Developing a Browser Extension for the Automated Detection of Deceptive Patterns in Cookie Banners

Juris Hannah Adorna[1], Aurel Jared Dantis[1],
Rommel Feria[2], Ligaya Leah Figueroa[2], and Rowena Solamo[2]

[1] University of the Philippines – Diliman, Quezon City 1101, Philippines
[2] Web Science Group, University of the Philippines – Diliman, Quezon City, 1101, Philippines
jurishannahadorna@gmail.com

Abstract. Interacting with web-based interfaces is often done with a particular objective in mind. However, deceptive patterns could interfere with these interactions by taking advantage of cognitive biases to either distract users from this objective or mislead them into non-ideal outcomes. These are found in cookie banners that nudge users to allow the tracking of their browsing patterns, infringing upon the user's right to informed consent regarding matters of their privacy. This paper discusses the implementation of a browser extension, Ariadne, that counteracts this by flagging deceptive patterns in cookie banners, in effect safeguarding the user's right to informed consent in data collection. The current implementation is divided into four units: a Naïve-Bayes model determining language clarity (Calliope), a convolutional neural network (CNN) based on VGG-19 determining option weight (Janus), an application programming interface (API) handling the classification and user reports (Dionysus), and the browser extension itself that allow these units to reach the user. The classifiers Calliope and Janus achieved respective accuracies of 85.00% and 100.00% upon unit validation and 80.00% and 66.67% upon unit testing. Integration testing resulted in an overall average accuracy of 68.70% based on the behavior of the browser extension given selected websites as recorded by thirty (30) observers. Acceptance testing was done through an alpha testing questionnaire yielding positive ratings from thirty (30) testers. This project intends to contribute to the larger body of knowledge surrounding the automated detection of deceptive patterns by implementing previous frameworks thereof and setting the groundwork for the creation of a system that can act as a toolbox of methods for the automated detection of deceptive patterns and corresponding methods for intervention.

Keywords: Deceptive Patterns, Cookie Banner, Browser Extension.

1 Introduction

1.1 Context and Definitions

Deceptive patterns (also known as “dark patterns”) [13] are a tool used in digital interfaces to coerce users into a particular route of interaction. These refer to design choices deliberately meant to affect an end-user’s decision-making process [3]. On cookie

J. Caro et al. (eds.), *Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2023)*, Atlantis Highlights in Computer Sciences 20,
https://doi.org/10.2991/978-94-6463-388-7_8

banners, an example of these design choices would be that the option to decline cookies is less visible or obvious than the one to accept them [17][35]. In general, these design decisions either hinder the user from achieving their goal (e.g., obscuring the path of action to deactivate an account) or adding unnecessary outcomes in the intuitive interaction path thereof (e.g., automatically opting into marketing emails when trying to fill out an order or subscription form) – both of which result in unintended alternatives.

Most of the literature on deceptive patterns explores its presence on cookie banners [17], websites [28], mobile applications [11], robots [23], advertisement [15], and games [46]. While examining where deceptive patterns are found provides an idea of what they are, Mathur et al. [29] note that it is difficult to find precise criteria that determine whether a pattern is deceptive. Additionally, they cite "a set of thematically related considerations" spanning several fields: psychology, economics, ethics, philosophy, and law, among others. Their work in "What Makes a Dark Pattern…Dark?" [29] explores a "conceptual foundation" for deceptive patterns by going over these various considerations.

This study's contribution is three-fold – (1) the creation of a toolbox of methods for the automated detection and intervention of deceptive patterns as a contribution to our field of study, (2) the development of a browser extension that helps identify deceptive patterns in cookie banners as the end-product, and (3) the tool itself that helps safeguard everyday users' rights to informed consent.

1.2 Statement of the Problem

Through deceptive patterns, users are taken advantage of through cognitive biases that could potentially manipulate behavior, manufacture consent, and result in unintended outcomes. To counteract this, the study aims to develop a browser extension that helps identify deceptive patterns in the form of unclear language and uneven options in cookie banners. This browser extension identifies the cookie banner on the website's interface and flags for indicators of deceptive patterns such as a lack of clarity in language or uneven weights between options using classifiers. Currently, the study uses a Naïve-Bayes classifier to identify language clarity and an image classifier built upon a pre-trained CNN model (VGG-19) to determine option weight.

The study's main objective is the creation of a system that can act as a toolbox of methods for automated detection of deceptive patterns and intervention thereof. This study aims to provide everyday users with a browser extension acting as a line of defense against deceptive patterns when browsing the internet, thus helping safeguard our right to informed consent.

2 Review of the State of the Art

At the time of writing, the Computer Science, Human-Computer Interaction, and Computing Ethics communities are exploring interventions for deceptive patterns – including legislation, intervention spaces, and automated detection. This study focuses on

automated detection of deceptive patterns on cookie banners and executing appropriate intervention through a browser extension.

2.1 Automated Detection

Soe et al. [41] posit that a unified criteria deciding whether a pattern is deceptive or not is a prerequisite to extensive development of methods for the automated detection of deceptive patterns. Despite the lack thereof, previous literature has explored the application of existing technology to automatically detect deceptive patterns. These works have a shared goal of informing people when they are subjected to deceptive patterns and blinded due to existing cognitive bias [34].

The proper detection of deceptive patterns is a prerequisite to successful intervention. Identifying when intervention is necessary would require knowing when deceptive patterns are employed and precise criteria thereof. There is a lack of consensus on what constitutes deceptive patterns, thus presenting another issue.

Researchers recommend reducing the task of detecting deceptive patterns into "discerning" [41] or "characterizing" [40] features, thus breaking up the task into detecting individual factors of deceptive patterns. Supervised learning could be applied to each feature to find patterns suggesting the presence of deceptive patterns. This allows certain taxa of deceptive patterns (e.g., nagging and roach motel) to be encoded which would not have been possible without accounting for multiple interfaces presented in succession. These types have visual and textual cues necessary for proper encoding. Visual cues can be processed through image processing and textual cues can be processed through natural language processing. The classifiers trained on Soe et al.'s dataset [40] performed with accuracy ranging from 50% to 72%, despite noting that the dataset was "relatively small and not well-balanced."

Mathur et al. [28] created a corpus of websites documenting the interaction path from the product page to the checkout page across shopping websites through a crawler. Classifying the websites was done using Webshrinker, though they pointed out that Alexa Web Services is a viable alternative. Product page URLs and resulting checkout pages' HTML source codes were extracted and stored before being clustered into different taxa of deceptive patterns. This presented a mechanism in detecting deceptive patterns prevalent in multiple websites.

Curley [9] proposes a framework for an application that identifies dark patterns in a single Web page, emphasizing the importance of using taxonomy to distinguish the different patterns. Curley proposes a set of definitions based on previous literature and separates the patterns into those that can be detected automatically, manually, or not at all. They propose the use of an ancillary Web page or overlays that highlight the possible presence of dark patterns on the website. They also suggest a reporting feature that users can use to report undetected cases of deceptive patterns.

3 Solution and Experiment Design

The goal of the study is to equip users with a tool that proactively checks for the presence of deceptive patterns on websites that they visit, allowing them to take the proper course of action afterward. The automated detection that the browser extension will perform must therefore be done in a way that yields acceptable accuracy while also requiring minimal intervention.

This project aims to lay groundwork on packaging detection and appropriate intervention through a browser extension that detects deceptive patterns in cookie banners and provides users with the necessary information to take the proper course of action. This approach is largely based on the definitions of Mathur et al. [29], the findings and data set of Soe et al. [41], and the framework discussed by Curley et al. [9] where deceptive patterns are shown by highlighting relevant areas in the web-based interface and providing an auxiliary window in the form of a browser extension's menu for displaying more information regarding the contents of the current tab and warnings shown.

3.1 Features

The proposed solution (1) automatically flags the use of unclear language in cookie banners, (2) automatically flags the use of uneven weights on options in cookie banners, and (3) allows users to manually report occurrences of deceptive patterns on websites (e.g., roach motel, price comparison prevention, disguised ads). These features were encased in a modular architecture to allow for future improvement and expansion for mechanisms used to detect and intervene with deceptive patterns.

3.2 Test plan

To evaluate the browser extension, various tests were performed to verify its functionality and usability. These tests aim to assess the performance of the project and its components.

Unit testing was done for the extension's capability to identify deceptive patterns from the text or image of the cookie banner. The models for language clarity and option weights will also be tested as units and as machine learning models.

Integration testing was done through selected websites with manually labeled results for both models. This allows for the end-to-end process to be observed between the tested units described above. The following websites were chosen:

- ABS-CBN at news.abs-cbn.com
- GMA at www.gmanetwork.com
- Manila Times at www.manilatimes.net
- Philippine Star at www.philstar.com
- TV5 at news.tv5.com.ph
- Inquirer at www.inquirer.net
- Pep.PH at www.pep.ph
- Rappler at www.rappler.com

Each of these websites were manually documented (as seen in the documentation repository[1] at `/website-photos`) to have used cookie banners and are each uniform in locale (i.e., based in the Philippines) and purpose (i.e., news or media outlets). These websites were selected due to the incentive that news and media outlets have for tracking the browsing behavior of their demographic.

Acceptance testing in the form of Alpha testing was conducted on Ariadne version 0.1.2 to show usable and effective aggregation of features from this study. This test is intended to assess the experience of users and user-developers through a questionnaire aiming to evaluate the understandability, documentation, installability, learnability, identity, copyright and licensing, and accessibility and answer the corresponding guiding questions based on the Software Sustainability Institute's Criteria-based Software Evaluation [19]. For this test, users are to be given access to the website, documentation, and the browser extension itself while user-developers are provided additional access to the repository storing the project itself[2]. This is to cover the insights of people who are engaging with the project as a browser extension to use and the insights of those who have the intention or capability to get involved in its development.

4 Browser Extension

4.1 Architecture

The proposed solution consists of five components:

1. *Ariadne*, the browser extension itself,
2. *Calliope*, a model for determining language clarity (i.e., whether the language used in requesting consent through the cookie banner is clear or unclear and whether it satisfies the GDPR requirements for informed consent),
3. *Janus*, a model for determining option weights (i.e., whether the cookie banner provides ample, non-coercive options for the user to agree or disagree to provide consent for the use of cookies), and
4. *Dionysus*, a server-side application responsible for making Calliope and Janus accessible through an HTTP API and managing reports of deceptive design that are sent by users of Ariadne.
5. *Olympus*, which is a Web-based application that allows the public to see an overview of the reports of deceptive design sent to Dionysus. This component is not integral to the functionality of the proposed solution.

For brevity, the component names above will be used for the rest of this paper. Fig. 1 serves as an overview of the proposed solution and shows how these components are linked to one another.

[1] https://github.com/wsg-ariadne/docs
[2] https://github.com/wsg-ariadne

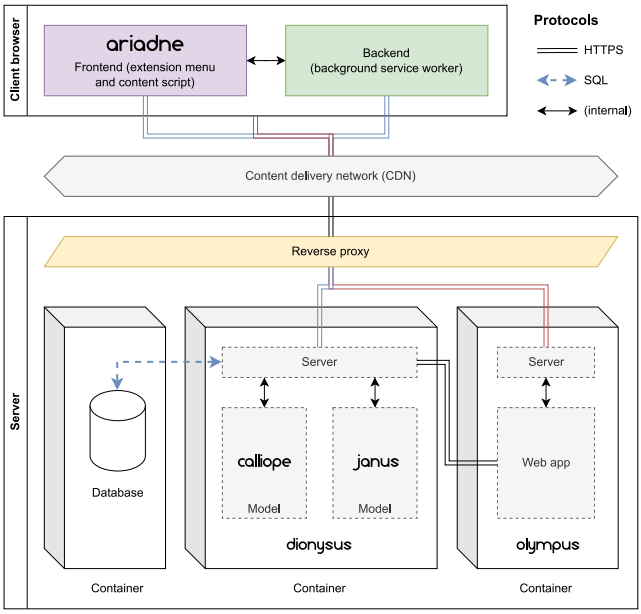

Fig. 1. Architecture of the proposed solution

4.2 Ariadne, the browser extension

Ariadne's functionality is centered around a six-step process:

1. *Read* the document object model (DOM) of the page that the user is viewing,
2. *Look* for an element in the DOM tree that meets predetermined criteria for candidate cookie banners,
3. *Extract* the text in the candidate cookie banner,
4. *Capture* the cookie banner as an image,
5. *Send* the extracted text and image data to Dionysus for classification by Calliope and Janus, and
6. *Show* the classification results to the user.

At the *Look* step, the browser extension considers an element to be a candidate for a cookie banner if its CSS properties resemble that of a "floating" element. The criteria for "floating" elements are discussed further in Sec. 4.2.2. If the browser does not find any element that matches this criteria, steps 3 to 5 in the six-step process are skipped. However, if a match is found, the browser proceeds to *Extract* the text it contains and *Capture* it as an image.

At the *Send* step, text extracted from the cookie banner along with an image of the cookie banner are sent to models that determine whether the cookie banner has features indicative of deceptive patterns. There are two models in this study, Calliope and Janus,

where the former receives the extracted cookie banner text as input and the latter receives the cookie banner image as input.

At the *Show* step, the results of the models above are displayed at the auxiliary screen, with a brief description of the deceptive pattern(s) that were flagged. If there are reports of deceptive patterns previously sent by a user for the page that the user is viewing, then the extension shows the number of such reports, both at the auxiliary screen and as a badge on the extension's icon.

Implementation. Ariadne is implemented according to Google's Manifest V3 specification for browser extensions [24], which is set to be the minimum requirement for the Google Chrome browser in the future [7]. Accordingly, the extension is developed to be compatible with versions 88 and newer of Google Chrome [6], along with other browsers based on the Chromium project.

The *Read* and *Look* steps of the process mentioned in Sec. 4.2.1 are achieved through a script that is injected by the browser into all pages; more specifically, the script is allowed to load on every page whose URL matches the patterns `http://*/*` and `https://*/*`. This script is known as a content script in Chrome Extension parlance [7] and is named content.js in the extension source code.

The content script waits for five seconds after the page finishes loading, then creates a copy of all the `<div>` elements on the page. Among those elements, the script looks for those that are floating, which is characterized as elements that:

- have a CSS z-index above the default of 0, and
- have their CSS position set to either fixed, absolute, or sticky.

Elements that match these criteria proceed to the *Extract* step. Here, the script creates a copy of the cookie banner's text content in a variable named `divText`, using the `.innerText` JavaScript property [30]. The extracted text is converted to lowercase for easier matching. The script then attempts to match the words cookies, consent, or trackers within `divText`. If a match is found, the script proceeds to the *Capture* and *Send* steps.

The *Send* step consists of two phases: one for the extracted text and another for the captured image. Both phases are done by a service worker script named background.js that runs in the background and waits for input from every instance of the content script that has been injected in every tab. Each input to the service worker is a JavaScript Object Notation (JSON) object with two keys: the first, `action`, is a string that determines the type of input, and the second, `args`, is a dictionary that contains the necessary data for the worker to process the input. For instance, in the first phase of the *Send* step, the content script will tell the service worker to feed `divText` to Calliope.

The *Capture* step is made possible by the html2canvas[3] JavaScript library, which allows for converting an arbitrary element or set of elements on a Web page to canvas data, which can then be saved as an image. Using this library, the content script captures

[3] https://html2canvas.hertzen.com/

the cookie banner as image data in a variable named `screenshot` and tells the service worker to forward the data to Janus.

The service worker forwards these requests to Dionysus, the server-side application responsible for calling Calliope and Janus, through an HTTP `POST` request. Dionysus will then reply with JSON objects containing the results from both models. This is further discussed in Sec. 4.5. The content script will then tell the service worker to change the extension icon's badge to purple if either model flags the input data for deceptive design. A notice is also visible in Ariadne's menu stating whether deceptive design was found on the page.

4.3 Calliope, the language clarity model

Enabling the flagging of deceptive patterns through the browser extension are models that aim to discern whether components of the cookie banner show features of deceptive patterns or not. The first of these models is Calliope, the language clarity model which aims to determine whether the language used in a cookie banner is clear or unclear in requesting consent for the use of cookies.

Implementation. Calliope is implemented through a Naïve-Bayes classifier trained on data from websites and previous studies describing cookie banners based on their compliance to GDPR standards of informed consent. The data points used to train Calliope were collected from various websites discussing examples of General Data Protection Regulation (GDPR) compliant and non-compliant cookie banners[4] and the repository[5] of Soe et al. [40].

These data points were manually labeled by the proponents of this study as seen in the repository[6] at `/data` Each data point is labeled either `GOOD` or `BAD` depending on whether the cookie banner text employs deceptive patterns or not, based on the criteria from both the EU's GDPR guidelines for cookie banners and Philippine data privacy laws, particularly Republic Act No. 10173. This criteria on what constitutes language clarity in cookie banners is as follows:

1. Explicit indication of a request of consent for non-essential cookies
2. Enumeration of specific purposes for cookie usage and collection of data
3. Referencing additional resources on how to access collected data, customize preferences, refusal or changing of preferences, or other auxiliary information and functions relating to cookie banners.

[4] https://www.cookieyes.com/blog/gdpr-cookie-consent-banner-examples/
https://www.enzuzo.com/learn/the-best-cookie-banner-examples-weve-seen-in-2022
https://www.vice.com/en/article/m7epda/its-bad-design-on-purpose-why-website-cookie-banners-look-like-that
https://blog.didomi.io/en/marketing-15-examples-cookie-banners-brands

[5] https://github.com/videoworkflow/cookiepopup

[6] https://github.com/wsg-ariadne/calliope

The data set in the repository has a dedicated column of reasons that provide the justifications of the proponents in labeling at `/data/data.pdf`. A comma-separated value (CSV) file containing the manually assigned labels and text can be viewed in the repository for Calliope and is read as its training data.

The data set from the work of Soe et al. [40] includes various forms of cookie notices in image format. The proponents filtered the data set to only contain images that resembled cookie banners. Then, optical character recognition was used to extract the text in the image. The extracted text was manually corrected to be free of any typographical errors and the text appearing as options on the cookie banners were removed for uniformity.

The combined data set was loaded into a Python notebook from the CSV file as a `pandas dataframe` before being standardized to remove punctuation. The string, representing the banner text, was also divided into a list of words for easier processing through the classifier training done with `nltk`.

Further standardization was done by making all letters lowercase before assessing for features that the classifier needs to consider when processing documents. The feature extractor takes five of the most frequently used words and uses these to learn how to predict the label of documents.

Classifiers using `nltk` call for the use of a training and a testing set, thus only using a subset of the data set and using the rest to assess the accuracy at the training phase. As seen in `/logs/logs.txt` the accuracy of the classifier at training is 76.25%. The classifier is then saved using `pickle5` for further use, deployment, and evaluation.

4.4 Janus, the option weight model

Soe et al. [41] discuss the numerous ways in which the cookie banner may be represented (i.e., image, text, and features) and the limitations each representation has. One of the benefits of using an image to represent the cookie banner is that it considers the layout in which these options could appear. The second classifier enabling the flagging of deceptive patterns through the browser extension is Janus, an option weight model which aims to determine whether the option to provide or deny consent to the use of cookies are fairly displayed on a cookie banner.

Implementation. Janus is implemented as an image classifier that determines whether an image of a cookie banner employs deceptive patterns in the form of either obscuring or removing the option to deny consent. This classifier is based on the VGG-19 model that classifies images into the following classes:

- ABSENT, indicating that the option to refuse cookies is not at all on the interface,
- WEIGHTED, indicating that the option to refuse cookies is made less obvious, less visible, or more tedious to select than the option to accept it, and
- EVEN, indicating that the options to accept and refuse cookies appear on the cookie banner and are equally obvious.

The photos used to train Janus were collected from the repository[7] of Soe et al. [40]. These data points were manually labeled by the proponents of this project as seen in the repository[8] at `janus/data`. Each data point is labeled either `ABSENT`, `WEIGHTED`, or `EVEN` depending on the way the options are presented on the interface. The data can be viewed in the repository at `/data` sorted into a folder for each label.

The data set from the work of Soe et al. [40] includes various forms of cookie notices in image format. The proponents filtered the data set to only contain images that resembled cookie banners before manually sorting them into the folders.

The classifier pre-processing and training follows the procedure detailed on a guide from Analytics Vidhya [43], implementing image classification on custom data sets. The data set in /data is then split into the training set (n = 60, evenly divided across the classes), its subset validation set (n = 12, evenly divided across the classes), and a testing set (n = 6, evenly divided across the classes). These images were then placed into the directory `/final-dataset` into their corresponding subdirectories `/final-dataset/train` for the training set, `/final-dataset/val` for the validation set, and `/final-dataset/test` for the testing set. This implementation makes use of a pre-trained CNN model, VGG-19, and takes a color image of 224 × 224 pixels to classify into one of three categories. This means that prior to its re-training, all the images need to be resized to the specifications of VGG-19 and turned into a `numpy` array representation of the resized image. The training, validation, and testing sets were all put into their corresponding arrays of `numpy` array representations and labels.

To customize VGG-19 into Janus, the last layer needs to be changed according to accommodate the three classes for the project. The `adam` optimizer was used to automatically decide the optimal learning rate. Implementing `EarlyStopping` to stop training once validation loss increases provide a way to preempt overfitting. The model was trained for ten (10) epochs and yielded an accuracy of 66.67% and loss of 0.8632 at the final performance graph found in `/logs/logs.txt`.

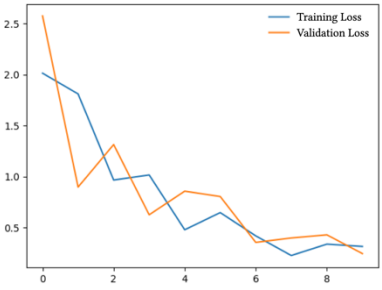

Fig. 2. Graph of Accuracy over Training of Janus

Fig. 3. Graph of Loss over Training of Janus

[7] https://github.com/videoworkflow/cookiepopup
[8] https://github.com/wsg-ariadne/janus

4.5 Dionysus, the classification and reporting API

Machine learning models such as Calliope and Janus require a certain level of performance and storage allocation from a host system to execute at an acceptable speed. The Janus model, for instance, is available as a set of files that take up around eighty megabytes, but the libraries required to run it such as TensorFlow take up hundreds of megabytes. Although the Janus model was found to take only a few seconds to classify each cookie banner image during testing, it was also found to take around three to four gigabytes of memory during runtime.

In the interest of positive user experience, the proponents opted to make a single running instance of both Calliope and Janus available through an HTTP API called Dionysus, instead of packaging both models inside the Ariadne browser extension installation file to run locally. The Dionysus API was also written to incorporate functionality for user-submitted reports, both to aid in improving model accuracy and for logging deceptive design patterns that cannot be detected by either model.

Implementation. Dionysus is written in Python for consistency with the Calliope and Janus models. It is a Flask[9] application that comes bundled with instances of Calliope and Janus, both of which are loaded into memory at launch. Furthermore, the application is packaged as a Docker image[10], in the interest of making Dionysus, Calliope, and Janus easy to redistribute and deploy [5][39].

Dionysus listens for HTTP requests on endpoints that are prefixed with `/api/v1`. The endpoints will be discussed with the assumption that they are prefixed as such, and that they expect and return JSON objects during operation. The full details of the API, including the format of these requests, are available at the project repository. The endpoints that concern Calliope and Janus, the classification models, are as follows:

- `/classify/text` which accepts cookie banner text as input, feeds that input to Calliope, and returns the result in the form of a Boolean value called `is_good`. A value of true indicates that the language used in the cookie banner text is clear, and a value of false indicates that it is unclear.
- `/classify/image` which accepts image data as input, feeds that input to Janus, and returns the result in the form of a string called classification. This string may have the value "WEIGHTED" if the cookie banner in the image might have weighted options, "EVEN" if the options are not weighted, and "ABSENT" if Janus could not see the option decline on the cookie banner.

It is possible for Ariadne to misidentify an element and potentially transmit sensitive text and image data to Dionysus for classification. For this reason, Dionysus does not store data sent to the endpoints above, and the input and result data for every detection request are lost from the server after the request is fulfilled.

Curley et al. [9] discuss the shortcomings of automated detection against the various forms of deceptive patterns and identify a reporting feature to fill the gap. This feature

[9] https://flask.palletsprojects.com/
[10] https://docs.docker.com/get-started/

is used to compensate for the fact that some deceptive patterns cannot be automatically detected, so users must do it manually and share the report to other users. For this reason, Dionysus also has endpoints for accepting and returning reports of deceptive patterns made by users:

- `/classify/report`, for submitting feedback on automated detections, i.e., reports on whether Calliope or Janus correctly or incorrectly flagged the Web page for deceptive design.
- `/reports`, for returning an overview of the most recent user-created reports, including information on the top reported Web page and the total number of reports, and accepting user-created reports of deceptive patterns, including free-form data that allows users to describe deceptive patterns aside from unclear language and uneven options.
- `/reports/by-id`, for obtaining the details of a report using its unique ID assigned in the /reports endpoint. These details include the kind of deceptive pattern, the timestamp of the first time a Web page was reported for having this deceptive pattern, and the number of times the Web page was reported for having that deceptive pattern.
- `/reports/by-url`. supplying Ariadne with that data, accepting the page's URL as input, and returning the appropriate report counts as output for Ariadne's menu (Fig. 4).

These reports are stored in a PostgreSQL database, which is also running in a Docker container created using the official PostgreSQL image[11]. It was mentioned that Olympus, the fifth component of our proposed solution, can show a publicly viewable summary of reports sent to Dionysus. This is made possible by the `/reports` endpoint. Together, Dionysus, Olympus, and the database run together as an orchestrated set of services using the Docker Compose tool[12].

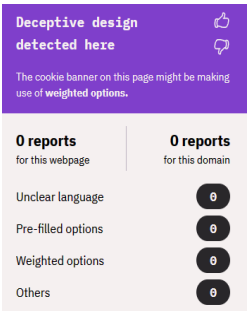

Fig. 4. Ariadne's extension menu, showing the report counts for a Web page.

[11] https://hub.docker.com/_/postgres
[12] https://github.com/docker/compose

5 Results and Discussion

As discussed, the study's objective is to create a system capable of acting as a toolbox of methods for the automated detection of deceptive patterns and intervention methods by providing everyday users with a browser extension. This browser extension, Ariadne, acts as a line of defense by flagging indicators of deceptive design such as the use of unclear language and uneven options through the classifiers Calliope and Janus, respectively.

5.1 Results

Ariadne was evaluated through the unit testing of Calliope and Janus, integration testing on eight (8) pre-selected websites and two (2) websites chosen by testers, and acceptance testing through the evaluation of thirty (30) alpha testers, six (6) of which were user-developers and the rest being users.

Unit testing. Both classifiers were separately tested. Classifier evaluation metrics, specifically accuracy, precision, recall, and the F-score were also computed. These values, summarized in Table 1, are indicators of different facets of the classifier's performance. To clarify, a positive or relevant event refers to the presence of deceptive patterns. Recalling that Calliope classifies whether unclear language is used or not and Janus classifies whether uneven banners are used. For Calliope, this means that a cookie banner being labeled `BAD` pertains to a positive. Similarly, a cookie banner being labeled `WEIGHTED` or `ABSENT` pertains to a positive.

Table 1. Summary of classifier evaluation metrics on Calliope and Janus

Metric	Formula	**Calliope** Validation	**Calliope** Testing	**Janus** Validation	**Janus** Testing
Accuracy, binary	$\frac{correct\ predictions}{total\ predictions}$	85.0%	80.0%	100%	66.67%
Accuracy, 3 classes	$\frac{correct\ predictions}{total\ predictions}$	-	-	91.67%	55.56%
Precision	$\frac{true\ positives}{predicted\ positives}$	0.9000	0.6000	1.0000	0.6667
Recall	$\frac{true\ positives}{actual\ positives}$	0.8182	1.0000	1.0000	0.8090
F-score	$2\left(\frac{precision \times recall}{precision + recall}\right)$	0.8571	0.7500	1.0000	0.7273

The accuracy for validation is consistently higher than that of testing. This is to be expected because validation is conducted with samples from the training data set while testing is conducted with samples that the model has not previously seen. As seen in Table 1, there are two rows for Accuracy, particularly for Janus. Accuracy (Binary) refers to whether the classifier can tell when a sample contains deceptive patterns or not while Accuracy (3 Classes) refers to whether Janus is able to correctly assign its three labels to the samples. Precision and recall are indicators of how well the classifier

can detect relevant or positive events, referring to the use or presence of deceptive patterns. From these metrics, one could see that Calliope's performance in validation and Janus' performance in testing could be improved.

Integration testing. The end-to-end process was tested through the observation of Ariadne's behavior on eight (8) selected websites (as seen in the documentation repository[13] at /website-photos) with manually labeled results for both models. The following websites, along with two (2) websites of the observer's choice were taken as samples for integration testing. A summary of the expected behavior and corresponding labels for each of the websites is seen in Table 2. The expected labels indicate the predicted label of Janus and Calliope from the unit level since the objective of integration testing is to see how the system performs end-to-end.

Table 2. Selected websites and their corresponding labels and expected flags

Website	Expected label		Expected flags	
	Calliope	Janus	Unclear language	Uneven weights
ABS-CBN	BAD	EVEN	✓	-
TV5	BAD	EVEN	✓	-
GMA Network	GOOD	WEIGHTED	-	✓
Inquirer	BAD	WEIGHTED	✓	✓
Manila Times	BAD	EVEN	✓	-
Pep.PH	GOOD	WEIGHTED	-	✓
Philippine Star	BAD	WEIGHTED	✓	✓
Rappler	GOOD	EVEN	-	-

The thirty (30) observers were tasked to install Ariadne, record whether the website's cookie banner was displayed, and then record the corresponding flags raised by Ariadne. Shown in Table 3 are their findings.

Table 3. Summary of observed behavior from Ariadne for selected websites

Website	Banner presence		Flags raised		
	Displayed	Not displayed	None	Unclear language	Uneven weights
ABS-CBN	26	4	6	24	16
TV5	29	1	4	26	9
GMA	28	2	18	11	11

[13] https://github.com/wsg-ariadne/docs/tree/main/website-photos

Network Inquirer	4	26	27	3	2
Manila Times	29	1	4	25	15
Pep.PH	28	2	4	0	26
Philippine Star	27	3	7	23	6
Rappler	8	22	26	4	0

Instances logged to have no displayed banner are to be deducted from the number of readings that indicate neither unclear language nor uneven weights were found on the interface to deduplicate the records. The accuracy is computed as the number of correct predictions is divided by the total number of predictions. In this case, the occurrence of a particular flag (i.e., none, unclear language, and uneven weights) is given by the quotient of the occurrences of the flag and the total number of samples with a banner displayed. From this, the accuracy is the % occurrence of flag if that flag is expected (i.e., ticked in Table 2) and 100% − % occurrence of flag otherwise. A summary of these values can be seen in Table 4, which shows volatility in its recorded performance across observers and across websites.

Table 4. Summary of accuracy rates for integration testing on Ariadne

Website	**Accuracy (%)**	
	Calliope	Janus
ABS-CBN	92.31	38.46
TV5	89.66	68.97
GMA Network	60.71	32.29
Inquirer	75.00	50.00
Manila Times	86.21	48.28
Pep.PH	100.00	92.86
Philippine Star	85.19	22.22
Rappler	50.00	100.00
Average	79.88	57.51

Acceptance testing. Alpha testing was conducted to evaluate the usability and effectiveness of the aggregation of features from this study. The thirty (30) alpha testers, consisting of six (6) user-developers and fifteen (15) users answered a questionnaire about the understandability, documentation, installability, learnability, identity, copyright and licensing, and accessibility of Ariadne. The alpha testers provided feedback in the form of Likert scale ratings, the averages of which are presented in Table 5.

Table 5. Average Likert scale ratings from alpha testers

Criterion	Rating (5)
Understandability	4.2700
Documentation	4.6150
Installability	4.8045
Learnability	4.7857
Identity	4.5190
Copyright & Licensing	4.8133
Accessibility	4.9667

5.2 Recommendations

Future developments could work on improving its components through expertise in machine learning and particular portions of the technology stack used to address the limitations and shortcomings of Ariadne version 0.1.2 as documented in this paper.

Ariadne. Because there is no standardization for the user interface of a cookie consent banner, as observed by Soe et al. [40][41], it is difficult to develop a tool that can programmatically determine whether a Web page employs deceptive patterns.

Computer vision techniques such as perceptual hashing and saliency mapping could be used instead of scraping the Web page for cookie banners. This could both simplify and increase the accuracy of the Read and Look steps of the browser extension's process, especially in Web pages that contain many HTML elements that the content script would have to sift through.

Calliope and Janus. On the unit level, improving the performance of both Calliope and Janus involves providing larger, high-quality data [28][35] and setting the groundwork by establishing firm criteria on what constitutes deceptive patterns [29]. Current and future efforts on a unified definition on deceptive patterns and taxonomy thereof are likely to provide clarity in future iterations of the models. Current and future developments in legislation [12] surrounding the data privacy [22], the use of cookies, and informed consent in the context of web-based interfaces could also help guide the creation of stricter, more deterministic criteria regarding deceptive patterns on cookie notices [40]. The transformation of the image to a square is also an issue because the resulting images may not include the text on the buttons at all. The nature of VGG-19 only accepting 224 × 224-pixel images present a limitation to Janus' performance. Since cookie banners are often long rectangles, the necessary resizing causes a loss of discernible features on the interface. Looking for alternative, less resource-demanding, pre-trained models that can process rectangular images to base Janus upon would be ideal for future iterations. The proponents were unable to devote significant time to exploring alternatives due to the study's priority being the creation of an overall system hosting the models, allowing for future work to be more targeted in improving individual components.

Dionysus. The handling of user reports through Dionysus would benefit from the development of a non-intrusive method to attribute reports to the user that submitted it to make it possible to request for the access or deletion of their data.

Ariadne informs the user of the presence of deceptive design by (1) turning the extension badge purple and (2) displaying a purple banner in the extension menu. This behavior is triggered by either a BAD result from the Calliope model, which means that the language of the cookie banner is unclear, or a WEIGHTED result from the Janus model, which means that the options for managing consent are not evenly presented on the cookie banner. Janus, however, also returns an ABSENT result if an option to decline cookies was not detected. Future iterations of Dionysus should include a separate label for classification of cookie banners with no visible options.

Although Dionysus does not require reports to contain information that could potentially identify someone, these reports are still user-generated content, and therefore there should be a way for a user to request deletion of their content. Created reports are not tied to a user, as Ariadne does not require the user to log in or otherwise identify themselves before use, but since IDs are created for every report, future work could include a way for a user to see the IDs for their reports and request deletion of these reports through the Olympus interface.

The proponents set out to lay down the groundwork of a toolbox for interventions against deceptive design that is accessible through a browser extension. Despite shortcomings and volatile accuracy rates seen in integration testing, this goal was accomplished to the extent that unit tests have demonstrated satisfactory performance of the language clarity model, Calliope and acceptable performance of the option weight model, Janus. Its performance was also found to be satisfactory and acceptance tests have demonstrated a positive evaluation among testers.

References

1. Edoardo Ardizzone, Alessandro Bruno, and Giuseppe Mazzola. 2013. Saliency Based Image Cropping. In Image Analysis and Processing – ICIAP 2013 (Lecture Notes in Computer Science), Alfredo Petrosino (Ed.). Springer, Berlin, Heidelberg, 773–782. https://doi.org/10.1007/978- 3- 642- 41181- 6_78
2. Luiz Adolpho Baroni, Alisson Andrey Puska, Luciana Cardoso de Castro Salgado, and Roberto Pereira. 2021. Dark Patterns: Towards a Socio-technical Approach. In Proceedings of the XX Brazilian Symposium on Human Factors in Computing Systems. ACM, Virtual Event Brazil, 1-7. https://doi.org/10.1145/3472301.3484336
3. H Brignull, M Leiser, C Santos, and K Doshi. 2023. Deceptive patterns - user interfaces designed to trick you. https://www.deceptive.design/
4. Christoph Bösch, Benjamin Erb, Frank Kargl, Henning Kopp, and Stefan Pfattheicher. 2016. Tales from the Dark Side: Privacy Dark Strategies and Privacy Dark Patterns. Proceedings on Privacy-Enhancing Technology 4 (2016), 18.
5. Tyler Charboneau. 2022. How to "Dockerize" Your Python Applications | Docker. https://www.docker.com/blog/how-to-dockerize-your-python-applications/.
6. Chrome Developers. 2020. Welcome to the Chrome Extension Manifest V3. https://developer.chrome.com/docs/extensions/mv3/intro/.

7. Chrome Developers. 2021. Chrome Extensions Manifest V2 Support Timeline. https://developer.chrome.com/docs/extensions/migrating/mv2-sunset/.
8. Gregory Conti and Edward Sobiesk. 2010. Malicious Interface Design: Exploiting the User. In Proceedings of the 19th International Conference on World Wide Web (Raleigh, North Carolina, USA) (WWW '10). Association for Computing Machinery, New York, NY, USA, 271-280. https://doi.org/10.1145/1772690.1772719
9. Andrea Curley, Dympna O'Sullivan, Damian Gordon, Brendan Tierney, and Ioannis Stavrakakis. 2021. The Design of a Framework for the Detection of Web-Based Dark Patterns. In 5th Workshop on Technology and Consumer Protection. ICDS 2021: The 15th International Conference on Digital Society, Nice, France, 8.
10. Gregory Day and Abbey Stemler. 2019. Are Dark Patterns Anticompetitive? https://www.ssrn.com/abstract=3468321.
11. Linda Di Geronimo, Larissa Braz, Enrico Fregnan, Fabio Palomba, and Alberto Bacchelli. 2020. UI Dark Patterns and Where to Find Them: A Study on Mobile Applications and User Perception. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. ACM, Honolulu HI USA, 1-14. https://doi.org/10.1145/3313831.3376600
12. Gertz, Michael and Martini, Mario, and Kramme, Inken and Seeliger, Paul and Drews, Christian and Kamke, Anton and Kraul, Felix and Reuter, Nicolas and Buchsbaum, Katharina and Urbanek, Ulrike. n.d.. Dark Pattern Detection Project - Dapde. https://dapde.de/en/.
13. Colin M Gray, Shruthi Sai Chivukula, and Ahreum Lee. 2020. What Kind of Work Do "Asshole Designers" Create? Describing Properties of Ethical Concern on Reddit. In Proceedings of the 2020 ACM Designing Interactive Systems Conference. Purdue University, Montreal QC Canada, 61-73.
14. Colin M. Gray, Yubo Kou, Bryan Battles, Joseph Hoggatt, and Austin L. Toombs. 2018. The Dark (Patterns) Side of UX Design. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. ACM, Montreal QC Canada, 1-14. https://doi.org/10.1145/3173574.3174108
15. Saul Greenberg, Sebastian Boring, Jo Vermeulen, and Jakub Dostal. 2014. Dark patterns in proxemic interactions: a critical perspective. In Proceedings of the 2014 conference on Designing interactive systems. ACM, Vancouver BC Canada, 523-532. https://doi.org/10.1145/2598510.2598541
16. Jonathan Harel, Christof Koch, and Pietro Perona. 2006. Graph-Based Visual Saliency.In-Advances in Neural Information. Processing Systems, B. Schölkopf, J. Platt, and T. Hoffman (Eds.), Vol. 19. MIT Press, Cambridge, 545-552.
17. Philip Hausner and Michael Gertz. 2021. Dark Patterns in the Interaction with Cookie Banners. arXiv:2103.14956 [cs] (March 2021). http://arxiv.org/abs/2103.14956 arXiv: 2103.14956.
18. L. Itti, C. Koch, and E. Niebur. 1998. A Model of Saliency-Based Visual Attention for Rapid Scene Analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 20, 11 (Nov. 1998), 1254-1259. https://doi.org/10.1109/34.730558
19. Mike Jackson, Steve Crouch, and Rob Baxter. 2011. Software Evaluation: Criteria-based Assessment.
20. Nehal Jaiswal and Yogesh K. Meghrajani. 2015. Automatic Image Cropping Using Saliency Map. In 2015 International Conference on Industrial Instrumentation and Control (ICIC). IEEE, Pune, India, 971-973. https://doi.org/10.1109/IIC. 2015.7150885
21. Jonathan Robie. 1998. What Is the Document Object Model? https://www.w3.org/TR/WD-DOM/introduction.html.

22. Sean Kellogg. 2020. How US, EU approach regulating 'dark patterns'. https://iapp.org/news/a/ongoing-dark-pattern-regulation/
23. Cherie Lacey and Catherine Caudwell. 2019. Cuteness as a 'Dark Pattern' in Home Robots. In 2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI). IEEE, Daegu, Korea (South), 374-381. https://doi.org/10.1109/HRI.2019.8673274
24. David Li. 2022. More details on the transition to Manifest V3. https://developer.chrome.com/blog/more-mv2-transition/
25. Jamie Luguri and Lior Jacob Strahilevitz. 2021. Shining a Light on Dark Patterns. Journal of Legal Analysis 13, 1 (Jan. 2021), 43-109. https://doi.org/10.1093/jla/laaa006 Number: 1.
26. Aditi M. Bhoot, Mayuri A. Shinde, and Wricha P. Mishra. 2020. Towards the Identification of Dark Patterns: An Analysis Based on End-User Reactions. In IndiaHCI '20: Proceedings of the 11th Indian Conference on Human-Computer Interaction. ACM, Online India, 24-33. https://doi.org/10.1145/3429290.3429293
27. Maximilian Maier and Rikard Harr. 2020. Dark Design Patterns: An End-User Perspective. Human Technology 16, 2 (2020), 170-199. https://doi.org/10.17011/ht/urn.202008245641
28. Arunesh Mathur, Gunes Acar, Michael J. Friedman, Eli Lucherini, Jonathan Mayer, Marshini Chetty, and Arvind Narayanan. 2019. Dark Patterns at Scale: Findings from a Crawl of 11K Shopping Websites. Proceedings of the ACM on Human-Computer Interaction 3, CSCW (Nov. 2019), 1-32. https://doi.org/10.1145/3359183
29. Arunesh Mathur, Mihir Kshirsagar, and Jonathan Mayer. 2021. What Makes a Dark Pattern... Dark?: Design Attributes, Normative Considerations, and Measurement Methods. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. ACM, Yokohama Japan, 1-18. https://doi.org/10.1145/3411764.3445610
30. Mozilla Developer Network. 2023. HTMLElement: innerText Property - Web APIs | MDN. https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/innerText.
31. Mozilla Developer Network. 2023. <iframe>: The Inline Frame Element - HTML: HyperText Markup Language | MDN. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe.
32. Mozilla Developer Network. 2023. Viewport Concepts - CSS: Cascading Style Sheets | MDN. https://developer.mozilla.org/en-US/docs/Web/CSS/Viewport_concepts.
33. Ernst Niebur. 2007. Saliency Map. Scholarpedia 2, 8 (Aug. 2007), 2675. https://doi.org/10.4249/scholarpedia.2675
34. Patricia A Norberg, Daniel R Horne, and David A Horne. 2007. The privacy paradox: Personal information disclosure intentions versus behaviors. Journal of consumer affairs 41, 1 (2007), 100-126.
35. Midas Nouwens, Ilaria Liccardi, Michael Veale, David Karger, and Lalana Kagal. 2020. Dark Patterns after the GDPR: Scraping Consent Pop-Ups and Demonstrating Their Influence. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (Honolulu, HI, USA) (CHI '20). Association for Computing Machinery, New York, NY, USA, 1-13. https://doi.org/10.1145/3313831.3376321
36. Oleksandr Paraska. 2020. Machine Learning on the Web for content filtering applications. https://www.w3.org/2020/ 06/machine- learning- workshop/talks/machine_learning_on_the_web_for_content_filtering_applications.html
37. Ziaur Rahman, Yi-Fei Pu, Muhammad Aamir, and Farhan Ullah. 2019. A Framework for Fast Automatic Image Cropping Based on Deep Saliency Map Detection and Gaussian Filter. International Journal of Computers and Applications 41, 3 (May 2019), 207-217. https://doi.org/10.1080/1206212X.2017.1422358
38. S. Hrushikesava Raju, Saiyed Faiayaz Waris, S. Adinarayna, Vijaya Chandra Jadala, and G. Subba Rao. 2022. Smart Dark Pattern Detection: Making Aware of Misleading Patterns

Through the Intended App. In Sentimental Analysis and Deep Learning (Advances in Intelligent Systems and Computing), Subarna Shakya, Valentina Emilia Balas, Sinchai Kamolphiwong, and Ke-Lin Du (Eds.). Springer, Singapore, 933-947. https://doi.org/10.1007/978-981-16-5157-1-72

39. Prokop Simek. 2019. Dockerizing | Developer Experience Knowledge Base. https://developerexperience.io/articles/dockerizing.
40. Than Htut Soe, Oda Elise Nordberg, Frode Guribye, and Marija Slavkovik. 2020. Circumvention by design - dark patterns in cookie consent for online news outlets. In Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society. ACM, Tallinn Estonia, 1-12. https://doi.org/10.1145/3419249.3420132
41. Than Htut Soe, Cristiana Teixeira Santos, and Marija Slavkovik. 2022. Automated detection of dark patterns in cookie banners: how to do it poorly and why it is hard to do it any other way. https://doi.org/10.48550/ARXIV.2204.11836
42. Erdinç Uzun. 2020. A Novel Web Scraping Approach Using the Additional Information Obtained From Web Pages. IEEE Access 8 (2020), 61726-61740. https://doi.org/10.1109/ACCESS.2020.2984503
43. Nithyashree V. 2021. Step-by-Step guide for Image Classification on Custom Datasets. https://www.analyticsvidhya. com/blog/2021/07/step- by- step- guide- for- image- classification- on- custom- datasets/.
44. Ari Ezra Waldman. 2020. Cognitive biases, dark patterns, and the 'privacy paradox'. Current Opinion in Psychology 31 (Feb. 2020), 105-109. https://doi.org/10.1016/j.copsyc.2019.08.025
45. Fiona Westin and Sonia Chiasson. 2020. Opt out of Privacy or "Go Home": Understanding Reluctant Privacy Behaviours through the FoMO-Centric Design Paradigm. In Proceedings of the New Security Paradigms Workshop (San Carlos, Costa Rica) (NSPW '19). Association for Computing Machinery, New York, NY, USA, 57-67. https://doi.org/10.1145/3368860.3368865
46. Zagal, José P and Björk, Staffan and Lewis, Chris. 2013. Dark patterns in the design of games. https://www.diva- portal.org/smash/get/diva2:1043332/FULLTEXT01.pdf.