Description of main product and outcomes of the research

Haoyu Chen
Faculty of Sciences, Engineering and Technology
The University of Adelaide
Adelaide Australia
a1867006@adelaide.edu.au

***Abstract*—In this report, we explain the development of a Chrome extension called "Browser Shield", which is designed to detect phishing or malicious websites and improve the quality of the user's safe online environment by monitoring and analyzing in real-time the URLs that the user enters and accesses to detect the risk of phishing websites. The extension has the following features: instant whitelist and blacklist checking of the URLs accessed by the user, the whitelist can be constantly updated, the blacklist is currently set to be updated every 3 days, and machine learning predictions are made if the URL address the user is trying to access is neither in the whitelist nor in the blacklist, the machine learning model is trained and stored on the server side and all the predictions are made on the server side The machine learning model is trained and stored on the server side and all predictions are made on the server side in order to serve some users with older computers, and the predictions are fed into the UI and icon. The plugin also allows users to make suspicious URL submissions, which are filtered by a suggested CAPTCHA system and a URL validation plugin to enhance server-side system security.**

Keywords –Phishing attacks, Blacklist, Machine learning, CAPTCHA, Security.

I. SYSTEM DESIGN AND IMPLEMENTATION

The system design initially designed by 3 parts: blacklist, classifier, and machine learning prediction, but after finishing the project A, the plan has changed because whitelist is very useful to improve the performance, and the machine learning part has already included some classifier functional, so in the end the design has changed to whitelist, blacklist and machine learning prediction. The flowchart below shows the logic of the extension tool:

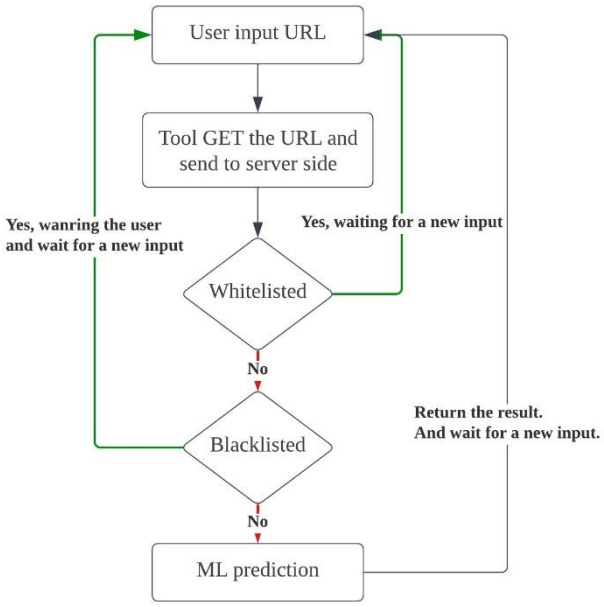

Fig 1. Flowchart of the system design (detection part).

All the whitelist, blacklist, and machine learning detection or prediction will be done on the server side. Since after improving the machine learning method, the performance has a huge improvement, so it is easily handling all 3 methods on the server side instead of putting the blacklist check on the user side for saving resources and time as we planned in project A.

Detailed progress will be listed below.

For the extension tool itself:

1. Open the Browser/Start
2. User Visits/input a URL.
    - The background script captures the URL.

3. Check if URL is in Local Storage (Whitelist)
    - Yes: Display "Safe" on Popup.
    - No: Proceed to Check Blacklist.
4. Check if URL is in Local Storage (Blacklist)
    - Yes: Display "Unsafe" on Popup.
    - No: Proceed to Server Validation.
5. Send URL to Server for Validation
    - Server checks against server-side blacklist.
    - Server runs URL through the Machine Learning model.
6. Receive Server Response
    - Safe: Update Local Storage (Whitelist), Display "Safe" on Popup.
    - Unsafe: Update Local Storage (Blacklist), Display "Unsafe" on Popup.
    - Unknown: Display "Unknown" on Popup.
7. User Interacts with Popup
    - Choose to report a URL.
8. Display CAPTCHA in Popup
    - Generate CAPTCHA and display it to the user.
9. User Submits Reported URL and CAPTCHA
    - CAPTCHA Correct?
        - Yes: Proceed to Save URL.
        - No: Increment CAPTCHA Fail Count.
10. Check CAPTCHA Fail Count
    - Less than 3 times failure Allow Retry.
    - 3 or More: Block User for 24 hours.
11. Save Reported URL and Log Entry
    - Validate Reported URL.
    - Save valid URL to reported_urls.csv.
    - Log the attempt to submission_log.txt.
12. End

For developers:

Manually download and update the blacklist from the PhishTank: http://data.phishtank.com/data/online-valid.csv, add more whitelist URLs if needed. Simply run the update_blacklist.py file will extract the URL from the downloaded csv file, will also add an updating time to the log.

The "Browser Shield" extension consists of several components:

1. Icon: the icon is not only a fixed figure, but it will also change depending on the result returned from the server side. There are 4 different types of icons, error, normal, safe and phishing:

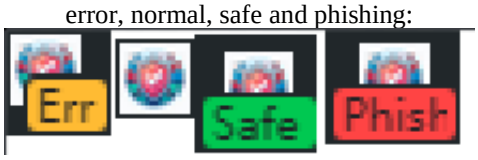

Fig 2. 4 different icons depending on status.

2. User Interface (UI): Presented to the user via pop-up windows, including a security status indicator, an input box for suspicious URLs, a CAPTCHA display and input box, and a red report button. The UI is designed to be clean and simple, with the goal of allowing users to quickly understand the security status of the current website and to report suspicious URLs if needed. the UI displays a green security icon for websites that are recognized as secure, and a red warning icon for websites that are recognized as insecure.

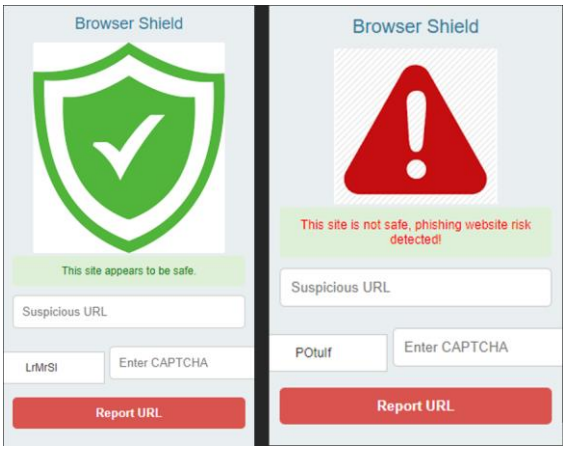

Fig 3. Safe/unsafe feedback and message of the popup UI.

3. Background Script: Checks URLs in real time while the user is browsing, determines their security status, and updates the UI. The background script also interacts closely with the user interface (UI). As soon as it detects a suspicious web request, it updates the badge text of the browser action, providing the user with intuitive feedback on the security status. In this way, the extension not only increases the user's trust, but also improves its actual security. For example, when a user visits a safe URL, the badge will display "Safe", and the green background of the badge becomes a safety shield in the user's mind. On the contrary, when visiting an unsafe URL, the badge displays "Phish", and the badge with a red background act as a warning flag to alert the user of potential risks.

To increase reliability, the background script also handles error conditions. If the server responds abnormally, or if the user has a problem with a network request, it will catch these errors and update the badge text to "Err", while ensuring that the user notices the change in status. This design considers the uncertainty of the network environment and ensures that the user receives the necessary feedback even in a changing network environment.

The background.js file is designed for checking the returning result from the server side and some other features .with a timed-triggered blacklist update mechanism, the extension can pull the latest blacklist data from the server at regular intervals, ensuring that even if users do not manually update the extension for a long period of time, its protection is still up to date.

Finally, to ensure a consistent user experience, the background script ensures that the badge text is reset whenever the extension is reloaded or installed. This ensures that the extension appears in a clean, consistent state every time the user launches the browser. Additionally, since the extension runs silently behind the scenes, special care was taken to ensure its performance efficiency and avoid any impact on the performance of the user's device.

4. Flask Server Side: Handles blacklist and whitelist comparison, performs machine learning model predictions, and logs suspicious URLs reported, updating datetime by users, developer updating blacklist and whitelist logs.

II. Machine Learning Prediction

We trained a logistic regression model to recognize phishing websites. The model used textual features of URLs as input and was obtained by training on historical phishing site data. The model achieved 96.5% accuracy on the test set.

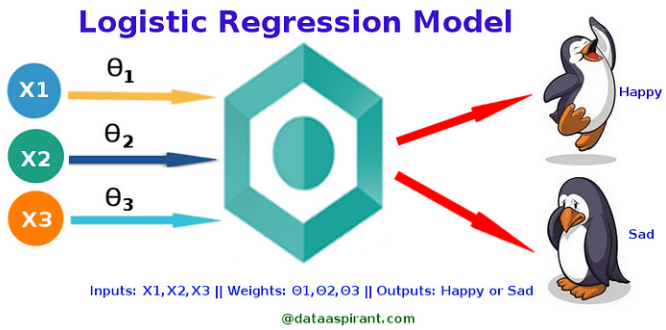

Fig 4. Logistic regression model example [1].

In the preprocessing step, we employ tokenization and stemming. Tokenization breaks down URLs into individual words or symbols, which helps us to identify and analyze patterns in URLs. Stemming extraction, on the other hand, removes prefixes and suffixes from words to summarize different forms of words into their basic forms. This reduces the number of features that the model needs to process while maintaining the ability to capture the underlying meaning in the original text.

The dataset used to train this model has 549346 samples, the URLs used for training are still considered to be relatively new and recent in date, so they are very suitable for training and prediction, and the accuracy of the resulting TESTs is very high, and a sample of the results will be given later in the OUTCOMES section. The model gets a big performance improvement compared to the previous versions of machine learning models, the reason is that the new dataset is very simple, only provides the URL and condition, the advantage is that we do not need to do a lot of preprocessing of the user input URL as in the previous models, the previous models need to carry out complex preprocessing led to a large amount of information from URLs. The previous model required complex preprocessing, which resulted in the features extracted from the URL not matching the

features needed by the model in some cases, leading to various errors or false positives. The new model does not have this problem.

III. CAPTCHA AND LOGS

To prevent malicious reports and automated attacks, we designed and implemented a simple CAPTCHA system. The system displays a randomly generated CAPTCHA when a user reports a suspicious URL, which the user needs to enter correctly to complete the report. If the user enters it incorrectly three times in a row, the system will prevent the user from continuing the report for 24 hours.

The "Browser Shield" extension has a dynamic CAPTCHA feature for verifying the authenticity of an action when a user reports a suspicious URL. The system challenges the user by generating a random CAPTCHA to ensure that the reported behavior was performed by a real user and not an automated script such that it protects server-side data security.

Workflow

1. CAPTCHA Generation:
    - Whenever the user interaction interface (UI) loads, the system automatically triggers a function to generate a random CAPTCHA consisting of a combination of 6 digits and letters.
    - The generated CAPTCHA is instantly displayed in the extension's pop-up window, and the user is required to enter the correct CAPTCHA before submitting a suspicious URL.
2. User input processing:
    - The user fills in the input box with the CAPTCHA and submits the suspicious URL.
    - The system will check if the CAPTCHA entered by the user is the same as the one displayed.
3. Verification Result:
    - If the CAPTCHA is correct, the user's report is sent to the locally running Flask server for further processing.
    - If the CAPTCHA is entered incorrectly, the system adds a count of incorrect attempts and alerts the user that the CAPTCHA is incorrect.
4. Security Measures:
    - If a user enters an incorrect CAPTCHA three times in a row, the system automatically locks the user's reporting capabilities to prevent malicious attempts and automated attacks.
    - Locked users will need to wait 24 hours before attempting to submit again.
5. User Feedback:
    - Regardless of success or failure, the user will receive a corresponding feedback message on the interface to confirm the status of the report or the error message.

Technical Implementation

The CAPTCHA system is implemented using JavaScript and the Chrome extension API, which is integrated in the front-end of the extension and in the interaction with the back-end server. Chrome.storage.local is utilized to track the number of attempts and block status of the user, ensuring that the user's restricted status is retained even if the extension or browser is restarted.

Security and User Experience

This CAPTCHA system protects users from phishing attacks while preventing automated tools from abusing reporting capabilities. Despite the added complexity of user operation, this trade-off is necessary given the increased security. The system was designed with special consideration for the intuitiveness of the user interface and the smoothness of user interaction to ensure that it does not cause too much disruption to normal user use.

Log generation and update

For the log part, the log file will be updated when a user successfully uploads a suspicious website. The log will record the time of submission, the URL address of the suspicious URL submitted by the user, and the address will be stored in a

separate server-side csv file, which can be processed by the developer to remove duplicate URLs at regular intervals and validate them manually or through an external third-party service. Users can submit suspicious websites that have been detected, or false positives. Developers can continuously maintain and update the blacklist by submitting and analyzing the results of subsequent blacklist updates to improve the accuracy of the extension.

IV. OUTCOMES AND RESULT

- **Whitelist testing:**

We need to do some basic testing for the whitelist method. We have saved 20 whitelisted URLs in the database. And we expected if the main domain is in the whitelist, then the extension tool will show the safe message and stop after return the status to user side.

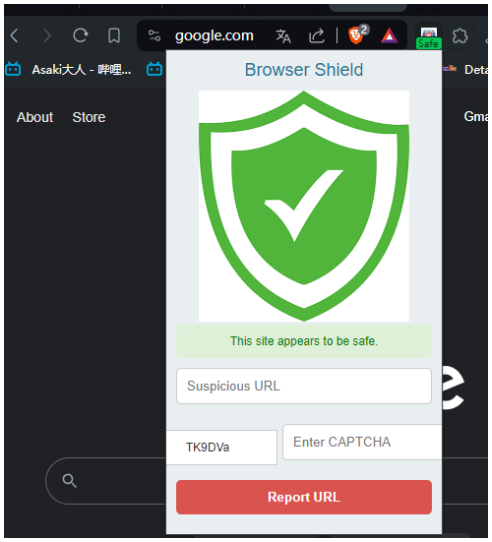

Fig 5. Whitelist test of google.com

Fig 6. Whitelist test of google.com (terminal screenshot)

As expected, the whitelisted is working perfectly. Next text will be searching something on the google so the URL will be changed:

Fig 7. Whitelist test of main domain whitelisted (google.com)

Fig 8. Whitelist test of searching URL of google (terminal screenshot)

Since the main domain of this long URL in the screenshot above also in the whitelist, the result is as expected. It returns safe and popup the safe message.

- **Blacklist testing:**

Blacklist testing is straighter forward, the blacklist is extract from the verified_online.csv file, then save to blacklist.db database file. Since the URLs in the database clearly not in the whitelist, it will fail to match the whitelist then pass to blacklist check. We have choose a URL from the blacklist:

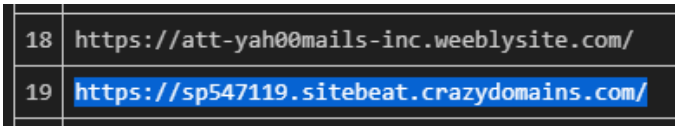

Fig 9. Blacklist example from blacklist.db

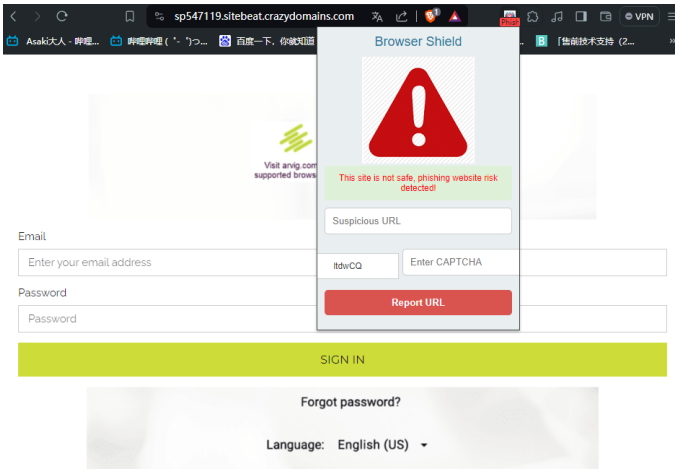

Fig 10. Blacklist test of the chosen example.

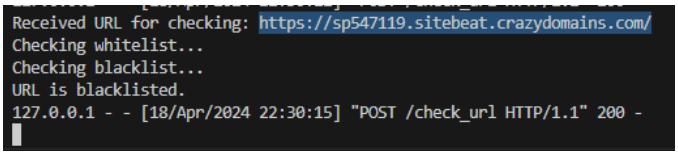

Fig 11. Blacklist terminal debug screenshot.

As the result we can see on the above screenshot, we can see that the blacklist is also working as expected.

- **Machine learning prediction:**

  Once the URL neither in the whitelist nor blacklist. The server side will hand over the URL to the machine learning model to do the predictions.
  Given a simple example of the Adelaide Uni website page. The URL is: https://www.adelaide.edu.au/student/

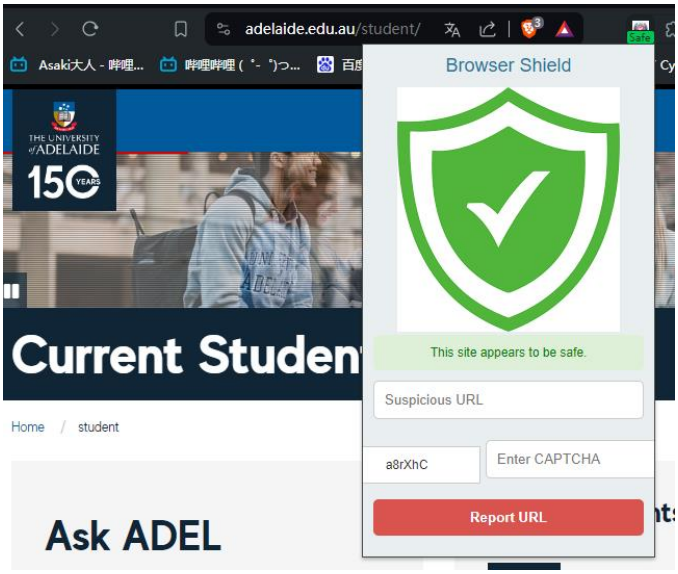

Fig 12. The screenshot of Adelaide Uni website.

Fig 13. The screenshot of the result of ML prediction of https://www.adelaide.edu.au/student/

As we can see the extension tool have checked both whitelist and blacklist then found the URL not in there, then the machine learning method start to tokenize and stemmed the URL, after getting necessary features, model predicted it and given a “good” result then return the “safe” status to the user side and let the popup UI shows the safe message.

We have manually tested about 50 URLs we commonly use in our daily life but not in the whitelist, they all are showing the safe message which is highly accuracy.

We have also tested a real-life case, a scam AU Post link which we received via the phone message, the link is https://aupost.auhrly.express:

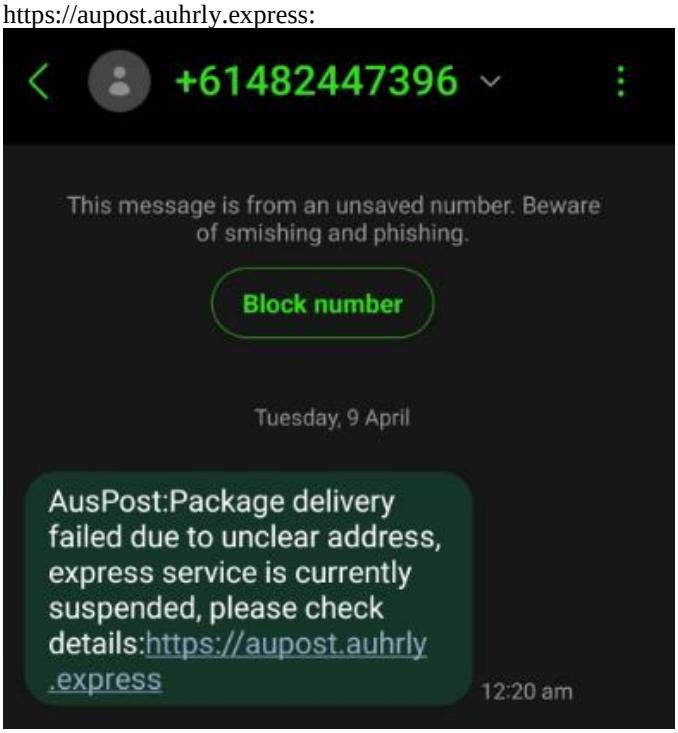

Fig 14. Scam message screenshot from mobile phone.

Although this website is down already for some reason but we can still see that the result shows this website is a

phishing website:

Fig 15. Real-world scam URL testing

```
Received URL for checking: https://aupost.auhrly.express/
Checking whitelist...
Checking blacklist...
Original URL: https://aupost.auhrly.express/
Cleaned URL: aupost.auhrly.express/
Tokens: ['aupost', 'auhrly', 'express']
Stemmed Tokens: ['aupost', 'auhr', 'express']
Processed URL for Prediction: aupost auhr express
Prediction:  bad
127.0.0.1 - - [18/Apr/2024 22:43:10] "POST /check_url HTTP/1.1" 200 -
```

Fig 16. Terminal feedback of scam URL testing

As we can see that this URL neither in the whitelist nor blacklist and the prediction did an excellent work.

◆ False negative/False positive case

Luckily, I found a false negative case of the call of duty official website:

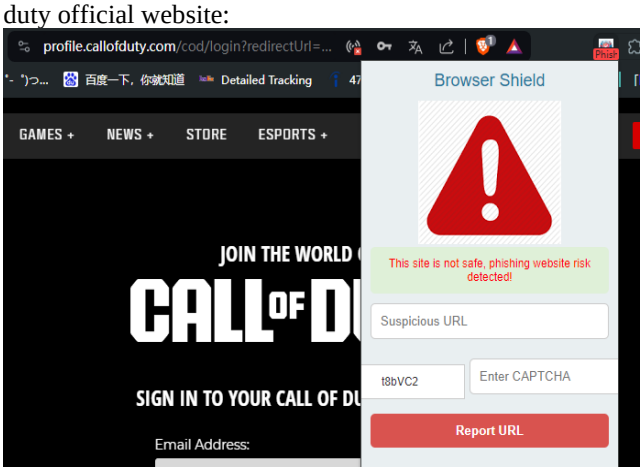

Fig 17. False negative case screenshot.

```
Received URL for checking: https://profile.callofduty.com/cod/login?redir
ns%2FredeemCode
Checking whitelist...
Checking blacklist...
Original URL: https://profile.callofduty.com/cod/login?redirectUrl=https%
de
Cleaned URL: profile.callofduty.com/cod/login
Tokens: ['profile', 'callofduty', 'com', 'cod', 'login']
Stemmed Tokens: ['profil', 'callofduti', 'com', 'cod', 'login']
Processed URL for Prediction: profil callofduti com cod login
Prediction:  bad
127.0.0.1 - - [18/Apr/2024 22:45:52] "POST /check_url HTTP/1.1" 200 -
```

Fig 17. Terminal feedback of false negative case screenshot.

This URL is not a phishing website but somehow the prediction said it is bad. This might because of the dataset is very limited, although it has 549346 samples, but still not include all the URLs, and machine learning is not 100% correct, as we mentioned earlier the test set accuracy is 96.5%, so that false negative and false positive case happens. Unfortunately, we did not find a false positive cases yet, manually tested some phishing URLs who submitted on the PhishTank, all the URLs given a warning phishing message.

V. Future Work and Improvement

1. Add more samples in the dataset to train the model to improve the accuracy.
2. Change the CAPTCHA to reCAPTCHA, this google CAPTCHA method is more secure than this simple CAPTCHA method, which will reduce the risk of server attacks.
3. In the popup UI, add 2 options: false positive or false negative. If user chooses one of it, when server-side saving the URLs, it will also save the reason why this user reported it, and it will help developers improve the accuracy of the extension tools.
4. Since PhishTank shut down the registration, and if by any chance they bring it back, we can use the API instead manually download the csv file and updating the blacklist.
5. More deeper machine learning methods can be applied to this extension project such as RNN or CNN method, which will highly improve the accuracy of the predction but this requires more detailed coding techniques, since it requires a large amount of features to extract and analysis, and it will cost more performance, if this can be applied to a

real server, the performance problem maybe can be solved.

6. Use HTTPS protocol to encrypt all data transmission, make sure that the extension uses HTTPS URLs when communicating with the server. we have set the server URL to always use https://. For example:

   fetch('http://127.0.0.1:5000/check_url'

   But a more rigorous approach has to do with obtaining an SSL certificate. We can get a free certificate from places like “Let's Encrypt”. For development and testing, you can use a self-signed certificate, but browsers will display a security warning. We've tried this approach but there were some bugs in the debug that affected the results, and in the end, we weren't able to fix them, so they weren't rendered in the final code. This needs to be improved in the future.

VI. References

[1] DataAspirant, “How Logistic Regression Model Works,” DataAspirant. [Online]. Available: https://dataaspirant.com/how-logistic-regression-model-works/. [Accessed: 14- Apr- 2024]