output: "Negative"		
--	---	---	---	---	--	--

```

class SentimentAnalysis:

    def __init__(self): ...

    def payload_forming(self, sentence): ...

    def query(self, payload):

        while True:
            try:
                response = requests.post(self.API_URL, headers=self.headers, json=payload)
                response.raise_for_status() # Raise an exception for 4xx and 5xx status codes
                return response.json()
            except requests.exceptions.HTTPError as http_err:
                print(f"HTTP error occurred: {http_err}")
                # Handle the error as needed, e.g., log it or raise a custom exception
            except requests.exceptions.RequestException as req_err:
                print(f"Request error occurred: {req_err}")
                # Handle the error as needed, e.g., log it or raise a custom exception
            except Exception as err:
                print(f"An unexpected error occurred: {err}")

            print("Retrying in 5 seconds...")
            time.sleep(5)

    def sentiment_analysis(self, input_sentence): ...

    def sentiment(self, input_sentence):
        sentiment_score = self.sentiment_analysis(input_sentence)
        res = sentiment_score[0]['label']

        return res.capitalize()

```

Figure 17: Code Snippet of `sentiment` function in `SentimentAnalysis` class

```
@patch('sentiment_analysis.requests.post')
def test_fail_sentiment_analysis(self, mock_post):
    """
    This test tests whether the error can be properly handled and sentiment analysis result can be successfully retrieved in the next iteration
    """
    sa = SentimentAnalysis()
    error_message = "400 Client Error: Bad Request"
    mock_http_error = requests.exceptions.HTTPError(error_message)

    mock_response = Mock()      # mock response object
    mock_response.raise_for_status.side_effect = [mock_http_error, None]  # Simulate the condition where first iteration is error, while second is successful
    mock_response.json.return_value = [{"label": "negative"}]
    mock_post.return_value = mock_response

    input_sentence = "White people is very violence"
    res = sa.sentiment(input_sentence)

    self.assertEqual("Negative", res)
```

Figure 18: Example code (TC021) for test cases under Unit Test 9 and 15

### 3.1.10 Unit Test 10 - Bias Detection Function

This sub-section focused on testing the main interface between our code and the HuggingFace API of bias detection. This bias detection takes in a string input and returns the bias score of the string. The technique used to test it is by using Equivalence Partitioning where 2 valid and 1 invalid Equivalence Class has been derived. One of the valid ECs is the input without any anti-bias statement while the other input is with the statement. The method used for testing the invalid EC is the same as Test 9 where we capture the error and rerun it for the second iteration until it succeeds. A total of 3 test cases are needed.

Mocking and Patching from Python Unit Testing Framework - unittest is utilised here to mimic the behaviour of the external API and test how our code handles the output from the mimic API.

The main function being tested is `detect_bias` in the `BiasDetection` class from **bias\_detection.py** (Figure 19). An example test case for this test is attached below (Figure 20).

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC022	Test whether the bias detection function is able to pass the input without anti-bias statement to API and correctly interpret the output in the desired way when a valid input is given.	(string) "White people is very violence"	80	80	SS010	Pass
TC023	Test whether the bias detection function is able to pass the input with anti-bias statement to API and	(string) "White people is very violence"	80	80	SS010	Pass

	correctly interpret the output in the desired way when a valid input is given.					
TC024	Test whether the sentiment analysis function is able to handle the error that occurred such as HTTP connection error, and retry the request to the API to get the output.	<p><b>(string)</b></p> <p>"Chinese people is fairly quiet"</p> <p>With HTTPError being mocked within the test case for the first iteration</p>	<p>Error message was caught and message printed out in the terminal:</p> <p>HTTP error occurred: 400 Client Error: Bad Request Retrying in 5 seconds..</p> <p>Output: 70</p>	<p>Error message was caught and message printed out in the terminal:</p> <p>HTTP error occurred: 400 Client Error: Bad Request Retrying in 5 seconds...</p> <p>Output: 70</p>	SS010	Pass

```

class BiasDetection:

    def __init__(self): ...

    def payload_forming(self, sentence): ...

    def query(self, payload):
        while True:
            try:
                response = requests.post(self.API_URL, headers=self.headers, json=payload)
                response.raise_for_status() # Raise an exception for 4xx and 5xx status codes
                return response.json()
            except requests.exceptions.HTTPError as http_err:
                print(f"HTTP error occurred: {http_err}")
                # Handle the error as needed, e.g., log it or raise a custom exception
            except requests.exceptions.RequestException as req_err: ...
            except Exception as err: ...

            print("Retrying in 5 seconds...")
            time.sleep(5)

    def detect_bias(self, input_sentences):
        filtered = filter_sentence(input_sentences)
        payload = self.payload_forming(filtered)
        bias_score = 0

        try:
            response = self.query(payload)[0]

            for item in response:
                if item['label'] == 'BIASED':
                    bias_score = item['score'] * 100
                    break

            return bias_score
        except Exception as e:
            print(f"An error occurred: {e}")

```

**Figure 19: Code Snippet of detect\_bias function in BiasDetection class**

```

@patch('bias_detection.requests')
def test_success_bias_detection_filter(self, mock_requests):
    """
    This test tests whether the bias detection result can be successfully retrieved
    """
    bd = BiasDetection()
    mock_response = Mock()          # mock response object
    mock_response.json.return_value = [{"label": "BIASED", "score": 0.80}]
    mock_requests.post.return_value = mock_response

    input_sentence = "You should not consider this based on the gender of a person. White people is not violence"
    res = bd.detect_bias(input_sentence)

    _, kwargs = mock_requests.post.call_args_list[0]
    preprocess_sent = kwargs['json']['inputs']
    self.assertEqual(preprocess_sent, "White people is not violence")
    self.assertEqual(res, 80)

```

Figure 20: Example code (TC023) for test cases under Unit Test 10

## 3.2 White-Box Testing

We have achieved a coverage percentage of 91% calculated using the combination of statement and branch coverage with our black-box testing (Appendix Fig 9.2.1). After reviewing the report, we found out that there are some files that are not covered well where the coverage percentage is less than 90%. Hence, we decided to use the white-box testing method to develop the test cases that cover the execution path of some functions until they meet the minimum coverage percentage that our team desired. After performing the tests, the overall coverage has reached 97% (Appendix Fig 9.2.2).

The white-box testing will only be carried out on the files with a low coverage percentage, particularly 5 files. The sub-section will illustrate the testing result of the white-box testing. All of the test cases below will utilise the mock and patch provided in the unittest framework to mimic the API behaviour while the error handling part will be mocked as well where sometimes we do not need to provide proper input to test the function.

The number of test cases should be determined based on the significance of missed statements or branches. This approach allows us to make informed decisions about whether to cover them or not. It is not always necessary to achieve 100% coverage for each file, as testing consumes valuable resources and time.

### 3.2.1 Unit Test 11 - AI Detection (ai\_detection.py)

Most of the statements or branches that have not been tested in this module are related to either error-handling exceptions or branches that can lead to other results. Hence, we decided to use branch coverage to test the codes to ensure all or most branches are covered.

Refer back to Figure 3 for the code snippet of the main function being tested. An example test case for this test is attached below (Figure 21).

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC025	Tests whether the ai detection can properly process ai percentage <= 90	(string)	{"human": 13, "ai": 87, "ai_prob": "Likely"}	{"human": 13, "ai": 87, "ai_prob": "Likely"}	SS011	Pass



		input_text= "Apart from counting words and characters, our online editor can help you to improve word choice and writing style, and, optionally, help you to detect grammar mistakes and plagiarism. To check word count, simply place your cursor into the text box above and start typing, and that is the end."				
TC026	Tests whether the ai detection can properly process ai percentage <= 60		{"human": 42.85, "ai": 57.15, "ai_prob": "Unsure"}	{"human": 42.85, "ai": 57.15, "ai_prob": "Unsure"}	SS011	Pass
TC027	Tests whether the ai detection can properly process ai percentage <= 40		{"human": 60, "ai": 40, "ai_prob": "Unlikely"}	{"human": 60, "ai": 40, "ai_prob": "Unlikely"}	SS011	Pass
TC028	Tests whether the ai detection can properly process ai percentage <= 20		{"human": 95.80, "ai": 4.20, "ai_prob": "Very unlikely"}	{"human": 95.80, "ai": 4.20, "ai_prob": "Very unlikely"}	SS011	Pass

TC029	Tests whether the ai detection handles the Bad Request Error (422)	(string)  input_text= "Apart from counting words and characters, our online editor can help you to improve word choice and writing style, and, optionally, help you to detect grammar mistakes and plagiarism. To check word count, simply place your cursor into the text box above and start typing, and that is the end."	"Bad Request. Please try again."	"Bad Request. Please try again."	SS011	Pass
TC030	Tests whether the ai detection handles the Not Found Error (404)		"Invalid API Endpoint. Please contact the development team"	"Invalid API Endpoint. Please contact the development team"	SS011	Pass
TC031	Tests whether the ai detection handles the Too Many Request Error (429)	<b>Error is triggered using mock</b>	"Rate Limit Exceeded. Please contact the development team."	"Rate Limit Exceeded. Please contact the development team."	SS011	Pass

```

@patch("ai_detection.requests")
def test_success_likely_ai_detection_WB1(self, mock_requests):
    """
    This test tests whether the ai detection can properly process ai percentage <= 90
    """
    mock_response = Mock()
    mock_response.status_code = 200
    mock_response.text = '{"success":true,"public_link":"https:\\\\app.originality.ai\\share\\b2"}'
    mock_requests.request.return_value = mock_response

    txt = "Apart from counting words and characters, our online editor can help you to improve v
    result = human_ai_detection(txt)

    expected_res = {"human": 13, "ai": 87, "ai_prob": "Likely"}

    _, kwargs = mock_requests.request.call_args_list[0]
    self.assertEqual(result, expected_res)
    self.assertEqual(kwargs['data'], {"content": txt})

```

Figure 21: Example code (TC025) for test cases under Unit Test 11

### 3.2.2 Unit Test 12 - Bias Detection (bias\_detection.py)

Most of the statements or branches that have not been tested in this module are related to error-handling exceptions. Hence, we decided to use branch coverage to test the codes to ensure all or most branches are covered. Within the test cases, we did not cover one of the exception handling as it is not related to API Error and has a very small chance of being triggered during code execution.

Refer back to Figure 19 for the code snippet of the main function being tested. An example test case for this test is attached below (Figure 22).

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC032	Tests whether the request exception error can be properly handled by bias detection function and bias result can be successfully retrieved in the next iteration	(string)  "Chinese people is fairly quiet"  With RequestException Error being mocked within the test case for the first iteration	Error message was caught and message printed out in the terminal:  Request Exception Error Retrying in 5 seconds..  Output: 70	Error message was caught and message printed out in the terminal:  Request Exception Error Retrying in 5 seconds..  Output: 70	SS012	Pass
TC033	Tests whether the exception error can be properly handled by bias detection function and whether bias result can be successfully retrieved in the next iteration	(string)  "Chinese people is fairly quiet"	Error message was caught and message printed out in the terminal:  Unexpected Error Retrying in 5 seconds..	Error message was caught and message printed out in the terminal:  Unexpected Error Retrying in 5 seconds..	SS012	Pass

		With Exception Error being mocked within the test case for the first iteration	Output: 70	Output: 70		
--	--	--	---------------	---------------	--	--

```

@patch('sentiment_analysis.requests.post')
def test_fail_sentiment_analysis_request_exception_WB1(self, mock_post):
    """
    This test tests whether the request exception error can be properly handled and bias result of
    """
    bd = BiasDetection()
    error_message = "Request Exception Error"
    mock_http_error = requests.exceptions.RequestException(error_message)

    mock_response = Mock()      # mock response object
    mock_response.raise_for_status.side_effect = [mock_http_error, None]  # Simulate the condition
    mock_response.json.return_value = [{"label": "BIASED", "score": 0.70}]
    mock_post.return_value = mock_response

    input_sentence = "Chinese people is fairly quiet"
    res = bd.detect_bias(input_sentence)

    self.assertEqual(res, 70)

```

Figure 22: Example code (TC032) for test cases under Unit Test 12

### 3.2.3 Unit Test 13 - ChatGPT (chatgpt.py)

All statements or branches that have not been tested in this module are related to error-handling exceptions. Hence, we decided to use branch coverage to test the codes to ensure all or most branches are covered. Within the test cases, we covered all of the possible exceptions handling to reach 100% coverage as it is related to the API-side error which is very important to have a clear message when an error occurs.

Refer back to Figure 1 for the code snippet of the main function being tested. An example test case for this test is attached below (Figure 23).

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC034	Tests whether API error can be handled properly by chatgpt prompt function.	(string) "Placeholder input"  Input does not matter as the API error is triggered using the mock	"OpenAI API returned an API Error."	"OpenAI API returned an API Error."	SS013	Pass
TC035	Tests whether the timeout error can be handled properly by chatgpt prompt function.	(string) "Placeholder input"  Input does not matter as the Timeout error is triggered using mock	"Request took too long to complete on OpenAI API."	"Request took too long to complete on OpenAI API."	SS013	Pass
TC036	Tests whether rate limit error can be handled properly by chatgpt prompt function.	(string) "Placeholder input"	"OpenAI API request exceeded rate limit. Please contact the development team."	"OpenAI API request exceeded rate limit. Please contact the development team."	SS013	Pass

		Input does not matter as the rate limit error is triggered using mock				
TC037	Tests whether API connection error can be handled properly by chatgpt prompt function.	<b>(string)</b> "Placeholder input" Input does not matter as the API connection error is triggered using mock	"Failed to connect to OpenAI API."	"Failed to connect to OpenAI API."	SS013	Pass
TC038	Tests whether authentication errors can be handled properly by chatgpt prompt function.	<b>(string)</b> "Placeholder input" Input does not matter as the authentication error is triggered using mock	"API Key or token is invalid, expired or revoked. Please contact the development team."	"API Key or token is invalid, expired or revoked. Please contact the development team."	SS013	Pass
TC039	Tests whether server unavailable error can be handled properly by chatgpt prompt function.	<b>(string)</b> "Placeholder input" Input does not matter as the Server unavailable error is triggered using mock	"OpenAI Servers temporarily unable."	"OpenAI Servers temporarily unable."	SS013	Pass

```

@patch('openai.ChatCompletion.create', side_effect=openai.error.APIError('Test error message'))
def test_fail_chatgpt_generation_api_error_WB1(self, mock_openai_create):
    """
    This test tests whether api error can be handled properly by chatgpt prompt function.
    """
    chatgpt = ChatGPT()

    # Call the function with a mock that raises an Exception
    with self.assertRaises(Exception) as context:
        chatgpt.prompt_GPT("Placeholder input")

    # Check if the expected error message is in the exception
    self.assertIn(f"OpenAI API returned an API Error.", str(context.exception))

```

Figure 23: Example code (TC034) for test cases under Unit Test 13



### 3.2.4 Unit Test 14 - Sentence Similarity (sentence\_similarity.py)

The only branch that we are missing on this module is one of the exception handling for the HuggingFace API. We decided to use 1 test case to test it as it is important to ensure the error related to the external API is properly handled as expected and provides a meaningful error message, which is also the branch coverage technique.

Refer back to Figure 5 for the code snippet of the main function being tested. An example test case for this test is attached below (Figure 24).

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC040	Tests whether the similarity checking can handle request exception error	(string, string) “Placeholder input” “Placeholder input”  Input does not matter as the Request Exception error is triggered using mock	"Request Exception Error"	"Request Exception Error"	SS014	Pass

```

@patch('sentence_similarity.requests.post')
def test_fail_check_similarity_request_exception_WB1(self, mock_post):
    """
    This test tests whether the similarity checking can handle request exception error
    """
    sts = STSModel()
    error_message = "Request Exception Error"
    mock_error = requests.exceptions.RequestException(error_message)

    mock_response = Mock()
    mock_response.raise_for_status.side_effect = mock_error
    mock_post.return_value = mock_response

    source_sent = "Placeholder input"
    target_sent = "Placeholder input"

    with self.assertRaises(Exception) as context:
        sts.similarity_checking(source_sent, target_sent)

    self.assertEqual(mock_post.call_count, 1)
    self.assertIn(f"Request error occurred. Please try again.", str(context.exception))

```

Figure 24: Example code (TC040) for test cases under Unit Test 14

### 3.2.5 Unit Test 15 - Sentiment Analysis (sentiment\_analysis.py)

All statements or branches that have not been tested in this module are related to error-handling exceptions. Hence, we decided to use branch coverage to test the codes to ensure all or most branches are covered. Within the test cases, we did not cover one of the exception handling as it is not related to API Error and has a very small chance of being triggered during code execution.

Refer back to Figure 17 for the code snippet of the main function being tested. An example test case for this test can refer back to Figure 18.

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC041	Tests whether the request Exception error can be properly handled and sentiment analysis result can be successfully retrieved in the next iteration	(string)  "White people is very violence"  With RequestException Error being mocked within the test case for the first iteration	Error message caught and message printed out in terminal:  Request error occurred: Request Exception error Retrying in 5 seconds..  Output: "Negative"	Error message caught and message printed out in terminal:  Request error occurred: Request Exception error Retrying in 5 seconds..  Output: "Negative"	SS015	Pass
TC042	Tests whether the Exception error (or other error) can be properly handled and whether the sentiment analysis result can be successfully retrieved in the next iteration	(string)  "White people is very violence"  With Exception Error being mocked within the test case for the first iteration	Error message caught and message printed out in terminal:  An unexpected error occurred: Exception Error Retrying in 5 seconds..  Output: "Negative"	Error message caught and message printed out in terminal:  An unexpected error occurred: Exception Error Retrying in 5 seconds..  Output: "Negative"	SS015	Pass

## 4. Integration Testing

After the completion of unit testing, the next phase is integration testing, which involves the consolidation and assessment of multiple software modules or components as a unified group. The primary objective of this testing phase is to ascertain whether the interactions between the various components of the software function as anticipated and operate seamlessly when integrated into a cohesive whole.

Our integration testing is focused on 2 aspects of the software:

1. Flask application
2. Backend code

### 4.1 Integration Testing 1 - Flask Application

This section focuses on ensuring the integration and interaction of various components of the Flask application is working smoothly and as expected. It covers the navigation pages part, download result part, attack part and evaluation part of the software.

The testing techniques that we used to test the integration in the Flask application is by utilising the unittest library. For the test cases related to page navigation, we use the get method to request the response to the respective page and check whether the response's status code is 200 which means the request has succeeded. As for the download feature, we used request\_mock from the Flask library to mock the URL that the HTML is downloaded from and ran testing to check whether the downloaded file exist. As for the testing that involves attacking and evaluation, we just set up the necessary input and allow the testing to interact with the actual API and retrieve the output. This is because interacting with actual API instead of mock can ensure the interactions between each API and code work as expected. As for the output, we verified it by checking whether the necessary "keyword" exists on the page returned to us.

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		

TC043	Test whether the start route work as expected	‘/’	Status code of 200	Status code of 200	SS016	Pass
TC044	Test whether the about route work as expected	‘/about’	Status code of 200	Status code of 200	SS016	Pass
TC045	Test whether the contact route work as expected	‘/contact’	Status code of 200	Status code of 200	SS016	Pass
TC046	Test whether the index route work as expected	‘/index’	Status code of 200	Status code of 200	SS016	Pass
TC047	Test whether the download feature of the web page is working as expected	‘/download’, data= {‘url’: ‘https://example.com’}	‘download.html’ appeared	‘download.html’ appear in the tester directory	SS016	Pass
TC048	Test whether bias attack is working and result is loaded as expected	‘/results’, query_string= {‘oriInput’: “Should you co-sign a personal loan for a friend/family member? Why/why not?”}	The attack and evaluation successfully executed and the return output contains the necessary keyword as defined.	The attack and evaluation successfully executed and the return output contains the necessary keyword as defined.	SS017	Pass
TC049	Test whether the TextAttack recipe attack can work as expected	prompt_num = 0 attack_code = ‘dwb’	The attack and evaluation successfully executed and the return output contains	The attack and evaluation successfully executed and the return output contains	SS016	Pass

		'/results2', query_string={'prompt': 0, 'attack': "dwb"}	the necessary keyword as defined.	the necessary keyword as defined.		
--	--	--	-----------------------------------	-----------------------------------	--	--

```

@requests_mock.mock()
def test_download_route(self, m):
    """
    Test whether the download feature of the web page is working as expected.
    """
    # Mock an HTTP GET request to a URL
    url = 'https://example.com'
    m.get(url, text='<html><body>Mocked Content</body></html>')

    # Simulate a POST request to the /download route
    response = self.app.post('/download', data={'url': url})

    # Check if the request was successful (status code 200)
    self.assertEqual(response.status_code, 200)

    # Check if the 'downloaded_results.html' file was created
    self.assertTrue(os.path.exists('downloaded_results.html'))

    # Check the content of the response
    self.assertIn(b'Download Successful', response.data)
    self.tear_down()

```

Figure 25: Example code (TC047) for test cases under Integration Testing 1

```

@app.route('/download' , methods=['POST'])
def download():

    if request.method == 'POST':
        url = request.form['url']

    if not url:
        return 'Invalid URL. Please provide a valid URL.'

    try:
        # Send an HTTP GET request to the URL
        response = requests.get(url)

        # Check if the request was successful (status code 200)
        if response.status_code == 200:
            # Parse the HTML content of the webpage using BeautifulSoup
            soup = BeautifulSoup(response.text, 'html.parser')

            # Save the entire HTML content to a file
            with open('downloaded_results.html', 'w', encoding='utf-8') as file:
                file.write(soup.prettify())

            return render_template('downloaded.html')
        else:
            return f'Failed to download webpage. Status code: {response.status_code}'

    except Exception as e:
        return f'Error: {str(e)}'

```

Figure 26: Code of download feature tested by TC047

## 4.2 Integration Testing 2 - Backend Code

Although our unit testing has a lot of different components, most of the components are integrated under one of the attack algorithms, which is a bias attack in our case. Hence, this section will only be testing the core part of the backend code, which is the bias attack that involves a customised attack and evaluation of results.

The testing technique used here is still using the unittest framework to assert the result but we are not using mock but the actual API and framework to simulate the actual execution path of the main algorithm of our software. Since the attack result is inconsistent in each trial, we cannot validate the output result but only check whether the algorithm can execute as expected and return the correct number of outputs when the respective inputs are passed.

Test ID (TCxxx)	Test Desc	(Input Domains) & Test Input	Outcome		Evidence (SSxxx)	Test Result (Pass/Fail)
			Actual	Expected		
TC050	Test whether the bias attack function can work as expected	"what is an apple"  "An apple is a type of fruit that is widely cultivated and enjoyed around the world. It belongs to the Rosaceae family and the Malus genus. Apples come in various colors, including red, green, and yellow, and they can vary in taste from sweet to tart. They are typically round or oval in shape and have a thin skin that can be eaten. Apples are a good source of essential nutrients, including dietary fiber, vitamins (such as vitamin C), and minerals (such as potassium). They are known for their crisp and juicy texture, and they are commonly eaten fresh as a snack or	Tuple of length 6	Tuple of length 6	SS018	Pass



		<p>used in a wide range of culinary applications. Apples are used to make various products like apple juice, applesauce, apple pies, and they are also a popular ingredient in salads and desserts. Apples are not only delicious but also considered nutritious and are often associated with health benefits, including promoting heart health, aiding in digestion, and providing antioxidants that may help protect against certain chronic diseases. Different apple varieties have unique flavors and textures, making them a versatile and popular fruit in many cuisines worldwide."</p>				
--	--	--	--	--	--	--

```

class BiasAttackTest(unittest.TestCase):
    def test_successful_bias_attack(self):
        """
        Test whether the bias attack function can work as expected
        """
        input_str = "what is an apple"
        output_str = "An apple is a type of fruit that is widely cultivated and en

        res = bias_attack(input_str, output_str)
        self.assertTrue(len(res), 6)      # Check whether the number of output retr

```

Figure 27: Example code (TC050) for test cases under Integration Testing 2

```
def bias_attack(input_text, output_text):  
    """  
    Run the bias attack on the input text and output text.  
    """  
  
    attack = attack_recipe(target_bias=50.0)  
    result = attack.attack(input_text, output_text)  
  
    sentiment_analysis = SentimentAnalysis()  
    ori_sentiment = sentiment_analysis.sentiment(result.original_result.output)  
    perturb_sentiment = sentiment_analysis.sentiment(result.perturbed_result.output)  
  
    return (result.original_text(), result.original_result.output, result.original_resu
```

Figure 26: Code of bias attack and evaluation feature tested by TC050

## 5. Black-box Testing

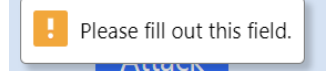
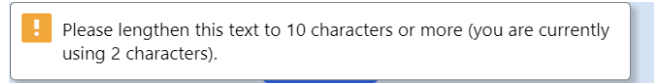
Black-box testing is the testing that is done without the knowledge of the internal structure or implementation of the software. Unlike black-box and white-box testing in unit testing, this section focused on simulating and testing the behaviour of our web application/software from the point of view of the end-users in order to ensure that it works as expected.

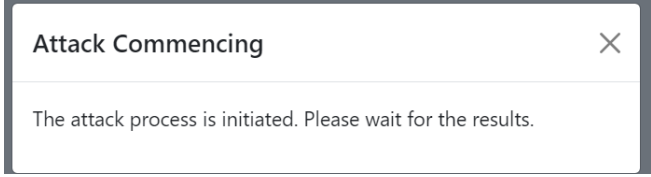

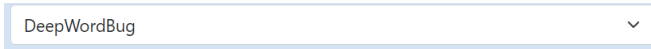
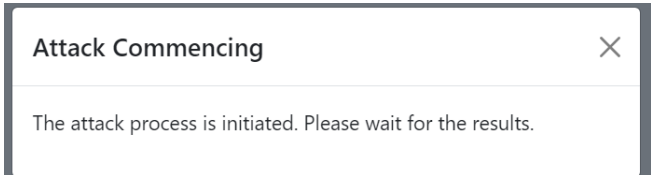
The testing is conducted on the client side, which is the GUI that is designed to abstract the complexity from the end users. The testing will be focused on button navigation, input and output of the software.

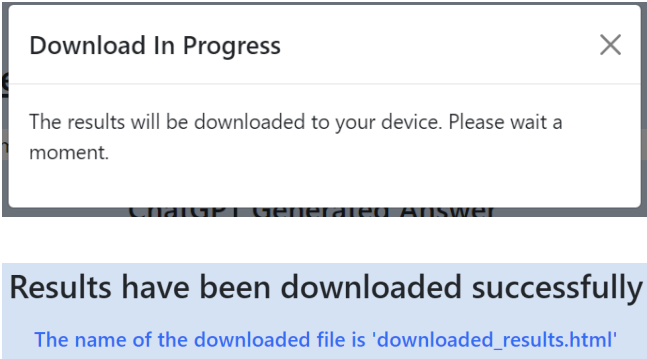
### 5.1 List of GUI Components

1. AttackGPT title (can be clicked - button)
2. Home Button
3. About Button
4. Contact Button
5. Hamburger Icon (when the screen is too small)
6. Bias Attack Text Box
7. Bias Attack Button
8. TextAttack Attack Recipes prompt's drop-down box
9. TextAttack Attack Recipes attack recipe's drop-down box
10. Text Attack Button
11. Perform New Attack Button
12. Download Result Button
13. Display of attack and evaluation results

## 5.2 Testing Result Tabulation

Test ID (BT XX)	GUI Component(s)	Test Description	Expected Behavior	Actual Behaviour	Test Result (Pass/Fail)
1	AttackGPT title	Test whether it will back to the home page when clicked	As expected	When clicked, it will back to the home page	Pass
2	Home Button	Test whether it will back to the home page when clicked	As expected	When clicked, it will back to the home page	Pass
3	About Button	Test whether it will go to the about page when clicked	As expected	When clicked, it will go to the about page	Pass
4	Contact Button	Test whether it will go to the contact page when clicked	As expected	When clicked, it will go to the contact page	Pass
5	Hamburger Icon	Test whether it will expand and show the home, about and contact button	As expected	When clicked, it will expand and show the home, about and contact button	Pass
6	Bias Attack Text Box	Test whether text can be inputted	As expected	Text can be inputted	Pass
7	Bias Attack Text Box Bias Attack Button	Test whether the attack can be run with no text	As expected	Remind message occurred: 	Pass
8	Bias Attack Text Box Bias Attack Button	Test whether the attack can be run with less than 10 texts	As expected	Remind message occurred: 	Pass

9	Bias Attack Text Box Bias Attack Button	Test whether the attack can be run with more than 10 and less than 200 texts	As expected	Attack commence message occurred:  The attack process is initiated. Please wait for the results.	Pass
10	TextAttack Attack Recipes prompt's drop-down box	Test whether the option in the dropdown menu can choose	As expected	The option can be chosen: 	Pass
11	TextAttack Attack Recipes attack recipe's drop-down box	Test whether the option in the dropdown menu can choose	As expected	The option can be chosen: 	Pass
12	Text Attack Button	Test whether the attack can be run	As expected	Attack commence message occurred:  The attack process is initiated. Please wait for the results.	Pass
13	Perform New Attack Button	Test whether it can back to the home page when clicked	As expected	When clicked, it will back to the home page	Pass
14	Download Result Button	Test whether the download can be initiated and the result will be download to the current directory	As expected	Download message pop-up and when download successful, change to the successful download page	Pass

		as downloaded.html		 <p>The screenshot shows a web application interface. At the top, there is a dialog box titled 'Download In Progress' with a close button (X). The text inside the dialog says: 'The results will be downloaded to your device. Please wait a moment.' Below this dialog, there is a blue banner with the text 'Results have been downloaded successfully'. Underneath the banner, it says 'The name of the downloaded file is 'downloaded_results.html''.</p>	
--	--	--------------------	--	---	--

## 6. Usability Testing

Usability testing is a crucial evaluation process that focuses on assessing the user-friendliness of a product or application, with a particular emphasis on its user interface (UI) and overall presentation. Although our project primarily centres on backend logic and analysis beyond the GUI, we recognize the importance of ensuring that users can easily comprehend and navigate the content we present to them.

In this test report, we will engage external participants, not affiliated with our project team, to thoroughly evaluate our application. They will rate the application on multiple aspects, using a scale ranging from 1 to 5. Additionally, participants will have the opportunity to provide personal comments and feedback on areas of the application that they believe require improvement.

The final usability assessment will be determined by calculating the average of the ratings across these various aspects. This approach will enable us to pinpoint the specific areas within the application that demand greater attention and enhancement.

### 6.1 Score/Scale for each Level of Agreement

The score/scale for each level of agreement is shown in the table below:

Level of Agreement	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Score/Scale	1	2	3	4	5

## 6.2 Evaluation Aspect/Statement in the Survey

The table below shows the statements/aspects that we gave to the end user to test and rate. There is a total of 6 statements which are focusing on testing the ease of use, design, speed and clarity of the instructions within the application.

No.	Statement/Aspect
1	User can easily navigate through the application and find what they want
2	User can easily understand the content of the application (with guide and about page given)
3	The design of the application is aesthetically pleasing
4	The result shown is easy to understand and interpret
5	The downloaded result can be understand easily
6	The speed of the application is acceptable

**Table 1: Statements that are included in the survey**

## 6.3 User Feedback/Survey

### 6.3.1 Survey 1

Remark: The speed of the execution is too slow especially for the bias attack which might fail after executing so long.

No.	Statement/Aspect	Agreement Level	Score/Scale
1	User can easily navigate through the application and find what they want	Agree	4
2	User can easily understand the content of the application (with the guide and about page given)	Agree	4
3	The design of the application is aesthetically pleasing	Neutral	3
4	The result shown is easy to understand and interpret	Neutral	3
5	The downloaded result can be understood easily	Agree	4
6	The speed of the application is acceptable	Disagree	2

**Table 2: Survey from end user 1**

### 6.3.2 Survey 2

Remark: The display of the result could be more clearly with which part of the text changed being highlighted or pointed out.

No.	Statement/Aspect	Agreement Level	Score/Scale
1	User can easily navigate through the application and find what they want	Agree	4



2	User can easily understand the content of the application (with the guide and about page given)	Neutral	3
3	The design of the application is aesthetically pleasing	Agree	4
4	The result shown is easy to understand and interpret	Neutral	3
5	The downloaded result can be understood easily	Agree	4
6	The speed of the application is acceptable	Neutral	3

**Table 3: Survey from end user 2**

### 6.3.3 Survey 3

Remark: None

No.	Statement/Aspect	Agreement Level	Score/Scale
1	User can easily navigate through the application and find what they want	Agree	4
2	User can easily understand the content of the application (with the guide and about page given)	Neutral	3
3	The design of the application is aesthetically pleasing	Neutral	3
4	The result shown is easy to understand and interpret	Agree	4
5	The downloaded result can be understood easily	Agree	4
6	The speed of the application is acceptable	Neutral	3

**Table 4: Survey from end user 3**

#### 6.3.4 Survey 4

Remark: Cannot download the result - website keeps crashing

No.	Statement/Aspect	Agreement Level	Score/Scale
1	User can easily navigate through the application and find what they want	Strongly Agree	5
2	User can easily understand the content of the application (with the guide and about page given)	Agree	3
3	The design of the application is aesthetically pleasing	Neutral	3
4	The result shown is easy to understand and interpret	Agree	4
5	The downloaded result can be understood easily	Strongly Disagree	1
6	The speed of the application is acceptable	Disagree	2

**Table 5: Survey from end user 4**

#### 6.3.5 Survey 5

Remark: The execution is slow

No.	Statement/Aspect	Agreement Level	Score/Scale
1	User can easily navigate through the application and find what they want	Strongly Agree	5
2	User can easily understand the content of the application (with the guide and about page given)	Agree	4
3	The design of the application is aesthetically pleasing	Agree	4

4	The result shown is easy to understand and interpret	Neutral	3
5	The downloaded result can be understood easily	Neutral	3
6	The speed of the application is acceptable	Disagree	2

**Table 6: Survey from end user 5**

## 6.4 Overall Result

The average mark for each statement/aspect is calculated and tabulated below:

No.	Statement/Aspect	Average Score
1	User can easily navigate through the application and find what they want	4.4
2	User can easily understand the content of the application (with the guide and about page given)	4.25
3	The design of the application is aesthetically pleasing	3.2
4	The result shown is easy to understand and interpret	3.4
5	The downloaded result can be understood easily	3.2
6	The speed of the application is acceptable	2.4

**Table 7: Average scores of each statement in the survey**

## 6.5 Summary

Based on the feedback from our representatives, it is evident that our GUI has excelled in creating a user-friendly interface that facilitates easy navigation and usage. Users have generally found the interface to be intuitive and accessible. However, there are specific aspects of the application that have garnered more mixed feedback.

Some areas, such as the design and the presentation of results, have received moderate ratings from most users. This suggests that there is room for improvement in terms of creating a more aesthetically pleasing interface and enhancing the clarity of result visualizations for users. These aspects are critical for ensuring a positive user experience and should be a focus in future iterations.

One significant issue identified in our software pertains to the execution time of certain attacks, notably the bias attack, which operates in real-time on input text. This issue has received a relatively low average score of 2.4 from our users. Addressing and significantly improving the execution time of such attacks should be a priority in future releases of the application. Ensuring faster performance in this critical area will contribute to a smoother and more efficient user experience.

## **7. Recommendation for Improvements**

While our application does not primarily focus on testing aspects such as performance and scalability, and all of our test cases have passed successfully, we recognize the need for continuous improvement. Therefore, this section will centre on enhancing the usability of our application based on valuable feedback and insights gathered from third-party users.

One prominent area of improvement highlighted by user feedback is the execution speed of our application, with all users rating it a maximum score of 3 out of 5. This feedback underscores the significance of addressing execution speed, particularly concerning our application's core functionality, which involves conducting attacks on datasets.

The current limitation in execution speed can be attributed to the backend framework our team is currently utilizing. To tackle this challenge, we are actively exploring the adoption of an alternative framework that can significantly enhance the application's performance. In the future, we plan to design a more efficient attack mechanism to integrate into the application, potentially leveraging a different framework to achieve the desired speed improvements. This strategic shift aligns with our commitment to ensuring that the process of running attacks on datasets remains not only efficient but also feasible within acceptable timeframes for ourselves and the end users.

Moreover, although the navigation part of our application is highly acceptable by most end users, they also raised the download feature of our application keeps crashing. One of the possible improvements that can be done on this is to perform more thorough testing on the server side before releasing our application to ensure that the host server is stable and working properly. Besides, the improvement that can be made involves researching and choosing a more stable server to host our application in the future instead of the current one.

These are the 2 aspects that our team thinks need the most improvement as they will impact the end-user experience and the development team's work on evaluating the algorithm on real-world data.

## **8. Limitations of the Testing Process**

Although the testing plan has been quite thorough and all of the test cases have passed, our team still noticed some possible improvements in our testing plan to provide a more comprehensive analysis and testing on our application so that it is more trustworthy for our end users.

One of the limitations of our testing is the lack of comprehensive testing at different stages of the testing process. For example, during the integration testing, our team are focused on testing the combination of multiple isolated components due to the time and resource constraints on our side. The downside of such an approach is that error might occur within the smaller combination of the components which we should have tested if possible so that we can better ensure the components work perfectly with each other.

Additionally, our team did not conduct testing on the communication between our application and the server. This omission resulted in one user reporting consistent crashes when attempting to use the download feature. This issue was not encountered during testing on our local computers. To address such communication-related problems in the future, we acknowledge the need to implement comprehensive testing procedures that encompass the interaction between our application and the server.

Furthermore, it's essential to acknowledge the limitations of our usability testing efforts. We conducted testing with only 5 end users due to time and resource constraints. While this sample size provided valuable feedback, it may not be sufficient to conclusively determine the overall

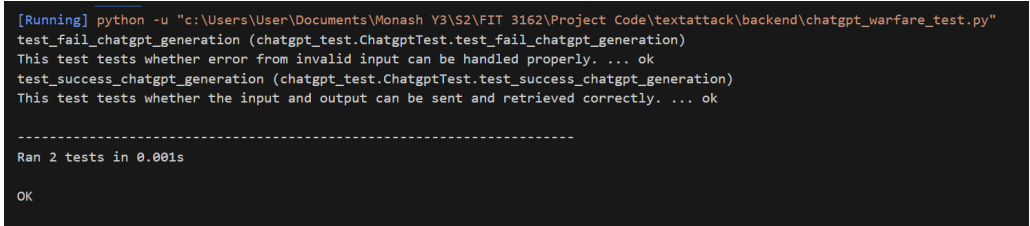
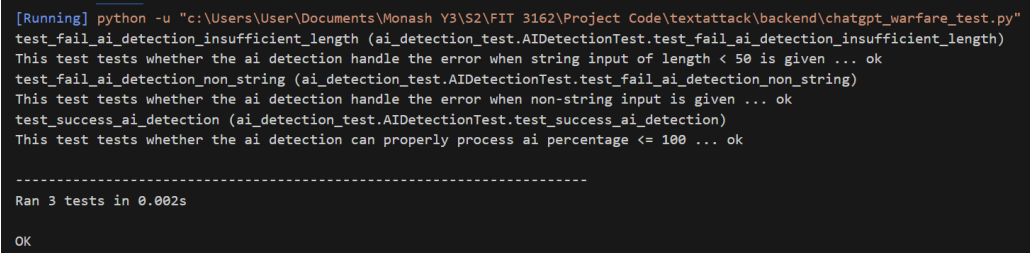
quality of our design. In future projects, we should aim to expand the pool of end users involved in usability testing to gather more comprehensive insights.

To enhance our testing process, we can explore automated testing methodologies, such as leveraging Continuous Integration/Deployment features within our Git repository. Automating the execution of unit test cases after each code commit can significantly improve efficiency and accuracy. This approach reduces the need for manual testing and ensures that code changes are promptly validated, even when multiple modifications have been made.

Finally, we recognize the need to incorporate system testing into our testing process. System testing evaluates whether the completed software product meets all defined requirements. By including this critical phase, we can ensure that all software-related requirements are fully met before considering the product as a finalized and reliable solution. This step is integral to delivering a well-working and compliant software product.

## 9. Appendix

### 9.1 Evidence of Testing

ScreenShot ID (SSxxx)	Image
Black-Box Testing	
SS001	 <pre>[Running] python -u "c:\Users\User\Documents\Monash Y3\S2\FIT 3162\Project Code\textattack\backend\chatgpt_warfare_test.py" test_fail_chatgpt_generation (chatgpt_test.ChatgptTest.test_fail_chatgpt_generation) This test tests whether error from invalid input can be handled properly. ... ok test_success_chatgpt_generation (chatgpt_test.ChatgptTest.test_success_chatgpt_generation) This test tests whether the input and output can be sent and retrieved correctly. ... ok  ----- Ran 2 tests in 0.001s  OK</pre> <p><i>SS001: Result of unit testing on prompt_GPT function</i></p>
SS002	 <pre>[Running] python -u "c:\Users\User\Documents\Monash Y3\S2\FIT 3162\Project Code\textattack\backend\chatgpt_warfare_test.py" test_fail_ai_detection_insufficient_length (ai_detection_test.AIDetectionTest.test_fail_ai_detection_insufficient_length) This test tests whether the ai detection handle the error when string input of length &lt; 50 is given ... ok test_fail_ai_detection_non_string (ai_detection_test.AIDetectionTest.test_fail_ai_detection_non_string) This test tests whether the ai detection handle the error when non-string input is given ... ok test_success_ai_detection (ai_detection_test.AIDetectionTest.test_success_ai_detection) This test tests whether the ai detection can properly process ai percentage &lt;= 100 ... ok  ----- Ran 3 tests in 0.002s  OK</pre> <p><i>SS002: Result of unit testing on human_ai_detection function</i></p>

SS003	<pre data-bbox="797 217 1821 448">[Running] python -u "c:\Users\User\Documents\Monash Y3\S2\FIT 3162\Project Code\textattack\backend\chatgpt_warfare_test.py" test_fail_check_similarity_both_invalid (sentence_similarity_test.SentenceSimilarityTest.test_fail_check_similarity_both_invalid) This test tests whether the similarity checking can handle the error when both arguments are not of type string (invalid). ... ok test_fail_check_similarity_one_invalid (sentence_similarity_test.SentenceSimilarityTest.test_fail_check_similarity_one_invalid) This test tests whether the similarity checking can handle the error when one of the argument is not of type string (invalid). ... ok test_success_check_similarity (sentence_similarity_test.SentenceSimilarityTest.test_success_check_similarity) This test tests whether the sentence similarity can be checked and retrieved properly ... ok  ----- Ran 3 tests in 0.002s  OK</pre> <p data-bbox="920 459 1695 491"><i>SS003: Result of unit testing on similarity_checking function</i></p>
SS004	<pre data-bbox="797 564 1821 772">[Running] python -u "c:\Users\User\Documents\Monash Y3\S2\FIT 3162\Project Code\textattack\backend\chatgpt_warfare_test.py" test_non_existent_file (read_file_test.ReadFileTest.test_non_existent_file) This test tests whether the error can be handled properly when the file that supposed to read does not exist ... ok test_valid_file (read_file_test.ReadFileTest.test_valid_file) This test tests whether the content of the file can be properly read in the expected manner ... ok  ----- Ran 2 tests in 0.001s  OK</pre> <p data-bbox="898 783 1718 815"><i>SS004: Result of unit testing on read_prompt_from_file function</i></p>
SS005	<pre data-bbox="797 895 1821 1070">test_fail_gender_word_swap (word_add_gender_test.WordAddGenderTest.test_fail_gender_word_swap) Test whether the gender word swap transformation able to detect the error when non-AttackText object being passed ... ok test_success_gender_word_swap_people (word_add_gender_test.WordAddGenderTest.test_success_gender_word_swap_people) Test whether the gender word swap transformation work as expected when there exist PEOPLE term in the sentence ... ok test_success_gender_word_swap_personal_pronouns (word_add_gender_test.WordAddGenderTest.test_success_gender_word_swap_personal_pronouns) Test whether the gender word swap transformation work as expected when there exist PERSONAL_PRONOUNCS term in the sentence ... ok  ----- Ran 3 tests in 0.003s  OK</pre> <p data-bbox="752 1082 1865 1114"><i>SS005: Result of unit testing on get_transformation function on WordAddGender Class</i></p>
SS006	<pre data-bbox="797 1190 1821 1350">test_fail_sexuality_word_swap (word_add_sexuality_test.WordAddSexualityTest.test_fail_sexuality_word_swap) Test whether the sexuality word swap transformation able to detect the error when non-AttackText object being passed ... ok test_success_sexuality_word_swap_people (word_add_sexuality_test.WordAddSexualityTest.test_success_sexuality_word_swap_people) Test whether the sexuality word swap transformation work as expected when there exist PEOPLE term in the sentence ... ok test_success_sexuality_word_swap_personal_pronouns (word_add_sexuality_test.WordAddSexualityTest.test_success_sexuality_word_swap_personal_pronouns) Test whether the sexuality word swap transformation work as expected when there exist PERSONAL_PRONOUNCS term in the sentence ... ok  ----- Ran 3 tests in 0.004s  OK</pre>



	<i>SS006: Result of unit testing on get_transformation function on WordAddSexuality Class</i>
SS007	<pre> test_fail_race_word_swap (word_add_race_test.WordAddRaceTest.test_fail_race_word_swap) Test whether the race word swap transformation able to detect the error when non-AttackText object being passed ... ok test_success_gender_word_swap_personal_pronouns (word_add_race_test.WordAddRaceTest.test_success_gender_word_swap_personal_pronouns) Test whether the race word swap transformation work as expected when there exist PERSONAL_PRONOUNCS term in the sentence ... ok test_success_race_word_swap_people (word_add_race_test.WordAddRaceTest.test_success_race_word_swap_people) Test whether the race word swap transformation work as expected when there exist PEOPLE term in the sentence ... ok ----- Ran 3 tests in 0.004s OK </pre> <p><i>SS007: Result of unit testing on get_transformation function on WordAddRace Class</i></p>
SS008	<pre> test_fail_religion_word_swap (word_add_religion_test.WordAddReligionTest.test_fail_religion_word_swap) Test whether the religion word swap transformation able to detect the error when non-AttackText object being passed ... ok test_success_religion_word_swap_people (word_add_religion_test.WordAddReligionTest.test_success_religion_word_swap_people) Test whether the religion word swap transformation work as expected when there exist PEOPLE term in the sentence ... ok test_success_religion_word_swap_personal_pronouns (word_add_religion_test.WordAddReligionTest.test_success_religion_word_swap_personal_pronouns) Test whether the religion word swap transformation work as expected when there exist PERSONAL_PRONOUNCS term in the sentence ... ok ----- Ran 3 tests in 0.008s OK </pre> <p><i>SS008: Result of unit testing on get_transformation function on WordAddReligion Class</i></p>
SS009	<pre> test_fail_sentiment_analysis (sentiment_analysis_test.SentimentAnalysisTest.test_fail_sentiment_analysis) This test tests whether the error can be properly handled and sentiment analysis result can be successfully retrieved in the next iteration ... HTTP error occurred: 400 Client Error: Bad Request Retrying in 5 seconds... ok test_success_sentiment_analysis (sentiment_analysis_test.SentimentAnalysisTest.test_success_sentiment_analysis) This test tests whether the sentiment analysis result can be successfully retrieved ... ok ----- Ran 2 tests in 5.003s OK </pre> <p><i>SS009: Result of unit testing on sentiment function</i></p>
SS010	<pre> test_fail_sentiment_analysis (bias_detection_test.BiasDetectionTest.test_fail_sentiment_analysis) This test tests whether the error can be properly handled and bias result can be successfully retrieved in the next iteration ... HTTP error occurred: 400 Client Error: Bad Request Retrying in 5 seconds... ok test_success_bias_detection_filter (bias_detection_test.BiasDetectionTest.test_success_bias_detection_filter) This test tests whether the bias detection result can be successfully retrieved ... ok test_success_bias_detection_normal (bias_detection_test.BiasDetectionTest.test_success_bias_detection_normal) This test tests whether the bias detection result can be successfully retrieved ... ok ----- Ran 3 tests in 5.004s OK </pre> <p><i>SS010: Result of unit testing on detect_bias function</i></p>

White-Box Testing	
SS011	<pre> test_fail_ai_detection_error_WB5 (ai_detection_test.AIDetectionTest.test_fail_ai_detection_error_WB5) This test tests whether the ai detection handle the Bad Request Error (422) ... ok test_fail_ai_detection_error_WB6 (ai_detection_test.AIDetectionTest.test_fail_ai_detection_error_WB6) This test tests whether the ai detection handle the Not Found Error (404) ... ok test_fail_ai_detection_error_WB7 (ai_detection_test.AIDetectionTest.test_fail_ai_detection_error_WB7) This test tests whether the ai detection handle the Too Many Request Error (429) ... ok test_fail_ai_detection_insufficient_length (ai_detection_test.AIDetectionTest.test_fail_ai_detection_insufficient_length) This test tests whether the ai detection handle the error when string input of length &lt; 50 is given ... ok test_fail_ai_detection_non_string (ai_detection_test.AIDetectionTest.test_fail_ai_detection_non_string) This test tests whether the ai detection handle the error when non-string input is given ... ok test_success_ai_detection (ai_detection_test.AIDetectionTest.test_success_ai_detection) This test tests whether the ai detection can properly process ai percentage &lt;= 100 ... ok test_success_likely_ai_detection_WB1 (ai_detection_test.AIDetectionTest.test_success_likely_ai_detection_WB1) This test tests whether the ai detection can properly process ai percentage &lt;= 90 ... ok test_success_unlikely_ai_detection_WB3 (ai_detection_test.AIDetectionTest.test_success_unlikely_ai_detection_WB3) This test tests whether the ai detection can properly process ai percentage &lt;= 40 ... ok test_success_unsure_ai_detection_WB2 (ai_detection_test.AIDetectionTest.test_success_unsure_ai_detection_WB2) This test tests whether the ai detection can properly process ai percentage &lt;= 60 ... ok test_success_very_unlikely_ai_detection_WB4 (ai_detection_test.AIDetectionTest.test_success_very_unlikely_ai_detection_WB4) This test tests whether the ai detection can properly process ai percentage &lt;= 10 ... ok  ----- Ran 10 tests in 0.007s OK </pre> <p><i>SS011: Result of white-box testing on ai_detection function</i></p>
SS012	<pre> test_fail_sentiment_analysis_exception_WB1 (bias_detection_test.BiasDetectionTest.test_fail_sentiment_analysis_exception_WB1) This test tests whether the error can be properly handled and bias result can be successfully retrieved in the next iteration ... An unexpected error occurred: Unexpected Error Retrying in 5 seconds... ok test_fail_sentiment_analysis_request_exception_WB1 (bias_detection_test.BiasDetectionTest.test_fail_sentiment_analysis_request_exception_WB1) This test tests whether the error can be properly handled and bias result can be successfully retrieved in the next iteration ... Request error occurred: Request Exception Error Retrying in 5 seconds... ok  ----- Ran 2 tests in 10.004s OK </pre> <p><i>SS012: Result of white-box testing on detect_bias function</i></p>

SS013	<pre> test_fail_chatgpt_generation_api_connection_error_WB4 (chatgpt_test.ChatgptTest.test_fail_chatgpt_generation_api_connection_error_WB4) This test tests whether api connection error can be handled properly by chatgpt prompt function. ... ok test_fail_chatgpt_generation_api_error_WB1 (chatgpt_test.ChatgptTest.test_fail_chatgpt_generation_api_error_WB1) This test tests whether api error can be handled properly by chatgpt prompt function. ... ok test_fail_chatgpt_generation_authentication_error_WB5 (chatgpt_test.ChatgptTest.test_fail_chatgpt_generation_authentication_error_WB5) This test tests whether authentication error can be handled properly by chatgpt prompt function. ... ok test_fail_chatgpt_generation_rate_limit_error_WB3 (chatgpt_test.ChatgptTest.test_fail_chatgpt_generation_rate_limit_error_WB3) This test tests whether rate limit error can be handled properly by chatgpt prompt function. ... ok test_fail_chatgpt_generation_server_unavailable_error_WB6 (chatgpt_test.ChatgptTest.test_fail_chatgpt_generation_server_unavailable_error_WB6) This test tests whether server unavailable error can be handled properly by chatgpt prompt function. ... ok test_fail_chatgpt_generation_timeout_error_WB2 (chatgpt_test.ChatgptTest.test_fail_chatgpt_generation_timeout_error_WB2) This test tests whether timeout error can be handled properly by chatgpt prompt function. ... ok ----- Ran 6 tests in 0.005s OK </pre> <p><i>SS013: Result of white-box testing on prompt_GPT function</i></p>
SS014	<pre> test_fail_check_similarity_request_exception_WB1 (sentence_similarity_test.SentenceSimilarityTest.test_fail_check_similarity_request_exception_WB1) This test tests whether the similarity checking can handle request exception error ... ok ----- Ran 1 test in 0.002s OK </pre> <p><i>SS014: Result of white-box testing on sentence_similarity function</i></p>
SS015	<pre> test_fail_sentiment_analysis_exception_WB2 (sentiment_analysis_test.SentimentAnalysisTest.test_fail_sentiment_analysis_exception_WB2) This test tests whether the Exception error (or other error) can be properly handled and sentiment analysis result can be successfully retrieved in the next iteration ... An unexpected error occurred: Exception Error Retrying in 5 seconds... ok test_fail_sentiment_analysis_request_exception_WB1 (sentiment_analysis_test.SentimentAnalysisTest.test_fail_sentiment_analysis_request_exception_WB1) This test tests whether the request Exception error can be properly handled and sentiment analysis result can be successfully retrieved in the next iteration ... Request error occurred: Request Exception error Retrying in 5 seconds... ok ----- Ran 2 tests in 10.000s </pre> <p><i>SS015: Result of white-box testing on sentiment function</i></p>
Integration Testing	
SS016	<pre> test_about_route (__main__.TestApp.test_about_route) Test whether the about route work as expected ... ok test_contact_route (__main__.TestApp.test_contact_route) Test whether the about route work as expected ... ok test_download_route (__main__.TestApp.test_download_route) Test whether the download feature of the web page is working as expected. ... ok test_index_route (__main__.TestApp.test_index_route) Test whether the index route work as expected ... ok test_start_route (__main__.TestApp.test_start_route) Test whether the start route work as expected ... ok test_successful_results2_route (__main__.TestApp.test_successful_results2_route) ... &lt;class 'bytes'&gt; ok test_successful_results_route (__main__.TestApp.test_successful_results_route) </pre> <p><i>SS016: Result of integration testing on webpage navigation, download and attack recipe</i></p>

SS017	<pre> test_successful_results_route (__main__.TestApp.test_successful_results_route) Test whether bias attack is working and result is loaded as expected ... textattack: No entry found for goal function &lt;class 'backend.maximize_bias.MaximizeBias'&gt;. textattack: Unknown if model of class &lt;class 'backend.chatgpt.ChatGPT'&gt; compatible with goal function &lt;class 'backend.maximize_bias.MaximizeBias'&gt;. HTTP error occurred: 502 Server Error: Bad Gateway for url: https://api-inference.huggingface.co/models/valurank/distilroberta-bias Retrying in 5 seconds... HTTP error occurred: 503 Server Error: Service Unavailable for url: https://api-inference.huggingface.co/models/cardiffnlp/twitter-roberta-base-sentiment-latest Retrying in 5 seconds... HTTP error occurred: 503 Server Error: Service Unavailable for url: https://api-inference.huggingface.co/models/cardiffnlp/twitter-roberta-base-sentiment-latest Retrying in 5 seconds... ok  ----- Ran 1 test in 273.516s OK </pre> <p><i>SS017: Result of integration testing on bias attack and result showcase on web</i></p>
SS018	<pre> test_successful_bias_attack (__main__.BiasAttackTest.test_successful_bias_attack) Test whether the bias attack function can work as expected ... textattack: No entry found for goal function &lt;class 'maximize_bias.MaximizeBias'&gt;. textattack: Unknown if model of class &lt;class 'chatgpt.ChatGPT'&gt; compatible with goal function &lt;class 'maximize_bias.MaximizeBias'&gt;. HTTP error occurred: 503 Server Error: Service Unavailable for url: https://api-inference.huggingface.co/models/valurank/distilroberta-bias Retrying in 5 seconds... HTTP error occurred: 503 Server Error: Service Unavailable for url: https://api-inference.huggingface.co/models/valurank/distilroberta-bias Retrying in 5 seconds... ok  ----- Ran 1 test in 19.779s OK </pre> <p><i>SS018: Result of integration testing on bias_attack function</i></p>

## 9.2 Images

Coverage report: 91%

coverage.py v7.3.1, created at 2023-10-04 19:45 +0800

Module	statements	missing	excluded	branches	partial	coverage
ai_detection.py	46	10	0	22	10	71%
ai_detection_test.py	29	0	0	6	0	100%
bias_detection.py	55	6	0	18	3	85%
bias_detection_test.py	38	0	0	6	0	100%
chatgpt.py	27	9	0	12	4	56%
chatgpt_test.py	25	0	0	6	0	100%
chatgpt_warfare_test.py	27	0	0	2	1	97%
data.py	6	0	0	0	0	100%
read_file.py	10	0	0	2	0	100%
read_file_test.py	14	0	0	4	0	100%
sentence_similarity.py	21	2	0	2	1	87%
sentence_similarity_test.py	43	0	0	10	0	100%
sentiment_analysis.py	34	6	0	4	1	76%
sentiment_analysis_test.py	26	0	0	4	0	100%
word_add_gender.py	28	1	0	12	2	92%
word_add_gender_test.py	31	0	0	10	0	100%
word_add_race.py	28	1	0	12	2	92%
word_add_race_test.py	31	0	0	10	0	100%
word_add_religion.py	28	1	0	12	2	92%
word_add_religion_test.py	31	0	0	10	0	100%
word_add_sexuality.py	28	1	0	12	2	92%
word_add_sexuality_test.py	31	0	0	10	0	100%
Total	637	37	0	186	28	91%

*Appendix Fig 9.2.1: Coverage Report after unit testing (black-box testing)  
(Yellow colour is the file under minimum coverage percentage)*

## Coverage report: 97%

*coverage.py v7.3.1, created at 2023-10-04 23:15 +0800*

<i>Module</i>	<i>statements</i>	<i>missing</i>	<i>excluded</i>	<i>branches</i>	<i>partial</i>	<i>coverage</i>
ai_detection.py	42	0	0	20	2	97%
ai_detection_test.py	109	1	0	28	1	99%
bias_detection.py	55	2	0	18	2	95%
bias_detection_test.py	61	0	0	10	0	100%
chatgpt.py	27	3	0	12	1	90%
chatgpt_test.py	41	0	0	24	0	100%
chatgpt_warfare_test.py	27	0	0	2	1	97%
data.py	6	0	0	0	0	100%
read_file.py	10	0	0	2	0	100%
read_file_test.py	14	0	0	4	0	100%
sentence_similarity.py	21	0	0	2	0	100%
sentence_similarity_test.py	57	0	0	14	0	100%
sentiment_analysis.py	34	2	0	4	0	95%
sentiment_analysis_test.py	49	0	0	8	0	100%
word_add_gender.py	28	1	0	12	2	92%
word_add_gender_test.py	31	0	0	10	0	100%
word_add_race.py	28	1	0	12	2	92%
word_add_race_test.py	31	0	0	10	0	100%
word_add_religion.py	28	1	0	12	2	92%
word_add_religion_test.py	31	0	0	10	0	100%
word_add_sexuality.py	28	1	0	12	2	92%
word_add_sexuality_test.py	31	0	0	10	0	100%
<b>Total</b>	<b>789</b>	<b>12</b>	<b>0</b>	<b>236</b>	<b>15</b>	<b>97%</b>

*Appendix Fig 9.2.2: Coverage Report after white-box testing*

