Automated Cookie Notice Analysis and Enforcement

Rishabh Khandelwal
University of Wisconsin–Madison

Asmit Nayak
University of Wisconsin–Madison

Hamza Harkous*
Google Inc.

Kassem Fawaz
University of Wisconsin–Madison

Abstract

Online websites use cookie notices to elicit consent from the users, as required by recent privacy regulations like the GDPR and the CCPA. Prior work has shown that these notices are designed in a way to manipulate users into making website-friendly choices which put users' privacy at risk. In this work, we present *CookieEnforcer*, a new system for automatically discovering cookie notices and extracting a set of instructions that result in disabling all non-essential cookies. In order to achieve this, we first build an automatic cookie notice detector that utilizes the rendering pattern of the HTML elements to identify the cookie notices. Next, we analyze the cookie notices and predict the set of actions required to disable all unnecessary cookies. This is done by modeling the problem as a sequence-to-sequence task, where the input is a machine-readable cookie notice and the output is the set of clicks to make. We demonstrate the efficacy of *CookieEnforcer* via an end-to-end accuracy evaluation, showing that it can generate the required steps in 93.7% of the cases. Via a user study, we also show that *CookieEnforcer* can significantly reduce the user effort. Finally, we characterize the behavior of *CookieEnforcer* on the top 100k websites from the Tranco list, showcasing its stability and scalability.

1 Introduction

Cookies are small blocks of data stored on users' browsers that allow the websites to record the state of the browsing session, enabling them to enhance the user experience. Previous works have shown that the majority of the cookies are used for advertising and tracking purposes [4, 7, 19, 20, 43] and that this trend has been rising in the last decade [45, 49]. To curb the associated privacy risks, governments have introduced privacy regulations (GDPR [2] and ePrivacy Directive [1]). These require websites to obtain explicit, specific, and informed user consent before collecting or processing user data.

To comply with these regulations, websites use cookie notices to obtain users' consent and (in some cases) to offer them options to control cookies. Still, in their current forms, cookie notices suffer from usability issues [26]. Websites often design the notices to nudge the users into giving consent [52] or make it harder for users to find the cookie settings.

Take https://www.dailymail.co.uk/ as an example (when visited from the UK in February 2023). To adjust their cookie settings, the user is presented with a blocking notice where they should first click on the "Cookie Settings" button to navigate to the settings menu. On that menu, there are 11 individual cookie settings, 9 of which are pre-enabled for "legitimate interests". Further, there is another view for per-vendor settings with over 100 listed vendors along with their options; all these options are also pre-enabled[1]. Both menus do not have an opt-out button; the user has to individually disable each cookie setting.

Prior work [5] has highlighted the difficulty of exercising cookie control, showing that users are more likely to agree to the default option when the alternative is difficult to find. There, users expressed interest in having easier ways to limit access to their data. To bridge this usability gap, several proposals have aimed at automating the interaction with cookie notices. These can be categorized into two approaches: those that work at the browser level – interacting with the underlying cookie storage – and those that work at the website UI level – interacting with the cookie notice itself.

Anti-tracking solutions [14, 22] take the browser-level route by blocking all third-party cookies but fail to address non-essential cookies from the first parties. Chen et al. [8] found that existing solutions like disabling third-party cookies or using filter lists do not adequately prevent user tracking. Recently, CookieBlock [4] was proposed as another browser-level solution that takes a machine learning (ML) approach to automatically classify both first and third-party cookies based on their values before blocking the undesired subset. However, as ML is inherently error-prone, its failures within this approach manifest in breaking websites' functionalities due to the deletion of necessary cookies. Furthermore, browser-level solutions prevent the user from leveraging additional opt-out mechanisms provided by the websites. Fouad et al. [21] found that a significant number of websites use cookie respawning, where a deleted cookie is automatically recreated.

In the notice-centric approach, the idea is to interact with the cookie notices themselves, as a human would do. So far, the solutions taking that route have relied on manually analyzing a subset of cookie notices (by major Consent Management Platforms (CMPs)) and hard-coding JavaScript snippets to enforce privacy-respecting cookie choices [24, 38, 46]. As we

*The opinions expressed in this publication are those of the author and not of Google.

[1] A video illustrating this: https://youtu.be/FI0oJF03eCI

show later, these fail to scale to the variety of cookie notices in the wild.

We note that the two solutions described above are orthogonal to each other. While the browser-level solutions override website owners' design and manipulate cookies, the notice-level approach interacts with the notice as a human would, relying on website-provided options. The Notice-centric approach also benefits from the sites' motivation to abide by regulations and from existing regulatory supervision. In the browser-level approach that manipulates cookies, it is often non-trivial to identify non-essential first-party cookies. For example, a popular browser-level solution PrivacyBadger [22] fails to disable non-essential cookies on www.gov.uk.

In this work, we take the notice-centric approach and propose *CookieEnforcer*, the first cookie enforcement controller system that operates at the website UI level while still generalizing to a wide variety of websites, without hard-coded rules. *CookieEnforcer*'s backend locates the fine-grained options for each notice, understands the semantics of the cookie settings, and automatically extracts instructions to disable non-essential cookies. These instructions consist of a set of clicks which can then be executed on the frontend, a Chrome browser extension. Achieving these objectives required (1) building a unified understanding of the cookie control settings that scales across web technologies and (2) providing users with options to enforce only the necessary cookies.

At the core of *CookieEnforcer* are two ML models. The first is an encoder-based classification model (based on BERT [16]) that identifies the cookie notice, leveraging its textual features. The second model is an encoder-decoder model (based on T5 [44]), which takes the text content and selection status of the notice's elements and produces the sequence of steps required to disable non-essential cookies. Around these models, we built several blocks to simulate user interaction with the target website. The final outcome per website is a JavaScript snippet that can enforce the sequence of steps from the decision model. A distinctive aspect of *CookieEnforcer* is that it provides the users with three interface options, ranging from fully automated to semi-automated methods, which cater to the tradeoff between automation on one hand and the sense of confidence in cookie enforcement on the other hand.

We evaluate *CookieEnforcer* from three angles: end-to-end accuracy, user perception, and scalability. First, we perform an end-to-end evaluation on 2000 websites from Tranco [36], assessing the core components. We show that our pipeline correctly generates a sequence of clicks required to disable non-essential cookies for 93.7% of the pages in our manually annotated dataset. Second, we conduct an online user study with 165 participants on Prolific to measure the effectiveness of *CookieEnforcer*'s client implemented as a browser extension. We show that it significantly reduces the time taken to adjust cookie settings on a set of 13 websites (0.9 seconds as compared to 24 seconds for manual baseline). Moreover, *CookieEnforcer* obtained a 37% higher score on System Usability Scale (SUS) compared to the manual baseline. Third, we run the backend of *CookieEnforcer* on the top-100k websites from the Tranco list [36]. This measurement showcases the low failure rate of *CookieEnforcer* and its ability to characterize cookie notices at scale.

2 Background and Related Work

Cookie Notice Studies: Websites use cookie notices to obtain users' consent and (in many cases) allow them to control such cookies. Consent Management Platforms (CMPs) help websites comply with the regulations [30] by offering APIs to manage cookie notices. These platforms are third party integrations, which provide easy solutions for obtaining and storing user consent. As of 2020, the adoption rate of these CMPs was limited to 10% of the 10k most popular websites [30], with many websites opting to implement customized versions of the cookie notice.

Cookie Notice Analysis: Sanchez-Rola et al. [47] conducted a manual analysis of 2k EU websites to determine the extent of cookie-based tracking. They found that 57% of these include cookie notices. Degeling et al. [15] measured the GDPR's impact on cookie notices by manually examining the top 500 websites in each of the EU member states. They found that 62% of the websites serve cookie notices. More recently, Kampanos et al. [33] used a list of common CSS selectors to detect cookie notices in 17k websites in the UK and Greece. They found that 45% of these serve a cookie notice. They also analyzed the notices to check for compliance to find that only a small fraction provides a direct opt-out option. Eijk et al. [18] used a similar methodology to understand the effect of user location on the presence of cookie notices. Matte el al. [40] compared the user options against those stored by the CMPs and found suspected violations. Bollinger el al. [4] analyzed 30k websites and identified several GDPR violations. Finally, Coudert et al. [12] used a keyword based scoring algorithm to detect cookie notices and analyzed them for detecting dark patterns.

Our approach differs from these works in two aspects. First, we present a more robust cookie notice detection that does not rely on keywords or handcrafted rules (which can easily become obsolete). Second, we go beyond detecting cookie notices and extracting dark patterns. We analyze the detected cookie notices to extract and understand its fine-grained options using a deep text-to-text model. We use this understanding to automatically disable non-essential cookies.

Users' Perception and Dark Patterns: Utz et al. [53] conducted a manual analysis to identify common properties of cookie notices. They investigated how these properties impact users' decision to accept/reject cookies, finding that nudging has a large effect on users' choice. Similarly, Machuletz et al. [39] studied how the number of options and presence of "select all" button influences users' decisions. Kulyk et

al. [35] reported that users find cookie notices annoying and disruptive. Nouwens et al. [42] studied the effect of CMPs on people's consent choices by scraping designs from popular CMPs in 10k UK websites, finding dark patterns on most of the websites.

Automated Enforcement: The widespread availability of dark patterns in cookie notices motivated approaches for automated interactions on the user's behalf. Particularly, the browser extensions Consent-O-Matic [46], Cliqz-Autoconsent [38] and Ninja-Cookie [24] automatically enforce users' choices for cookie notices. However, these extensions employ rule-based enforcement and rely on the presence of specific CMPs. This approach does not scale to the majority of websites implementing customized cookie notices. Similarly, other works [4, 9, 31] classify cookies into pre-set categories and provide options to remove these cookies from the browser storage. In these approaches, the user is still required to interact with the cookie notices. We address this limitation by emulating users' interaction with cookie notices.

Another set of works [10, 27, 34] analyze privacy settings pages to present them in a more accessible manner. Specifically, Chen et al. [10] and Khandelwal el al. [34] automatically detect hard-to-find privacy settings on Android and the web respectively. Habib et al. [27] analyze the privacy policies of the websites to determine the opt-out links and presents them to the user. These approaches operate on fairly static webpages, and the user still has to manually interact with the settings. Our work differs in two aspects: First, we cope with the highly-dynamic nature of cookie notices. For example, the cookie settings can be dynamically injected after the user interacts with the notice (*e.g.,* clicks on "More Options"). Second, these systems do not model the choices' semantics. In *CookieEnforcer*, we use this modeling to enable the user to automatically disable the non-essential cookies.

HTML Analysis Techniques To detect cookie notices, *CookieEnforcer* leverages techniques from the HTML rendering process. HTML rendering can be abstracted as a relative ordering of layers (HTML elements) along an imaginary z-axis. The ordering of these layers, *i.e.,* which element is at the top and so on, is determined using the stacking context[2] and stacking order. The stacking order refers to the position of web elements on this imaginary z-axis. Without special attributes, the stacking order is generally the same as the order of appearance in the HTML. This ordering can be altered via a special CSS attribute called *z-index*, where higher *z-index* results in a higher position in the stacking order. The *z-index* is set to "auto" for the elements where it is not specified.

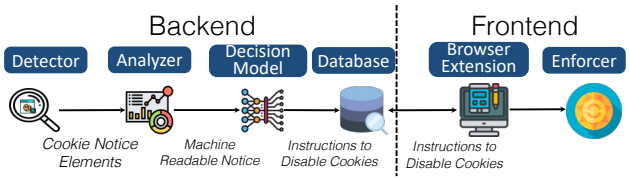


Figure 1: An overview of *CookieEnforcer*'s components. *Backend* generates the machine readable representation of cookie notices whereas *Frontend* uses them to disable non-essential cookies.

3 System Overview

The three objectives of *CookieEnforcer* are to transform the cookie notices into a machine readable format, determine the cookie setting configuration to disable non-essential cookies (whenever possible), and generate a set of instructions that can be used to enforce this configuration. A high level overview of *CookieEnforcer* is in Fig. 1; it utilizes two components to achieve its objectives: *backend* and *frontend*.

Backend: The backend of *CookieEnforcer* consists of three modules. The *Detector* module (Section 4) takes as input a domain name and identifies the web element corresponding to a cookie notice (if present). Then the *Analyzer* module (Section 5) mimics the behavior of a human user by dynamically interacting (performing click actions) with the cookie notice to locate all the adjustable settings. This module accounts for the cases where settings become unhidden or are dynamically injected upon user interaction. It outputs a list of all interactive elements and their associated text description. Next, the *Decision Model* (Section 6) utilizes semantic text understanding to decide on the sequence of actions to disable the non-essential cookies.

Frontend: The frontend consists of the *CookieEnforcer* browser extension which fetches the information for each website from the *backend* and allows the users to disable the non-essential cookies. The extension allows the users to choose between three interfaces (Section 7.2) that provide users with control over the level of automation. Finally, the *Enforcer* (Section 7.1) module performs the actions to disable the non-essential cookies, with or without user intervention – depending on the interface type selected by the user. Note that the cookie notice might not appear if the cookie settings have been decided on before (by the user or the extension).

Challenges: In order to achieve the goals of *CookieEnforcer*, we must overcome three main challenges:

- First, *CookieEnforcer* must identify the cookie notice present on the website. This problem is challenging due to the flexible nature of HTML implementation. For example, prior work [18] that used CSS selectors to detect cookie notices had a high false negative rate of 18%.

[2]For more details on stacking contexts: https://web.dev/learn/css/z-index/#stacking-context.

- Second, *CookieEnforcer* must extract the configurable settings along with their context from the cookie notice. This task is challenging as the interactable elements can (1) be dynamically injected in the notice using JavaScript and (2) exhibit dynamic effect when clicked. This challenge renders a static analysis approach ineffective. For example, in Fig. 3(b), *Save Settings* button submits user preferences whereas the switch disables/enables cookies.
- Third, *CookieEnforcer* must understand the context of each cookie setting. This task is also challenging since the context of the settings (provided by the text describing them) comes from free form natural language, and is diverse. Keyword-based approaches cannot cope with the diversity of text in cookie notices. For example, on `www.virginmedia.com`, the element that reveals fine-grained settings has the text: *"Open the cookie jar"*.

4 Cookie Notice Detector

The *Detector* module detects the presence of cookie notices on webpages. As indicated earlier, this task is challenging as the open nature of HTML allows different implementations of the cookie notices. For example, it is possible to design the cookie notices as floating pop-ups with custom elements, inline frames (*IFrames*), *shadow-roots*,[3] or simply as *div* elements. *CookieEnforcer* addresses these challenges by relying on the global stacking order of HTML.

4.1 Candidate identification

A website serving a cookie notice is expected to surface the notice over the content of the websites to ensure that the user sees the notice and has the opportunity to provide consent. Hence, the elements corresponding to the cookie notices should be higher in the stacking order of the HTML.[4] As described in Section 2, the stacking order determines which element the user sees on the top most layer of the webpage. The *Detector* module leverages this invariant behavior. It looks for non-negative *z-index* attributes within the stacking context to mark candidate elements. However, in practice, not all implementations of cookie notices utilize the z-index to surface the cookie notices. For example, the website `www.gov.uk` shows the notice as the first element in the HTML tree, without utilizing z-index. To capture such instances, the *Detector* module expands the candidate set to include the first three and the last three visible elements of the webpage. We acknowledge that the latter is a heuristic stemming from our empirical observation that, in the cases where the z-index is not used, the notice is present in the top or the bottom of the DOM tree.

[3]For more details: `https://developer.mozilla.org/docs/Web/Web_Components/Using_shadow_DOM`

[4]Technically, cookie notices should be higher in the stacking order of the HTML within the root stacking context. We omit references to the stacking context for simplicity.

4.2 Text Classifier

After obtaining the candidates, our goal is to identify the cookie notice element. We rely on the text in the candidate elements and use a text classifier to perform this task.

Baseline Approach: One approach to perform this classification is to use a keyword-based model as the cookie notice is expected to convey information about the use of cookies. However, this approach is not effective for cases which provide notice and choice without explicitly talking about the cookies. For example, when accessed from the United Kingdom, the cookie notice on `www.arizona.edu` reads: *I have read, understand, and consent to UA's enrollment management Privacy Policy. Consent, Decline*. Therefore, we need a classification model that relies on the text semantics to determine if the candidate element is a cookie notice.

Classifier Choice: We use a text classifier based on BERT (Bidirectional Encoder Representations from Transformers), a Transformer-based encoder pretrained on masked language modeling and next sentence prediction objectives [17]. BERT has been the model of choice for achieving strong performance on a variety of text classification tasks, such as sentiment analysis and topic classification [51]. The key advantage of such a large pretrained model is that it is readily trained on a large corpus, so it can be finetuned on a downstream task with a relatively small dataset. In this work, we finetune the $BERT_{Base\text{-}Cased}$ variant (case-sensitive with 12 layers).

Training/Testing Sets Curation: We create the data for the classifier by sampling 250 websites from the top-50k most popular website list from Tranco [36]. We first extract the candidate elements for each website from this set by using the candidate identification methods. One of the authors then manually annotated each website, indicating whether it showed a cookie notice. The annotation task involved looking at the screenshots of the webpages and identifying if a cookie notice was present. As the task is fairly easy for an expert, we only require one annotation per website. We obtain 112 websites with cookie notices and 138 without cookie notices. We extract at most two candidate elements from each website to obtain a total of 505 candidate elements, 112 of which are notice elements. We keep aside a test set of 100 candidates, 50 cookie notices elements and 50 non-cookie notice elements.

For each candidate, we first extract its text by concatenating the text of all its elements. For example, in Fig. 3(a), the input text for the classifier would be: *We use cookies to improve your browsing experience...to manage your cookie settings, click "More Information". Accept Cookies More Information.*

Training and Performance: Next, we finetune the model on the training set with 62 notice elements and 343 non-notice elements. We use oversampling to ensure that both classes were represented equally. We trained the $\text{BERT}_{\text{Base-Cased}}$ model with a learning rate of $2e^{-5}$ for 10 epochs and used the last

Instances	Support	Recall	Precision	F1-score
Not Cookie notice	50	0.96	0.98	0.97
Cookie notice	50	0.98	0.96	0.97
Total Pages	100	0.97	0.97	0.97

Table 1: Classifier's performance on the test set.

model checkpoint for evaluation. Table 1 shows the performance of the classifier on the test set. The classifier achieves an average F1-score of 0.97, indicating that the model learned to distinguish cookie notice elements from the rest. Analyzing the failure cases, we observe that, in a few cases where the text contained topics other than cookies, the model was confused. We attribute this to the fact that as text about other topics increase, the information about cookie notices present in the text gets diluted, resulting in mis-classification.

5 Cookie Notice Analyzer

The *Analyzer* module takes the HTML element corresponding to the cookie notice as its input and extracts the cookie settings, their current state (selected or not-selected), and the text corresponding to the settings.[5]

The flexible nature of HTML implementations presents two challenges for the *Analyzer* module. First, cookie notices are frequently dynamic. On several websites, the elements corresponding to cookie settings only load when another button is clicked. This renders the static analysis of HTML ineffective. Second, the fine-grained cookie settings in many of the cookie notices are initially hidden. In order to change the fine-grained settings, users have to navigate to a different view (usually by clicking buttons like "Cookie Settings"). This second view is usually a different element in the DOM tree. As a result, *CookieEnforcer* has to keep track of the browser state with respect to the different cookie elements as well as different views of the cookie notice.

CookieEnforcer addresses these challenges by mimicking the actions of real users: it interacts with the cookie notices and observes the behavior of the webpage after each interaction. The *Analyzer* starts by first discovering the elements in the notice with which the user can interact. Next, the *Analyzer* clicks on each element to identify any dynamically injected elements. Finally, it identifies the cookie settings and extracts the text corresponding to those settings.

5.1 Identifying Interactive Elements

CookieEnforcer leverages the *tabbing* feature of HTML to identify the interactive elements within the cookie notice. This feature allows users to access interactive elements via the *Tab* key. Prior work, analyzing the HTML pages to detect privacy settings, also used this technique [34]. The key

[5]Video with steps is at: https://youtu.be/ViyKxbY3rAM.

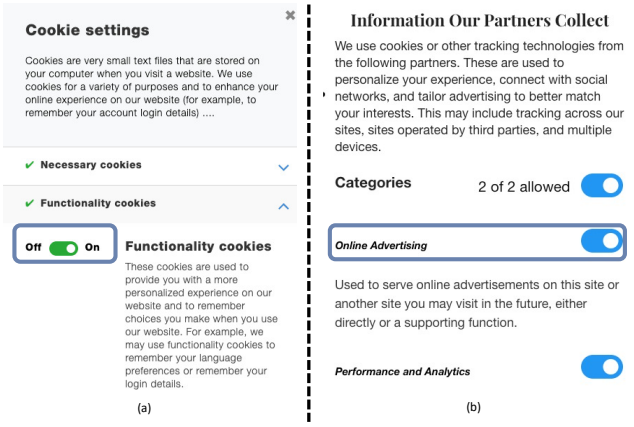


Figure 2: Examples of different types of text extraction. (a) Switch on www.horiba.com has no *aria-label*, text is extracted via HTML code and on-screen distance. (b) The label for switch on www.justinbeaber.com has *aria-label* as Online Advertising.

idea is that, since the users need to interact with the cookie settings to adjust the preferences, we can simulate this interaction via *tabbing* and obtain a set of candidates for cookie settings. By relying on this invariant behavior of the HTML, *CookieEnforcer* extracts the set of candidate cookie settings.

This set of candidates does not contain dynamically injected elements, which are loaded as a result of an interaction with another element. For example, in Fig. 3, the settings appearing after clicking on "More Information" button are dynamically loaded. The *Analyzer* module recursively checks for these elements by clicking each visible element from the candidate set and querying again to find new elements.

After obtaining the candidate elements set, the *Analyzer* module excludes the elements that redirect the user to a different page or open a separate tab. This way, we filter out links for cookie policies, explanations about cookies, and cookie vendor details. A side effect of this decision is that the module also filters out elements which take users to dedicated webpages for cookie settings. We further cover this in evaluation (Section 8.1) and discuss the usability implications of this decision in Section 9.

5.2 Extracting Cookie Settings

At this point, we the analyzer has found all interactable elements in the cookie notice. The next step is to extract the text that describes these settings. This text, combined with the state of the element (selected/not-selected) is needed for the decision model (Section 6) to semantically understand the cookie notice.

Here, we use two independent signals to extract descriptive and concise text corresponding to an HTML element. First, we leverage the *aria-label* attribute,[6] wherever available. This

[6]For more information: https://www.w3.org/TR/wai-aria/#aria-label

attribute allows assisted technologies to read and consume webpages, thereby making web content accessible to users with disabilities. For example, the *aria-label* attribute for the highlighted switch in Fig. 2(b) has a value of "Online Advertising" which describe what setting the switch adjusts.

In the absence of *aria-label* attribute, we design a text extraction technique inspired by Khandelwal et al. [34]. For each interactable element, it searches for the closest parent node in the DOM tree that contains text. However, this parent node might contain other text such as the description of the setting. For example, in Fig. 2(a), ideally we would like the text corresponding to the switch to be *Functionality cookies*, as opposed to *Functionality cookies* together with the description below it. We address this limitation by relying on the on-screen distance to identify the element describing the setting. Specifically, we find the closest (on-screen) text containing element from the cookie setting. In cases with multiple elements with the same on-screen distance, we break the tie first using the x coordinate (and then the y coordinate, if needed). For example, in Fig. 2(a), the closest text element for the switch (marked with the box) is *Functionality cookies*.

The final step in this stage is indexing each extracted HTML element. Prior work [25] has used XML Path Language (*XPath*) [11] to reference the HTML elements. However, we empirically found that, due to the dynamic nature of the notices, *XPaths* for cookie notices are highly vulnerable to change upon page updates (*e.g.* in the DOM tree, notice element can be injected before or after another div element is loaded for ads); hence they are not suitable. Instead, we rely on the `querySelector()` HTML function[7] (which returns the element matching a specified CSS selector or group of selectors in the HTML). Using this function, we construct a path that can be used to identify the elements, even when the placement of the element is dynamic.

5.3 Execution Roles

In order to represent a cookie notice in a machine readable format, *CookieEnforcer* determines the execution role of the elements by interacting (performing the click action) with them and analyzing the effect on the webpage. We define the execution role for all interactive elements within the cookie notice as described in Table 2. These roles categorize the possible outcomes when the user clicks an element in the cookie notice. Type A elements allow a user to adjust their preference for a particular setting. Type B elements reveal new cookie notices. Type C elements reveal hidden settings within a cookie notice (e.g., "Functional and Personalization" tab in Fig. 3). Finally, Type D elements are used to submit the choices. Type D elements typically conclude the users' interaction with the cookie notice. A detailed description of the above types is provided in Appendix A.1

[7]For more details: https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector

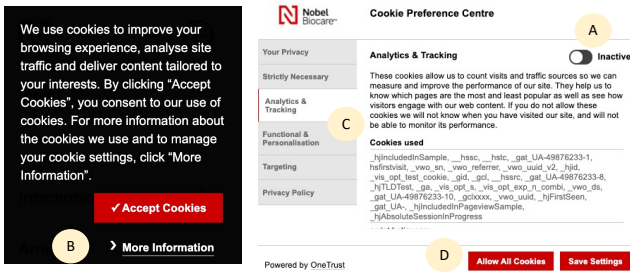


Figure 3: Cookie notices on www.nobelbiocare.com showing elements with different execution roles. (A) Type A element used to enable/disable Analytics cookie. (B) Type B element reveals the second banner shown on the right. (C) Type C element reveals the hidden settings. (D) Type D element to submit the preferences.

Type	Execution Role	Example
A	Configuring choices	A switch enabling/disabling marketing cookies
B	Uncovering hidden notices	Cookie Settings button in Fig. 3 (B) that reveals another notice when it is clicked
C	Uncovering hidden settings	Analytics and Tracking Cookies tab in Fig. 3 (C) that reveals setting which was previously not visible
D	Enforcing choices	Accept Button in Fig. 3 (D) that completes the users' interaction with the notice.

Table 2: Definition of the execution roles with examples.

6 Decision Model

At this stage, we have extracted all the interactable cookie settings, associated them with text, identified their execution role, and determined their addresses. The next step is to determine the actions required to disable non-essential cookies. We employ a decision model that understands the context (as provided by the setting text and the execution role) in which the user interacts with the settings. We then use the contexts of all the settings to determine the configuration required to disable the non-essential cookies.

6.1 Why Natural Language Understanding?

To determine the step required to disable cookies, *CookieEnforcer* needs to parse the different settings across views of the cookie notice and semantically understand them.

One approach to perform this task is to simply deselect all the enabled options and determine which element to click to save the configuration. However, this approach has two main limitations. First, the existing settings are not always enabled or disabled by default. The user might be required to interfere to enable/disable cookies (e.g., www.microsoft.com). Second, the cookie setting might be worded in a way where the element needs to be selected to disable non-essential cookies.

For example, the option can be: *Only allow necessary cookies*. Deselecting this option will lead to undesirable outcomes. Hence, it is important to account for the text of the element.

Another approach to formulate the task is: given the text associated with the element and its execution role, determine if it should be clicked. The major drawback with this approach is that it models the task as a series of decisions without considering the interplay between these decisions. For example, if a site shows three buttons: "Accept all," "Save custom preferences," and "Reject all," we want to click on "Reject all." However, if another site only shows "Save custom preferences" and "Accept," we want to select "Save custom preferences," so it is sub-optimal to treat them independently.

Thus, we observe that an effective decision model should meet two requirements: a) semantically understand the text corresponding to the options and b) determine the series of actions required by simultaneously considering for all options.

6.2 Extracting Actions to Disable Cookies

Our goal here is to develop a decision model that takes in the text corresponding to all the cookie settings and their current state (selected or not-selected), and determines the actions required to disable the non-essential cookies. We model this problem as a sequence-to-sequence learning task where the model gets the text and the state and determines the steps required. Specifically, we train a Text-To-Text Transfer Transformer (T5) model as the decision model.

The T5 model, introduced by Raffel et al. [44], proposes a unified framework that treats all NLP tasks as text-to-text problems. This model has been shown to have a strong performance on a variety of NLP tasks ranging from classification to generation problems [23, 29, 48]. The general approach of serializing structured steps into text has also been used to achieve state-of-the-art results in the data-to-text generation community [28, 32]. For our purposes, we fine-tune a T5-Large model (770 million parameters) to produce a sequence of steps (clicks) required to disable the cookies.

Guided by the elements' execution roles (Table 2), we first transform the information stored about the cookie notice in a single sentence format. Specifically, Type A elements have a state associated with them (selected/not-selected) whereas the other elements do not. The state of Type A elements allows the model to understand that these elements are configurable. Then we train the model to produce a text indicating which elements to click, given the text representation. The input and output for the T5 model would take the following format:

Input-Output format for the Decision model.

Input: < notice_0_tag_0> - <notice_0_tag_0_text>,
<notice_0_tag_0_state> || <notice_0_tag_1> - <notice_0_tag_1_text>,
<notice_0_tag_1_state> - . . . **
<notice_1_tag_0> - <notice_1_tag_0_text> - . . . <end>
Output: Click <notice_0_tag_0> | Click <notice_0_tag_2> ** Click
<notice_1_tag_2>

The ** symbol separates multiple notices' contents in the input and the output. The || symbol separates the settings options within the same notice in the input. The | symbol separates the click steps within the same notice in the output. Note that the state for an element is only defined if it belongs to Type A. For example, Table3 shows the input and output for the T5 model corresponding to the cookie notices on `www.askubuntu.com` shown on (1) and (4) in Fig. 4.

We note here that some websites provide an option to opt-out of non-essential cookies on the first cookie notice but can have pre-selected options on the second. When creating the training data, we chose to disable the cookies from the first notice only. This emulates the behavior of the human who would not click to see more options if the option to reject non-essential cookies was provided. The model learns this behavior too upon training. This way, the decision model, given all the options available on a given webpage, can predict what actions to take to disable non-essential cookies.

6.3 Training and Performance

To create the dataset for the decision model, we first sample 300 websites with cookie notices from Tranco's top-50k popular website list [36]. Next, we analyze the sites using the *Detector* and the *Analyzer* module to extract the options and their states (selected or not-selected). Then one of the authors manually determined the series of clicks required to disable the non-essential cookies. This resulted in a dataset of 300 labeled websites. Next, we keep 60 websites aside for the test set. We further ensure that the test set consists of customized in-house notices as well as notices from different CMPs to ensure diversity.

We test the performance of the model by measuring its quality via the exact match percentage metric: the generated sequence should be exactly the same as the ground truth. We find the exact match % of the model on the test set to be 95%, indicating that the model has succeeded in learning the task across a variety of websites. For example, take the input:
Input : switch0 - do not allow non-essential cookies, not selected || button1 - save || button2 - accept <end> .
The model correctly generates:
Output : Click switch0 | Click button1.
This selected wording (involving a double negative) was not present in the training set. The most similar phrase was: *do not sell personal information*. The failure cases included notices

Website	Input	Output
netflix.com	button1 - learn more about our use of cookies and information. || button4 - accept || button5 - reject || button6 - personalise my choices || button7 - close ** button0 - close ... || save settings <end>	Click button5.
tata.com	button0 - sweet! || button1 - sorry, i'm on a diet <end>	Click button1.
askubuntu.com	button0 - customize settings || button1 - accept all cookies ** switch3 - performance cookies, not selected || switch4 - functional cookies, not selected || switch5 - targeting cookies, not selected || button6 - confirm my choices || button7 - accept all cookies || button8 - cancel <end>	Click button0 ** Click button6.

Table 3: Examples demonstrating the application of *Decision model* on cookie notices for a few websites. We show the screenshots corresponding to these cookie notices in Fig. 11 (in Appendix A.4). Note that for www.tata.com, the options are non-standard but the decision model is still able to reject the cookies. More examples can be found in Table. 9.

where the number of setting options is large and the input to the model is truncated. Note that we further evaluate the performance of the decision model with a larger dataset in the end-to-end evaluation (Section 8).

It is worth noting too that the exact match % is a conservative metric. In practice, it can be relaxed depending on the output sequence. For example, the relative order of clicking on two switches is not important, but clicking the "Save" button before clicking a switch might give undesirable outcomes.

Table 3 shows three examples from applying the decision model on a diverse set of cookie notices (the screenshots for these notices are shown in Fig. 11 of Appendix A.4). Notably, we see that, for www.netflix.com, there are two views for the cookie notice with second view consisting of fine grained options. However, since the first view contains a *reject* button, the decision model only clicks on it. Another interesting example is www.newscientist.com. Apart from the regular switches, the second view for the cookie notice on this website contains an option to *object to legitimate interests* for basic ads. This option can be easily missed by the users as they have to expand an additional frame to see that. *CookieEnforcer* not only finds this option, but also understands the semantics and decides to object. These examples showcase that the model learns the context and generalizes to new examples.

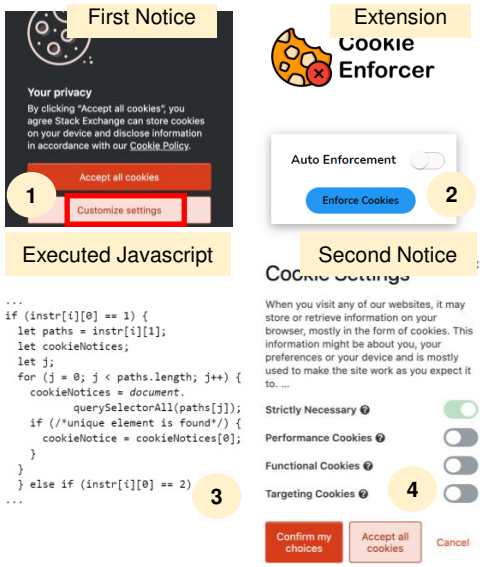


Figure 4: A typical workflow of *CookieEnforcer* extension with semi-automated enforcement. (1) First the user visits www.askubuntu.com. (2) User activates the plugin and instructs the extension to disable non-essential cookies. (3) *CookieEnforcer* retrieves the information (locally) and generates the Javascript required. (4) Adjusted settings before submitting preferences.

7 Frontend

The frontend of the *CookieEnforcer* is a browser extension for Google Chrome. It has two components: the *Enforcer* module and the user interface.

7.1 Enforcer

The frontend receives the set of instructions for a specific website from the backend.[8] These instructions contain the `CSS selector` path for the cookie notices. Further, for all cookie setting elements, the instructions also contain 1) path of the element relative to cookie notice, and 2) desired state of the setting element. Using the `CSS selector`, the extension first determines whether the cookie notice is present. Then, it accesses each cookie setting using the relative paths and adjusts it to get the desired state. For example, for a slider, if the desired state is `disabled` but the current state is `enabled`, the *Enforcer* clicks on the element to change the state. A more detailed example, including a snippet of the Javascript code is shown in the Appendix A.2.

Why Not Replay Cookies: An alternate solution to performing the clicks at client side is to extract the actual cookies set by websites after setting the optimal configuration in the backend and replay them at the client side at run-time. There would be no need to perform the clicks in this case. Note that the *backend component of CookieEnforcer is still required to determine the optimal configuration*. This solution, however, has several drawbacks: (1) The websites do not need to store consent string as a cookie - they can store it at other locations in the browser. For example, https://seminolestate.edu stores the consent string in local storage. (2) Some cookies can also

[8] We discuss the deployment options to deliver instructions in Section 9

have encrypted ids before the preferences. In these cases, it not clear how the cookies interacts with the server database and whether it breaks the usability. (3) In some cases, cookies are set before the user interacts with the notice and replayed cookies do not overwrite existing cookies.[9]

Additionally, we note that one major advantage of performing clicks over replaying cookies is that it helps *CookieEnforcer* identify whether there has been a change in the cookie notice implementation on a website. For example, if a website changes CMPs, the cookie replay solution will be ineffective. However, *CookieEnforcer* can detect the staleness of its paths and trigger a re-analysis to update them. Generally, performing clicks is a preferable solution as it works within the framework provided by the websites, which can be expected to remain an invariant due to usability aspects.

7.2 User Interfaces

The core functionality of *CookieEnforcer* is to disable non-essential cookies (wherever possible). As discussed above, this requires *CookieEnforcer* to perform clicks on behalf of the users, who may have different levels of comfort with such a form of automated enforcement. To account for such preferences, *CookieEnforcer* consists of three interfaces (described below) that allow various degrees of control. Each interface comes with its own usability aspects and lies on one point of the tradeoff line between full automation and full control. We provide a discussion on this trade-off in Section 8.2.2.

Semi-Automated Enforcement: Upon detecting the cookie notice, the user first activates the extension by clicking on its icon, thus showing an "Enforce Cookies" button on the extension popup. Clicking on it activates the *Enforcer* module (discussed below) which performs the actions. This mode allows the users to see how the enforcement is happening by artificially introducing delays between the clicks. The workflow for this mode is shown in Fig. 4. A demo for this mode is shown at https://youtu.be/gasSjHo8Zwk.

Fully Automated Enforcement: The extension detects the cookie notice and automatically triggers the *Enforcer* module to disable non-essential cookies. Furthermore, the extension hides the cookie notice(s) and performs the clicks in the background so that the browsing experience of the users is not impacted. Thus, in this mode, the user does not interact with the cookie notice or the extension. A demo for this mode is shown at https://youtu.be/f8rtTUwIHlU.

Informed Enforcement: The extension first checks whether the cookie notice is present. Upon detecting the cookie notice, the website then overlays a pre-recorded GIF showing the actions that the extension will perform. The user can see the GIF and click to provide informed consent to the extension, which then triggers the *Enforcer* module. A demo for this mode is shown at https://youtu.be/eh7a35oaKlU.

[9] We noticed such behavior on https://ffii.org/.

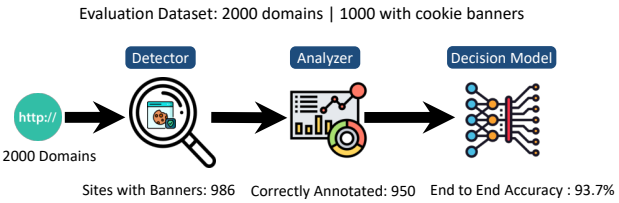


Figure 5: The results from evaluation of *CookieEnforcer* shows that the system performs well on the test set.

8 Evaluation

We perform multiple experiments to evaluate the accuracy, usability and stability of *CookieEnforcer*, as well as to showcase its utility on a large-scale dataset. We seek to answer these questions:

Q1. *What is the end-to-end performance of CookieEnforcer?*
Q2. *Does CookieEnforcer improve the user experience?*
Q3. *Can CookieEnforcer analyze cookie notices at a scale?*

8.1 End-to-End Evaluation

We conducted a manual end-to-end quality evaluation of *CookieEnforcer* on 2000 sites to assess the accuracy of extracting a machine-readable representation of cookie notices (if present) on previously unseen domains. Fig. 5 provides an overview of these steps. We also evaluated the generalizability of *CookieEnforcer* by analyzing sites from different locations (London, California, and Illinois). Finally, we demonstrated the temporal stability of the extracted cookie notice representations, indicating the feasibility of offline deployment.

Dataset: The evaluation dataset for CookieEnforcer consisted of 2000 diverse domains, including 250 from the top-1k of the Tranco list [36] and the remaining domains in the 1k-50k range. To increase the likelihood of encountering cookie notices, the evaluation was conducted from a GDPR-covered location (London, UK) via a VPN. The dataset for the *Detector* module was manually annotated[10] by taking screenshots of the websites and judging the presence of a cookie notice. Annotating the dataset for the *Analyzer* module *beforehand* was infeasible due to lack of unique identifiers for cookie setting options, so we manually verified the presence of cookie settings *after* passing the data through the *Analyzer*. To create the annotated dataset for *Decision Model*, we obtained input strings from the *Analyzer* module and manually wrote the expected output strings required to disable cookies.

Findings: The evaluation set's 2000 domains were first processed by the *Detector* module. The module identified 986 domains as having cookie notices, with 2 false positives and 16 false negatives. The *Analyzer* module was then used to identify the various cookie settings present in the notices

[10] Since these tasks are objective/deterministic in nature, one of the authors did the manual annotation.

from these domains. Manual verification was performed, and a website was counted as an error if the *Analyzer* missed at least one cookie setting. The Analyzer correctly identified the options in 950 domains. It also filtered out the false positives from the previous stage as the elements on those pages only had out-of-page links. The *Decision Model* was applied to the elements from the remaining 950 domains. The model's outputs were compared to the manually written ones using exact sequence match as the metric (cf. Sec. 6.3). The decision model accurately predicted the steps for 937 domains, resulting in an accuracy of 93.7% for *CookieEnforcer* in the end-to-end evaluation.

Error Analysis: The Detector module failed to detect cookie notices on a total of 16 domains. Seven of these domains had the notice present in a "shadow-root" element[11], which is rendered separately from the document's main DOM tree and is inaccessible using Selenium. Four domains had notices that were only displayed for a short time (∼6 seconds) and were missed by the tool due to its included delay to allow all elements to load. The remaining four websites had notices with missing z-index attributes. The *Analyzer* module failed on 36 domains. In 12 of these, the cookie settings were present on a different URL and were filtered out, resulting in erroneous representation. On the remaining 22 domains, the cookie notice had non-standard behavior, such as elements that were not reachable via tabbing, long delays between the first and second notice, or non-standard implementation of "checkboxes" and "radio" buttons. Finally, the *Decision Model* failed on domains with large numbers of settings, due to model input truncation[12]. For example, https://www.blu-ray.com/ has over 100 settings, leading to a very long input.

Privacy Implication of Errors: In *CookieEnforcer*, errors can arise due to: (a) the *Detector* missing the notice (b) the *Analyzer* missing cookie settings, or (c) the *Decision Model* resulting in incorrect instructions. Detection errors (16/1000) and errors in the decision model due to input string size (15/1000) do not pose a privacy risk as *CookieEnforcer* does not take any actions on these websites and leaves the notice for the user to interact with. However, instances where the *Analyzer* misses some settings may lead to keeping non-essential cookies enabled, posing a privacy risk. These instances occurred 3.6% of the time in the evaluation set.

Generalizability: To evaluate *CookieEnforcer*'s generalizability, we analyzed cookie notices from 937 correctly annotated websites in the testing set via VPNs in London (LDN), California (CA), and Illinois (IL). Comparing LDN vs. CA, 70.6% of the analyzed domains had the same notice while 17.2% did not show a cookie notice when accessed from CA. The remaining 12.2% had different notices, some of which provided a "Do Not Sell My Information" link or lacked fine-grained opt-out options. However, *CookieEnforcer* correctly generated instructions to disable non-essential cookies for these domains. We note that different notices based on location do not pose issues during deployment because we can identify the location of web requests and serve the appropriate instructions. When comparing domains accessed from CA vs. IL, we found that only 1.2% websites had a different notice. For example, https://www.prada.com/ had an option to reject cookies on the first notice when accessed from CA, but not from IL. Notably, *CookieEnforcer* generated accurate instructions to disable non-essential cookies for all websites with different notices.

Feasibility of Offline Deployment: We evaluate the feasibility of offline deployment by performing temporal analysis on the generated instructions Specifically, we generate instructions to disable non-essential cookies on websites when accessed from London and verify their stability over one month. We note that we only use the 937 correctly annotated websites as we want to measure the stability of generated instructions. For these sites, we generate the instructions on December 5, 2022 and then manually verify whether the non-essential cookies are disabled using these instructions over a period of one month. At the end of this period, we find that the instructions only fail in less than 1% of the websites, primarily because the cookie notice has either disappeared or changed. This demonstrates the stability of *CookieEnforcer*-generated instructions over time. For cases where the notice changed, *CookieEnforcer* was able to re-analyze and generate the correct instructions. We also note that the evaluation set included websites with frequent layout changes, such as news websites.

Comparison with Existing Tools: We also compared the performance of Ninja Cookie [24] and Consent-o-matic [46] (both discussed in Sec. 2) with CookieEnforcer on our evaluation set (250 websites out of the 1000 with notices). We define the failure metric as the fraction of websites on which the cookie notice remained visible, despite the availability of the plugin. We observe that Ninja Cookie fails in 50% of the websites, whereas Consent-o-matic fails in 76% of the websites, as compared to the failure rate of 9% for CookieEnforcer. Further, both plugins appeared to hide the notice only on those websites that included CMPs, which is consistent with a rule-based approach. Note that a rule-based approach for CMPs has pitfalls as the websites can adapt the CMP to their use case by customizing settings.

Even with the variations in the HTML and the dynamic nature of elements in the cookie notice, our pipeline accurately generates the steps required to disable non-essential cookies in 937/1000 websites (overall accuracy of 93.7%). Unlike the alternative in-browser solutions, *CookieEnforcer*'s errors do not lead to breaking functionality and carry a minimal risk (3.6%) of keeping non-essential cookies enabled.

[11] For more details: https://shorturl.at/hiuY6

[12] We ran the T5 model inference with 256 tokens only, but it is possible to run it with a much larger number.

8.2 User-based Evaluation

We conducted a user study on Prolific to evaluate *CookieEnforcer* with 165 US-based participants who had a >95 approval rate and at least 10 accepted submissions on the platform. 52% of participants were female, 42% were male, and 48% held a Bachelor's degree. The average age range was 25-34 years. The study, which was approved by the IRB at our institute, lasted an average of 17.3 minutes with a median time of 15.4 minutes. Participants were paid $3 and no personally identifiable information was collected.

8.2.1 Study Design

We evaluate the usability of interfaces of *CookieEnforcer* by designing a *within-subject* study, where the same user was exposed to four conditions. The interfaces of *CookieEnforcer* (Section 7.2) make up three of the conditions, while the manual baseline is the last condition. We instructed each participant to visit four websites and disable all non-essential cookies using these conditions. As the goal of this study was to measure the usability of the extension, we did not obscure the goal of disabling cookies in the experiment.

Website Selection: As our primary objective in this study is to measure the usability of *CookieEnforcer*, we require that the participants are able to see and interact with the cookie notices on the selected websites. Therefore, to minimize overlap with users' browsing history, we purposefully select a set of 13 non-popular websites. Of the 13 websites, 10 websites use 9 distinct Consent Management Platforms (CMPs) whereas 3 websites implement their own cookie notice. Additionally, 6 websites have an option similar to "Reject All" that allows disabling multiple cookies with one click. These selection criteria ensure that the selected websites have a diverse set of cookie notices. More details on the website selection and a full list of websites can be found in Appendix A.3.2.

Study Flow: During the study, participants first install the *CookieEnforcer* extension from the Google Chrome Webstore. Then four websites are chosen at random from those in our pool such that the browser has no cookies for them. In the manual condition, the participants interact with the cookie notice to disable the non-essential cookies whereas, in the other three conditions enabled by the extension (cf. Section 7.2), the participants use the given interface to complete the task. For each interface type, the participants are shown how to perform the task on a sample website before they attempt the task. Note here that the order of conditions, as well as condition assignment per website, were randomized to counteract learning and fatigue effects.

As the participants interact with each cookie notice, we record the total time and the number of clicks it takes for them to adjust the cookie settings. We define the start time as the time when the website has loaded in the browser (measured via the extension). We also monitor the elements clicked, which are used to determine the end time for the task. After each task, the participants fill the System Usability Score questionnaire [6]. After completing all tasks, the participants are required to fill a post-study questionnaire consisting of two questions: (1) Their preferred interface choice for setting cookie preferences (including the manual baseline). (2) Their level of comfort with the error rate of *CookieEnforcer*. Specifically, we first informed the participants that *CookieEnforcer* could make errors that might result in some tracking cookies being enabled. We then asked them about the error rate they would be comfortable with. After both questions, the participants were asked to explain their choice in an open-ended question. Finally, there was an open-ended question asking for general feedback at the end of the survey. More details about the user study are included in the Appendix. A.3.

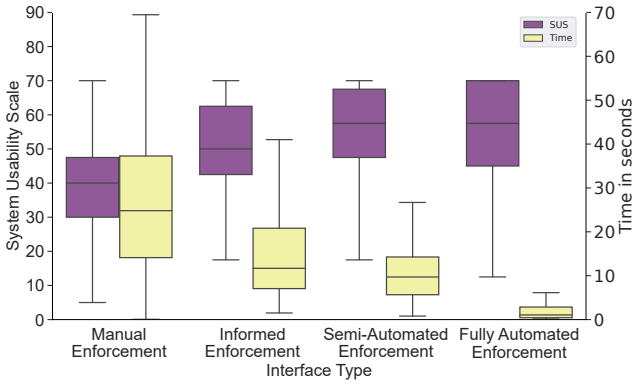


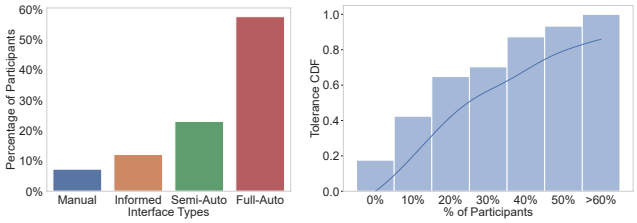
Figure 6: The results from the usability study showing the variation in usability and time needed for enforcement with different interface types. We find that the usability score of **Fully Automated mode** is significantly higher than the manual baseline ($p = 1.2e^{-16}$).

Ethical Considerations We collect user clicks as part of the study to determine the time taken to finish the tasks and estimate user effort. Participants of the study are made aware of this ahead of time. Further, we note that there is a safeguard built in the plugin which allows data transfer only for websites in the user study. Additionally, we instructed the participants to delete the extension at the end of the study to ensure that we do not collect any data outside of the user study.

8.2.2 Findings

We assess the usability of the four interfaces (three interfaces for *CookieEnforcer* and the manual baseline) using two metrics: 1) usability score as measured by the SUS survey and 2) user effort as measured by the time taken by participants to disable non-essential cookies.

Fig. 6 compares the System Usability Scale (SUS) score for the four interface types. SUS scores [3] are used in the literature to evaluate different UI designs. With an average score of 54, the fully automated interface outperforms the



(a) Preference for interface type. **(b)** Error Tolerance for *CookieEnforcer*.

Figure 7: (a) Majority of the participants preferred the fully automated interface as to not interact with the cookie notice. (b) 87% of the users are comfortable with an error rate of 10%.

manual baseline (SUS: 38) and informed interface (SUS: 50). We test the statistical significance of the change in usability score using the Wilcoxon signed-rank test [54] and find that the result is statistically significant (after correcting for multiple hypothesis testing). We note that the semi-automated interface had a close SUS to the fully automated interface.

Next, we compare the average time taken by the participants to complete the task in Fig. 6. We find that the fully automated interface required the least amount of user effort, with the median time being 0.9 seconds compared to > 9 seconds for the other interfaces (24 seconds for manual baseline). We again test the statistical significance using the Wilcoxon signed-rank test [54] and find the results to be statistically significant. Additionally, in the baseline (manual) system, users required an average of seven clicks (not known apriori) to complete the tasks whereas the user had to click on a consistent button in the informed enforcement mode and on the extension icon (plus another consistent button) in the semi-automated mode. Thus, the *CookieEnforcer* plugin significantly reduces the time taken by users to disable non-essential cookies. In Appendix A.3.3, we include the full pairwise statistical significance comparisons among the various interfaces for the SUS and the time metrics.

User Preference for Interface: We analyze the participants' preference for user interface while adjusting cookie settings (Fig. 7a). We find that 92.73% of the participants preferred an interface of *CookieEnforcer*, with 62% choosing the fully automated interface. This is reflective of the value users have found in usability evaluation (Fig. 6). Further, analyzing the qualitative responses, we observe that participants liked the fully automated interface due to its efficiency. For example, one user stated "*It is the most hands-off option, and it helps me know that my data is taken care of without me having to remember to do so after each page.*" On the other hand, participants who preferred the manual interface indicated that they liked the fact that it puts humans in charge. It is noteworthy that despite the requirement of informed consent (by GDPR, CPRA), the users overwhelmingly favored the automated interface due to its ease of execution, which is an increased evidence on the convenience of delegating consent [41].

Usability of *CookieEnforcer* with Errors: Next, we analyze the self-reported error tolerance of the participants. We define error rate here as the fraction of websites where at least one of the non-essential cookies is enabled. Fig. 7b shows the error rate that users are willing to tolerate while using *CookieEnforcer*. We find that 83% of the participants have $\geq$ 10% tolerance for errors (self reported). We also note that majority of the remaining 17% of users who do not tolerate any tracking cookies desire to set their cookie preferences manually. Recall that *CookieEnforcer* was found to allow non-essential cookies in only 3.6% of the cases. Comparing this with the tolerance rate reported by the users, and considering usability scores of baseline and fully automated interface, we conclude that *CookieEnforcer* is a usable tool that can reduce the user effort required to adjust cookie preferences.

8.3 Scalability Analysis

We demonstrate the scalability of *CookieEnforcer* by running it on a total of 100k websites and showcasing the insights that can be gleaned.

8.3.1 Dataset and Setup

We use the top-100k websites from the Tranco list[13] generated on 21 March 2022 [37]. We filter out 7,784 websites that have non-English cookie notices using the *langdetect* library [13]. Additionally, we filter out 6,743 websites that were not accessible using the automated browser. At the end of this filtering, we perform our analysis on 85,473 websites by running the backend of *CookieEnforcer*. Further, to capture maximum cookie notices, we perform this analysis by accessing the websites from the United Kingdom via a VPN.

8.3.2 Analysis

Characterizing the Dataset: Table 4 shows a breakdown of the cookie notices obtained using *CookieEnforcer* on the 85.47k websites in our dataset. First, we find that ***CookieEnforcer* detects a cookie notice on 52.7% of the websites**. Previous works that manually analyzed notices at a small scale found notices in 57-62% of the websites [15, 47], indicating that our estimate is within the expected range.[14]

Second, we note that **35.4% of the websites with notices in our dataset had cookie notices with multiple views while 64.6% had a single-view cookie notice**. The average number of settings options for single view notices was 2, whereas, for notices with multiple views, it was 28 (with a standard deviation of 103). This can be attributed to the

[13] Available at https://tranco-list.eu/list/K2JLW/1000000

[14] The caveat when comparing against previous automated approaches is that they operated at a smaller scale (with different data) and had lower detection accuracy. For instance, those using keyword-based CSS selectors found that 40-45% of the websites in their dataset contained cookie notices [18, 33].

Type	% Websites	Avg. # Settings (per site)	One Click Opt-Out (% websites)
No Notice	47.3	–	–
Single View	64.6	2.17 (3.01)	11.5
Mutliple Views	35.4	28.7 (103)	9.96

Table 4: Details about the analyzed websites. After the first row, the percentage is calculated with total number of websites with a cookie notice (45,044).

fact that some websites have over 1000 options to choose from, including individual options from individual vendors (https://www.formula1.com/). We do note that number of websites with such behavior is < 5%. As such, the median number of settings on multiple views is 9, potentially inviting users to navigate multiple notices and interact with tens of settings. With *CookieEnforcer*, the users can disable cookies without interacting (with as low as zero clicks).

Finally, we approximate the number of websites that provide a One-click opt-out mechanism by analyzing the output of the *Decision model* (Section 6) and counting instances where only one click was required. We discard websites with a single view notice and one setting option (usually "Accept All"). We find that in total, **only 21.5% of the websites with cookie notices provide a One-click opt-out mechanism**, indicating the need for a higher cognitive load on users' behalf to determine the steps to disable non-essential cookies.

Timing Analysis: We calculated the average time taken by the three major components of *CookieEnforcer*: (1) Cookie Notice Detection (Section 4) (2) Analysis of First View (3) Analysis of Additional Views. On average, *CookieEnforcer* is able to detect notices in 26.6 seconds, whereas the analysis of first and additional views takes 103 and 300 seconds, respectively. This shows that *CookieEnforcer* can analyze a website with notice(s) within minutes, an important desirable feature for scalability. Furthermore, this analysis is highly amenable to parallelization and further time optimization, both per website and across websites.

Failure Cases: As indicated in Sec. 8.1, there are cases where *CookieEnforcer* would fail to generate a machine-readable representation of a cookie notice. Our analysis of the 85,473 websites allows us to measure the cases where *CookieEnforcer* fails at a scale. These cases primarily include HTML implementations of cookie notices and settings that are not part of *CookieEnforcer*'s design, such as using *div* to implement checkboxes. We found that such failure cases account for less than 5% of the analyzed websites and do not impact the scalability of *CookieEnforcer*. We obtain this error rate by examining notices, which consisted of multiple views but no Type A elements, and by analyzing the errors in logs.

9 Discussion

Deployment Aspects: *CookieEnforcer*'s frontend needs to access the enforcement instructions per website in order to operate. There are multiple options for such storage that lie on the usability-privacy spectrum. The default option we will provide is the privacy-preserving one with the instructions prepackaged with the extension. These instructions can be alternatively hosted on a third-party server to reduce the storage overhead. This involves disclosing first-time visits to websites, so the user has to select a trusted server. Other solutions can leverage private information retrieval techniques to balance storage requirements and privacy properties.

Informed Consent: One challenge with automated solutions for privacy enforcement, such as *CookieEnforcer*, is informed consent. The philosophy of cookie notices is to allow the user to be aware of and control how each website uses cookies to track them. An automated solution might deprive the users of exercising informed consent. However, we note that recent regulations, such as California's CCPA [50], contain provisions for "authorized agents."[15] Consumers might exercise their privacy rights through a registered authorized agent. A legal entity can incorporate *CookieEnforcer* to act on behalf of the users regarding cookie notices. The ensuing challenge relates to potential errors in *CookieEnforcer*'s operation. As an automated solution, errors are inevitable. While participants in our user study indicate some error tolerance, solutions like *CookieEnforcer* have to be clear about the possibility of errors and their impact on users' privacy.

Limitations: A major limitation for *CookieEnforcer* comes from variability in HTML implementation. For example, *CookieEnforcer* relies on an accessibility feature (*tabbing*) to identify the interactive elements in the cookie notices. However, as we noted in our evaluation, the websites can implement buttons that do not fit these criteria. However, we empirically observe such websites to be rare.

We consider dedicated cookie setting pages as another limitation for *CookieEnforcer*. While extracting the interactable elements in the *Analyzer* module, we filter out out-of-page elements, including elements pointing toward a dedicated cookie settings page. A potential solution is to analyze the page, determine if it is a cookie settings page, and notify the user to adjust preferences.

Finally, we note that *CookieEnforcer* can fail during enforcement on the client-side. This failure can result from a change in cookie notice or the elements within it going stale. These failure modes can be detected via the plugin, which can (after user consent) trigger a re-generation of the instructions by the *Backend* of *CookieEnforcer*. These failures would only result in the cookie notices staying on the screen, and the user can then submit their preferences.

[15] https://privacy.microsoft.com/en-us/ccpa-guidance

10 Conclusion

In this paper, we present *CookieEnforcer*, which, given a domain, automatically detects the cookie notice, extracts the options provided and transforms the cookie notice into a machine readable format. It then uses a text-to-text deep learning model to understand the different options provided and determines the steps required to disable non-essential cookies. The machine readable format of the cookie notice further enables more usable interfaces to be built. Finally, we have extensively evaluated *CookieEnforcer* for performance, stability, usability and scalabiltiy. We find that it accurately annotates the cookie notices of a given domain, and is scalable. Further, the users also found *CookieEnforcer*'s interface more usable compared to the existing baseline.

Acknowledgments

This work was supported by the NSF through awards: CNS-1942014 and CNS-2003129, and by gifts from NVIDIA and Google. Finally, we thank the reviewers for their fruitful discussions and recommendations.

References

[1] European parliament, council of the european union. directive 2002/58/ec of the european parliament and of the council of 12 july 2002 concerning the processing of personal data and the protection of privacy in the electronic communications sector (directive on privacy and electronic communications), july 2002. https://eur-lex.europa.eu/eli/dir/2002/58/oj.

[2] European parliament, council of the european union. regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation), april 2016. http://data.europa.eu/eli/reg/2016/679/2016-05-04.

[3] A. Bangor, P. T. Kortum, and J. T. Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

[4] D. Bollinger. Analyzing cookies compliance with the gdpr. Master's thesis, 2021.

[5] I. Borberg, R. Hougaard, W. Rafnsson, and O. Kulyk. "so i sold my soul": Effects of dark patterns in cookie notices on end-user behavior and perceptions.

[6] J. Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[7] J. Brookman, P. Rouge, A. Alva, and C. Yeung. Cross-device tracking: Measurement and disclosures. *Proc. Priv. Enhancing Technol.*, 2017(2):133–148, 2017.

[8] Q. Chen, P. Ilia, M. Polychronakis, and A. Kapravelos. Cookie swap party: Abusing first-party cookies for web tracking. In *Proceedings of the Web Conference 2021*, WWW '21, page 2117–2129, New York, NY, USA, 2021. Association for Computing Machinery.

[9] Q. Chen, P. Ilia, M. Polychronakis, and A. Kapravelos. Cookie swap party: Abusing first-party cookies for web tracking. In *Proceedings of the Web Conference 2021*, WWW '21, page 2117–2129, New York, NY, USA, 2021. Association for Computing Machinery.

[10] Y. Chen, M. Zha, N. Zhang, D. Xu, Q. Zhao, X. Feng, K. Yuan, F. Suya, Y. Tian, K. Chen, et al. Demystifying hidden privacy settings in mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 570–586. IEEE, 2019.

[11] J. Clark, S. DeRose, et al. Xml path language (xpath), 1999.

[12] R. Coudert. Automatically detect dark patterns in cookie banners. 2020.

[13] M. M. Danilak. Langdetect, python library to detect language.

[14] C. T. David Cancel. Ghostery. https://www.ghostery.com/, 2010.

[15] M. Degeling, C. Utz, C. Lentzsch, H. Hosseini, F. Schaub, and T. Holz. We value your privacy... now take some cookies: Measuring the gdpr's impact on web privacy. *arXiv preprint arXiv:1808.05096*, 2018.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] R. v. Eijk, H. Asghari, P. Winter, and A. Narayanan. The impact of user location on cookie notices (inside and outside of the european union). In *Workshop on Technology and Consumer Protection (ConPro'19)*, 2019.

[19] S. Englehardt and A. Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1388–1401, 2016.

[20] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web*, pages 289–299, 2015.

[21] I. Fouad, C. Santos, A. Legout, and N. Bielova. Did i delete my cookies? cookies respawning with browser fingerprinting. *ArXiv*, abs/2105.04381, 2021.

[22] E. F. Foundation. Privacybadger. https://privacybadger.org/, 2014.

[23] D. Furrer, M. van Zee, N. Scales, and N. Schärli. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*, 2020.

[24] T. Goudout. Ninja cookie. https://gitlab.com/ninja-cookie/ninja-cookie, 2020.

[25] R. Gunawan, A. Rahmatulloh, I. Darmawan, and F. Firdaus. Comparison of web scraping techniques: regular expression,

html dom and xpath. In *International Conference on Industrial Enterprise and System Engineering (IcoIESE 2018) Comparison*, volume 2, pages 283–287, 2019.

[26] H. Habib, S. Pearman, J. Wang, Y. Zou, A. Acquisti, L. F. Cranor, N. Sadeh, and F. Schaub. "it's a scavenger hunt": Usability of websites' opt-out and data deletion choices. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020.

[27] H. Habib, Y. Zou, A. Jannu, N. Sridhar, C. Swoopes, A. Acquisti, L. F. Cranor, N. Sadeh, and F. Schaub. An empirical analysis of data deletion and opt-out choices on 150 websites. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, 2019.

[28] H. Harkous, I. Groves, and A. Saffari. Have your text and use it too! end-to-end neural data-to-text generation with semantic fidelity. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2410–2424, Barcelona, Spain (Online), Dec. 2020. International Committee on Computational Linguistics.

[29] H. Harkous, S. T. Peddinti, R. Khandelwal, A. Srivastava, and N. Taft. Hark: A deep learning system for navigating privacy feedback at scale. In *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[30] M. Hils, D. W. Woods, and R. Böhme. Measuring the emergence of consent management on the web. In *Proceedings of the ACM Internet Measurement Conference*, pages 317–332, 2020.

[31] X. Hu, N. Sastry, and M. Mondal. Cccc: Corralling cookies into categories with cookiemonster. In *13th ACM Web Science Conference 2021*, pages 234–242, 2021.

[32] M. Kale and A. Rastogi. Text-to-text pre-training for data-to-text tasks. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 97–102, Dublin, Ireland, Dec. 2020. Association for Computational Linguistics.

[33] G. Kampanos and S. F. Shahandashti. Accept all: The landscape of cookie banners in greece and the uk. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 213–227. Springer, 2021.

[34] R. Khandelwal, T. Linden, H. Harkous, and K. Fawaz. Prisec: A privacy settings enforcement controller. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[35] O. Kulyk, A. Hilt, N. Gerber, and M. Volkamer. this website uses cookies": Users' perceptions and reactions to the cookie disclaimer. In *European Workshop on Usable Security (EuroUSEC)*, 2018.

[36] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, pages 1–15. Internet Society, 2019.

[37] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, Feb. 2019.

[38] S. Macbeth. Cliqz autoconsent. https://github.com/ghostery/autoconsent, 2020.

[39] D. Machuletz and R. Böhme. Multiple purposes, multiple problems: A user study of consent dialogs after gdpr. *Proceedings on Privacy Enhancing Technologies*, 2020(2):481–498, 2020.

[40] C. Matte, N. Bielova, and C. Santos. Do cookie banners respect my choice? : Measuring legal compliance of banners from iab europe's transparency and consent framework. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 791–809, 2020.

[41] B. Nissen, V. Neumann, M. Mikusz, R. Gianni, S. Clinch, C. Speed, and N. Davies. Should i agree? delegating consent decisions beyond the individual. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[42] M. Nouwens, I. Liccardi, M. Veale, D. Karger, and L. Kagal. 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*, pages 1–13, 2020.

[43] P. Papadopoulos, N. Kourtellis, and E. Markatos. Cookie synchronization: Everything you always wanted to know but were afraid to ask. In *The World Wide Web Conference*, pages 1432–1442, 2019.

[44] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[45] F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against {Third-Party} tracking on the web. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 155–168, 2012.

[46] Rolf, Bagge, Janus, Bager, and Kristensen. Consent-o-matic. https://github.com/cavi-au/Consent-O-Matic, 2020.

[47] I. Sanchez-Rola, M. Dell'Amico, P. Kotzias, D. Balzarotti, L. Bilge, P.-A. Vervier, and I. Santos. Can i opt out yet? gdpr and the global illusion of cookie control. In *Proceedings of the 2019 ACM Asia conference on computer and communications security*, pages 340–351, 2019.

[48] P. Shaw, M.-W. Chang, P. Pasupat, and K. Toutanova. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 922–938, Online, Aug. 2021. Association for Computational Linguistics.

[49] K. Solomos, P. Ilia, S. Ioannidis, and N. Kourtellis. Clash of the trackers: Measuring the evolution of the online tracking ecosystem. *arXiv preprint arXiv:1907.12860*, 2019.

[50] State of California. California Consumer Privacy Act (CCPA). https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375, June 2018. Assembly Bill No. 375.

[51] C. Sun, X. Qiu, Y. Xu, and X. Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[52] C. Utz, M. Degeling, S. Fahl, F. Schaub, and T. Holz. (un) informed consent: Studying gdpr consent notices in the field. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 973–990, 2019.

[53] C. Utz, M. Degeling, S. Fahl, F. Schaub, and T. Holz. (un)informed consent: Studying gdpr consent notices in the field. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 973–990, New York, NY, USA, 2019. Association for Computing Machinery.

[54] F. Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

A Appendix

A.1 Details on Execution Roles

The click command in Selenium emulates the click operation on a given element. However, the click action can only be performed if the element is visible on the browser (and is not overlayed by another element). For instance, if the first click on the *Save Settings* button removes the notice, a second click on the same element will result in an error as the element is no longer visible. Another example is when an element allows users to configure a choice. There, we should be able to click it multiple times to change the choice. We leverage these behaviors to identify the different execution roles for the elements.

As mentioned in Section 5.3, there are 4 types of execution role assigned to each element by the *Analyzer* module. They are as follows:

1. **Type A:** An element belongs to Type A if it is visible after two clicks and its state (selected or not-selected) changes with the clicks. For example, the switch element in Fig. 3 changes state and is visible after the clicks. Note that it is possible to implement Type A elements such that the state cannot be queried; however, empirically, we found that to be very rare.
2. **Type B:** Elements belonging to Type B reveal another cookie notice. Thus, to identify these elements, we check (1) if the element disappears after the clicks, and (2) the *Detector* module returns the new notice. For example, when we click the button "More information" in Fig. 3, the new notice (the right plot in Fig. 3) appears. Thus, we determine the execution role of the button to be Type B.
3. **Type C:** To identify Type C elements, we require that (1) the element can be clicked twice, and (2) that its *checked* attribute should not change with clicks. These elements are used for internal navigation within the notice.
4. **Type D:** Such elements result in closing the cookie notice. We identify these elements by requiring (1) failure in the second click, and (2) no new cookie notice appearing after first click.

A.2 Generation of JavaScript Code

Here we illustrate how *CookieEnforcer* disables non-essential cookies on `www.askubuntu.com` by executing JavaScript via the extension. First, the extension retrieves a set of instructions with `CSS selectors` for cookie notices. For each cookie setting present on the notice, the instructions also contain 1) paths for the setting element relative to the cookie notice, and 2) desired state of the setting element. The state and the corresponding actions are tracked using a variable `code`. The set of all codes are shown in Table 5. For example, `code` = 3 instructs the extension to click the setting element if it is visible. We also note that the paths are stored relative to the cookie notice as the absolute path (xpath) of the setting elements may change (ref. Section 5).

For the `http://askubuntu.com`, a partial set of instructions is shown in Fig. 8.

```
...
[
   code: 1  //Look for cookie notice,
   css_selector: "div[class*=\"z-nav-fixed js-consent-banner\"]",
]
[
    code: 3 //Click on element if visible,
   xpath: "div/button[2]"
],

[
    code: 5 //Click on element if it is selected,
   xpath: "div/div[1]/div[3]/div[2]/div[1]/input[1]"
],
...
```

Figure 8: A Partial instruction set for askubuntu website for illustration purposes.

The extension executes the instructions sequentially, starting by first detecting the cookie notice using the query selectors (lines 3-10 in Fig. 9). Next, after identifying the cookie notice, the *Enforcer* module finds the setting elements using the relative paths present in the instructions (line 13-15, Fig. 9). Finally, it checks the desired state of the element and performs actions as required (lines 17-18 and lines 23-24 in Fig. 9.

```
 ...
1 if (code == 1) { // code == 1 instructs to check for notice element
2    // ccs_selector extracted from the instructions
3    let css_selector = "div[class*=\"z-nav-fixed js-consent-banner\"]"
4    let cookieNotice; // cookie notice html element
5    // Check if the notice exists, if it does not, return
6    try {
7        cookieNotices = document.querySelectorAll(css_selector);
8    } catch (error) {
9            return;
10       }
11 }
12 else if (code === 3) { // code = 3 instructs to click if visible
13    let xpath = "div/button[2]" // relative path to cookie notice
14    // access the element using the relative path
15    let el = document.evaluate(xpath, cookieNotice)
16    // click if the element is visible
17    if (el.style.visibility !== "hidden")
18        el.click();
19 }...
20 else if (code === 5) { //code=5 instructs to click element if enabled
21    let xpath = "div/div[1]/div[3]/div[2]/div[1]/input[1]"
22    let el = document.evaluate(xpath, cookieNotice)
23    if (el.checked)
24        el.click(); // adjust the state if required
25 }...
```

Figure 9: *CookieEnforcer*'s JS generation code. Ellipses are used to skip non-important parts of the code.

Code	Purpose
1	Check if element is displayed and if not then wait for 2 sec at 40ms intervals
2	Check if element is displayed and if not raise error
3	Click on the element if it is displayed
4	Click on element unconditionally
5	Click on element if selected
6	Click on element if not selected
99	iFrame injection

Table 5: Table for codes used in the extension and their purpose.

A.3 Details of the User Study

A.3.1 User-Based Evaluations

Here we provide more details about the User Study we conducted to evaluate the usability of our extension. We first asked users to install our custom chrome browser extension which detected a subset of websites from Table 6 which had never been visited by the user. From that subset we choose 4 website, one for each of the 4 types of user interface as mentioned in 7.2 for the study. Then they were prompted to visit these website as shown in Fig. 10.

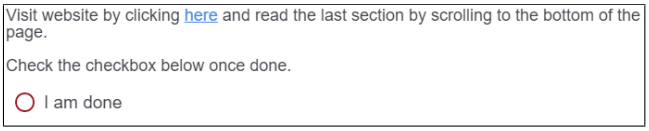


Figure 10: Prompting Users to visit the website.

A.3.2 Websites Used in the User Study

To choose websites for the study we looked at websites that used some of the most common CMPs like TrustArc and OneTrust as well as some that used custom cookie notices. We also selected some websites with a one click opt-out button and some that required multiple clicks. Table 6. shows the full list of websites we used for the user study.

A.3.3 Usability Evaluation

Participants were asked to fill out a standard System Usability Scale questionnaire [6] for each of the 4 interfaces they used.

The p-value for SUS score of all the combinations of interfaces are given in Table 7 below:

Interface Type	Manual	Informed	Semi	Auto
Manual	–	$3.5e-12$	$7.4e-21$	$1.2e-16$
Informed	$3.5e-12$	–	$1.78e-03$	$3.98e-03$
Semi	$7.4e-21$	$1.78e-03$	–	$7.66e-01$
Auto	$1.2e-16$	$3.98e-03$	$7.66e-01$	–

Table 7: SUS score p-values of all interfaces' combinations.

Website	CMP Type
seminolestate.edu	sscCookieStatement
statsperform.com	Optanon/OneTrust
gordonramsay.com	CybotCookiebot
crowe.com	Optanaon/OneTrust
financialsense.com	Custom
horiba.com	Custom
vogella.com	QuantCast
schroders.com	Custom
decathlon.co.uk	Didomi
justinbiebermusic.com	Evidon
hydroflask.com	TrustArc
piwik.pro	PIWIK pro
huntingpeaks.com	Iubenda

Table 6: List of Websites and their CMP types.

Since the p-value of the SUS Scores for user preference between manually setting cookies against that of using CookieEnforcer extension is < 0.05 we can reject the null hypothesis and say that these observations are statistically significant.

We also noted down the time taken by participants to use each of the interface. The p-value between the time taken for each of the interfaces is given in Table 8.

Interface Type	Manual	Informed	Semi	Auto
Manual	–	$4.34e-13$	$9.52e-21$	$2.12e-34$
Informed	$4.34e-13$	–	$7.09e-03$	$3.36e-31$
Semi	$9.52e-21$	$7.09e-03$	–	$9.74e-30$
Auto	$2.12e-34$	$3.36e-31$	$9.74e-30$	–

Table 8: p-values of time taken for all interfaces' combinations.

Since the p-value of the time taken to manually set cookies against the time taken by any of the CookieEnforcer interface is < 0.05 we can reject the null hypothesis and say that these observations are statistically significant.

A.4 Examples of Cookie Notices

In Figure 11, we show some examples of the cookie banners that are discussed in Section 6.

Website	Input	Output
www.amazonaws.com	button0 - customize cookie preferences || button1 - accept all cookies ** switch0 - allow performance category , selected || switch1 - allow functional category , not selected || switch2 - allow advertising category , not selected || button3 - cancel customizing cookie preferences || button4 - save customized cookie preferences <end>	Click button0 ** Click switch0 | Click button4.
reddit.com	button0 - reject non-essential || button1 - accept all <end>	Click button0.
wordpress.com	button0 - customize || button1 - accept all ** switch3 - analytics: these cookies allow us to optimize performance by collecting , selected || switch4 - advertising: these cookies are set by us and our advertising , not selected || button5 - accept selection <end>	Click button0 ** Click switch3 | Click button5.
www.facebook.com	button1 - more options || button2 - allow all cookies ** switch4 - cookies from other companies we use tools from other companies , not selected || button8 - allow only essential cookies || button9 - allow selected cookies <end>	Click button1 ** Click button8.
newscientist.com	button1 - i accept || button2 - show purposes ** button4 - select basic ads; object to legitimate interests ... switch23 - analytics cookies , not selected || button62 - confirm my choices	Click button2 ** ... Click button4 | Click button62.

Table 9: Examples demonstrating the application of *Decision model* on cookie notices for a few websites.

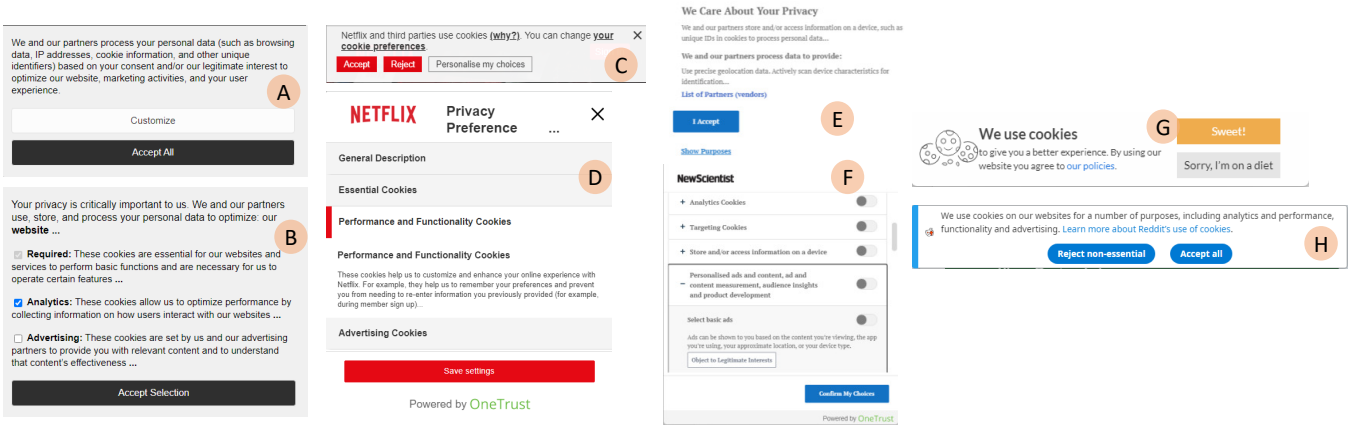


Figure 11: Screenshots of Cookie Notices for the websites listed in Table 9 where (A) is the first view for WordPress and (B) is the second view. (C) is the first view for Netflix and (D) is the second view. (E) is the first view of NewScientist and (F) is the second view. (G) is the first view of TATA and (H) is the first view of Reddit.