

Automatically Detecting Online Deceptive Patterns

Asmit Nayak, Shirley Zhang*, Yash Wani*, Rishabh Khandelwal, Kassem Fawaz

University of Wisconsin – Madison

Abstract

Deceptive patterns in digital interfaces manipulate users into making unintended decisions, exploiting cognitive biases and psychological vulnerabilities. These patterns have become ubiquitous on various digital platforms. While efforts to mitigate deceptive patterns have emerged from legal and technical perspectives, a significant gap remains in creating usable and scalable solutions. We introduce our *AutoBot* framework to address this gap and help web stakeholders navigate and mitigate online deceptive patterns. *AutoBot* accurately identifies and localizes deceptive patterns from a screenshot of a website without relying on the underlying HTML code. *AutoBot* employs a two-stage pipeline that leverages the capabilities of specialized vision models to analyze website screenshots, identify interactive elements, and extract textual features. Next, using a large language model, *AutoBot* understands the context surrounding these elements to determine the presence of deceptive patterns. We also use *AutoBot*, to create a synthetic dataset to distill knowledge from ‘teacher’ LLMs to smaller language models. Through extensive evaluation, we demonstrate *AutoBot*’s effectiveness in detecting deceptive patterns on the web, achieving an F1-score of 0.93 when detecting deceptive patterns, underscoring its potential as an essential tool for mitigating online deceptive patterns.

We implement *AutoBot*, across three downstream applications targeting different web stakeholders: (1) a local browser extension providing users with real-time feedback, (2) a Lighthouse audit to inform developers of potential deceptive patterns on their sites, and (3) as a measurement tool designed for researchers and regulators.

1 Introduction

Deceptive patterns, also known as ‘*Dark Patterns*,’ are design choices that manipulate users into making unintended decisions as they interact with applications. These patterns exploit cognitive biases and psychological vulnerabilities to influence user behavior, often in ways that benefit the service provider at the user’s expense [12, 13]. The growing use of deceptive patterns negatively impacts the quality of user experiences across various online activities, such as online purchases, engaging with social media, playing video games, or simply browsing the web [47]. The widespread nature of these patterns is well documented, with notable examples from resources like Harry Brignull’s `deceptive.design`¹ and Caroline Sliders’ analysis at `The Pudding`².

Despite the recent push toward better web experience, fueled by user awareness, media revelations, and privacy regulations [17, 2], deceptive patterns continue to pose substantial challenges [39]. As a result, users are at risk of harm, such as financial loss [61], privacy violations [8], and the exploitation of vulnerable populations, including children [69]. In response, researchers have explored different approaches to detect and classify deceptive patterns on the web. Earlier efforts included manual analysis to analyze the distribution of deceptive patterns on the Internet [47, 30, 12]. Such approaches are infeasible at scale

*Equal Contribution

¹<https://www.deceptive.design/>

²<https://pudding.cool/2023/05/dark-patterns/>



Figure 1: OpenAI’s GPT4.5 incorrectly identifies the color of the “Reject All” button as being less prominent than the other, leading to an incorrect classification of “nudge”. Similarly, Gemini 2.5 Pro incorrectly notes that the two buttons in the cookie notices are visually distinct from each other, resulting in a misclassification of “nudge”.

due to the sheer volume, dynamic nature, and diversity of website interfaces. More recent efforts include heuristic- and ML-based methods [12, 30, 47, 76, 68, 6, 18, 55, 46]. However, these approaches often exhibit limited accuracy when identifying deceptive patterns in the wild, as we show later.

Recognizing the limitations of current approaches, there is a pressing need for automated tools to assist web stakeholders in navigating and mitigating online deceptive patterns. Such an automated tool must perform two primary tasks: 1) accurately identify deceptive patterns and 2) precisely *localize* their position within the website. The ability to both identify and localize these patterns offers several benefits to web stakeholders. First, web users can be alerted to deceptive patterns on websites they visit, enabling informed decision-making. Second, regulators can leverage such tools to identify deceptive patterns at scale, facilitating enforcement and policy development. Third, developers can gain insights into potentially problematic elements within their websites, promoting more ethical design practices [70].

We propose an automated deceptive pattern detection framework, *AutoBot*, to address these limitations. *AutoBot* accurately identifies and localizes the deceptive patterns from a screenshot of a webpage. It does not rely on the underlying HTML code of the webpage, which tends to be less stable than screenshots. HTML implementation and code can vary significantly across webpages and even different accesses, while screenshots and text remain more consistent [35]. *AutoBot* adopts a modular design, breaking down the task into two distinct modules and leveraging existing state-of-the-art models for each. Specifically, *AutoBot* utilizes a *Vision Module*, which analyzes screenshots to accurately localize UI elements, extracting essential features into a structured, text-only format (*ElementMap*). It feeds this *ElementMap* to a *Language Module* that employs a Large Language Model (LLM) to analyze the *ElementMap* and assign a deceptive pattern based on a defined taxonomy (Section 2.1).

AutoBot’s design avoids the pitfalls of directly prompting Vision Large Language Model (VLLM) for

end-to-end analysis. These models, as we show in Figure 1, are known to hallucinate, leading to false positives undermining their reliability [64]. Furthermore, VLLMs currently lack the capability for accurate localization of UI elements, as shown in recent works [19, 82] and in Section 4.4.2. While fine-tuning VLLMs could potentially improve performance, the substantial demand for large annotated datasets and significant computing resources renders this approach impractical [9].

AutoBot leverages the capabilities of specialized vision models to help with the localization task and utilizes LLMs to perform accurate deceptive pattern identification. While LLMs have shown strong performance in detecting deceptive patterns (see Section 5.3), large-scale use of these LLMs might be prohibitive due to cost, latency, and privacy concerns. In this work, we show how we can create a synthetic dataset using an LLM as the ‘teacher’ and distill its knowledge into smaller language models (SLMs), like *Qwen2.5-1.5B*, and very small language models (vSLMs), like *Flan-T5*. We show a detailed evaluation of these models in Section 5.3.

We demonstrate the practical applications of *AutoBot* in three instantiations, targeting different web stakeholders. First, we design, and implement a browser extension (Section 7.1) using *AutoBot* to automatically detect and highlight deceptive patterns on websites, providing real-time user assistance on personal computers. Second, we create a custom Lighthouse audit (Section 7.2) that leverages *AutoBot* to inform developers of potential deceptive patterns on their sites, integrating directly into developer workflows and providing a quantifiable score. Third, we demonstrate how researchers and regulators can use *AutoBot* to perform a large-scale measurement and analysis of the online deceptive patterns landscape (Section 7.3). Our measurement on over 11,000 websites across popular (Tranco) and e-commerce (Shopify) domains highlighted the prevalence of deceptive patterns, with many websites exhibiting several patterns on a single webpage.

2 Background and Related Works

Web deceptive patterns refer to website interface design choices that manipulate or deceive users into making decisions they might not otherwise make [12]. Examples of such patterns include hidden costs, forced continuity, misdirection in e-commerce websites, and privacy-invasive default settings in social media platforms. Here, we present a filtered taxonomy to categorize web deceptive patterns based on existing work. We also survey existing works on detecting deceptive patterns on websites.

2.1 Taxonomy for Deceptive Patterns

Brignull et al. presented the first taxonomy of deceptive patterns in 2010 [12]. Conti et al. expanded this taxonomy to include ‘malicious interface designs’ [21]. Bösch et al. [11] introduced a similar taxonomy called ‘privacy dark patterns’, which included more privacy-centric categories such as ‘Forced Registration’ and ‘Hidden Legalese Stipulations’.

More recently, Gray et al. [30] created a unified corpus to detect deceptive designs in user interfaces, building on previous taxonomies and categories. Since then, various works have further adapted the taxonomy for specific domains. For instance, Lewis et al. [40] codified deceptive patterns for mobile apps and games, while Mathur et al. [47] adapted the taxonomy to focus on deceptive patterns present in shopping websites. Work by Chen et al. [18] and Mansur et al. [46] extended the taxonomy to detect deceptive patterns in mobile and web apps. Although prior works developed various taxonomies to classify deceptive patterns, these efforts are inconsistent and often domain-specific. To address these issues, Gray et al. [31] introduced a unified ontology of deceptive patterns integrating past literature across regulatory reports and academic works.

2.1.1 Filtered Taxonomy

Since our work, like others in literature, focuses on detecting patterns at the page level, some patterns fall outside its scope. For instance, the deceptive pattern ‘*Sneak into Basket*’ – where websites quietly add

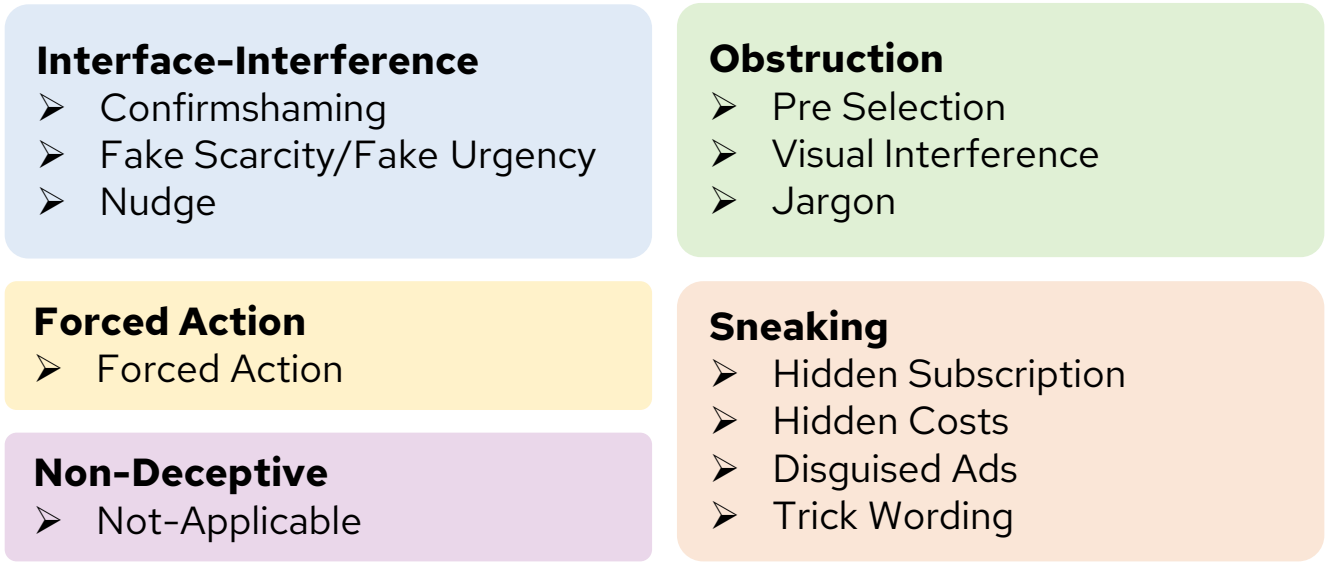


Figure 2: Taxonomy of Deceptive Patterns. *AutoBot* classifies text elements into five high-level deceptive pattern categories: Interface Interference, Obstruction, Forced Action, Sneaking, and Non-Deceptive.

unwanted items like magazine subscriptions to a user’s cart – creates an unwanted transaction that would require the detection system to understand user action across multiple pages over a period of time.

We classify deceptive patterns into two types – (1) static patterns with no temporal dimension and are visible upon page render, and (2) dynamic patterns that rely on user interactions or time-based triggers related (e.g., ‘Nagging,’ ‘Hard-to-Cancel,’ and ‘Bait-and-Switch’). Since our system relies on screenshots, in this work, we limit our analysis to static deceptive patterns, as defined in Figure 2. The detection of dynamic patterns requires temporal analysis of the webpages, which is out of scope for this work. Our system presents a building block for future systems to detect dynamic patterns by analyzing webpage code, temporal activities, and performing actions.

We filter Gray et al.’s ontology [31] focusing only on static deceptive patterns. Figure 2 shows the filtered taxonomy, comprising four high-level categories and 11 low-level sub-types. The complete taxonomy is provided in Section 11.1. This taxonomy includes several high-level categories from Gray et al. [31]. It includes the sub-types from the Brignull et al. taxonomy [12], which map to low-level patterns in Gray et al.’s ontology. These sub-types include textual descriptions of the pattern, which we use to prompt the LLMs, as shown later. We map our taxonomy to Gray et al.’s ontology [31] in Section 11.3. Finally, the taxonomy excludes a subset of the deceptive patterns that cannot be detected through screenshots, i.e., the dynamic ones.

2.2 Detecting Deceptive Patterns

Researchers developed several mechanisms to detect and measure the prevalence of online deceptive patterns [12, 30, 47, 76, 68, 6, 18, 55, 46]. Curley et al. categorized earlier manual, automated, or semi-automated detection mechanisms [24].

2.2.1 Human-based Manual Annotations

Early efforts to identify online deceptive patterns relied on manual exploration. Brignull et al. [12] manually explored the web to compile a “Hall of Shame”: a list of websites with deceptive patterns. Gray et al. [30] expanded this corpus by performing keyword searches on social media for posts highlighting websites with deceptive patterns, which were then manually validated. While highly accurate, this methodology is limited to domain experts and lacks scalability due to manual validation requirements. To expedite the manual exploration process, Mathur et al. [47] proposed a clustering-based pipeline to group similar websites based

on their text content, which is then manually inspected. Although this process accelerates data collection, it still lacks scalability.

2.2.2 Probabilistic Text & Image Models

Attempts to automate the manual inspection process include Tung et al.’s *Naive Bayes* classifier [76], Soe et al.’s gradient-boosted tree to flag texts showcasing deceptive patterns [68], and Adorna et al.’s combined *Naive Bayes* classifier and VGG-19 model to identify deceptive designs in cookie notices [6]. However, these works are often domain-specific and cannot readily generalize to new domains. Additionally, most works rely solely on text-based classifiers or heuristics, limiting their ability to detect visual deceptive patterns, such as those based on colors or trick wording.

2.2.3 Heuristic Based Methods

Recent works by Chen et al., Mansur et al., and Raju et al. [18, 46, 55] focus on automated detection of deceptive patterns in mobile apps. Chen et al. [18] and Mansur et al. [46] used predefined heuristics, limiting detection to simpler deceptive patterns like disguised ads. Prior works used the HTML code of websites to identify and detect deceptive patterns. For example, Raju et al. [55] employed rule-based source code analysis to detect patterns like ads, forced action, and nagging [55]. However, due to the versatile and open nature of HTML, this task proves to be very challenging. While HTML implementation and code vary significantly across webpages, and even across different accesses of the same page, screenshots and text remain more consistent. As a result, recent approaches have moved completely towards text or screenshots of websites to detect and identify deceptive designs.

2.2.4 Classical ML Models

In the broader domain of Privacy, Safety, and Security (PSS), researchers have developed tools to help users navigate specific deceptive designs on websites. For instance, works by Khandelwal et al. [35, 36] enable users to find and adjust privacy settings and automatically disable non-essential cookies. Similarly, OptOutEasy by Kumar et al. [10] automatically finds opt-out links from privacy policies and surfaces them to users. These domain-specific approaches do not readily apply to deceptive pattern detection as they require retraining for each deceptive pattern.

2.2.5 Large Language Models (VLLMs)

Prior work have explored using LLMs to detect deceptive patterns. Sazid et al. [59] used GPT-3.5-Turbo to detect deceptive text with 92.57% accuracy, although their method is limited to singular text lines, only 7 types of patterns, and ignores any surrounding visual or textual context. Similarly, Schäfer et al. [60] utilized GPT-4o to remove deceptive elements from synthetic HTML, reducing manipulateness in 91% of cases. This approach, however, is constrained due to the versatile and non-standardized nature of HTML [71] and the challenge of fitting inflated, real-world website source code into an LLM’s context window.

More recently, a concurrent work with ours, Shi et al. [64], introduces *DPGuard* to automatically detect deceptive patterns from screenshots of mobile apps and websites by directly prompting VLLMs. We show later in Section 3 that VLLMs perform poorly in detecting deceptive patterns from screenshots and are often prone to hallucinations and false positives. Additionally, *DPGuard* only detects the presence of these deceptive patterns without any positional reference. Our work identifies deceptive patterns with significantly higher accuracy and extracts their positions, enabling us to show users exactly where these patterns occur.

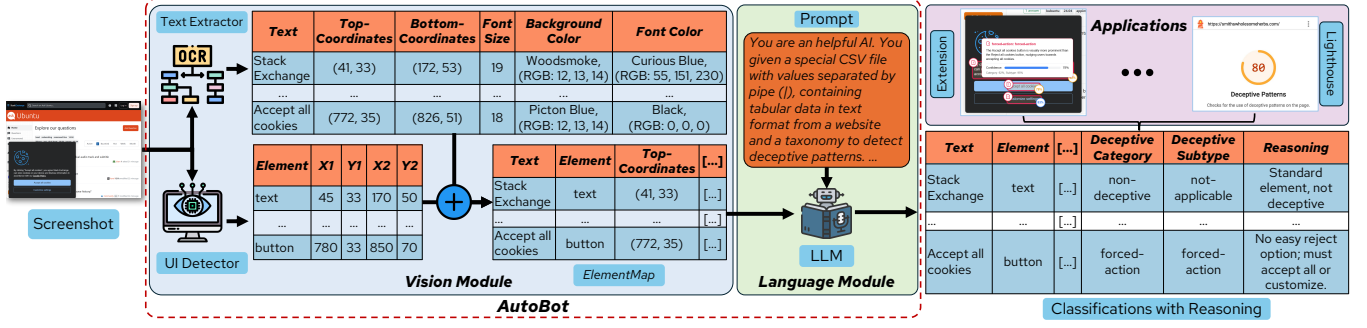


Figure 3: Overview of *AutoBot*’s working process. *AutoBot* takes a screenshot through a multi-stage framework consisting of the Vision Module and the Language Module, and returns the Deceptive Pattern classification, Subtype, and reasoning for each element in the screenshot. The results are then used by applications such as browser extensions and Lighthouse.

3 System Overview

This work presents *AutoBot*, an end-to-end framework that identifies the deceptive patterns and extracts their positions on a webpage. After receiving a screenshot of a webpage, it identifies UI elements on the page and feeds the identified elements along with the associated text to an LLM. The output is a mapping of each element to a corresponding deceptive pattern from Section 2.1.

Why Screenshots? Prior works analyzing websites [35, 36] have primarily relied on HTML analysis. This approach faces significant challenges due to the dynamic nature of websites. Websites increasingly employ *JavaScript* frameworks that modify the Document Object Model (DOM) on the fly, rendering static HTML analysis insufficient. Furthermore, the diversity in the coding practices and obfuscation techniques provide additional challenges in HTML-based analyses. To address these challenges, *AutoBot* adopts a novel approach focusing on the invariant aspect of websites: *the user experience*. By leveraging the visual signals and associated text, *AutoBot* models how users perceive and interact with websites. This approach offers several advantages: (1) It is resilient to change in underlying technologies as it captures the actual rendered content. (2) It allows us to analyze the same information that the user encounters, providing a more accurate representation of the potential deceptive patterns.

Vision Large Language Models (VLLMs) Recent advances in VLLMs offer a venue for analyzing screenshots and highlighting patterns with proper prompting. To this end, we empirically evaluate the effectiveness of GPT4.5 [5] and Gemini 2.5 Pro [72] in detecting web deceptive patterns. Our experiments showed that while these models can detect deceptive patterns, but they often hallucinate and give false positive answers. As shown in Figure 1, both Gemini 2.5 Pro and GPT-4.5 struggle to identify the deceptive patterns in screenshots. A recent work in this domain by Shi et al. [64], *DPGuard*, uses these models directly to detect, not localize, deceptive patterns. Consistent with Shi et al.’s evaluation, we find that *DPGuard* faces performance issues, struggling to generalize over a variety of websites (refer to Section 6), achieving only a macro score of 0.3452 in detecting deceptive patterns on websites.

In addition, VLLMs frequently struggle with the precise location of elements in an input image [19, 82, 41], and cannot be used to localize deceptive patterns out of the box. However, when given bounding box coordinates, these models can effectively reason about the spatial arrangements of objects [65].

Modules *AutoBot* leverages the above insights to automatically detect and localize deceptive patterns on websites, as shown in Figure 3. It breaks the localization problem and deceptive pattern detection into two tasks: vision and language. This breakdown allows *AutoBot* to independently leverage vision techniques for the precise localization of elements and powerful language models for the accurate classification of patterns.

1. **Vision Module:** The *Vision Module* maps a screenshot of a webpage to a table of elements as shown in Figure 3. We refer to this tabular representation as *ElementMap*. The *ElementMap* contains the text associated with the element along with its other features: element type, bounding box coordinates, font size, background color, and font color. This module addresses the issues of high false positive rates and localization by parsing the screenshot of a webpage.
2. **Language Module:** The *Language Module* (Section 5) takes the *ElementMap* as input and maps each element to a deceptive pattern from the taxonomy in Section 2.1. This module reasons about each element considering its spatial context and visual features. We explore different instantiations of this module with different trade-offs in terms of cost, need for training, and accuracy.

4 Vision Module

The *Vision Module* generates an *ElementMap* from a screenshot of the website through the following three steps, as shown in Figure 3.

1. **Text Extraction:** Extract the text, the bounding boxes of the text, and the associated features from the screenshot.
2. **Web-UI Element Detection:** Localize UI elements, extract their bounding boxes, and identify their types from the screenshot.
3. ***ElementMap* Generation:** Merge the results of the above steps into a tabular representation of the website to generate an *ElementMap*.

4.1 Text Extraction

The Text Extraction step starts by performing Optical Character Recognition (OCR) on the website screenshot. We employ the Google Vision OCR API ³ as it has high accuracy in extracting text from images of varying scales and resolutions. The API returns blocks of detected text. We concatenate these blocks based on proximity to form coherent text blocks. These blocks are a list of bounding boxes with their respective text content. Then, we retrieve the font size, font color, and background color of each bounding box, as illustrated in Figure 3. We calculate font size by subtracting the bottom y-coordinate from the top y-coordinate of the bounding box. To determine color information, we utilize the *extcolors* [16] package, extracting the most prominent color (background) and the second most prominent color (font).

This approach addresses a key limitation identified by Soe et al. [68] in the previous ML methods: the lack of representation of the UI richness that a user perceives, such as text placement and contrast between the font and background colors. Our approach captures these detailed text features, along with the relative location of text elements on the website, providing a comprehensive representation of the textual content experienced by users.

4.2 Web-UI Element Detection

The Web-UI Element Detection step uses the same screenshot to identify and localize these 7 *Web-UI Elements*: buttons, checkboxes (☑, ■), radio buttons (●, ○), and toggle switches (🔘, 🔘). This step provides context to the extracted text and enables a more comprehensive understanding of the webpage’s structure. For instance, distinguishing between a clickable button and a static text block can be significant, as a seemingly simple text might be a deceptive call to action when recognized as a button. Similarly, identified checkbox states (checked or unchecked) can reveal pre-selected options that users might overlook.

³<https://cloud.google.com/vision/docs/ocr>

In the following, we describe how we survey existing Web-UI detection methods and the underlying datasets. As we find these methods and datasets to not be appropriate for the deceptive patterns detection task, we describe how we train a real-time *Web-UI Element Detector* using YOLOv10 on a custom dataset.

Limitations of Existing Web-UI Detection Methods While Web-UI Element Detection is a well-studied problem, the existing approaches face several limitations in our deceptive pattern detection task. Detectors like *Ominparser 2* [44], *Ferret UI 2* [42], *UEID* [80], and *Element Detector* [79] do not distinguish between checked and unchecked states. Detectors like *UISketch* [62] report low performance on real-world websites. *ScreenRecognition* [84] shows promise for this step but was trained on mobile screenshots and is closed-source, preventing its use or testing on websites.

Limitations of Existing Web-UI Datasets To improve the detector performance, we explored modifying existing datasets used to train them. *Omniparser 2* uses a manually annotated dataset of popular websites, but the dataset is not public [44]. *UEID* [80] was trained on the *RICO Dataset* [43], which contains manually annotated mobile app UIs. We could not modify this dataset to train a detector for webpages. *Ferret UI 2* [42] and *Element Detector* [79] use *WebUI* [79], a popular framework with 400k websites which is publicly available, but the automatically computed labels from accessibility trees⁴ are noisy and miss a lot of elements on a given website [79, 78]. *UISketch* contains hand-drawn images to identify *Web-UI Elements* based on their sketches and does not generalize well to real-world websites [62].

4.2.1 Dataset Curation

We develop a novel approach to create a diverse and representative dataset to address the lack of suitable *Web-UI Element Dataset* for training our *Web-UI Element Detector*, as shown in Figure 4. Instead of relying on manual scraping and labeling [84, 62, 81, 44], we leverage AI-powered tools to generate a synthetic dataset that reflects the current web landscape. Our approach allows for precise control over element positioning and labeling, which allows us to scale the dataset generation.

We generated 2.5K ideas for diverse websites using GPT-4 [5]. We passed these ideas to v0⁵ (refer to Section 11.4 for examples), an AI-based website generator, to generate three websites per idea. v0 uses *shadcn* [63], a customizable UI-Library, which allows us to get bounding boxes and the state (checked/unchecked) of every UI element. Three members of the research team manually verified the 7.5K websites generated by v0. With a failure rate of less than 2%, this process was relatively fast, as errors in websites are detected by the compiler. These errors consisted mainly of typos and missing libraries that the researchers manually fixed. We then rendered each v0 generated website (7.5K) using randomized components from 6 popular UI-libraries, like Material UI and Bootstrap⁶. This process resulted in 62K screenshots, where each screenshot had a bounding box and a label for each UI element. Recall that we have 7 labels for the UI elements. We refer to this dataset as the *Web-UI Elements Dataset*.

4.2.2 Training Web-UI Element Detector

We use the *Web-UI Elements Dataset* to train an ensemble of YOLOv10 models. In particular, we randomly divide the dataset into a training set consisting of 85% of the images and a validation set consisting of the remaining 15%. We present the performance of the trained ensemble in Section 4.4.2 on real-world websites.

Why YOLOv10? We adopt the YOLOv10 model (You Only Look Once (YOLO) architecture [75], a real-time object detector for recognizing UI elements from a screenshot. We chose YOLOv10 over Convolution Neural Networks (CNNs) [35, 46, 18] and VLLMs like Molmo [25]. CNN-based detectors, such as

⁴Accessibility trees are generated using aria labels which developers have to optionally add to a website.

⁵<https://v0.dev/>

⁶<https://mui.com/material-ui/>, <https://getbootstrap.com/>

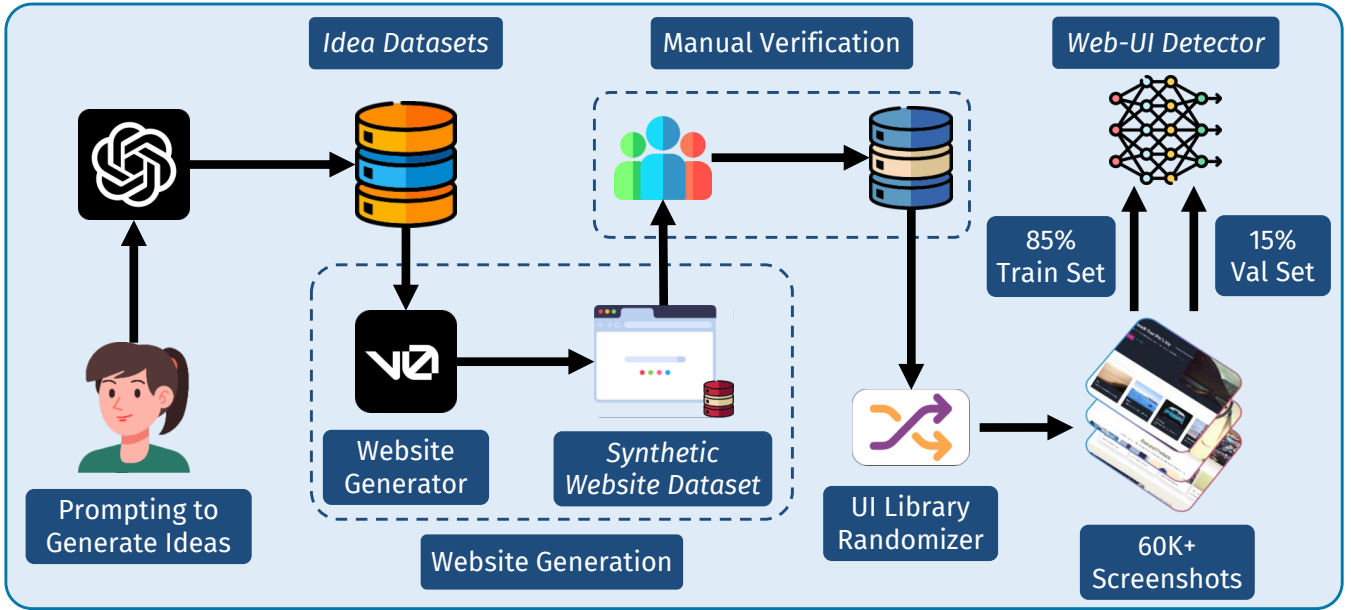


Figure 4: Pipeline of Generating Web-UI Element Dataset to train YOLOv10. We used GPT-4 to generate 2.5K ideas (*Idea Datasets*), which were then processed by v0 to create 7.5K websites (*Synthetic Website Dataset*). After manually verifying these sites for rendering errors and randomizing their UI library, we capture over 60K screenshots to train our YOLOv10 model.

Faster R-CNN [57], provide predictions with high accuracy, but require considerable computational power and time [56, 57]. VLLMs, despite their strong capabilities in understanding image context, demonstrate significant limitations in image classification tasks [85], specifically object detection tasks [83]. Evaluating Molmo [25] on detecting *Web-UI Elements* yielded poor results, as shown in Table 1. YOLO models are comparatively lightweight, around 40MB, allowing for various deployment options without requiring extensive compute resources.

Ensemble of YOLOv10 During training, we observed that YOLOv10 models could not distinguish between the 7 labels (first column of Table 1) accurately. We attribute the reason to the labels being visually similar, such as a checked switch and a checked radio button. As such, we trained three YOLOv10 models, each focused on distinguishing between 2-3 different elements. The first model labeled button and ; the second labeled , , and ; and the third labeled and . We found that each model performed much better than one trying to handle all the classes at once. The same observation has been made in literature before for YOLO-based object detection [48, 74, 53]. We combined the outputs of the three models by simply performing a union over the detected UI elements. In case of overlap, we took the label with the higher confidence classification. We refer to this detection method as the YOLOv10 **Ensemble**.

4.3 *ElementMap* Generation

The *ElementMap* Generation step merges the *Web-UI Elements* from the *Web-UI Element Detector* with the text blocks from the *Text Extraction* step. In particular, it iterates over each detected *Web-UI Element* and applies spatial heuristics depending on the element type to find the most likely text block corresponding to the element. For example, buttons are matched based on overlap, while checkboxes and radio buttons are paired with nearby text. The closest matching text block is then relabeled with the element type. This labeling results in an *ElementMap*, where each row contains an element label, the text, the bounding box coordinates, the font size, the background color, and the font color. The *Language Module* uses the *ElementMap* to detect the deceptive patterns on a page.

4.4 Vision Module Evaluation

We create a dataset to evaluate the real-world performance of the YOLOv10 ensemble.

4.4.1 Vision Dataset

We curate a labeled dataset of UI elements from the deceptive pattern websites dataset of Mathur et al. [47]. We manually annotated over 1.5K website screenshots using Label Studio [73]. In particular, one author manually annotated each screenshot by drawing bounding boxes around each UI element and assigning it a type. The type is one the 7 *Web-UI Elements*: buttons, checkboxes (☒, ☐), radio buttons (☒, ☐), and toggle switches (☒, ☐). Another author independently verified the annotations. Both authors then discussed and resolved the conflicts in annotations. We refer to this dataset as the *Real-UI Dataset*.

4.4.2 Vision Module Evaluation

We measure the accuracy of the YOLOv10 ensemble of models on the *Real-UI Dataset*. We report our results using the IoU metric, which measures the overlap between the predicted bounding box and the ground truth box by dividing the area of their intersection by the area of their union. A higher IoU indicates better localization accuracy, and we consider a detection to be correct if the IoU exceeds 0.5 and the model confidence exceeds 0.3. We choose a lower IoU threshold to account for the distribution shift between training on a synthetically labeled dataset and a human-annotated one. The bounding boxes from both will be different. Using a high IoU threshold would result in more false negatives, which would affect the subsequent steps in *AutoBot*’s pipeline.

Table 1: Performance of the *YOLOv10 Ensemble* and *Molmo* [25] on our *Real-UI Dataset*

Class	Precision		Recall		F1-Score		# Elements
	YOLO	Molmo	YOLO	Molmo	YOLO	Molmo	
button	0.94	0.87	0.88	0.41	0.91	0.55	5226
<input checked="" type="checkbox"/>	0.85	0.97	0.95	0.47	0.89	0.63	113
<input type="checkbox"/>	0.98	0.92	0.76	0.44	0.86	0.60	246
<input checked="" type="radio"/>	0.85	0.86	0.93	0.29	0.89	0.43	76
<input type="radio"/>	0.86	0.84	0.89	0.35	0.87	0.49	132
<input checked="" type="checkbox"/>	0.96	0.96	0.98	0.42	0.97	0.59	52
<input type="checkbox"/>	0.91	0.93	0.94	0.47	0.93	0.63	34
Total	0.91	0.90	0.91	0.42	0.91	0.57	5879

Table 1 shows the F1-scores of our YOLOv10 Ensemble and Molmo [25] for each of the 7 classes. We observe that the ensemble outperforms Molmo for all the UI elements. Note that few-shot prompting of local VLLMs is not possible for image inputs. The ensemble exhibits a relatively lower performance on the unchecked radio button and unchecked check box, because they look visually similar to the other UI elements.

5 Language Module

The *Language Module* assigns a deceptive pattern (from the taxonomy in Section 2.1) to each element present in an *ElementMap*. This is the input/output structure of the language module as shown in Figure 5.

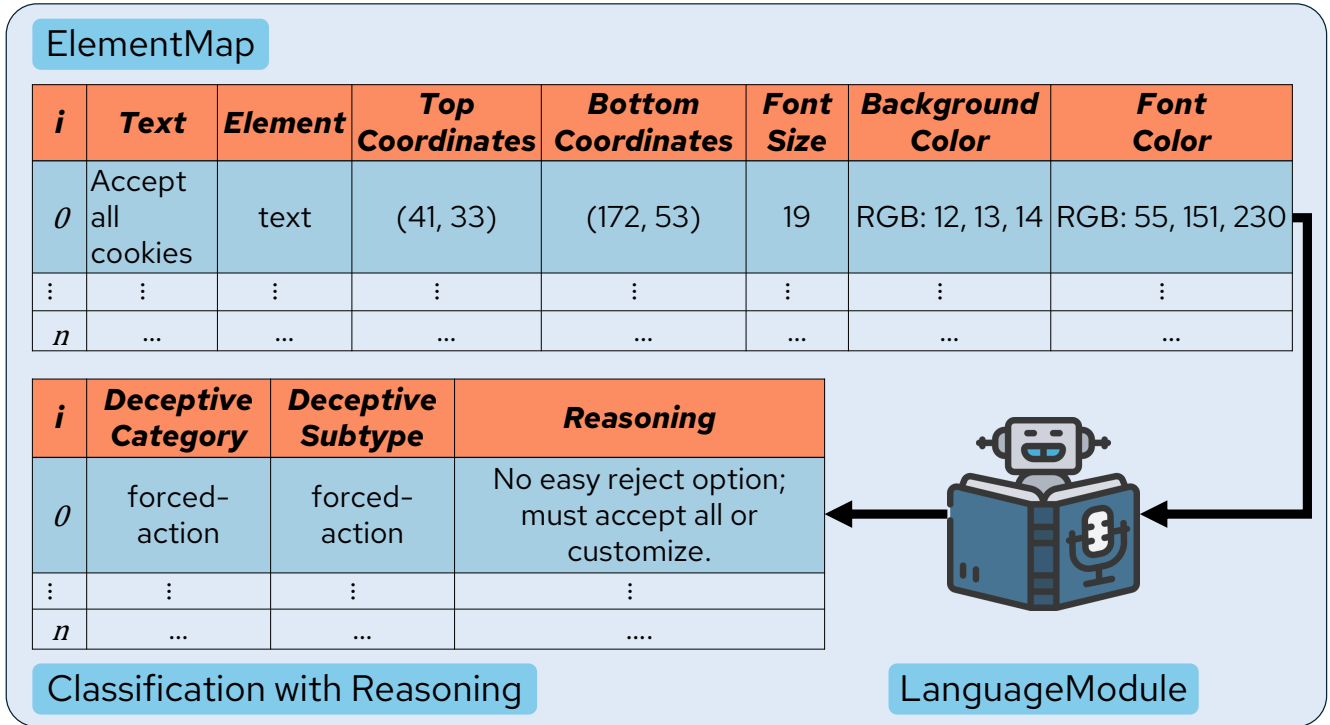


Figure 5: The input and output structure of our language module. The input ElementMap consists of key features of a web element, and the output contains a deceptive category, subtype, and reasoning of classification.

5.1 Possible Solutions

Prior works have shown that LLMs [49] are suitable for reasoning tasks similar to our task. These models fall into three categories: 1) large Language Models (LMs) like Gemini and GPT-4, 2) small LMs such as Gemma and Qwen, and 3) very small LMs like T5. These models have varying capabilities and trade-offs, as described in Table 2.

Table 2: Comparison of Language Models: Gemini vs Qwen2.5 vs T5

Feature	Large LM	Small LM	Very Small LM
Size (parameters)	Large (>200B)	Medium (1.5B)	Very Small (700M)
Context Window	1M	128K	<1K
Deployment	Cloud API only	Can be run locally	Can be run locally
Required Memory	N/A	~4.5 GB	~700MB
License	Proprietary	Open Source	Open Source
Latency	Higher	Medium	Very low
Cost	High	Free	Free
Data Privacy	Data leaves device	Data stays on device	Data stays on device

5.1.1 Large Language Models (LLMs):

LLMs are very effective at performing a wide range of tasks [23, 7, 66] and are able to closely follow user instructions [52, 86]. However, as shown in Table 2, using these LLMs is cost-prohibitive and has potential data privacy concerns [3].

5.1.2 Small Language Models (SLMs):

Unlike LLMs that follow complex instructions when performing a task, SLMs often struggle to do so. This limitation of SLMs can be overcome by fine-tuning them on specific tasks, improving their performance to match that of LLMs [45]. Moreover, as shown in Table 2, SLMs have two key advantages over LLMs: 1) being compute efficient, they can be deployed locally across a wide range of platforms, and 2) they alleviate any data privacy concern associated with API based LLMs.

5.1.3 Very Small Language Models (vSLMs):

SLMs, while compute-efficient, are not the best solution to use in an extremely resource-constrained platform, such as in-browser or on devices with no specialized GPU. In such environments, we can leverage vSLMs like **Flan-T5** [20]. Models like **Flan-T5** need to be finetuned or distilled from LLMs for specific tasks to achieve high accuracy [33]. We show the detailed distillation steps we performed in Section 5.2.4.

In summary, we observe that large LMs, such as **Gemini**, are considerably effective in detecting deceptive patterns from an *ElementMap*. However, as described in Table 2, utilizing such large and closed-source models presents challenges like high usage cost, considerable latency, and potential data-privacy concerns as the *ElementMap* is sent to an external service. Smaller LMs such as **Qwen** and **T5** address these challenges and have been shown to perform well on such specific tasks after finetuning [45, 33].

5.2 Our Solution

To combine the strengths of large and small LMs, we adopt a distillation approach where we use large LMs as teachers and small LMs as students to complete our task. Specifically, we create a synthetic dataset of deceptive pattern classification from *ElementMap* using **Gemini**. We then use this dataset to distill smaller student models, i.e., **Qwen** and **T5**. As such, *AutoBot* comprises three language models to detect deceptive patterns, each presenting different trade-offs as described in Table 2.

5.2.1 Prompting the LLM

We utilize proven techniques like Chain-of-Thought (CoT) [77], few-shot prompting [15], and prompting the model to reason about its classification [51, 32, 49] to help the model understand the task and identify deceptive patterns. Specifically, our system prompt, P_{system} ⁷, provides the LLM with a plan on how to detect deceptive patterns and instructs the LLM to generate three columns — *Deceptive Category*, *Deceptive Subtype*, and *Reasoning* — for each element in the *ElementMap*, as shown in Figure 5. This prompt allows us to not only generate precise labels and the associated reasoning but also minimize hallucinations. An added benefit of generating the classification with ‘Reasoning’ is that we can use these ‘rationales’ to further train smaller LMs. This *Distilling Step-by-Step* methodology has been shown to be very effective by Hsieh et al. [33].

During our initial evaluation of **Gemini 1.5 Pro**, using P_{system} , we observed that the model could identify all deceptive patterns in the *ElementMap*, but would often mis-classify ‘non-deceptive’ patterns as deceptive, resulting in a high false positive rate and a low precision score. To address this problem, we utilized the **Gemini 2.0-Flash-Thinking** reasoning model. Reasoning models have shown promising results in being able to reason about a task [45, 51]. We prompted the LLM to re-evaluate the elements identified as deceptive and correct misclassifications⁷. Next, we utilized the **Gemini 2.0-Flash-Thinking** model to re-verify the labels on every website with one or more elements classified as deceptive. This additional step in generating the labels, significantly reduced the number of false positives we observed. We note that we do not use this model as our base model because of its limited availability, making it infeasible to be used at scale. We show that this approach results in high precision and recall in Section 5.3.

⁷Please find the system prompt under *Files* here: https://osf.io/tha2d/?view_only=4fd2116fa2e94c99857679eddfe5937

5.2.2 Creating Distillation Dataset

We create $\mathcal{D}_{\text{distill}}$ by scraping and analyzing 11K websites from the Tranco list [54]. We filter out non-English websites using the `langdetect` library [67] and adult websites using the `NudeNet` package [50], leaving us with 6,626 websites. For these websites, a significant portion was non-deceptive or had very few deceptive patterns. To increase websites with deceptive patterns, we opted to look at e-commerce websites, as these websites tend to have deceptive patterns [47]. As such, we analyze an additional 4,492 featured websites from *Shopify Partners Directory*⁸.

Overall, we have 11,118 websites in our dataset. We run *AutoBot* on these websites, using the **Gemini 1.5 Pro** and **Gemini 2.0 Flash-Thinking** models. To reduce randomness and have deterministic outputs, we limit the *temperature* = 0 and *top_p* = 0.1 [58]. We incorporate the final classification and reasoning produced by *AutoBot* in $\mathcal{D}_{\text{distill}}$. The distribution of samples (a sample is defined as a single row of the *ElementMap*) is shown in Table 3.

Table 3: Distribution of Samples in $\mathcal{D}_{\text{distill}}$.

Category	Subtype	# Samples
Non Deceptive	<i>Not Applicable</i>	160934
Forced Action	<i>Forced Action</i>	3403
Interface Interference	<i>Nudge</i>	1335
	<i>Fake Scarcity / Fake Urgency</i>	933
	<i>Confirmshaming</i>	428
Obstruction	<i>Visual Interference</i>	689
	<i>Pre-Selection</i>	234
Sneaking	<i>Trick Wording</i>	904
	<i>Hidden Costs</i>	233
	<i>Hidden Subscription</i>	3495
	<i>Disguised Ads</i>	5980

5.2.3 Distilling Small Language Models (Qwen2.5-1.5B)

DeepScaleR [45] has shown that small language models (SLMs) finetuned for specific tasks are able to mirror the performance of larger models on those same tasks. Using this insight, we distill a **Qwen2.5-1.5B** model to be able to mimic **Gemini**’s performance at detecting deceptive patterns. We use the $\mathcal{D}_{\text{distill}}$ dataset to perform full-finetuning of the **Qwen2.5-1.5B** model. As shown in Table 3, the distribution of samples in the distillation dataset is highly imbalanced, with over 92% of the samples being ‘non-deceptive’. Training on such an imbalanced dataset may introduce bias towards the majority class [38]. To mitigate such bias, we perform Random Under Sampling of the ‘non-deceptive’ class, which has shown to improve performance [26], to achieve a more balanced distribution of about 55% ‘non-deceptive’ samples. Our distilled version of **Qwen2.5-1.5B** was trained on 34.7M tokens on 4 Nvidia-A6000 GPUs for 2 epochs. We present the performance of the trained SLM in Section 5.3.

⁸<https://www.shopify.com/partners/directory>

5.2.4 Distilling Very Small Language Models (T5):

To distill the knowledge from LLMs into very small language models like T5, we use and improve the paradigm introduced by Hsieh et al. [33]. We split $\mathcal{D}_{\text{distill}}$ into 90% training and a 10% validation set grouped by the sites from which the samples were extracted. This ensures that samples from the same site are not present in training and testing sets. We formally define the training dataset, D_{train} , as:

$$d_i \in D_{\text{train}} \subset \mathcal{D}_{\text{distill}} \quad (1)$$

$$d_i = (x_i, y_i, z_i, r_i) \quad (2)$$

where D_{train} is a subset of the balanced $\mathcal{D}_{\text{distill}}$ from Section 5.2.3. Here, x_i represents the input to classify, y_i represents the deceptive design category, z_i represents the deceptive design subtype, and r_i represents the associated reasoning for the category and subtype. Since, the context window of T5 is significantly smaller than that of SLMs and LLMs, for each web element that is to be classified, x_i , we provide its neighboring web elements, in a sliding window fashion: $x_{i-1 \rightarrow i-n}$ to $x_{i+1 \rightarrow i+n}$. For the distillation task, we used $n = 4$.

Baseline Approach Based on this initial methodology, we train a T5 model, f , on a multi-task problem: predict the deceptive design category, subtype as the label, and reasoning as the rationale. We use the task-specific prefix `[classify]` to generate the label and `[reason]` for the reasoning.

We define the model and loss functions as:

$$f(x, t) = \begin{cases} (\hat{y}_i \oplus \hat{z}_i), & \text{if } t = [\text{category}] \\ \hat{r}_i, & \text{if } t = [\text{reason}] \end{cases} \quad (3)$$

$$\mathcal{L} = \mathcal{L}_{\text{label}} + \alpha \mathcal{L}_{\text{reason}} \quad (4)$$

Here, $\mathcal{L}_{\text{label}}$ and $\mathcal{L}_{\text{reason}}$ are the label prediction loss and reason generation loss, respectively, and are defined as:

$$\mathcal{L}_{\text{label}} = \frac{1}{N} \sum_{i=1}^N \ell(f(x_i, t_{\text{category}}), y_i \oplus z_i) \quad (5)$$

$$\mathcal{L}_{\text{reason}} = \frac{1}{N} \sum_{i=1}^N \ell(f(x_i, t_{\text{reason}}), r_i), \quad (6)$$

where ℓ is the cross entropy loss between the predicted and target tokens, and \oplus is a string concatenation function.

Each web element that is to be classified, x_i , is provided alongside its neighboring web elements, $x_{i-1 \rightarrow 0}$ to $x_{i+1 \rightarrow N}$. The model’s performance is measured as the exact match of its `[category]` outputs with the ground truth data. An example of the model’s sample input and expected output is shown below.

Input Sample

Input: `[category]:` Line 14,Preferences,checked checkbox,...</s>Line 10,"MAGIC We use cookies to personalise ...</s>Line 11,COMING SOON ,text,...</s>

Output: *obstruction,pre-selection*

Input: `[reason]:` Line 14,Preferences,checked checkbox,...</s>Line 10,"MAGIC We use cookies to personalise ...</s>Line 11,COMING SOON ,text,...</s>

Output: *Cookie banner option is pre-selected to indicate users to allow extra cookies.*

After training T5 on the $\mathcal{D}_{\text{distill}}$ dataset for 2 epochs, we observed the training accuracy to saturate to $\sim 48\%$. Here, accuracy refers to the correct prediction of both category and sub-types.

Our Approach: To overcome the low accuracy in the baseline approach, we split the labeling task into two separate tasks: category and subtype, introducing an additional “task prefix” [subtype] for the new task. We redefine the model and loss function to:

$$f(x, t) = \begin{cases} \hat{y}_i, & \text{if } t = [\text{category}] \\ \hat{z}_i, & \text{if } t = [\text{subtype}] \\ \hat{r}_i, & \text{if } t = [\text{reason}] \end{cases} \quad (7)$$

$$\mathcal{L} = \alpha(\mathcal{L}_{\text{category}} + \mathcal{L}_{\text{subtype}}) + (1 - \alpha)\mathcal{L}_{\text{reason}}, \quad (8)$$

where α is a tuning factor.

By separating label into category and subtype in addition to the reason tasks, the model can learn the relation between the category and the subtype and how they both relate to the reasoning. A sample of the new inputs and expected outputs is shown below.

Input Sample	
Input: [category]: Line 14,Preferences,checked check-box,...</s>Line 10,"MAGIC We use cookies to personalise ...</s>Line 11,COMING SOON ,text,...</s>	
Output: <i>obstruction</i>	
Input: [subtype]: Line 14,Preferences,checked checkbox,...</s>Line 10,"MAGIC We use cookies to personalise ...</s>Line 11,COMING SOON ,text,...</s>	
Output: <i>pre-selection</i>	
Input: [reason]: Line 14,Preferences,checked checkbox,...</s>Line 10,"MAGIC We use cookies to personalise ...</s>Line 11,COMING SOON ,text,...</s>	
Output: <i>Cookie banner option is pre-selected to indicate users to allow extra cookies.</i>	

After training the T5 model on the new loss function and tasks using the same ground truth dataset and the same number of epochs, we observe the test accuracy of detecting deceptive patterns saturate to $\sim 95\%$. We present these results in the following section.

5.3 Language Module Evaluation

We create a dataset to evaluate the real-world performance of the different models in the language module.

5.3.1 Language Dataset

To evaluate the *Language Module* of our pipeline, we curate a dataset, referred to as *LangEval* dataset. In particular, we randomly choose 200 websites from the *D3 Dataset* (in particular the Mathur et al. portion) described in Section 6.1. Next, for these 200 websites, one of the authors manually annotated the UI element classifications in the *ElementMap* to provide ground truth UI labels. Thus, the *LangEval* dataset contains the manually labeled *ElementMap* of each website associated with the manually labeled deceptive patterns.

5.3.2 Evaluating Language Models

We evaluate the various language models discussed in this section on the *LangEval* dataset. The performance is shown in Table 4. We report the performance of the language models in two ways:

1. *Binary Classification*: This metric assesses models’ ability to detect whether a UI element *is deceptive*, regardless of the specific categorization. We use this metric as it provides a performance measure while considering the inherent subjectivity of deceptive pattern classification – an element classified as ‘trick-wording’ could also be classified as ‘hidden-cost’. This classification is provided at the bottom of each evaluation table, with labels: “Deceptive” and “Non-Deceptive”.
2. *Class-wise Classification*: This is the *detailed classification* result, across categories and subtypes, between the ground truth data and the model-generated result.

Table 4: Performance of different models in *LanguageModule* on *LangEval*. The table has three sections, each showing performance at the category, subtype, and binary levels

Pattern Type		Precision			Recall			F1-Score		
		Gemini	Qwen	T5	Gemini	Qwen	T5	Gemini	Qwen	T5
Category	Non Deceptive	1.00	0.98	1.00	0.99	0.99	0.93	0.99	0.98	0.96
	Forced Action	0.89	0.85	0.84	0.79	0.83	0.86	0.84	0.84	0.85
	Interface Interference	0.85	0.79	0.51	0.85	0.55	0.69	0.85	0.65	0.59
	Obstruction	0.53	0.49	0.17	0.91	0.95	1.00	0.67	0.64	0.29
	Sneaking	0.87	0.73	0.50	0.96	0.71	0.84	0.91	0.72	0.63
	Not Applicable	1.00	0.98	0.99	0.99	0.99	0.89	0.99	0.98	0.94
Subtype	Confirmshaming	0.80	0.90	0.86	0.80	0.82	1.00	0.80	0.86	0.92
	Disguised Ads	0.76	0.58	0.33	0.96	0.58	0.73	0.85	0.58	0.46
	Fake Scarcity / Fake Urgency	0.96	0.91	0.61	0.93	0.63	0.76	0.95	0.75	0.68
	Forced Action	0.89	0.85	0.78	0.79	0.83	0.88	0.84	0.84	0.82
	Hidden Costs	0.60	-	0.05	0.60	-	0.20	0.60	-	0.08
	Hidden Subscription	0.90	0.78	0.56	0.95	0.72	0.77	0.92	0.75	0.64
	Nudge	0.74	0.57	0.17	0.80	0.37	0.68	0.77	0.45	0.28
	Pre-Selection	0.67	0.65	0.33	1.00	1.00	1.00	0.80	0.79	0.50
	Trick Wording	0.86	0.61	0.34	0.80	0.61	0.47	0.83	0.61	0.39
	Visual Interference	0.50	0.36	0.06	0.83	0.80	1.00	0.62	0.50	0.11
	Deceptive	0.89	0.84	0.65	0.95	0.80	0.95	0.92	0.82	0.77
	Not Deceptive	1.00	0.98	0.99	0.99	0.99	0.95	0.99	0.98	0.97

These results highlight the trade-offs between the different language models. As expected, Gemini exhibits the highest performance, reaching near perfect precision and recall on most deceptive patterns. It struggles in two pattern subtypes: hidden-costs and visual interference. In second place is the Qwen model, which struggles in more subtypes. The distilled T5 model exhibits generally acceptable performance, but struggles for pattern categories: interface-interference and obstruction.

6 End-to-End Evaluation

We perform an end-to-end evaluation of *AutoBot* on a real-world dataset, consisting of 1.1K websites, manually annotated by 2 authors (described in Section 6.1). We perform our evaluation with all three LLMs in the *LanguageModule* (see Section 5.1). For this End-to-End Evaluation, our objective is to measure the accuracy of our entire framework (including the vision and language models) in detecting deceptive patterns on websites.

6.1 Dataset

To create the dataset for the end-to-end evaluation, we crawl the websites in the deceptive pattern dataset from Mathur et al. [47]. We use this dataset since it is the latest and most comprehensive dataset of websites with deceptive patterns. We filter out the non-English and offline websites from this dataset, leaving us with 555 websites out of the 1400 sites mentioned. Next, we crawled the top 1000 websites from the Tranco [54] list⁹, utilizing the same methodology in Section 5.2.2 to filter out non-English and NFSW websites, resulting in 597 websites. Additionally, for the Tranco websites, we did not scrape just the landing pages of websites. Rather, we queried “site:domain” on DuckDuckGo and programmatically counted the number of interactable elements for each of the top ten resulting pages. The page with the highest count was selected for analysis. In total, we have 1152 websites in our end-to-end evaluation dataset.

Next, we manually labeled the screenshots of these websites to identify and localize the deceptive patterns, associating each screenshot with an *ElementMap*. Specifically, two authors annotated the same randomly selected 100 websites with high inter-annotator agreement ($\kappa = 0.89$) [37]. Then, each researcher separately labeled the rest of the screenshots. We refer to this dataset as the golden Deceptive Design Dataset (*D3 dataset*). We show the distribution of the samples in *D3 Dataset* in Table 5.

Table 5: Distribution of Samples in *D3 Dataset*.

Category	Subtype	# Samples
Non Deceptive	<i>Not Applicable</i>	22618
Forced Action	<i>Forced Action</i>	414
Interface Interference	<i>Nudge</i>	143
	<i>Fake Scarcity / Fake Urgency</i>	211
	<i>Confirmshaming</i>	42
Obstruction	<i>Visual Interference</i>	48
	<i>Pre-Selection</i>	18
Sneaking	<i>Trick Wording</i>	190
	<i>Hidden Costs</i>	28
	<i>Hidden Subscription</i>	379
	<i>Disguised Ads</i>	306

6.2 Findings

We analyze the websites in the *D3 Dataset* using *AutoBot* and compare the predicted deceptive patterns to the ground truth annotations.

The results from the evaluation are shown in Table 6. We observe that the performance of the *AutoBot* framework is mainly dependent on the type of language model used. We note here that **Gemini** is the teacher model in our framework, and **Qwen** and **T5** are the student models, as described in Section 5.1. As such, we see that Gemini has the best performance, followed by **Qwen** and **T5**.

We also evaluate Shi et al.’s [64] *DPGuard* framework. To evaluate their framework, we first create a mapping between their taxonomy and the filtered taxonomy (see Section 11.7). For our evaluation, we only consider the categories mappable to the filtered taxonomy. Additionally, as the *DPGuard* framework does not provide localization capabilities, our evaluation is solely focused on *DPGuard*’s ability to identify deceptive patterns present within the webpage in our *D3 Dataset*. Our evaluation shows that *DPGuard* is unable to identify deceptive patterns with high accuracy, especially struggling to classify ‘*hidden-subscription*’ and ‘*trick-wording*’. These findings are consistent with the evaluation performed by Shi et al. [64].

⁹<https://tranco-list.eu/list/QGJ74/1000000>

Table 6: Performance of *AutoBot* (with three underlying language models: Gemini, Qwen, and T5) and *DP-Guard* [64] on the *D3 Dataset*. The table has three sections, each showing performance at the category, subtype, and binary levels.

	Pattern Type	Precision				Recall				F1-Score			
		Gemini	Qwen	T5	<i>DPGuard</i> [64]	Gemini	Qwen	T5	<i>DPGuard</i> [64]	Gemini	Qwen	T5	<i>DPGuard</i> [64]
Category	Non Deceptive	1.00	0.98	1.00	–	0.99	0.99	0.96	–	0.99	0.99	0.98	–
	Forced Action	0.97	0.89	0.82	–	0.94	0.85	0.83	–	0.95	0.87	0.82	–
	Interface Interference	0.89	0.76	0.55	–	0.95	0.64	0.78	–	0.92	0.69	0.64	–
	Obstruction	0.70	0.57	0.13	–	0.97	0.71	0.93	–	0.81	0.63	0.23	–
	Sneaking	0.87	0.79	0.52	–	0.94	0.73	0.85	–	0.90	0.76	0.64	–
Subtype	<i>Not Applicable</i>	1.00	0.98	1.00	0.78	0.99	0.99	0.91	0.74	0.99	0.99	0.95	0.76
	<i>Confirmshaming</i>	0.93	0.75	0.59	0.05	1.00	0.69	0.85	0.32	0.97	0.72	0.70	0.09
	<i>Disguised Ads</i>	0.74	0.62	0.34	0.49	0.95	0.61	0.81	0.62	0.83	0.61	0.48	0.55
	<i>Fake Scarcity / Fake Urgency</i>	0.94	0.87	0.68	–	0.96	0.71	0.89	–	0.95	0.78	0.77	–
	<i>Forced Action</i>	0.97	0.89	0.72	0.40	0.94	0.85	0.84	0.51	0.95	0.87	0.77	0.45
	<i>Hidden Costs</i>	0.77	0.31	0.14	–	0.91	0.28	0.38	–	0.83	0.29	0.21	–
	<i>Hidden Subscription</i>	0.93	0.84	0.51	–	0.96	0.72	0.83	–	0.95	0.78	0.63	–
	<i>Nudge</i>	0.79	0.52	0.10	0.23	0.89	0.43	0.48	0.58	0.83	0.47	0.17	0.33
	<i>Pre-Selection</i>	0.64	0.64	0.25	0.02	0.94	0.94	1.00	0.14	0.76	0.76	0.40	0.03
	<i>Trick Wording</i>	0.85	0.74	0.57	0.00	0.82	0.69	0.67	0.00	0.83	0.71	0.62	0.00
	<i>Visual Interference</i>	0.73	0.54	0.03	0.09	0.98	0.62	0.93	0.22	0.84	0.58	0.05	0.13
	Deceptive	0.90	0.88	0.72	–	0.97	0.81	0.95	–	0.93	0.84	0.82	–
	Not Deceptive	1.00	0.98	0.97	–	0.99	0.99	0.97	–	0.99	0.99	0.98	–

–: the DP classification is not supported by the model.

6.2.1 Error Analysis

In the end-to-end evaluation, we observe that, overall, **Gemini** performs extremely well in identifying deceptive patterns. However, for certain subtypes, it has comparatively lower performance. For instance, **Gemini** has reduced performance in identifying ‘*pre-selection*’. Similarly, for ‘*disguised-ads*’, it has a slightly higher false positive rate. Further analysis shows that these false positives are mainly due to product placement: typically benign content resembling deceptive advertising. For instance, a website showcasing template-based shopping apps may include screenshots of user-created apps, some of which contain promotional text. Although harmless, this text is often misclassified as ‘*disguised-ads*’ due to its advertising-like appearance. Furthermore, through empirical analysis, we have observed **Gemini** interchangeably using ‘*nudge*’ and ‘*forced-action*’, especially when classifying cookie notices.

For the smaller models, **Qwen** and **T5**, we observe that the *AutoBot* sometimes fails to identify deceptive pattern categories/subtypes correctly. Investigating the error cases further, we find instances where *AutoBot* misclassified the category or sub-category of the deceptive pattern (while correctly identifying that the pattern is deceptive). For example, there are instances where *AutoBot* incorrectly identifies *forced-action* as *nudge*. We also find that on Wikipedia, *AutoBot* incorrectly tags a non-deceptive pattern as deceptive. The classification text contains money or cost-related text, causing the model to classify the text as *trick-wording* incorrectly.

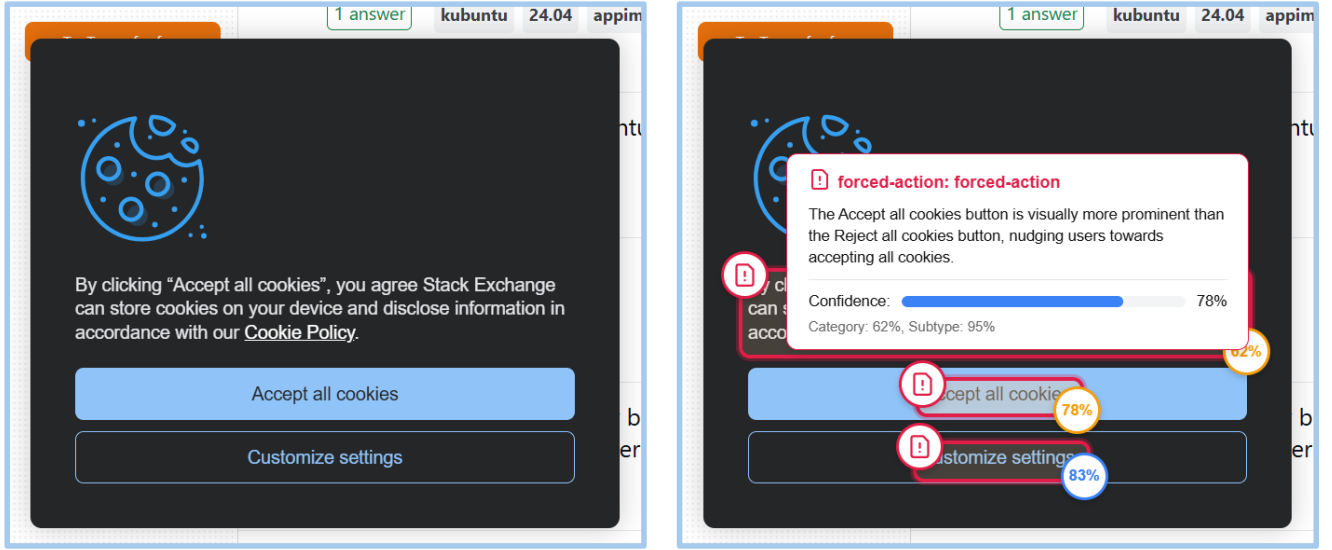
6.2.2 Impact of Errors

We note that the impact of the misclassifications due to category or sub-category mismatch is minimal on the users as the users will still be notified of a deceptive pattern. For false positives, the user gets notified for a pattern where none exists. Upon further inspection, users can safely ignore the notification, causing minimal distraction. For false negatives - the user impact can be severe. In such cases, users might get into a sense of false security and get deceived by the deceptive pattern because of *AutoBot*’s error. We emphasize that high recall of *AutoBot* ensures that such cases will be minimal.

7 Applications

We instantiate the *AutoBot* framework across three potential downstream tasks, each designed to serve a stakeholder in the web ecosystem: users, developers, and regulators and researchers.

7.1 Browser Extension for Web Users



(a) Screenshot of a website.

(b) Screenshot of *AutoBot* running on browser and detecting deceptive patterns.

Figure 6: Screenshot of website (left) and *AutoBot* running (right).

Our first instantiation is a browser extension that directly helps web users, the primary target of deceptive patterns. The extension takes a screenshot of the user’s active page and analyzes it using the *AutoBot* framework. The active page may contain sensitive information of the user. To mitigate data privacy concerns, the extension performs the analysis locally using a distilled version of *Flan-T5* Section 5.2.4. Once processed, the extension highlights deceptive patterns to the users as shown in Figure 6. Specifically, it shows bounding boxes around the UI elements where deceptive patterns are found, and informs the users as they hover over elements.

Extension Architecture The browser extension utilizes a hybrid architecture consisting of lightweight browser components with a locally running Flask server to run the *AutoBot* framework, similar to the *Zotero* extension [22]. On the browser side, we use a popup script to allow users to activate the extension. Once active, the extension takes a screenshot and sends it to the local Flask server for analysis. The Flask server hosts the *AutoBot* framework, using *Flan-T5* as the language model. After analysis, the server returns the classifications to the extension, which are then rendered by the content script of the extension on the user’s screen. This rendering is shown in Figure 6.

System Level Performance For the browser extension to be practical, it must perform its analysis in real time. We, therefore, conducted latency tests on its core *Flan-T5* (Language) and *YOLOv10 Ensemble* (Vision) models using three different machine configurations representing various hardware capabilities (modern high-end with GPU, older mid-range, ARM-based) and comparing CPU versus GPU performance. Our results (detailed in Section 11.5) show that while performance on older hardware using only the CPU takes roughly **1.5 to 3 seconds** per module (e.g., **1.95 seconds** for *YOLOv10 Ensemble* on a CPU-only i5 laptop), newer devices with GPUs achieve near real-time performance. Specifically, on the 2023 laptop with a GPU, *T5* inference (with 800 tokens as input) took less than **0.5 seconds**, and *YOLOv10 Ensemble* performed its classifications in less than **0.3 seconds**. Our experiments show the feasibility of implementing *AutoBot* as a browser extension, as modern hardware with GPU support enables a responsive experience without major performance delays [34, 4].

7.2 Lighthouse Reports for Web Developers

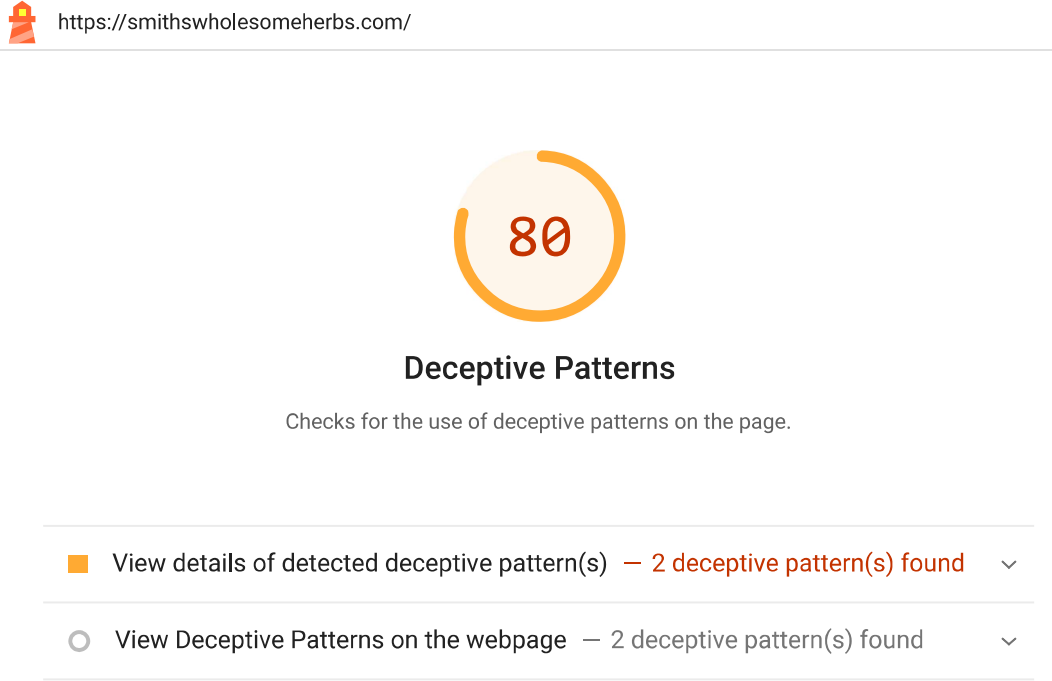


Figure 7: Custom Audit on a Lighthouse Report.

Research shows that while developers are often unaware of deceptive patterns on their websites and their impacts on users, they are open to addressing these patterns if properly informed [70]. To that end, we integrate *AutoBot* into Lighthouse [1, 29], a popular tool developers use to receive automated audits on website quality. Google bundles this tool with ChromeDev Tools, and major platforms such as Shopify, Wix, and Squarespace¹⁰ integrate it into their workflows. We created a custom Lighthouse audit using *AutoBot* to fit directly into a web developer’s workflow. This audit uses an existing **Lighthouse Gatherer** [27] to capture screenshots of the website, processes it using *AutoBot*, and finally incorporates the findings into a Lighthouse report, as shown in Figure 7.

We package Lighthouse with our custom audit in a docker container for better usability. The docker container maintains all functionality of Lighthouse while adding our custom audit. The container simply takes an URL as input to run the entire audit and give the developer a report. The report provides the developer with *DeceptivePattern Score* based on the number of deceptive patterns (n) found on the page using the following scoring scheme:

$$S(n) = \begin{cases} 100, & \text{if } n = 0 \\ 89, & \text{if } n = 1 \\ \max(100 - 10n, 0), & \text{if } n > 1 \end{cases}$$

The scoring scheme scales inversely with the number of deceptive patterns. It assigns a score of 0.89 for a single deceptive pattern to ensure that the audit shows a failure condition. An example of the Lighthouse Report as a developer would see is shown in Figure 7.

7.3 Enabling Web-Scale Analysis

Our last instantiation of the *AutoBot* framework is a tool designed for scalable website analysis. This tool serves researchers and regulators, providing them with the necessary automated capabilities to investigate

¹⁰<https://www.shopify.com/>, <https://www.wix.com/>, <https://www.squarespace.com/>

the broader landscape of online deceptive practices. It takes input from a list of URLs. Then, it crawls each URL, takes a screenshot, extracts the *ElementMap* from each screenshot, and passes the *ElementMap* to the Language Module. We use the Gemini model with batch API in this tool to generate accurate analysis at a low cost.

Measurement on Shopify and Tranco Websites We use this tool to analyze 11,118 websites, consisting of 6,626 diverse, popular domains selected from the Tranco list [54], and 4,492 e-commerce sites from the *Shopify Partners Directory*¹¹. We detail our website selection process earlier in Section 5.2.2.

The complete distribution of the identified deceptive patterns across the two domains is in Figure 8. We observe that *sneaking* is the most prevalent deceptive pattern across both domains. On Shopify websites, the second prevalent is *interface-interference*. For example, e-commerce websites tend to use fake scarcity or urgency (e.g., “Limited time offer: get 20% off for next 5 min”), which is classified as *fake-scarcity-fake-urgency*. On Tranco websites, *forced-action* is the next most prevalent pattern after *sneaking*. Our analysis also reveals that many websites employ multiple distinct categories of deceptive patterns simultaneously. For example, *bedbathandbeyond.com* (Section 11.2) consists of ‘*forced-action*’ (by providing no clear option to reject cookies) alongside ‘*obstruction*’ (by presenting mailing list terms and conditions in an excessively small font).

8 User Studies

We conducted two exploratory user studies to evaluate the usability of two *AutoBot* applications: browser extension and lighthouse report. In our first study, we evaluate the usability of a website with highlighted deceptive patterns. Next, we reached out to Shopify developers from the Shopify developer list (see Section 7.3), with the Lighthouse report of their website.

8.1 Website Usability

We perform an exploratory user-based evaluation to explore how highlighting deceptive patterns affects the usability of a website. We recruited 151 U.S.-based participants from Prolific, compensating them \$2 for a task with a median completion time of 7 minutes. We did not collect demographic data and asked Prolific to distribute the survey evenly across the demographics. The IRB at our institution determined that the proposed activity is not research involving human subjects as defined by DHHS and FDA regulations.

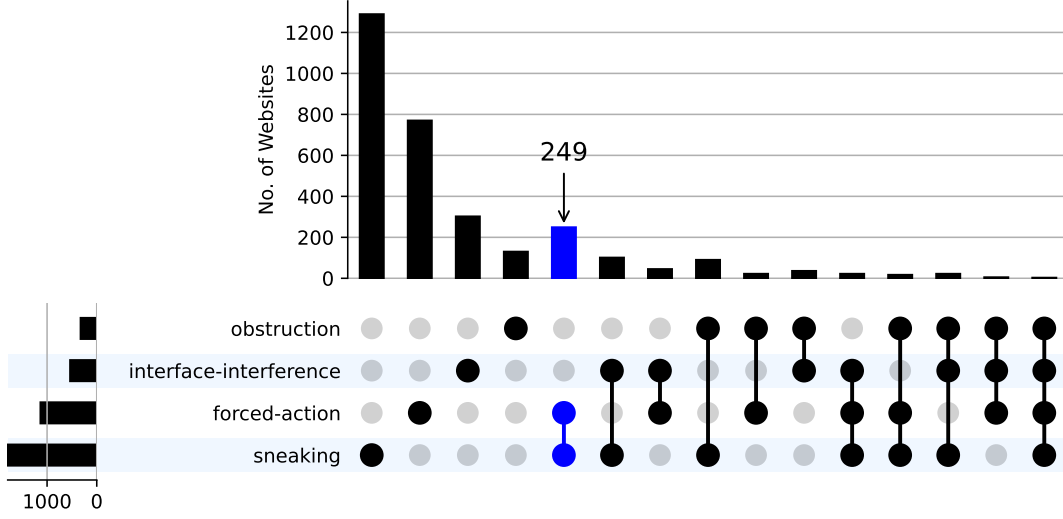
8.1.1 Study Design

We conducted a within-subject study to assess the usability impact of highlighting deceptive patterns with *AutoBot*. We informed the participants that the objective of the survey was to test the usability of a web interface. Next, participants were asked to visit two custom-made websites, one with and one without highlighting, and fill out a System Usability Scale questionnaire [14] after each. In both these websites, the participants were asked to perform one of the four tasks: sign up, download, do shopping, or read a news article. Once the participants visit a website and hover over the highlighted text, a banner cautioning them about the deceptive pattern is shown. Note that the participants interacted with highlighted websites directly without installing a browser extension.

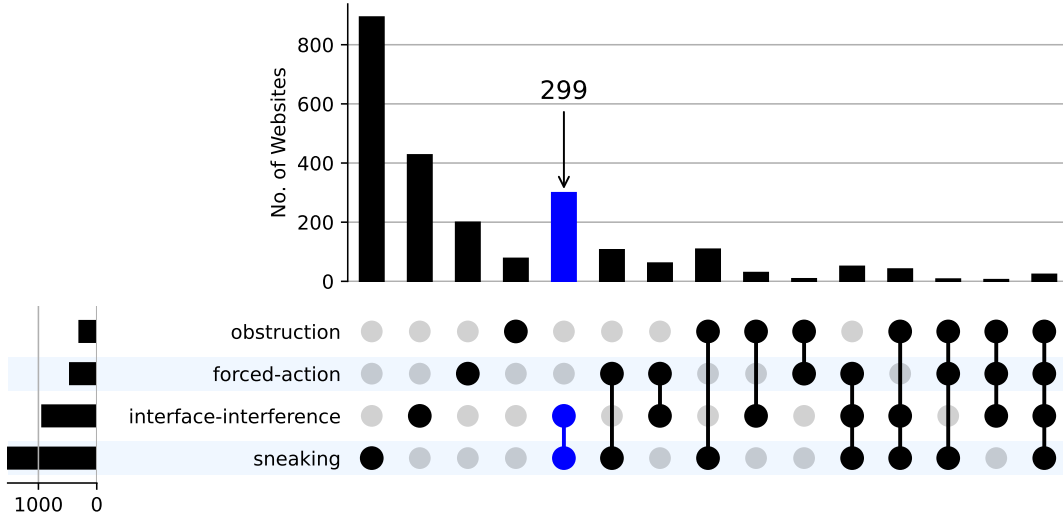
These custom-made websites were created by the authors based on various real websites, available at *UXP² Dark Patterns*¹², caught using deceptive patterns. An example of such a website is shown in Section 11.8. After visiting each website, the participant is asked to complete a System Usability Scale

¹¹<https://www.shopify.com/partners/directory>

¹²<https://darkpatterns.uxp2.com/>



(a) Distribution of Deceptive Patterns identified by *AutoBot* on Tranco websites



(b) Distribution of Deceptive Patterns identified by *AutoBot* on Shopify Websites

Figure 8: Distribution of deceptive patterns across various domains such as (a) Most visited Tranco websites and (b) Shopify based e-commerce websites

questionnaire [14]. Additionally, we asked the participants to fill out a post-study questionnaire consisting of three questions:

- Q1. *Did you find the hints about the deceptive patterns useful?*
- Q2. *Did you feel that the highlighted box encouraged you to notice the deceptive patterns?*
- Q3. *Did you feel that the highlighted box encouraged you to change your choice?*

8.1.2 Findings

We evaluated the website's usability with and without highlights using the SUS metric, , as shown in Figure 13. In this case, we consider the non-hypothesis to be that the website's usability is unaffected after highlighting. Based on the Wilcoxon signed-rank statistical test, the p -value for comparing two groups is 0.106, indicating no statistically significant difference between the SUS scores for websites. Therefore, the null hypothesis stands that usability is unaffected. We do note, however, that the mean SUS score for the highlighted website was 2 points higher.

Furthermore, based on participant feedback to the three questions mentioned above (Figure 14), we find that most found the highlight box (65.5%) and its corresponding hint (56.3%) helpful for recognizing deceptive patterns. Additionally, 38% of the participants reported they would change their behavior after being shown the highlighted patterns.

8.1.3 Future Studies

Reflecting on our preliminary user study results suggests that the design of highlighting deceptive patterns on webpages does not affect website usability, and can help users recognize deceptive patterns. Follow-up studies should investigate the usability and utility of real-time interventions that detect and present deceptive patterns on the web. For example, such studies can evaluate different realizations of *AutoBot*'s extension that involve different trade-offs between real-time performance, usability, and utility. In such studies, participants can be prompted to install the extension, visit real-world websites, and answer surveys based on their experience.

8.2 Developer Outreach

To evaluate the effectiveness of the Lighthouse reports, we notified the Shopify web developers of the 4,492 website we analyzed in Section 7.3 about various deceptive patterns present on their websites.

8.2.1 Outreach Design

We crawled the *Shopify Partners Directory* in search of developers and their most popular Shopify-based website. Next, we analyzed these websites for deceptive patterns as detailed earlier in Section 7.3. Through that process we created Lighthouse reports for each of the website. Next, we reached out to the web developers of the website through email, inquiring about their opinion of the Lighthouse report. Particularly, we asked them 4 questions:

- Q1. *Was this report is useful to you?*
- Q2. *Would you like more or less information included in the report, and if so, what?*
- Q3. *In light of this report, would you be willing to make any changes?*
- Q4. *Would you like to get access to this tool to run on your complete website?*

8.2.2 Ethical Consideration

In our email to the developers, we clearly identified ourselves as developers and stated our purpose of creating a system to automatically create Lighthouse reports that identify deceptive patterns on the web, and that we wanted to understand their perspective on these reports. We also stated that no personally identifiable data was being collected and that their responses to our email were optional. Since, no PII data was collected for our study, the IRB at our institution certified our study as "not human subject research," and we were not required to obtain consent before sending out the emails.

8.2.3 Findings

Unfortunately, the study yielded only three usable replies. Two responses indicated that removing DPs was an owner-level decision for which they lacked authority, while the third cited the desire to maintain customer retention as the reason for not making changes.

8.2.4 Future Studies

The limited response rate suggests that a different approach is needed to interact with developers. While prior work had success with email-based notifications for web developers [70], those emails included language about vulnerabilities and legal liabilities. We opted not to use such language with the developers,

which contributed to a reduced response rate. We suggest that future studies actively recruit developers and interview them about the Lighthouse reports. Another follow-up study can potentially help regulators generate variants of Lighthouse reports that check compliance with relevant laws, such as the French Data Protection Act.

9 Limitations

Scope of Detectable Patterns A limitation for *AutoBot* comes from the static UI analysis. Our approach inherently restricts the scope of detectable deceptive patterns to those visually present on a single page at a given time. Deceptive patterns such as *nagging* cannot be detected through this approach, and thus, we filter Gary et. al.’s taxonomy [31] to focus on static deceptive patterns.

Hardware Constraints Another practical limitation arises due to hardware limitations associated with deploying *AutoBot*’s extension using T5. The model requires approximately 1GB of memory, which might not be easily available on older devices. Furthermore, our experiments show that while near-real-time latency is achievable on modern hardware, the lack of GPU acceleration can significantly affect latency.

Multilingual Support *AutoBot*’s focus on English websites presents another limitation as the language module does not support non-English languages due to presence of language-specific datasets in the distillation pipeline. However, we note that it is possible to extend *AutoBot* to other languages by using a multi-lingual language models, and curating a language specific distillation dataset.

YOLOv10 Ensemble Label Set Finally, the *YOLOv10 Ensemble* is limited to identifying only a set of 7 common UI elements. Other interactive elements prevalent on modern websites, such as sliders and date pickers, are not explicitly recognized by the model. While this could potentially reduce the contextual information available to the *Language Module*, we have empirically observed that deceptive patterns rarely use other interactive UI elements.

10 Future Work

AutoBot is a framework that takes a screenshot and returns localized deceptive patterns. We provide three sample applications that build on top of it. We envision that applications like `lighthouse-ci` [28] can easily extend our work to integrate *AutoBot* into developer CI/CD workflows. Regulators can also use *AutoBot* with models aligned to their regulations to enforce policies automatically.

Another direction for future work is to enhance *AutoBot*’s multilingual capabilities. While deceptive patterns are language-agnostic, the current framework’s training and evaluation datasets were focused on English-language websites. Future efforts could explore language-specific versions of the *LanguageModule* or integrate advanced LLMs that can reason across multiple languages.

Lastly, we also envision using *AutoBot* as a tool to generate large-scale datasets of websites with elements automatically labeled for deceptive patterns. A significant bottleneck for finetuning/retraining a VLLM for this task is the lack of large-scale annotated training data. Datasets created using *AutoBot* can be utilized to overcome this challenge and potentially improve the performance of VLLMs in detecting deceptive patterns in the future.

11 Conclusion

In this paper, we introduce *AutoBot*, a framework to automatically detect deceptive patterns on websites. *AutoBot* employs a modular approach: first, it captures a screenshot of the website and processes it using the *Vision Module* to provide a textual representation of the website (*ElementMap*). It then analyzes the

ElementMap using a *Language Module* to identify deceptive patterns and their type. We evaluate *AutoBot* on a dataset of real-world websites to demonstrate its accuracy in identifying and localizing deceptive patterns. We then instantiate *AutoBot* in three settings: a user-facing browser extension, a developer-facing Lighthouse report, and a researcher/regulator-facing website analysis tool.

Acknowledgments

This work was supported by the NSF through awards CNS-1942014 and CNS-2247381, and by a research grant from the Google PSS Privacy Faculty Award program. Finally, we thank the reviewers for their thoughtful recommendations.

References

- [1] Lighthouse - Chrome Web Store.
- [2] 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). <https://eur-lex.europa.eu/eli/reg/2016/679/oj>, 2016. Accessed: 2024-09-02.
- [3] The 2023 State of API Security Report - Global Findings, Apr. 2023.
- [4] WebGPU, Feb. 2024. Defines a modern graphics and compute API for the Web.
- [5] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [6] J. H. Adorna, A. J. Dantis, R. Fera, L. L. Figueroa, and R. Solamo. Developing a browser extension for the automated detection of deceptive patterns in cookie banners. In *Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2023)*, volume 20, page 101. Springer Nature, 2024.
- [7] V. Agarwal, N. Thureja, M. Krishan Garg, S. Dharmavaram, D. Kumar, et al. " which llm should i use?": Evaluating llms for tasks performed by undergraduate computer science students in india. *arXiv e-prints*, pages arXiv-2402, 2024.
- [8] S. aka Steve Spiker. Tweet by spike aka steve spiker on x. <https://x.com/spjika/status/1686492710910427137>, 2023. Accessed: 2024-09-02.
- [9] Anthropic. Fine-tune claude 3 haiku in amazon bedrock. <https://www.anthropic.com/news/fine-tune-claude-3-haiku>, July 2024. Accessed: April 15, 2025.
- [10] V. Bannihatti Kumar, R. Iyengar, N. Nisal, Y. Feng, H. Habib, P. Story, S. Cherivirala, M. Hagan, L. Cranor, S. Wilson, F. Schaub, and N. Sadeh. Finding a Choice in a Haystack: Automatic Extraction of Opt-Out Statements from Privacy Policy Text. In *Proceedings of The Web Conference 2020*, pages 1943–1954, Taipei Taiwan, Apr. 2020. ACM.
- [11] C. Bösch, B. Erb, F. Kargl, H. Kopp, and S. Pfattheicher. Tales from the dark side: Privacy dark strategies and privacy dark patterns. *Proceedings on Privacy Enhancing Technologies*, 2016.
- [12] H. Brignull. Deceptive patterns. <https://www.deceptive.design/>.
- [13] H. Brignull, M. Leiser, C. Santos, and K. Doshi. Deceptive patterns – user interfaces designed to trick you, 4 2023.
- [14] J. Brooke. Sus: A quick and dirty usability scale. *Usability Evaluation in INdustry/Taylor and Francis*, 1996.
- [15] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [16] T. Cairns. extcolors: Extract colors from an image using k-means clustering. <https://pypi.org/project/extcolors/>, 2021. Accessed: 2024-09-02.

- [17] California Privacy Protection Agency. California Privacy Rights Act (CPRA). <https://thecpra.org/>, 2024. Accessed: 2024-09-04.
- [18] J. Chen, J. Sun, S. Feng, Z. Xing, Q. Lu, X. Xu, and C. Chen. Unveiling the tricks: Automated detection of dark patterns in mobile applications. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–20, 2023.
- [19] K. Chen, Z. Zhang, W. Zeng, R. Zhang, F. Zhu, and R. Zhao. Shikra: Unleashing multimodal llm’s referential dialogue magic. *arXiv preprint arXiv:2306.15195*, 2023.
- [20] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.
- [21] G. Conti and E. Sobiesk. Malicious interface design: exploiting the user. In *Proceedings of the 19th international conference on World wide web*, pages 271–280, 2010.
- [22] Corporation for Digital Scholarship. Zotero. <https://www.zotero.org/>.
- [23] H. Cui, Z. Shamsi, G. Cheon, X. Ma, S. Li, M. Tikhonovskaya, P. Norgaard, N. Mudur, M. Plomecka, P. Racuglia, et al. Curie: Evaluating llms on multitask scientific long context understanding and reasoning. *arXiv preprint arXiv:2503.13517*, 2025.
- [24] A. Curley, D. O’Sullivan, D. Gordon, B. Tierney, and I. Stavrakakis. The design of a framework for the detection of web-based dark patterns. 2021.
- [25] M. Deitke, C. Clark, S. Lee, R. Tripathi, Y. Yang, J. S. Park, M. Salehi, N. Muennighoff, K. Lo, L. Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. *arXiv preprint arXiv:2409.17146*, 2024.
- [26] E. A. Elsouid, M. Hassan, O. Alidmat, E. Al Henawi, N. Alshdaifat, M. Igtait, A. Ghaben, A. Katrawi, and M. Dmour. Under sampling techniques for handling unbalanced data with various imbalance rates: A comparative study. *International Journal of Advanced Computer Science & Applications*, 15(8), 2024.
- [27] Google. full-page-screenshot.js. <https://github.com/GoogleChrome/lighthouse/blob/main/core/gather/gatherers/full-page-screenshot.js>.
- [28] Google. Lighthouse-ci. <https://github.com/GoogleChrome/lighthouse-ci>.
- [29] Google Chrome Developers. Lighthouse Overview.
- [30] C. M. Gray, Y. Kou, B. Battles, J. Hoggatt, and A. L. Toombs. The dark (patterns) side of ux design. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pages 1–14, 2018.
- [31] C. M. Gray, C. Santos, and N. Bielova. Towards a preliminary ontology of dark patterns knowledge. In *Extended abstracts of the 2023 CHI conference on human factors in computing systems*, pages 1–9, 2023.
- [32] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [33] C.-Y. Hsieh, C.-L. Li, C.-K. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, and T. Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.
- [34] Hugging Face. Running models on WebGPU. <https://huggingface.co/docs/transformers.js/en/guides/webgpu>, 2024. Accessed: 2025-04-13.
- [35] R. Khandelwal, T. Linden, H. Harkous, and K. Fawaz. {PriSEC}: A Privacy Settings Enforcement Controller. pages 465–482, 2021.
- [36] R. Khandelwal, A. Nayak, H. Harkous, and K. Fawaz. Automated Cookie Notice Analysis and Enforcement.
- [37] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.
- [38] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya. A survey on addressing high-class imbalance in big data. *Journal of Big Data*, 5(1):1–30, 2018.
- [39] M. Leiser and C. Santos. Dark patterns, enforcement, and the emerging digital design acquis: Manipulation beneath the interface. 2023.

- [40] C. Lewis. Gameful Patterns. In C. Lewis, editor, *Irresistible Apps: Motivational Design Patterns for Apps, Games, and Web-based Communities*, pages 33–50. Apress, Berkeley, CA, 2014.
- [41] M. Li, R. Zhang, J. Chen, J. Gu, Y. Zhou, F. DERNONCOURT, W. Zhu, T. Zhou, and T. Sun. Towards visual text grounding of multimodal large language model. *arXiv preprint arXiv:2504.04974*, 2025.
- [42] Z. Li, K. You, H. Zhang, D. Feng, H. Agrawal, X. Li, M. P. S. Moorthy, J. Nichols, Y. Yang, and Z. Gan. Ferret-ui 2: Mastering universal user interface understanding across platforms. *arXiv preprint arXiv:2410.18967*, 2024.
- [43] T. F. Liu, M. Craft, J. Situ, E. Yumer, R. Mech, and R. Kumar. Learning design semantics for mobile apps. In *The 31st Annual ACM Symposium on User Interface Software and Technology*, UIST ’18, pages 569–579, New York, NY, USA, 2018.
- [44] Y. Lu, J. Yang, Y. Shen, and A. Awadallah. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*, 2024.
- [45] M. Luo, S. Tan, J. Wong, X. Shi, W. Y. Tang, M. Roongta, C. Cai, J. Luo, T. Zhang, L. E. Li, R. A. Popa, and I. Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. <https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2>, 2025. Notion Blog.
- [46] S. H. Mansur, S. Salma, D. Awofisayo, and K. Moran. Aidui: Toward automated recognition of dark patterns in user interfaces. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1958–1970. IEEE, 2023.
- [47] A. Mathur, G. Acar, M. J. Friedman, E. Lucherini, J. Mayer, M. Chetty, and A. Narayanan. Dark patterns at scale: Findings from a crawl of 11k shopping websites. *Proceedings of the ACM on human-computer interaction*, 3(CSCW):1–32, 2019.
- [48] V. Mohankumar and S. Anbalagan. A benchmark dataset and ensemble YOLO method for enhanced underwater fish detection. *ETRI Journal*, n/a(n/a). _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.4218/etrij.2024-0383>.
- [49] A. Nayak, R. Khandelwal, E. Fernandes, and K. Fawaz. Experimental security analysis of sensitive data access by browser extensions. In *Proceedings of the ACM on Web Conference 2024*, pages 1283–1294, 2024.
- [50] notAI tech. Nudenet: Nudity detection with deep neural networks. <https://github.com/notAI-tech/NudeNet>, 2019. Accessed March 19, 2025.
- [51] OpenAI. Learning to reason with LLMs. <https://openai.com/index/learning-to-reason-with-llms/>.
- [52] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- [53] V. Pham, L. D. T. Ngoc, and D.-L. Bui. Optimizing YOLO Architectures for Optimal Road Damage Detection and Classification: A Comparative Study from YOLOv7 to YOLOv10, Oct. 2024. *arXiv:2410.08409 [cs]*.
- [54] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. *arXiv preprint arXiv:1806.01156*, 2018.
- [55] S. H. Raju, S. F. Waris, S. Adinarayna, V. C. Jadhala, and G. S. Rao. Smart dark pattern detection: Making aware of misleading patterns through the intended app. In *Sentimental Analysis and Deep Learning: Proceedings of ICSADL 2021*, pages 933–947. Springer, 2022.
- [56] J. Redmon. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- [57] S. Ren. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.
- [58] M. Renze. The effect of sampling temperature on problem solving in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7346–7356, 2024.
- [59] Y. Sazid, M. M. N. Fuad, and K. Sakib. Automated detection of dark patterns using in-context learning capabilities of gpt-3. In *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*, pages 569–573. IEEE, 2023.

- [60] R. Schäfer, P. M. Preuschoff, R. Niewianda, S. Hahn, K. Fiedler, and J. Borchers. Don't detect, just correct: Can llms defuse deceptive patterns directly? In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2025.
- [61] Schneider Wallace. Dark patterns: Making online subscriptions harder to cancel draws lawsuits, settlements, and government scrutiny. <https://www.schneiderwallace.com/media/dark-patterns-making-online-subscriptions-harder-to-cancel-draws-lawsuits-settlements-and-government-s> 2023.
- [62] V. P. Sermuga Pandian, S. Suleri, and P. D. M. Jarke. Uisketch: a large-scale dataset of ui element sketches. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2021.
- [63] shadcn. Build your component library - shadcn/ui.
- [64] Z. Shi, R. Sun, J. Chen, J. Sun, M. Xue, Y. Gao, F. Liu, and X. Yuan. 50 shades of deceptive patterns: A unified taxonomy, multimodal detection, and security implications. In *Proceedings of the ACM Web Conference 2025 (WWW'25)*, 2025.
- [65] F. Shiri, X.-Y. Guo, M. G. Far, X. Yu, G. Haffari, and Y.-F. Li. An empirical analysis on spatial reasoning capabilities of large multimodal models. *arXiv preprint arXiv:2411.06048*, 2024.
- [66] S. Shool, S. Adimi, R. Saboori Amleshi, E. Bitaraf, R. Golpira, and M. Tara. A systematic review of large language model (llm) evaluations in clinical medicine. *BMC Medical Informatics and Decision Making*, 25(1):117, 2025.
- [67] N. Shuyo and others (ported to Python). langdetect: Language detection library ported to python. <https://pypi.org/project/langdetect/>, 2021. Version [Specify version if needed, e.g., 1.0.9].
- [68] T. H. Soe, O. E. Nordberg, F. Guribye, and M. Slavkovik. Circumvention by design-dark patterns in cookie consent for online news outlets. In *Proceedings of the 11th nordic conference on human-computer interaction: Shaping experiences, shaping society*, pages 1–12, 2020.
- [69] L. Stein. Tweet by luke stein on x. <https://x.com/lukestein/status/1150014732486742016>, 2019. Accessed: 2024-09-02.
- [70] A. Stöver, N. Gerber, H. Pridöhl, M. Maass, S. Bretthauer, I. Spiecker Gen. Döhmman, M. Hollick, and D. Hermann. How Website Owners Face Privacy Issues: Thematic Analysis of Responses from a Covert Notification Study Reveals Diverse Circumstances and Challenges. *Proceedings on Privacy Enhancing Technologies*, 2023(2):251–264, Apr. 2023.
- [71] J. Tan, Z. Dou, W. Wang, M. Wang, W. Chen, and J.-R. Wen. Htmlrag: Html is better than plain text for modeling retrieved knowledge in rag systems. In *Proceedings of the ACM on Web Conference 2025*, pages 1733–1746, 2025.
- [72] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- [73] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov. Label Studio: Data labeling software, 2020-2025. Open source software available from <https://github.com/HumanSignal/label-studio>.
- [74] R. Walambe, A. Marathe, K. Kotecha, and G. Ghinea. Lightweight Object Detection Ensemble Framework for Autonomous Vehicles in Challenging Weather Conditions. *Computational Intelligence and Neuroscience*, 2021:5278820, Oct. 2021.
- [75] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding. Yolov10: Real-time end-to-end object detection. *arXiv preprint arXiv:2405.14458*, 2024.
- [76] D. Wang, J. Anthony Ribando, D. Mo, and N. Tung. insite. a chrome extension that protects consumers from marketing tricks when they shop online., 2019.
- [77] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [78] J. Wu, R. Krosnick, E. Schoop, A. Swearngin, J. P. Biggam, and J. Nichols. Never-ending Learning of User Interfaces, Aug. 2023. arXiv:2308.08726 [cs].

- [79] J. Wu, S. Wang, S. Shen, Y.-H. Peng, J. Nichols, and J. Bigham. Webui: A dataset for enhancing visual ui understanding with web semantics. *ACM Conference on Human Factors in Computing Systems (CHI)*, 2023.
- [80] M. Xie, S. Feng, Z. Xing, J. Chen, and C. Chen. Uied: a hybrid tool for gui element detection. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1655–1659, 2020.
- [81] M. Xie, S. Feng, Z. Xing, J. Chen, and C. Chen. Uied: a hybrid tool for gui element detection. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1655–1659, New York, NY, USA, 2020. Association for Computing Machinery.
- [82] H. You, H. Zhang, Z. Gan, X. Du, B. Zhang, Z. Wang, L. Cao, S.-F. Chang, and Y. Yang. Ferret: Refer and ground anything anywhere at any granularity. *arXiv preprint arXiv:2310.07704*, 2023.
- [83] Y. Zang, W. Li, J. Han, K. Zhou, and C. C. Loy. Contextual object detection with multimodal large language models. *International Journal of Computer Vision*, 133(2):825–843, 2025.
- [84] X. Zhang, L. De Greef, A. Swearngin, S. White, K. Murray, L. Yu, Q. Shan, J. Nichols, J. Wu, C. Fleizach, et al. Screen recognition: Creating accessibility metadata for mobile applications from pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2021.
- [85] Y. Zhang, A. Unell, X. Wang, D. Ghosh, Y. Su, L. Schmidt, and S. Yeung-Levy. Why are visually-grounded language models bad at image classification? *arXiv preprint arXiv:2405.18415*, 2024.
- [86] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Appendix

11.1 Full Taxonomy

Table 7: Filtered taxonomy of Deceptive Patterns. The table shows the Category, the Sub-Type, the description of the deceptive pattern, and an example.

Category	Sub-Type	Description	Examples
Interface Interference	Confirmshaming	Guilt-tripping users into making a specific choice	“No, I prefer to pay more”
	Fake-Scarcity/ Fake-Urgency	Create a false sense of urgency/scarcity to pressure users into making a choice	“Only 3 left in stock”
	Nudge	Nudge a user towards a specific choice.	“Accept All” in bright colors, while “Reject” is hard to notice
Forced-Action	Forced-Action	Design tactic forcing users to complete a specific task to proceed.	Pop-up ads
Obstruction	Pre-Selection	Choices given to users are already selected.	The checkbox of “Sign up for news and updates” is checked by default.
	Visual Interference	Misleading design elements that distract or mislead users from important information.	The term of use for service users are signing up for is in tiny font and cannot be clearly seen on sites.
	Jargon	The use of non-user-friendly language to prevent users from understanding important information.	“By affirming this selection, you consent to the perpetuation of automatic pecuniary transactions at designated intervals.”
Sneaking	Hidden Subscription	Users are not clearly informed that they are signing up for a service.	“By signing up for this email, you are agreeing to news and information from us”
	Hidden Costs	Users are not clearly informed about all the costs associated with a service.	During checkout, unexpected fees such as “handling charges” appear.
	Disguised Ads	Visually misleading ads embedded into page content.	Prominent “Download” button at the top of a page that redirects the user to unrelated adware.
	Trickwording	The use of non-user-friendly language to trick users into making certain choices	“Newsletter subscription by default, tick here to unsubscribe”
Non-Deceptive		Common, user-friendly design element that does not show any deceptive pattern.	Any text or web element that does not exhibit any deceptive behavior.

11.2 Website with Multiple Deceptive Pattern

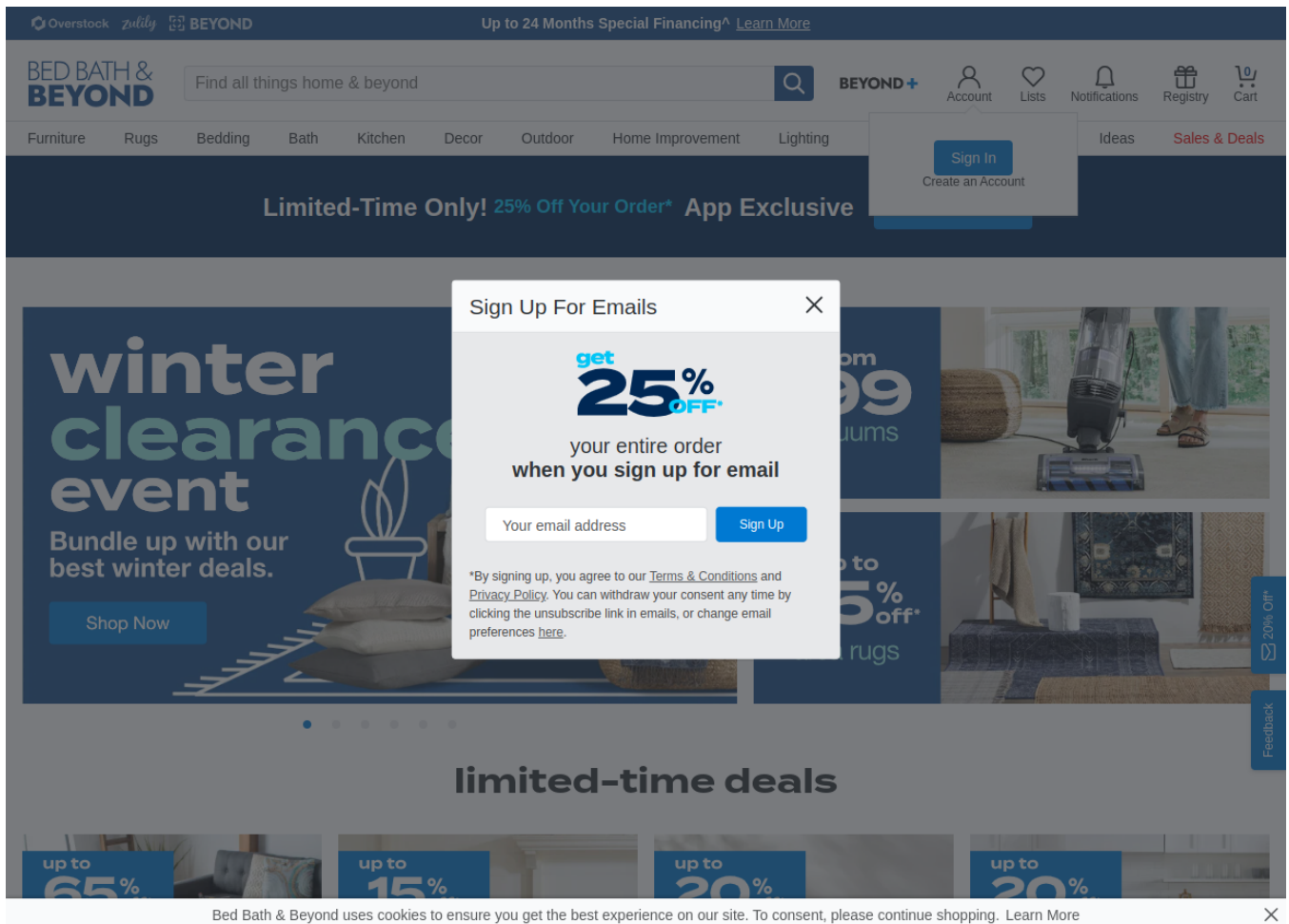


Figure 9: The screenshot of Bed, Bath, and Beyond, is identified as having four deceptive pattern categories on its web page.

11.3 Mapping to Gary et al.’s Taxonomy

Table 8: Mapping taxonomy of Deceptive Patterns to Gray et al. [31]

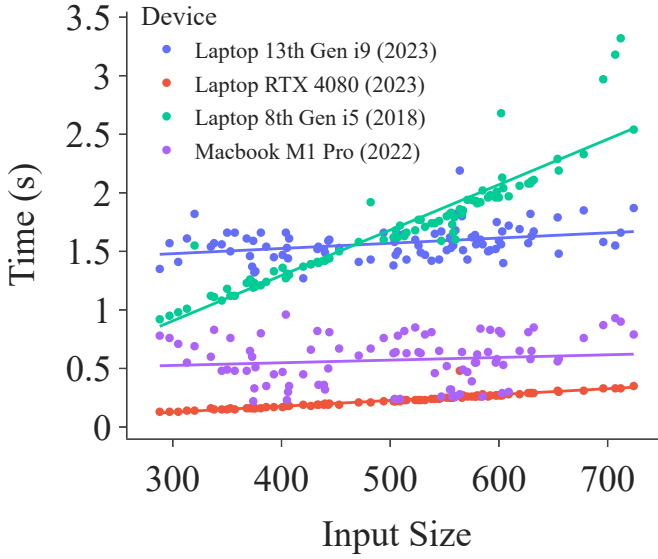
Category	Sub-Type	Mapping
Interface Interference	Confirmshaming	Social Engineering; Personalization
	Fake-Scarcity / Fake-Urgency	Social Engineering; Urgency
	Nudge	Interface Interference; Manipulating Visual Choice Architectures
Forced Action	Forced-Action	Forced Action
Obstruction	Pre-Selection	Interface Interference; Bad Default
	Visual Interference	Obstruction; Creating Barriers
	Jargon	Obstruction; Creating Barriers
Sneaking	Hidden Subscription	Sneaking; Hiding Information
	Hidden Costs	Sneaking; Hiding Information
	Disguised Ads	Sneaking; Bait and Switch
	Trick Wording	Sneaking; (De)contextualizing Cues

11.4 Website Generation Prompts

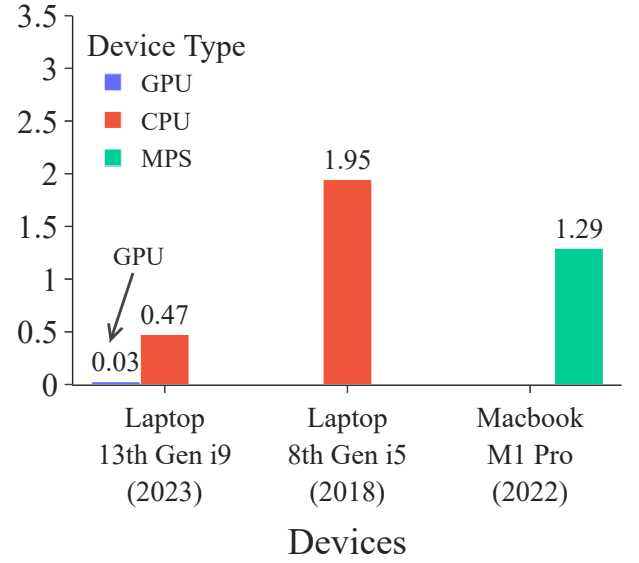
To generate websites using v0 we used prompts from GPT4. Some example prompts are shown below:

1. Create a virtual learning platform with courses, webinars, and interactive tools for students of all ages.
2. Create an online gourmet food shop featuring high-quality ingredients, kitchen tools, and gourmet recipes.
3. Design a warm and inviting UI for a pet care blog that radiates friendliness and approachability. The primary color palette should include soft, earthy tones with playful accents. Use a clean, easy-to-read sans serif font and incorporate elements like paw prints or pet silhouettes to enhance the thematic appeal. The homepage must prominently feature an engaging welcome message, and sections for various pets like dogs, cats, birds, and more, encouraging user navigation. Include a dynamic sidebar with widgets for pet care tips, a search bar, and featured posts. Dropdown menus should be intuitive, providing categories such as nutrition, training, health, and grooming. Visuals are key: integrate heart-warming pet images and infographics to explain care practices. End with footer links to contact details, social media, and a cute, animated pet mascot offering useful tips periodically.

11.5 Latency on Machines



(a) T5's latency when running on the three machines.



(b) YOLO's latency when running on the three machines

Figure 10: T5 and YOLOv10 model inference time running on the 3 machines

11.6 Mapping Multi-Token Labels to Single-Token Labels

Table 9: Mapping between multi-token and single-token representations of dark patterns and user interface concepts

Label	Multi Tokens	→	Single Token	Label
interface-interference	[3459, 18, 3870, 11788, 1]	→	[18805, 1]	distraction
forced-action	[5241, 18, 4787, 1]	→	[10472, 1]	obligation
obstruction	[26359, 1]	→	[12515, 1]	barrier
sneaking	[14801, 53, 1]	→	[14801, 1]	sneak
non-deceptive	[529, 18, 221, 6873, 757, 1]	→	[26213, 1]	irrelevant
confirmshaming	[3606, 7, 1483, 53, 1]	→	[12447, 1]	shame
fake-scarcity-fake-urgency	[9901, 18, 7, 1720, 6726..., 1]	→	[9554, 1]	manufactured
nudge	[3, 29, 13164, 1]	→	[3292, 1]	push
hard-to-cancel	[614, 18, 235, 18, 1608, 7125, 1]	→	[19885, 1]	sticky
pre-selection	[554, 18, 7, 15, 12252, 1]	→	[356, 1]	set
visual-interference	[3176, 18, 3870, 11788, 1]	→	[21634, 1]	obscure
jargon	[3, 5670, 5307, 1]	→	[11100, 1]	mystery
hidden-subscription	[5697, 18, 7304, 11830, 1]	→	[23808, 1]	conceal
hidden-costs	[5697, 18, 11290, 7, 1]	→	[594, 1]	price
disguised-ads	[31993, 26, 18, 9, 26, 7, 1]	→	[6543, 1]	ads
trick-wording	[7873, 18, 6051, 53, 1]	→	[21050, 1]	uncertain
not-applicable	[59, 18, 27515, 75, 179, 1]	→	[26213, 1]	irrelevant

11.7 Taxonomy Mapping from *DPGuard* to *AutoBot*

Table 10: Mapping taxonomy of DPGuard. The table shows deceptive patterns from DPGuard and the deceptive pattern Category/Sub-Type of *AutoBot*.

Deceptive Pattern	Category	Sub-Type
no dp	Non Deceptive	Not Applicable
nagging	—	—
roach motel	—	—
price comparison prevention	Sneaking	Hidden Subscription
intermediate currency	Forced Action	Forced Action
forced continuity	Sneaking	Hidden Subscription
hidden costs	Sneaking	Hidden Costs
sneak into basket	Sneaking	Hidden Subscription
hidden information	Obstruction	Visual Interference
preselection	Obstruction	Pre-Selection
toying with emotion	Interface Interference	Confirmshaming
false hierarchy	Interface Interference	Nudge
disguised ads	Sneaking	Disguised Ads
tricked questions	Sneaking	Trick Wording
small close button	—	—
social pyramid	Forced Action	Forced Action
privacy zuckering	Forced Action	Forced Action
gamification	Forced Action	Forced Action
countdown on ads	Forced Action	Forced Action
watch ads to unlock features or rewards	Forced Action	Forced Action
pay to avoid ads	Forced Action	Forced Action
forced enrollment	Forced Action	Forced Action

11.8 User Study Mock Website Samples

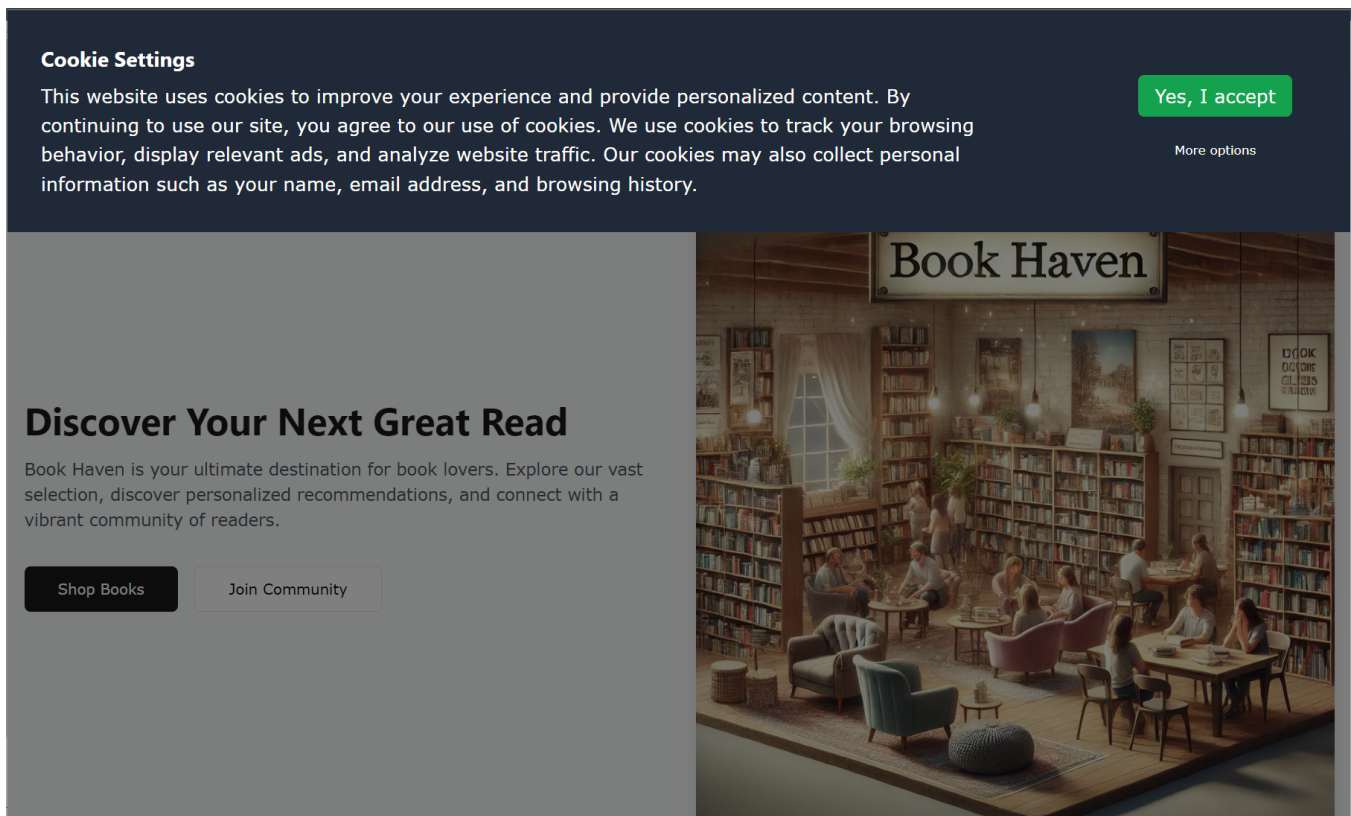


Figure 11: A website showing a book store's homepage with a cookie notice, nudging users to accept cookies.

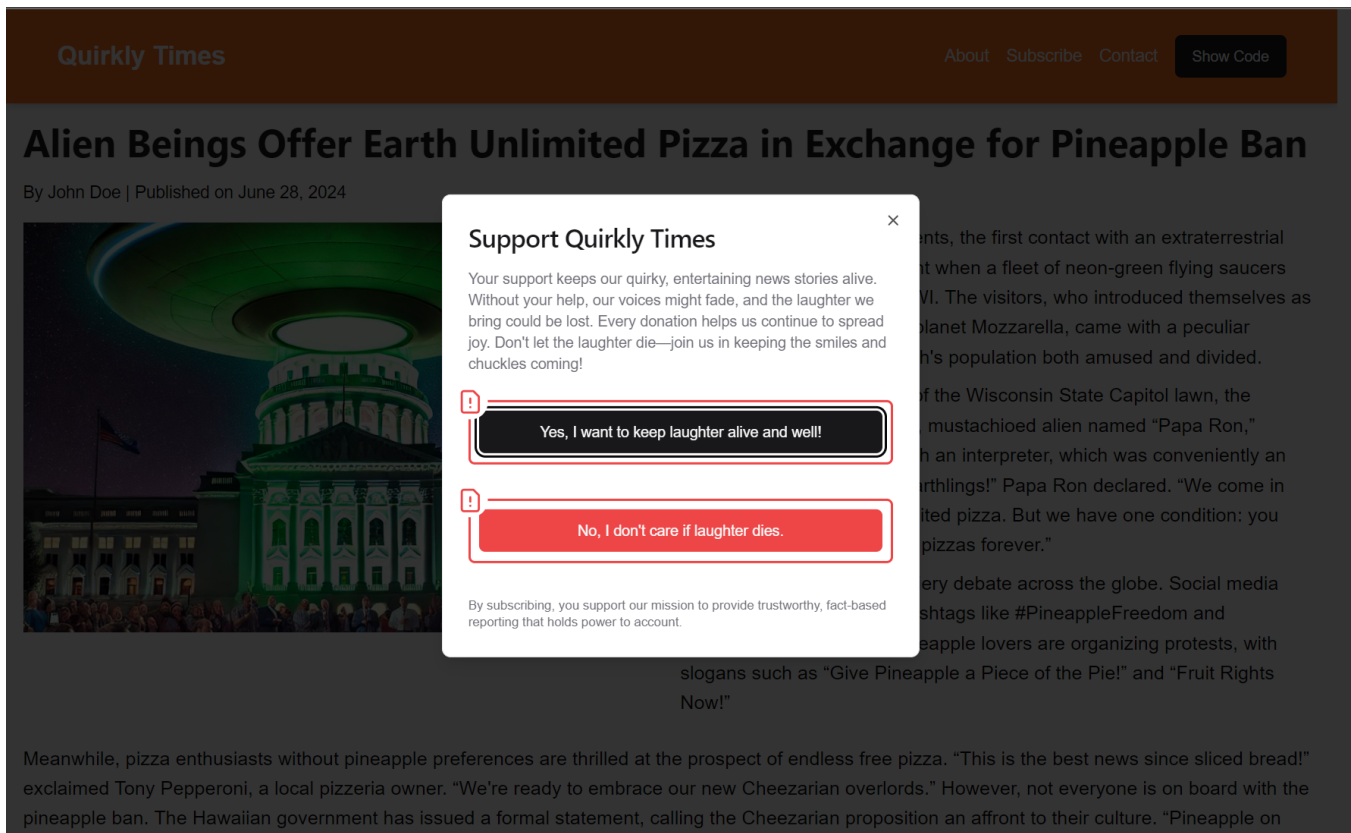


Figure 12: A news organization’s website asking users to donate. The deceptive patterns are highlighted in this example with a red box around them.

11.9 User Study Findings

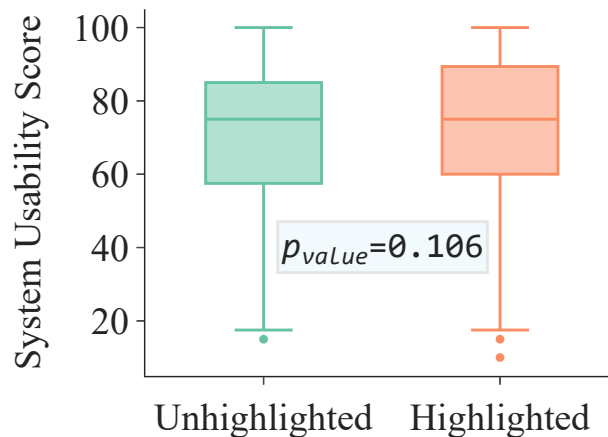


Figure 13: The results from the usability study show how highlighting affected the website’s user experience. We find that usability is not affected by the highlights ($p = 0.106$).

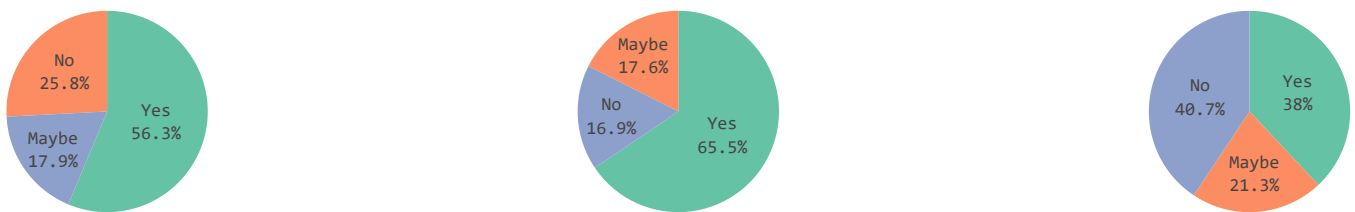


Figure 14: The distribution of participant responses to qualitative Q1-3 asked after the user study.