



CHROMA EYES: Detecting Inconsistencies of User Interface Elements between Light and Dark Modes of Web Applications

K C SHWETA, Korea University, Republic of Korea

BYUNGCHUL TAK*, Kyungpook National University, Republic of Korea

TEGAWENDÉ F. BISSYANDÉ, University of Luxembourg, Luxembourg

DONGSUN KIM*, Korea University, Republic of Korea

Dark mode interfaces in web applications have gained widespread adoption because of improved user comfort and reduced power consumption. While these interfaces can be implemented through built-in support or browser extensions that convert light mode layouts, inconsistencies frequently arise during the conversion process, including invisible UI elements, misplaced components, and incorrect color mappings. Despite the prevalence of these issues, *no existing approaches systematically detect such inconsistencies between light and dark mode interfaces*. Our preliminary study shows that popular commercial vision language models and accessibility issue detectors are ineffective for this task.

This paper presents CHROMA EYES, a novel approach to automatically detecting inconsistencies of graphical user interface elements between light and dark mode layouts of web applications. Detecting such inconsistencies is inherently challenging given that, since mode conversion intentionally changes colors and contrast, UI elements in light and dark modes are *expected* to look different. Thus, pixel-wise or visual comparison cannot distinguish intentional adaptations from actual errors. CHROMA EYES addresses this challenge by analyzing semantic roles and functional meanings of UI elements, enabling accurate correspondence detection between visually distinct but functionally equivalent components. We evaluate our approach on 2,009 screenshot pairs captured from 196 *real* web applications (147 with native dark mode support and 49 with browser extension-based conversion). CHROMA EYES achieves 96.19% accuracy at the screenshot level and 97.95% at the application level, significantly outperforming vision-language models (e.g., GPT-4o) and state-of-the-art accessibility issue detectors (e.g., OwlEye, axe DevTools).

CCS Concepts: • **Software and its engineering** → **Software maintenance tools**; **Empirical software validation**; **Software testing and debugging**; • **Computing methodologies** → **Object detection**; • **Applied computing** → **Optical character recognition**.

Additional Key Words and Phrases: dark mode layouts, GUI transformation, GUI inconsistencies, web applications, visual defect detection

ACM Reference Format:

K C Shweta, Byungchul Tak, Tegawendé F. Bissyandé, and Dongsun Kim. 2026. CHROMA EYES: Detecting Inconsistencies of User Interface Elements between Light and Dark Modes of Web Applications. *Proc. ACM Softw. Eng.* 3, ISSTA, Article ISSTA056 (October 2026), 23 pages. <https://doi.org/10.1145/3832147>

*Corresponding authors.

Authors' Contact Information: K C Shweta, Korea University, Department of Computer Science and Engineering, Seoul, Republic of Korea, devshwetak@gmail.com; Byungchul Tak, Kyungpook National University, School of Computer Science and Engineering, Daegu, Republic of Korea, bctak@knu.ac.kr; Tegawendé F. Bissyandé, University of Luxembourg, Luxembourg, tegawende.bissyande@uni.lu; Dongsun Kim, Korea University, Department of Computer Science and Engineering, Seoul, Republic of Korea, darksw@korea.ac.kr.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2994-970X/2026/10-ARTISSTA056

<https://doi.org/10.1145/3832147>

1 Introduction

Dark mode interfaces have transformed from a niche preference to a mainstream feature in modern web applications. This design paradigm, characterized by dark backgrounds and light text, addresses user demands for visual comfort and energy efficiency. Recent surveys demonstrate widespread adoption: 92% of software engineers prefer dark mode in development environments [15], while 82.7% of mobile users have set dark mode as their default system preference [17]. The business impact is equally compelling, implementing custom dark mode can reduce bounce rates by 60% and increase page engagement by 170% according to recent reports [14].

The appeal of dark mode stems from both user experience and technical benefits. Users report increased comfort when interacting with dark interfaces [55], particularly in low-light environments [24, 50]. Additionally, dark mode can significantly reduce power consumption on OLED (Organic Light-emitting Diode) screens by displaying fewer bright pixels, extending battery life on mobile devices [6, 31, 35, 56].

Web applications implement dark mode layouts through several different methods. One common approach involves manually designing native dark mode support, where developers designate specific background, text, and icon colors for dark themes and replace images as necessary. Another method utilizes browser extensions [18] that dynamically convert layouts into dark mode. While this approach eliminates the need for manual design effort, it frequently causes incorrect conversions in certain cases.

Dark mode layouts often exhibit inconsistencies that misalign with their corresponding light mode interfaces, regardless of implementation method [60]. These inconsistencies manifest as layout elements such as icons and buttons being removed, hidden, or misplaced during mode conversion. Other problems include elements becoming invisible or difficult to read due to improper color adjustment. Figure 1 shows real examples of these inconsistencies, where problematic elements are highlighted by red bounding boxes.

Although the modern dark mode in web applications was adopted as a mainstream feature [42] by Android and iOS in 2019 [27], there are a limited number of studies on such inconsistencies in the research community. The field is only beginning to explore the types of inconsistencies, their characteristics, and the development practices that contribute to them. Existing studies [26, 42] only focus on analyzing inconsistencies and providing recommendations to mitigate them, rather than developing automated detection techniques. Most of the inconsistency detection processes were manually conducted with pre-defined inconsistency types that were also manually identified. Automated inconsistency detection is still in its early stages.

Detecting inconsistencies between light and dark modes is intrinsically challenging because proper mode conversion *requires* visual changes. An approach that simply compares pixel-wise identicalness would flag every correctly converted element as different with white text becoming light gray, dark backgrounds replacing light ones, and icons changing color are all expected behaviors. Ironically, the opposite is also problematic: if a button or text block appears identical in both modes, this signals an incomplete conversion rather than consistency. Consequently, detecting true inconsistencies requires understanding the semantic role and functional meaning of each UI element, rather than relying on visual similarity.

In this paper, we present a novel approach, CHROMAEYES, to automatically detect inconsistencies between light and dark mode layouts of web applications. To the best of our knowledge, *this is the first approach to addressing the inconsistency problem between the modes*. Our approach automatically navigates a web application in light and dark modes, respectively, by scrolling and visiting pages, and captures its screenshot pairs at each location. Then, it identifies inconsistencies for each screenshot pair by comparing the representational meanings of the elements between the modes,

instead of simple visual identicalness, after recognizing interface elements of each mode, such as edges, text, objects, and layouts.

We evaluate CHROMAEYES on 147 *real* web applications, implementing light and dark modes, collected from the lists of popular and active websites [11, 16, 19, 20]. Note that *this is the first benchmark dataset to evaluate approaches to detecting inconsistencies between light and dark mode layouts of web applications*, to the best of our knowledge. Our approach automatically captures 2,009 screenshot pairs by navigating the web applications, and identifies inconsistencies in the pairs. The results show that it successfully detects inconsistencies in the pairs with 96.19% accuracy. Application-wise, the approach achieves 97.95% accuracy as well. The approach significantly outperforms popular vision language models, such as `gpt-4o` [8] and `claude-opus-4` [2], and the state-of-the-art accessibility issue detectors such as OwlEye [40] and axe DevTools [3]. In addition, CHROMAEYES is also effective in detecting inconsistencies of 49 applications whose dark mode layouts are implemented by browser extensions for dark mode conversion, with 98.88% accuracy.

In summary, this paper contributes the following:

- CHROMAEYES, a novel approach to detecting inconsistencies between light and dark modes of web applications. To the best of our knowledge, this is the first study to address the problem.
- Empirical results of evaluating our approach on 2,009 screenshot pairs captured from 147 *real* web applications, which show the effectiveness of the detection modules.
- Experiment results of comparing our approach with four popular vision language models [29] such as `gpt-4o` [8], and existing accessibility issue detection tools such as OwlEye [40].

2 Backgrounds

2.1 Dark Mode GUIs

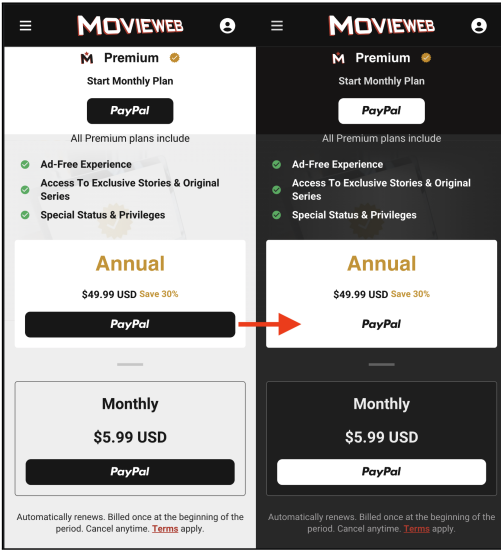
In 2018, MacOS Mojave adopted the system-wide dark mode GUI setting while integrating the modern GUI components of light mode [28]. The dark mode was thoughtfully crafted, with a sophisticated color palette balancing classic aesthetic and real-world visualization. Following the footsteps of this design aesthetic, iOS 13, iPadOS, and Android released the dark mode in 2019 [27, 34]. This design has gained massive popularity and acceptance among the users. Because of growing demand and system-wide adoption of dark mode in OS, many third-party applications and web applications introduced the dark mode to align with this design trend.

If a dark mode is carefully designed, adhering to design guidelines, GUI elements such as images and icons stand out from the background. This aesthetic creates a visually appealing experience for the user. The dark design emits less light from the screen, making the design very useful in low-light environments and reducing eye strain for the user who spends long hours in front of the digital screens. Dark mode is also known to consume less battery power for the OLED screen [31, 35].

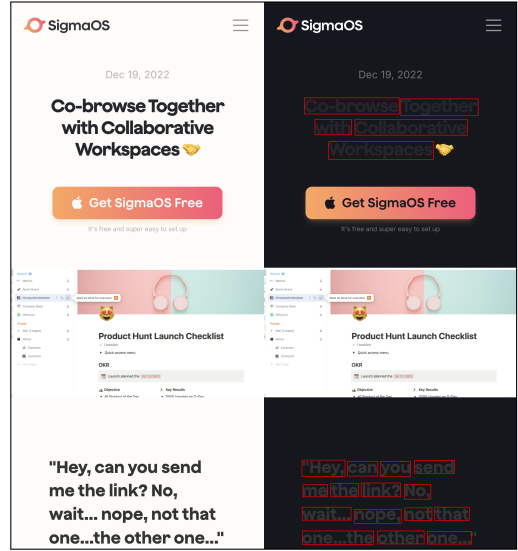
On the other hand, a poorly designed GUI can create inconsistencies between light and dark modes, negatively affecting the user experience. A well-crafted interface is crucial for building user trust. Macintosh is a good example that retained 20% more brand loyalty compared to Windows [57]. Many users favor dark mode over light mode. If the GUI fails to deliver a seamless experience, it can break user trust, ultimately affecting the success of the application.

2.2 Inconsistencies between Light and Dark Mode GUIs

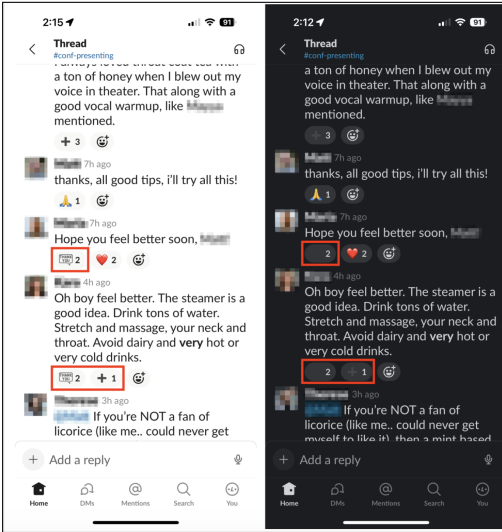
The term inconsistency may refer to a GUI state where its elements are well-designed in light mode, appearing visually cohesive, functional, and aligned with brand identity, but in dark mode, they may be poorly crafted, with low-quality graphics, weak contrast, and disrupted brand aesthetics (or vice versa). Prior to this, there have been no clear and formal definitions of inconsistency between light and dark modes among existing work [26, 42].



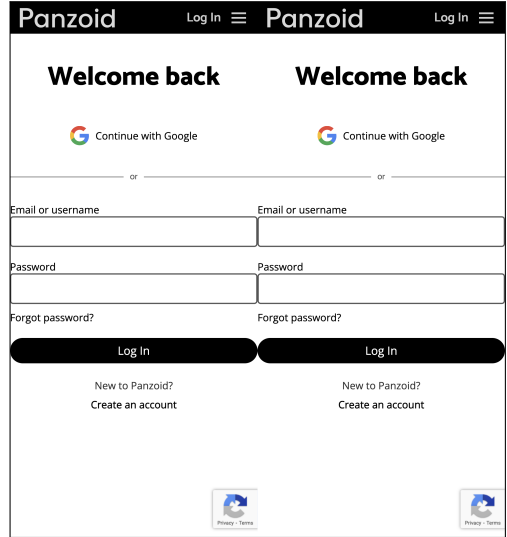
(a) Inconsistent layout due to incorrect edge conversion (button shape missing).



(b) Less decipherable text due to incorrect color conversion.



(c) Invisible icons due to incorrect conversion.



(d) Incomplete layout conversion (the dark mode layout on the right-hand-side is identical with the light mode of the left-hand-side, **but identical**).

Fig. 1. Common types of inconsistencies between light and dark modes of *real* web applications. More descriptions on other types of inconsistencies are available in our replication package [23].

Some previous work explored and discussed these inconsistencies. Ma et al. [42] defined the inconsistencies by enumerating the inconsistency categories they found during the evaluation of app development tools' ability to detect UI issues. Their inconsistencies included text & image contrast issue, an issue with too small touch target, missing item labels, and an overlapping clickable item issue. Andrew et al. [26] used the term inconsistency to indicate whether a certain app supported

both the light and dark modes within the target mobile platform. That is, if an app supported only one mode in iOS platform as an example, the app was considered inconsistent. Their work focused more on finding the variability of UI elements between light and dark modes, rather than discovering incorrectly rendered UI elements in one of the modes.

To clarify the inconsistencies between the modes in this study, we use the following definition:

DEFINITION 1. (*Light-Dark Mode Inconsistency*): An inconsistency occurs when a UI element or layout component that is visible/functional in one mode becomes invisible, illegible, misplaced, or functionally different in the other mode, beyond the expected color/contrast adaptations.

2.2.1 Inconsistencies in the Wild. In addition to our preliminary literature study of recent work [26, 42], we tried to figure out real cases of inconsistencies between the modes. Since there was no standard and widely accepted taxonomy or benchmark datasets for inconsistencies between light and dark mode GUIs yet, we first explored real web applications to identify the inconsistency cases. We collected issues related to dark mode from online sources, such as GitHub, X (formerly Twitter), and Google Play, using keywords such as “dark mode”, “dark mode issue,” and “night mode”. As a result, we collected and examined reports and reviews mentioning inconsistencies between dark and light modes: **104 issues** reported in GitHub projects, **54 app reviews** available in X (formerly Twitter)/Reddit, and **55 reviews** for Chrome dark mode extensions. The list of reports and reviews we inspected is available in our replication package [23].

On the basis of our investigation, we found the four most common types of inconsistency cases as shown in Figure 1. These are inconsistencies all found in **real** web applications. The types are (1) inconsistent layouts (Figure 1.a), (2) less decipherable text (Figure 1.b), (3) invisible icons (Figure 1.c), and (4) incomplete or no conversion to the dark mode (Figure 1.d). Although there are other minor types of inconsistencies we observed, this study focuses on the four types since we believe that they degrade the user experience more significantly than others.

2.2.2 Cause of Inconsistencies. The main causes of how inconsistencies of UI elements between light and dark modes arise can largely be divided into the following two types.

- **Color & contrast mismatch:** An inappropriate choice of color or contrast for UI elements is the most common cause of inconsistency. A suitable color of UI in one mode may not necessarily be suitable in another mode with different background colors. This renders texts invisible, the boundary of shapes disappear, and the depth of color ineffective.
- **Errors in mode conversion:** In some cases, the conversion of color or contrast from light to dark mode may not have been applied or incompletely applied due to bugs in the conversion logic.

2.3 Motivation: Limitations of Existing Tools

2.3.1 Limitation of Vision Language Models. Asking a pretrained multimodal model is one of the easiest ways to automatically detect GUI inconsistencies between light and dark modes. Specifically, there are vision language models (VLMs) such as ChatGPT [8] and Claude [2] that can answer a multi-modal query about given images. For example, one can write a prompt asking “(1) This pair of images are screenshots of light and dark mode, respectively. ... (more contextual details) ... Is this pair consistent or not? (2) If the pair is inconsistent, draw bounding boxes for each inconsistent area”.

Unfortunately, popular commercial VLMs are unable to correctly detect and localize inconsistencies. Figure 2.b shows a result of Claude-Opus-4 for the above prompt with an inconsistent pair of screenshots. While the model can figure out that the pair is inconsistent, it fails to correctly identify inconsistent GUI elements with bounding boxes. Note that the correct localization (by our approach described in Section 3) is shown in Figure 2.a. The comparison of inconsistency detection performance between our approach and VLMs is shown in Table 4.

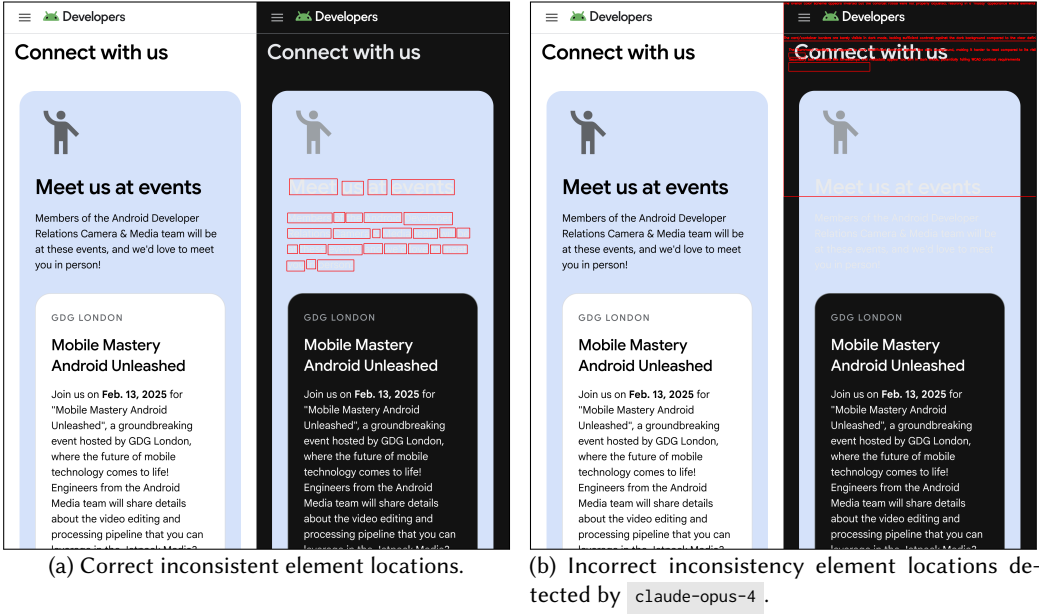


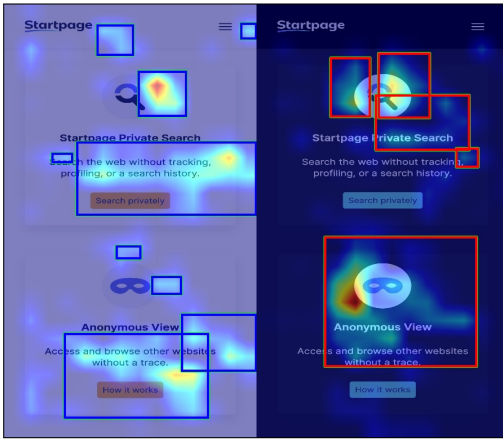
Fig. 2. Correctly vs. Incorrectly localized inconsistent GUI elements. While our approach shown in (a) can correctly identify inconsistent locations, `claude-opus-4` shown in (b), a vision language model, cannot precisely locate the inconsistencies.

2.3.2 Limitation of GUI Accessibility Issue Detectors. As VLMs are not effective for detecting inconsistencies (Section 2.3.1), another possible approach is to try applying popular accessibility issue detectors. These detectors can identify general accessibility issues such as less decipherable text and icons in web pages. We applied two off-the-shelf tools, OwlEye [40] and axe Devtools [3], to a pair of screenshots and identified inconsistencies by comparing accessibility issues detected by the tools for each mode. The accessibility issue detectors often incorrectly classify a consistent screenshot pair as inconsistent. As shown in Figure 3, the tools identify accessibility issues in each mode, but the issues are inconsistently detected across the consistent screenshot pairs. Moreover, the identified issues are false positives. The comparison of inconsistency detection performance between our approach and accessibility issue detectors is shown in Table 4.

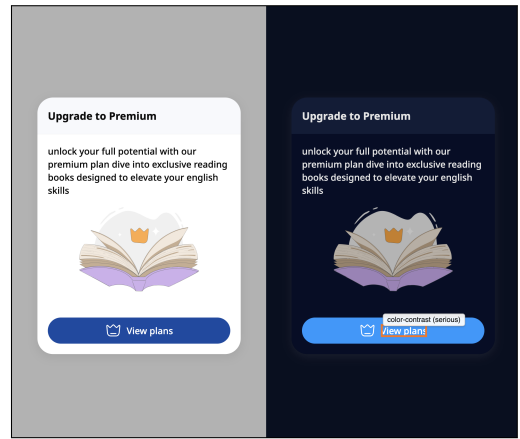
2.3.3 Limitation of Pairwise Comparison of GUI Elements. While VLMs (Section 2.3.1) and accessibility issue detectors (Section 2.3.2) are not successful in identifying inconsistencies, one possible approach is a pairwise comparison of GUI elements detected by text and object detectors in images. There are multiple text and object detectors publicly available to use. We selected Upstage OCR [22] and UIED [58] in our preliminary study to detect text and objects in screenshots, respectively.

However, this pairwise element comparison often produces false positives due to the instability of detection results. As shown in Figure 4, the detectors inconsistently identify GUI elements between the pair of screenshots when the pair is actually consistent. Developers and users may not make use of an inconsistency detector between light and dark modes if the detector produces too many false positives. Our approach overcomes this limitation and its results are shown in Table 3.

2.3.4 Limitation of Detecting Incomplete Conversion. In addition to the above examples, we found that existing tools could not identify inconsistencies that arose due to incomplete conversion (see Figure 1.d). It is challenging for the existing tools to recognize what has **NOT** been changed, and which should be different from each other when switching between the modes. The tools are often

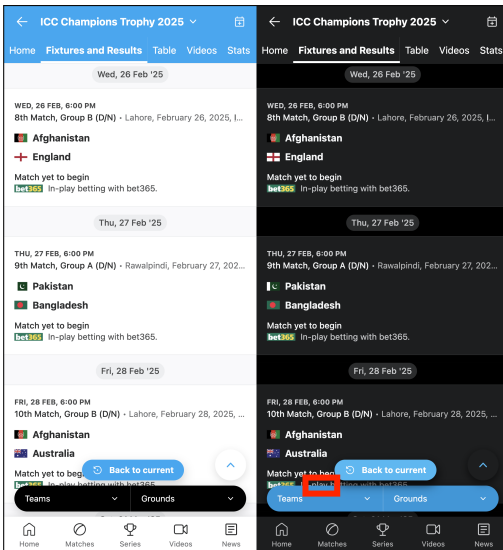


(a) Incorrect inconsistency detection by OwlEye [40] for a consistent screenshot pair.

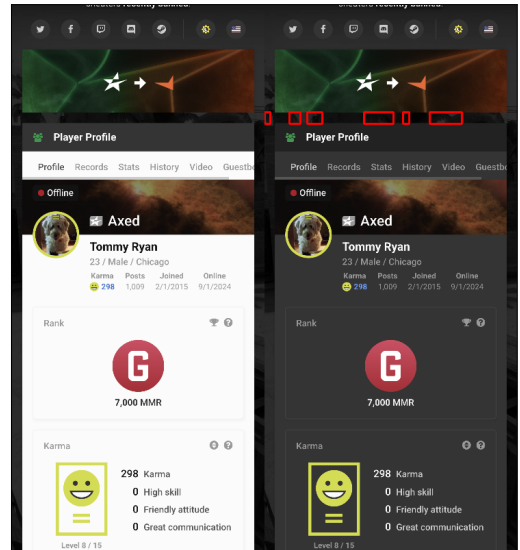


(b) Incorrect inconsistency detection by axe Devtools [3] for a consistent screenshot pair.

Fig. 3. Incorrect inconsistency detection due to the limitations of the existing UI accessibility issue detection tools: (a) OwlEye [40] and (b) axe Devtools [3].



(a) False positive due to the limitation of the text detection model [22].



(b) False positive due to the limitation of the object detection model [58].

Fig. 4. Incorrect detection of inconsistencies when pairwise GUI element comparison.

just designed to detect what is missing. However, to detect incomplete conversion inconsistencies, we need a tool that correctly identifies what and how GUI widget elements should be different from or identical with the corresponding elements in the opposite modes.

2.4 Challenge: Distinguishing Intentional Differences from Errors

It is NOT just about the identicalness of color, contrast, and brightness. Inconsistencies between light and dark mode layouts *cannot* be identified by merely checking the identicalness of GUI elements in the light mode with the corresponding counterparts in the dark mode as described in Section 2.3. Fundamentally, converting a light mode into a dark mode (or vice versa) implies

that the elements are changed in some manner, and thus each corresponding element pair between the modes should likely be different pixel-wise. Consequently, *if a pair of corresponding elements is pixel-wise identical, it is highly likely inconsistent as it might be **incorrectly** converted.*

It is a consistency problem rather than an accessibility issue. Thus, the challenge is to determine the consistency of GUI element pairs while they are not necessarily pixel-wise identical. The consistency between two different modes of the same application should be addressed as functional and representational consistency rather than visual identicalness. The existing studies are limited to manual detection [26, 42], and no approaches can automatically identify inconsistencies. Existing accessibility detection tools, such as UIS-Hunter [59], OwlEye [40], NightHawk [41], and axe DevTools [3], cannot identify inconsistencies since they are just designed for detecting accessibility issues in a single mode.

Novelty. To address the challenge, we propose a novel approach, CHROMA EYES, to detecting inconsistencies between light and dark modes of web applications, as described in Section 3. Although this approach does not address all types of inconsistencies that can be found in the web industry, it detects *major types* listed in Figure 1, such as edge, text, icons, emoticons, and incomplete conversion inconsistencies [60]. CHROMA EYES compares representational properties of GUI elements in a web application between the light and dark modes rather than simply detecting accessibility issues in each mode. Our novel approach outperforms existing techniques for detecting accessibility issues, as shown in Section 5.

3 Our Approach

Our approach, CHROMA EYES, detects inconsistencies between light and dark modes, as shown in Figure 5. To the best of our knowledge, *this is the first approach to address the problem of inconsistencies between the modes in an automated way.* After taking a URL of the target web application as input, the approach initiates navigation of the application pages. This automated navigation is recorded and repeated in both the light and dark modes to capture screenshot pairs of the target application. The screenshot pairs are first fed into detectors that localize three different GUI elements: objects, text, and edges. The approach identifies (1) layout inconsistencies based on the detected edges, (2) GUI widget elements (object and text) inconsistencies based on the detected edge, object, and text information, and (3) incomplete conversion inconsistencies based on the detected objects. The inconsistency identification results are analyzed screen-wise and application-wise, respectively, and CHROMA EYES returns the verdict. If the target application is classified as “inconsistent”, our approach provides an inconsistency report that lists inconsistent screenshots and the affected UI elements in the web application.

3.1 Automated Browsing and Screenshot Capturing

The first step of detection is exploring the target web application to collect screenshot pairs of light and dark modes at each identical location. This approach uses automated exploration since the search space of a web application could be very large. Some pages in an application can be expanded asynchronously and *almost infinitely*, and each page can be too long to explore within a limited time. Therefore, CHROMA EYES explores a feasible subset of a web application within a given time budget.

For a given URL of the target application, our approach explores the application by using two actions; (1) scrolling and (2) visiting links, as in Algorithm 1. The algorithm repeats scrolling a certain amount of page area and capturing the screenshots until it reaches either the end of the page or the maximum number of screens of a single page (Lines 8–15). The scrolling positions (I_s) and the maximum number of pages to capture (n_p) are customizable. Then, the approach repeats

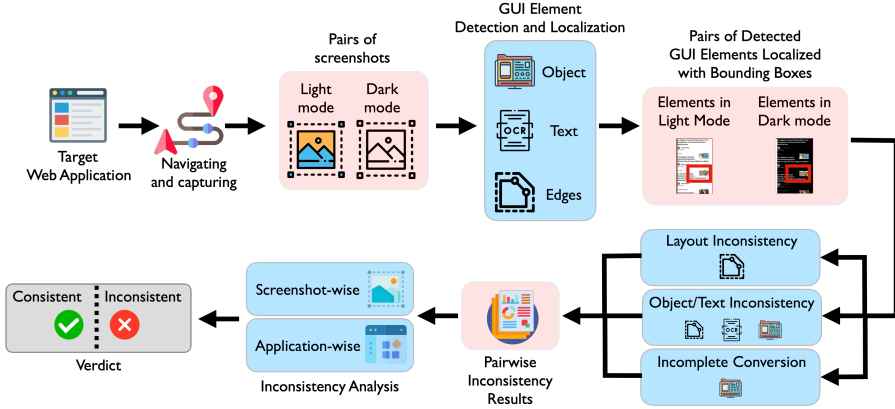


Fig. 5. Overview of CHROMAEYES, our inconsistency detection approach.

Algorithm 1: *screenShotPairs()*: Navigation to capture screenshot pairs between light and dark modes.

Input: Starting page of the target application URL: P_{init}
Input: Max. number of pages to visit: n_p
Input: Scroll interval (a list, e.g., [10, 20, ...]): I_s
Output: light and dark mode screenshot pairs: $\mathbb{S}$

```

1 Function screenShotPairs( $P_{init}, n_p, I_s$ ) :
2    $P_{toVisit} = \{P_{init}\}, P_{visited} = \mathbb{S} = \emptyset;$ 
3    $site = \text{getSite}(page_{init})$  // site information to find out pages to visit in the same web application.
4   Function capture( $P, n_p, I_s, site$ ) :
5      $P_{new} = \text{collectLinks}(P, site)$ 
6      $\mathbb{S}_{page} = \emptyset$ 
7      $h = \text{getScrollHeight}(P)$ 
8     foreach  $s \in I_s$  do
9        $pos = \frac{h \times s}{100}$  // compute an actual position of each % of the page.
10       $\text{mode}(P, \text{light})$  // change to the light mode.
11       $s_{light} = \text{takeScreenshot}(P, pos)$ 
12       $\text{mode}(P, \text{dark})$  // change to the dark mode.
13       $s_{dark} = \text{takeScreenshot}(P, pos)$ 
14       $\mathbb{S}_{page} = \mathbb{S}_{page} \cup \{(s_{light}, s_{dark})\}$ 
15    return ( $\mathbb{S}_{page}, P_{new}$ )
16  for  $P_{toVisit} \neq \emptyset \wedge \text{size}(P_{visited}) < n_p$  do
17     $P = \text{randomPop}(P_{toVisit})$ 
18    if  $P \notin P_{visited}$  then
19      ( $\mathbb{S}_{page}, P_{new}$ ) = capture( $P, n_p, I_s, site$ )
20       $P_{toVisit} = P_{toVisit} \cup P_{new}$ 
21       $\mathbb{S} = \mathbb{S} \cup \mathbb{S}_{page}$ 
22  return  $\mathbb{S}$ 

```

visiting the same scrolling positions after switching the mode (e.g., light to dark). Mode switching can be achieved by Selenium [12].

After scanning links available in the current page (in the same application as shown in Line 5 of Algorithm 1), CHROMAEYES jumps to another page of the same application sequentially. The approach repeats page-visiting until there are no more links available to visit or the maximum number of pages is visited (Line 16). After visiting pages and capturing screenshots in a mode (e.g., light mode), the navigation history is memorized and repeated in the same sequence for another mode (e.g., dark mode). Then, this step produces a set of screenshot pairs $\mathbb{S}$ (Line 22). Even if the web application is based on a single-page web framework such as React [10], our approach can scan the pages and capture the screenshots as long as the pages can be navigated by Selenium [12].

Algorithm 2: *detectLayoutInc()*: Layout inconsistency detection.

Input: Pair of sets of edge locations: $(edge_{light}, edge_{dark})$
Output: Inconsistency verdict based on edges (Boolean): V_{edge}
Output: Overlay image of light and dark mode edges: I_{edge}

```

1 Function detectLayoutInc( $s_{light}, s_{dark}$ ) :
2   // Assign color red color to light mode edges and green color to dark mode edges.
3    $coedge_{light} = coloringEdges(edge_{light}, color(red))$ 
4    $coedge_{dark} = coloringEdges(edge_{dark}, color(green))$ 
5    $I_{edge} = overlay(coedge_{light}, coedge_{dark})$ 
6   if  $color(red) \in I_{edge} \vee color(green) \in I_{edge}$  then
7     |  $V_{edge} = true$  // if any red or green pixel is included in the overlay, there are edge inconsistency.
8   else
9     |  $V_{edge} = false$  // if all pixels are cyan(=red+green), there is no edge inconsistency.
10  return  $V_{edge}, I_{edge}$ 

```

Our approach does *not* capture a screenshot pair if a *web page is dynamic* at a specific position, but it minimizes the impact of the dynamic pages. A page can display different screen layouts for the same request and the same position on every refresh. If a page is dynamic at a specific position, it is not able to correctly detect the inconsistencies between the light and dark modes at that position as it is necessary to refresh the page to switch modes. Fortunately, most of the dynamic pages are due to embedded advertisements, and *ad blockers* such as Adblock [1] can minimize the dynamic elements. CHROMA EYES applies the ad blockers when capturing screenshots to mitigate the impact of the dynamic pages due to advertisements.

3.2 GUI Element Detection and Localization

CHROMA EYES collects GUI elements and their locations in a pair of screenshots (i.e., s_{light} and $s_{dark} \in \mathbb{S}$ collected by Algorithm 1) by applying publicly available vision models. Our approach feeds each screenshot from the previous step to object (O), text (T) and edge (M) detection models, respectively. The result of each model is a set of bounding boxes designating the locations of the detected elements. Figure 2.a shows an example of text detection results.

The detected elements can be formalized as:

- Edge locations: $edge_{light} = \mathbb{M}(s_{light})$ and $edge_{dark} = \mathbb{M}(s_{dark})$
- Text locations: $text_{light} = \mathbb{T}(s_{light})$ and $text_{dark} = \mathbb{T}(s_{dark})$
- Object locations: $obj_{light} = \mathbb{O}(s_{light})$ and $obj_{dark} = \mathbb{O}(s_{dark})$

Rather than relying on specific detection models, the users of this approach can select and plug in any models into the approach. Note that the contribution of this approach is on leveraging the results of existing vision element detection models for identifying the inconsistencies between light and dark GUI modes. Our experiment setup (see Section 4) specifies vision models to evaluate the approach.

3.3 Layout Inconsistency Detection

Once we have a set of edge information in the screenshots captured from a web application, our approach inspects inconsistencies of layouts between the screenshots of each mode. As the layout of common web applications is often organized by panes and sections, the edges of those panes and sections should be visually recognizable after converting. However, many applications fail to maintain edge consistency after converting the mode.

After getting the edge locations as inputs, our approach colors the edges red for the light mode images and green for the dark mode images, respectively (Lines 3 and 4). The approach overlays

Algorithm 3: *detectInconsistentElement()*: Inconsistent text and object detection.**Input:** Sets of detected elements: $text_{light}, text_{dark}, obj_{light}, obj_{dark}, edge_{light}, edge_{dark}$ **Output:** Inconsistency verdict due to missing elements (Boolean): $V_{element}$ **Output:** Images with bounding boxes of missing elements for each mode: $I_{light}^{miss}, I_{dark}^{miss}$

```

1 Function detectInconsistentElement( $e_{light}, e_{dark}$ )
2   Function collectMissingElement( $E_s, E_t, edge_s, edge_t$ ) :
3      $P_{inc} = \emptyset$ 
4     foreach  $bb_s \in E_s$  and  $bb_t \in E_t$  do
5       // search for the element corresponding to  $bb_t$  in  $E_t$  at the same location  $bb_s$ .
6       if  $\neg any(Adjacent(bb_s, bb_t))$  then
7         // if no matching element found, check edges between the source and target at the bounding box
          of the source.
8         if  $getEdge(edge_s, bb_s) \neq getEdge(edge_t, bb_s)$  then  $P_{inc} = P_{inc} \cup \{bb_s\}$ ;
9     return  $P_{inc}$ 
10   $I_{dark}^{miss} = collectMissingElement(text_{light}, text_{dark}, edge_{light}, edge_{dark})$ 
       $\cup collectMissingElement(obj_{light}, obj_{dark}, edge_{light}, edge_{dark})$  // invisible in the dark mode.
11   $I_{light}^{miss} = collectMissingElement(text_{dark}, text_{light}, edge_{dark}, edge_{light})$ 
       $\cup collectMissingElement(obj_{dark}, obj_{light}, edge_{dark}, edge_{light})$  // invisible in the light mode.
12  if  $I_{light}^{miss} \neq \emptyset \vee I_{dark}^{miss} \neq \emptyset$  then  $V_{element} = True$  ;
13  else  $V_{element} = False$  ;
14  return ( $V_{element}, I_{light}^{invis}, I_{dark}^{invis}$ )

```

both images together to compute a new color of each pixel (Line 5). The resulting pixels (i.e., I_{edge}) are one of red, green, and cyan. The cyan edges are regarded as consistent ones. If the overlay image contains any red (edges only in the light mode) or green (edges only in the dark mode) pixels, there are edge inconsistencies between the modes (Line 7). Otherwise, it is not able to detect any inconsistency (Line 9). CHROMAEYES reports the verdict of inconsistency (V_{edge}) and overlay pixel map (I_{edge}) of edges (Line 10).

3.4 Object and Text Inconsistency Detection: Two-Step Detection

To detect text and object inconsistencies (shown in Figure 1.b and c), CHROMAEYES first compares GUI widget elements in the screenshots, detected by $T(s_*)$ and $O(s_*)$ as described in Section 3.2. For each widget element detected, our approach tries to locate a corresponding element in the opposite screenshot (Line 6 in Algorithm 3). The approach repeats *this pairing step* twice (light mode screenshot against dark mode screenshot, and vice versa).

If it is unable to localize any corresponding element in the opposite screenshot (Line 6 in Algorithm 3), our approach leverages edge information together when detecting inconsistencies as the *second step*. As shown in the examples in Figure 4, a simple pairwise comparison can often produce false positives due to the unstable results of the element detection models. Thus, CHROMAEYES additionally compares edge information in the areas of bounding boxes if an element is **NOT** detected in one of the screenshot pair (Line 8 in Algorithm 3). Figure 6 shows a visualized version of the comparison process.

3.5 Incomplete Conversion Detection

Our approach detects inconsistencies that arise from incomplete conversions (see Figure 1.d) by identifying and comparing the backgrounds of the pair of screenshots. CHROMAEYES first extracts the background area of screenshots in each mode after collecting bounding boxes of non-background objects (Lines 3–4 of Algorithm 4). Then, it computes the pixel-wise color differences between the two background areas of the screenshot pair (Line 6). After computing the number of “too-different”

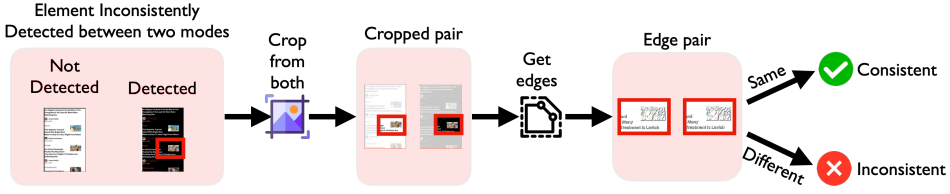


Fig. 6. Detailed process of how to use edge information when there is a (possibly) missing element in one of the pair during the detection of “Object/Text Inconsistency” in Figure 5. This process specifically corresponds to Line 8 in Algorithm 3.

Algorithm 4: *detectIncompleteConversion()*: Incomplete conversion detection.

Input: Pair of screenshots collected by Algorithm 1: $(s_{light}, s_{dark}) \in \mathbb{S}$
Input: Sets of detected GUI objects: obj_{light}, obj_{dark}
Input: Threshold for color difference: $colDiff$
Input: Threshold for background area difference: $areaDiff$
Output: Inconsistency due to incomplete conversion (Boolean): $V_{incomplete}$
Output: Pixel map representing incomplete conversion inconsistency: $s_{incomplete}$

```

1 Function incIncompleteConversion( $s_{light}, s_{dark}$ ):
2   // compute the background area of each mode.
3    $b_{light} = s_{light} - \bigcup obj_{light}.getBoundingBox()$ 
4    $b_{dark} = s_{dark} - \bigcup obj_{dark}.getBoundingBox()$ 
5   // compute the pixel-wise color difference between the two background areas.
6    $p_{diff} = getColorDiff(b_{light}, b_{dark})$ 
7   // count the number of pixels whose color difference is larger than  $colDiff$ .
8    $n_{diff} = countDiffPixels(p_{diff}, colDiff)$ 
9    $V_{incomplete} = \frac{n_{diff}}{|p_{diff}|} \geq areaDiff$ 
10  // highlight a pixel-wise overlay of the background color difference on the dark mode screenshot.
11   $s_{incomplete} = overlay(s_{dark}, p_{diff})$ 
12  return ( $V_{incomplete}, s_{incomplete}$ )

```

pixels (n_{diff}) based on the threshold $colDiff$ (Line 8), it calculates the ratio of the “too-different” pixels against the background area. If the ratio is larger than the threshold $areaDiff$, the dark mode has an incomplete conversion inconsistency. Our approach returns the verdict ($V_{incomplete}$) and the pixel map ($s_{incomplete}$) highlighted with inconsistent areas in the dark mode screenshot.

3.6 Analysis and Verdict

After incorporating the inconsistency detection results, CHROMAEYES analyzes them and reports a verdict. For each screenshot pair, if $V_{edge} \vee V_{element} \vee V_{incomplete} == True$, our approach reports that the pair is **inconsistent** (i.e., *screenshot-wise*) and provides corresponding evidence, I_{edge} , I_{invis}^{light} , I_{invis}^{dark} , and $s_{incomplete}$. In addition, the approach reports that the application is inconsistent if any screenshot pair is **inconsistent** (i.e., *application-wise*).

4 Experimental Setup

4.1 Research Questions

Our experiments investigate the following research questions:

- (1) **RQ1. (Detection)** How effective is CHROMAEYES at detecting inconsistencies between the light and dark mode of *real* web applications?
- (2) **RQ2. (Comparison)** Does CHROMAEYES outperform vision language models [29] and accessibility testing tools [3, 40]?

- (3) **RQ3. (Dark mode extensions)** How effective is CHROMAEYES at detecting inconsistencies of the dark mode converted by browser extensions?

4.2 Subjects

To assess the effectiveness of CHROMAEYES, we collected 147 *real* web applications with explicit light and dark mode switching, from the following sources: Tranco [11], Alexa¹ [16], DataForSEO [19], and Moz [20]. Note that *there is no standard benchmark to evaluate detectors for inconsistencies between light and dark modes* to the best of our knowledge. Thus, all of our experiments are conducted on the subjects we collected by the following procedures.

4.2.1 Subject Selection Procedure. We first inspected the top-ranked applications in the four sources (1,000 each) without creating an account to avoid any bias². The initial step collected 2,297 unique web applications after cleaning up obvious duplicates (i.e., identical URLs across the sources). The second step filtered out 775 applications (1) with CAPTCHA or specific human interaction requirements on the landing page, (2) unreachable or inactive, (3) insecure, and (4) with inappropriate content. Among the remaining 1,522 applications after filtering, the third step identified 311 web applications with *explicit* light and dark mode switching. The other 1,211 applications implicitly supported light and dark modes or their switching methods were unknown, and thus, they were excluded to eliminate ambiguity.

Note that our selection procedure just focuses on applications with **explicit** mode switching features for more effective evaluation of our approach. The fourth step removes redundant 74 applications (e.g., different URLs, but they are aliases), 64 applications with logging requirements on the first landing page, and 26 applications with dynamic pages (refreshing makes the pages always show different content). As a result of this selection procedure, our experiment subjects include 147 web applications with light and dark mode support. The experiments for RQ1 and RQ2 use these subjects whose list is available in our replication package [23].

4.2.2 Applications Converted by Browser Extensions. The experiment for RQ3 examines screenshot pairs of web applications converted from a light mode into a dark mode by browser extensions. We collected browser extensions that can convert a web application into dark mode. These extensions are popular as they are useful to switch any web application without native dark mode support. Based on the popularity, we selected 7 dark mode converter extensions.

To evaluate CHROMAEYES on web applications converted by the 7 extensions, we collected applications that supported the light mode only. Among the top-ranked applications in Tranco [11], we selected 7 top applications. As a result, we have 49 ($=7 \times 7$) combinations as additional applications for answering RQ3.

4.3 Metrics

Although there are no widely accepted metrics for the inconsistency detection problems, we can use accuracy, precision, recall, and F1-score as it can be regarded as a binary classification problem. First, we can classify the outcomes of CHROMAEYES as follows:

- True Positive (TP): actual inconsistent (screenshot or application) pair correctly classified as inconsistent.
- False Positive (FP): actual consistent pair incorrectly classified as inconsistent.
- True Negative (TN): actual consistent pair correctly classified as consistent.
- False Negative (FN): actual inconsistent pair incorrectly classified as consistent.

¹Although Amazon discontinued this listing service, the web applications on the list are still very popular and active (<https://aws.amazon.com/data-exchange/resources/end-of-life-of-alexa-web-information-service/>).

²We collected 500 web applications from Moz [20] as it lists only 500 applications.

Then, our evaluation computes the following metrics to measure the performance based on each set of the above outcomes:

- Accuracy = $\frac{|TP \cup TN|}{|TP \cup FP \cup TN \cup FN|}$
- Precision (Inconsistent) = $\frac{|TP|}{|TP \cup FP|}$
- Precision (Consistent) = $\frac{|TN|}{|TN \cup FN|}$
- Recall (Inconsistent) = $\frac{|TP|}{|TP \cup FN|}$
- Recall (Consistent) = $\frac{|TN|}{|TN \cup FP|}$
- F1-Score = $\frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

4.4 Parameters

To run CHROMAEYES, its parameters should be specified. First, our experiments adopt the Canny model [21] for $\mathbb{M}$, UpstageAI's OCR model [22] for $\mathbb{T}$, and UIED [58] for $\mathbb{O}$, respectively, to detect and localize edges, text, and objects as described in Section 3.2. We chose these models since they were easy to use, publicly available, and in particular, performed sufficiently accurately and stably for our purpose. Comparing the performance of different models is out of scope for our study, and it remains as our future work.

Other parameters are specified and justified as follows. For Algorithm 1, our experiments set $n_p = 5$ (maximum number of pages to visit) and $I_s = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$ (scroll interval), which are sufficient values to show the effectiveness of our approach. Note that the values should be expanded in real use to traverse the web application completely. This experiment utilizes the value of the Intersection over Union (IoU) threshold [13] as 0.5 for *adjacent*(...) in Line 6 of Algorithms 3 to check for the overlapping areas between the corresponding bounding boxes of light and dark modes. Additionally, we use $colDiff = \frac{80}{250}$ and $areaDiff = 0.1$, respectively, for Algorithm 4, by following design guidelines for light and dark modes [4, 5]. The sensitivity analysis results of these parameters (i.e., IoU, colDiff, and areaDiff) are reported in Section 6.2.

4.5 Experiment Procedures

4.5.1 Detection. To answer RQ1, we applied the detection process of CHROMAEYES to 147 applications. As described in Section 3.6, we collected the screenshot-wise and application-wise consistency results. Since there is no existing ground-truth dataset for inconsistencies between light and dark modes, two authors independently labeled the screenshot pairs as either consistent or inconsistent. For the 1,470 screenshot pairs used in the RQ1 evaluation, the two raters agreed on 1,337 cases and disagreed on 133 cases, yielding an observed agreement P_o of 0.9095. To statistically quantify the inter-rater reliability, we further computed Cohen's kappa. The expected agreement P_e was 0.5208, resulting in Cohen's K of 0.8112. This indicates strong agreement between the two raters beyond chance. Disagreements were resolved through joint re-examination. For contrast-related cases, the raters additionally used the WCAG contrast-ratio criteria as an objective reference: contrast ratios below 4.5:1 for text and below 3:1 for non-text elements were treated as evidence of inconsistency. The resolved labels were then used as the ground truth for computing the screenshot-wise and application-wise performance metrics. Note that it is challenging to completely detect inconsistencies even with manual inspection. Therefore, our inspection results might be vulnerable to missing inconsistencies. Nevertheless, we have thoroughly inspected the results to mitigate this issue. Our manually labeled dataset is publicly available [23].

4.5.2 Comparison with Vision Language Models and Accessibility Issue Detectors. As the problem of inconsistency detection between light and dark mode GUIs has been rarely explored by other studies, there are no available techniques to compare with CHROMAEYES side-by-side, to the best of our knowledge. Instead, we ask vision language models [29] whether a given pair is inconsistent, since language models perform well for many downstream tasks, including identifying issues in given images, which takes images and text and returns text or images.

To answer RQ2, our experiment employs four popular language models; `gpt-4o` [8] of OpenAI, `gemini-2.5-pro` [7] of Google, `claude-opus-4` [2] of Anthropic, and `grok-4.2-reasoning` [9] of xAI. The experiment uses a few-shot prompt [30, 52] template carefully engineered to simulate a task [37, 46] of GUI testers and developers (the prompt template is available in our replication package [23]). Our few-shot prompts consist of text parts that set the role of the LLM as an experienced UI/UX designer and provide two pairs of consistent and another two pairs of inconsistent screenshots. One inconsistent pair contains examples of inconsistent edges as well as indecipherable icons and text. Another pair shows the case of incomplete conversion. We feed screenshot pairs and the prompt to each model, and ask them to return a verdict (inconsistent or consistent) and bounding boxes if they are inconsistent.

In addition to the vision language models, our experiment also includes a comparison with accessibility issue detectors. Although they are not designed to detect inconsistencies ‘between light and dark modes’, the tools can still detect accessibility issues in each mode independently. After applying the tools to each mode, we can cancel out the corresponding issues between the modes. If there are remaining issues in any mode, we assume that inconsistencies between light and dark modes are detected.

After inspecting the availability and usability of existing accessibility issue detectors, we chose OwlEye [40] and axe DevTools [3]. There are other tools such as UIS-Hunter [59] and NightHawk [41]. However, they are currently not available. We have contacted the authors of both techniques. The authors of UIS-Hunter confirmed that they retracted the paper [59] a couple of years ago, and the replication package has been invalidated. We could not receive any response from the authors of NightHawk, and it turns out that the technique is not reproducible with its replication package.

4.5.3 Detection for Applications Converted by Browser Extensions. Similar to the experiment described in Section 4.5.1, we applied our approach to the 7 applications after converting them by 7 extensions each. As a result, our approach detected inconsistencies in 49 ($=7 \times 7$) combinations. Then, we followed the same procedure with Section 4.5.1 and computed the metrics (Section 4.3).

5 Results

5.1 RQ1: Inconsistency Detection

CHROMAEYES effectively detects inconsistencies between the modes as shown in Table 1. As described in Section 4.5.1, we computed the performance with two categories: screenshot-wise and application-wise. Our approach achieved high accuracy both for screenshots (96.19%) and applications (97.95%). The classification performance is more precise for application-wise (97.65%) than screenshot-wise (91.66%), while the approach can detect the inconsistency pairs for all applications (100%) and most for screenshots (99.66%).

Our approach over-approximates the inconsistencies for application-wise results as shown in the recall for consistent pairs (86.36%) of Table 1. This indicates that there are several false positives in which CHROMAEYES classifies the applications as “inconsistent” while they are actually “consistent”. Nevertheless, it minimizes missing inconsistencies, as shown by the precision for inconsistencies (97.65%).

While the overall classification results show that CHROMAEYES is effective, the performance of individual detection modules differs from each other. Table 2 lists the results for each inconsistency type; edge, text, object, and incomplete conversion, discussed in Section 2.2. While the accuracy of all types are higher than 95%, the precision values of text and object inconsistencies are relatively lower (86.42% and 77.94%, respectively) than other two types (91.45% and 89.13%). This implies that the approach slightly over-approximates consistent screenshots as inconsistent screenshots (i.e., false positives). This will be discussed in Section 6.3.

Table 1. Detection results of 147 web applications with explicit mode conversion support described in Section 4.2.

Metric	Screenshot-wise	Application-wise
Accuracy	0.9619	0.9795
Precision (Inconsistent)	0.9166	0.9765
Precision (Consistent)	0.9975	1.0000
Recall (Inconsistent)	0.9966	1.0000
Recall (Consistent)	0.9382	0.8636
F1-Score (Inconsistent)	0.9549	0.9881
F1-Score (Consistent)	0.9669	0.9268

The above values are based on our classification results (TP, TN, FP, FN): screenshot-wise (594,820, 54, 2), and application-wise (125, 19, 3, 0), respectively.

Our approach effectively mitigates the false positives by applying the process shown in Figure 6. Table 3 shows the comparison results between the performance of CHROMA EYES without and with the process described in Figure 6. In particular, the process improves the precision values from 80.83% to 86.42% for text inconsistencies, and from 51.55% to 77.94% for object inconsistencies, respectively. This indicates that the process effectively filters out incorrect detection results such as the cases shown in Figure 4, i.e., false positives.

Answer to RQ1: CHROMA EYES successfully detects inconsistencies between light and dark modes with 96.19% screenshot-wise accuracy and 97.95% application-wise accuracy.

5.2 RQ2: Comparison with Vision Language Models and Accessibility Issue Detectors

Our approach outperforms popular vision language models as well as accessibility issue detectors. As described in Section 4.1, we fed the same screenshot pairs collected from the subjects (Section 4.2 to four different vision language models [2, 7–9] and two accessibility issue detectors [3, 40] as described in Section 4.5.2. Then, we asked them for a verdict (inconsistent or consistent) and bounding boxes if the pair is inconsistent. As shown in Table 4, the accuracy of CHROMA EYES is higher than all four popular vision language models and the two state-of-the-art accessibility issue detectors, both for screenshot and application-wise.

Vision language models and accessibility issue detectors exhibited unstable performance when applied to the inconsistency detection task. For example, `grok-4.2-reasoning` [9] classified all screenshots as inconsistent, and thus, its precision and recall values for consistent pairs converged to zero. The axe DevTools [3] exhibited better overall performance than other tools, but the screenshot and application-wise accuracy values were limited to 58.65% and 82.31%, respectively.

In addition, our approach is more effective for identifying bounding boxes of inconsistent GUI elements as well as for inconsistency detection. For example, Figure 2 shows the bounding boxes indicating areas of inconsistent elements, identified by (a) CHROMA EYES and (b) `claude-opus-4` [2], respectively. After inspecting the results of all vision language models, we found **no** correct bounding boxes for inconsistent elements. In the case of the accessibility issue detectors, the bounding boxes were more stable than those of the vision language models, but their inconsistency detection performance was outperformed by CHROMA EYES.

Table 2. Detection results for each inconsistency types described in Section 2.2 and Figure 1.

Metric	Layout Inc.	Text Inc.	Object Inc.	Incomplete Conversion
Accuracy	0.9795	0.9782	0.9707	0.9965
Precision (Inconsistent)	0.9145	0.8642	0.7794	0.8913
Precision (Consistent)	1.0000	0.9983	0.9902	1.0000
Recall (Inconsistent)	1.0000	0.9896	0.8907	1.0000
Recall (Consistent)	0.9738	0.9765	0.9777	0.9965
F1-Score (Inconsistent)	0.9553	0.9227	0.8313	0.9425
F1-Score (Consistent)	0.9867	0.9873	0.9839	0.9982

These results are for 147 web applications with native mode conversion support described in Section 4.2.

Table 3. Ablation study results between “without vs. with” the two-step GUI widget element inconsistency detection process shown in Figure 6 and Line 8 in Algorithm 3.

Metric	Text Inconsistency without	Text Inconsistency with	Object Inconsistency without	Object Inconsistency with
Accuracy	0.9673	0.9782	0.8972	0.9707
Precision (Inconsistent)	0.8083	0.8642	0.5155	0.7794
Precision (Consistent)	0.9983	0.9983	0.9906	0.9902
Recall (Inconsistent)	0.9897	0.9896	0.9312	0.8907
Recall (Consistent)	0.9638	0.9765	0.8931	0.9777
F1-Score (Inconsistent)	0.8899	0.9227	0.6636	0.8313
F1-Score (Consistent)	0.9808	0.9873	0.9393	0.9839

Table 4. Detection results of CHROMAEYES, vision language models, and accessibility issue detectors.

Metric	CHROMAEYES		gpt-4o		gemini-2.5-pro		claude-opus-4		grok-4.2		OwlEye		axe DevTools	
	S-wise	A-wise	S-wise	A-wise	S-wise	A-wise	S-wise	A-wise	S-wise	A-wise	S-wise	A-wise	P-wise	A-wise
Accuracy	0.9619	0.9795	0.6061	0.8435	0.4843	0.8639	0.5761	0.8299	0.5965	0.8639	0.0129	0.8571	0.5865	0.8231
Precision (I)	0.9166	0.9765	0.5042	0.8551	0.4359	0.8639	0.4437	0.8550	0.4855	0.8897	0.0129	0.8571	0.5558	0.8984
Precision (C)	0.9975	1.0000	0.6738	0.0000	0.7969	0.0000	0.6404	0.4444	0.6704	0.5454	0.0000	0.0000	0.6982	0.3157
Recall (I)	0.9966	1.0000	0.5068	0.9841	0.9327	1.0000	0.3743	0.9593	0.4947	0.9603	1.0000	1.0000	0.8698	0.8984
Recall (C)	0.9382	0.8636	0.6715	0.0000	0.1794	0.0000	0.7036	0.1666	0.6621	0.2857	0.0000	0.0000	0.3022	0.3157
F1-Score (I)	0.9549	0.9881	0.5055	0.9151	0.5942	0.9270	0.4061	0.9042	0.4901	0.9236	0.0000	0.9230	0.6782	0.8984
F1-Score (C)	0.9669	0.9268	0.6726	0.0000	0.2929	0.0000	0.6705	0.2424	0.6662	0.3750	0.0000	0.0000	0.4218	0.3157

These results are for 147 web applications with native mode conversion support described in Section 4.2. “S-wise” and “A-wise” stand for *screenshot-wise* and *application-wise* performance, respectively. “I” and “C” indicate *Inconsistent* and *Consistent*, respectively. Specifically for axeDevTools [3], the column name “P-wise” is used since the tool works only for *page-wise* instead of *screenshot-wise*. The difference of the detection results between CHROMAEYES and VLMs/OwlEye/axeDevTools are statistically significant (by the McNemar’s test [43]).

Note that the detection performance results are significantly different between CHROMAEYES and baselines. The results are statistically tested by McNemar’s test [43]. As the detection is a binary classification, there are four different outcomes: (1) both CHROMAEYES and another tool *correctly* detect the (in)consistency between screenshot pairs, (2) both *incorrectly* detect, (3) only CHROMAEYES *incorrectly* detects, or (4) only another tool *incorrectly* detects. We tested screenshot-wise and application-wise results, and the p-values were all smaller than 0.01. Thus, the higher performance of CHROMAEYES shown in Table 4 is statistically significant.

Time Cost: We measured the runtime cost of CHROMAEYES and other tools. Our approach takes 70 seconds on average to automatically navigate one page and collect 10 pairs of screenshots. For inconsistency detection, CHROMAEYES takes 7.11 seconds per screenshot pair on average. This runtime is lower than that of the commercial VLM baselines, including GPT-4o (10.50s), Claude (13.54s), Gemini-2.5-Pro (24.69s), and Grok-4.20 (53.80s). Although accessibility detectors are faster, with OwlEye taking 0.42 seconds and axe DevTools taking 5.7 seconds per screenshot pair, they show substantially lower detection accuracy, as discussed above. These results indicate that CHROMAEYES provides a practical trade-off between detection accuracy and runtime cost.

Monetary Cost: We also analyzed the monetary cost of CHROMAEYES. The commercial API used by ChromaEyes is Upstage Document OCR, which costs \$0.0015 per image. Since our evaluation includes 2,009 screenshot pairs, CHROMAEYES processed 4,018 screenshots in total, resulting in an OCR API cost of \$6.027. This cost is substantially lower than the cost of applying commercial VLMs to the same screenshot pairs. Moreover, this cost can be further reduced or eliminated by replacing the commercial OCR service with open-source OCR models, provided that they achieve comparable detection accuracy. These results indicate that CHROMAEYES is practical not only in terms of detection accuracy and runtime, but also in terms of monetary cost.

Answer to RQ2: CHROMAEYES outperforms popular vision language models, such as gpt-4o [8], as well as state-of-the-art accessibility issue detectors such as OwlEye [40]. In addition, our approach is more effective not only for inconsistency detection but also for identifying the inconsistent areas (i.e., bounding boxes) than other tools.

5.3 RQ3: Inconsistency Detection when Extensions are Applied

CHROMAEYES appropriately works for 49 web applications with dark mode layouts converted by browser extensions. As listed in Table 5, our approach accurately detects inconsistencies with 98% screenshot-wise accuracy. The outcomes are (TP, TN, FP, FN) = (492, 41, 4, 2), which implies that they are highly vulnerable to making inconsistencies when automatically converting applications into dark mode.

Note that all the 49 ($=7 \times 7$) combinations contained at least one inconsistency as there were 494 inconsistencies ($=TP+FN=492+2$), which indicated a single combination had 10 inconsistencies on average. Indeed, all combinations had at least one. Our approach correctly detected at least one inconsistency in each combination. Thus, the application-wise accuracy is 100%. This implies that it is necessary to improve the reliability of the extensions.

Answer to RQ3: CHROMAEYES also successfully detects inconsistencies between the modes with 98.88% accuracy even if the dark mode is automatically generated by browser extensions.

6 Discussions

6.1 Reporting the Inconsistencies to the Developers

Table 5. Detection results of 49 combinations of 7 apps only with the light mode and 7 dark-mode conversion extensions described in Section 4.2.

Metric	Screenshot-wise
Accuracy	0.9888
Precision (Inconsistent)	0.9919
Precision (Consistent)	0.9534
Recall (Inconsistent)	0.9959
Recall (Consistent)	0.9111
F1-Score (Inconsistent)	0.9939
F1-Score (Consistent)	0.9318

As all combinations have at least one inconsistency (i.e., the number of true positives is 49), we do not report on the performance of application-wise results.

It was challenging to report the inconsistencies detected by CHROMAEYES to the developers who were in charge of the corresponding web applications. It is common to report glitch detection results in many previous studies [48, 49, 53]. Unfortunately, most of the web applications in our subjects listed in Section 4.2 have **no** public open source repositories where we can report issues, even though the applications are popular and active. Moreover, there was no contact information (e.g., email addresses) to report any issues with the web applications. Even when the information was available, we could not receive any response from the developers or administrators of the applications. Therefore, it is necessary to come up with a way to report the inconsistencies detected by automated techniques so that the developers can repair the problem immediately.

6.2 Sensitivity Analysis of Parameters

For the three parameters described in Section 4 (i.e., IoU, colDiff, and areaDiff), we additionally conducted a sensitivity analysis. As repeating the experiment with all screenshots as well as different parameters is time-consuming and takes too much effort, we first randomly sampled 323 screenshot pairs out of 2,009 screenshot pairs (i.e., 95% of confidence interval and 5% of error margin). Then, we varied the values of each parameter by $\pm 10\%$ and $\pm 20\%$, which implies that we compared the performance of CHROMAEYES with five different values for each parameter.

The analysis results show that the selected values perform best when compared with other adjacent values (i.e., $\pm 10\%$ and $\pm 20\%$). When IoU = 0.5, our approach achieves the best F1-scores both for consistent and inconsistent screenshot pairs. The F1-score of inconsistent pairs is less and that of consistent pairs is higher when colDiff = $\frac{64}{200}$ and $\frac{96}{200}$ (i.e., $\pm 20\%$ of $\frac{80}{200}$) while it is not different from colDiff = $\frac{80}{200}$ when colDiff = $\frac{72}{200}$ and $\frac{88}{200}$ (i.e., $\pm 10\%$ of $\frac{80}{200}$). As our focus is on detecting inconsistent screenshot pairs, we chose colDiff = $\frac{80}{200}$ in particular. The performance of CHROMAEYES is not different when varying the value of areaDiff. The detailed values and plots of this sensitivity analysis are available in our replication package [23].

6.3 False Positives and False Negatives

Although our approach outperforms large commercial VLMs and state-of-the-art accessibility issue detectors as shown in Table 4, it may incorrectly detect consistent or inconsistent screenshot pairs.

To identify false positives and false negatives by CHROMAEYES, we manually analyzed the detection results shown in Table 1. We found 54 pairs of false positives and two pairs of false negatives.

False positives are mostly due to dynamic elements and the limitations of object/text detection models. For example, the object detection model [58] used in our approach can inconsistently detect GUI elements in the screenshot pairs between light and dark modes, as shown in Figure 7(a). Note that the object detection model unnecessarily detects cubes and small objects in the dark mode, while it does not detect them in the light mode. Thus, it is detected as an inconsistent pair, but it is actually consistent.

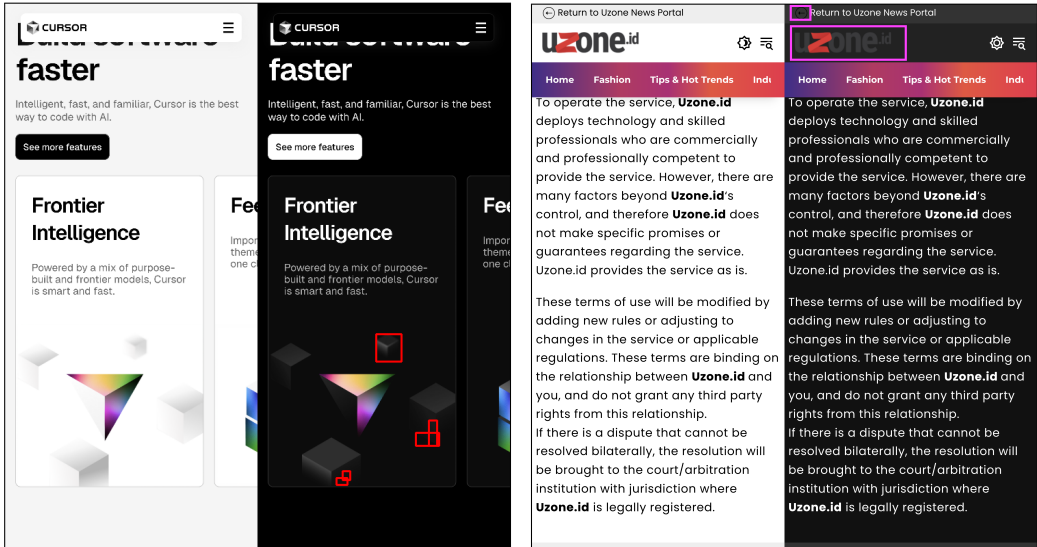
Our approach missed two false negative pairs, mainly due to subtle color shifts between the modes. As shown in the example of Figure 7(b), CHROMAEYES failed to detect the inconsistency on the top of the screenshot pairs, while the pair should be detected as an inconsistent pair since the icon and logo are not clearly decipherable. This false negative occurs because the text and object detection models [58] identified the low-visibility logos and icons as valid UI elements, causing CHROMAEYES to overlook their reduced decipherability due to poorer visual clarity.

6.4 Threats to Validity

Internal validity: Due to the limitation of vision models (i.e., edge, text, and object detection), our approach currently works correctly only for mobile-sized screenshots. To mitigate this limitation, we collected subjects that could correctly show layouts in small resolutions. Fortunately, most of the top-ranked applications were listed in the sources we searched [16, 19, 20].

External validity: Since there is no existing standard benchmark for inconsistencies between light and dark mode layouts yet, our approach is not evaluated on the subjects collected from other studies. This threat has been mitigated by collecting subjects from the sources [16, 19, 20] listing widely-visited applications.

Conclusion validity: In this study, we manually labeled the inconsistent and consistent screenshot pairs since no benchmark is available publicly. The labeling could be subject. To mitigate this issue, two of the authors independently labeled the screenshot pairs.



(a) False positive due to the limitation of the object detection model [58].

(b) False negative due to the sensitivity of the object detection model [58].

Fig. 7. Examples of false positives and false negatives identified by a manual analysis.

7 Related Work

The study of inconsistencies between light and dark modes in web applications is beginning to appear. Andrew et al. [26] conducted studies to gain insights into the implementation of light and dark modes across platforms and interviewed developers to understand how they typically create alternative models. Their goal was to come up with guidance and recommendations for better mode implementations. Ma et al. [42] performed an in-depth analysis of Android app dark modes to identify the prevalence of dark mode implementation across different categories of Android apps. These studies focus on comparing and analyzing various aspects of light and dark mode characteristics, whereas we specifically focus on detecting inconsistencies.

Related to the dark mode layouts, there have been some studies that primarily focused on readability, user behavior, and adaptability. For example, Koning et al. [39] analyzed the user behavior in dark mode. Palmen et al. [44] studied the effect of the variable font grade on the readability under light and dark modes. Similarly, Chen et al. [32] investigated the effect of font type, weight, and color on the readability of Chinese characters in dark modes. Another set of work explored visual fatigue in Optical See-Through Head-Mounted Displays [36, 38]. There also exist studies that assessed usability and productivity [45, 54].

There are several studies focusing on improving the GUI accessibility and design motivated by the increasing popularity of mobile and web applications. For instance, UIS-Hunter [59] is an automatic tool that detects 11 types of UI component violations based on Material Design guidelines. Xbot [33] uses data-flow analysis and app instrumentation to identify issues such as item labels and text contrast. Similarly, other studies [40, 41] propose tools to detect accessibility violations such as text overlap and missing images, while some studies [25, 61, 62] propose solutions to repair the GUI issues. Ryou and Ryu [47] present an approach to detecting layout inconsistencies of responsive web applications when resizing the application dynamically. These tools are designed for light-theme interfaces, offering limited support for dark-mode design.

8 Conclusion

In this work, we presented a novel approach, CHROMAEYES, to detecting inconsistencies between light and dark mode layouts of web applications. Although the inconsistencies have negative impacts on user experience, there has been no research effort to detect and fix those inconsistencies automatically. We showed that our approach effectively detects the inconsistencies of 2,009 screenshot pairs captured from 147 *real* web applications, with 96.19–97.95% accuracy. In addition, our approach outperforms four popular vision language models such as gpt-4o [8], as well as the state-of-the-art accessibility issue detectors such as OwlEye [40]. Our future work includes improving the accuracy of baseline text/object detection models, detecting inconsistencies in large screenshots, and an automated technique that creates “hot-fixes” to repair inconsistencies.

9 Data Availability

We make the replication package publicly available, which includes all the code and datasets to reproduce our experiments at <https://github.com/PALM-Group-Korea-University/ChromaEyes-ISSTA-2026> [23] as a repository and at <https://doi.org/10.5281/zenodo.21715790> [51] as an archived file.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP)-ICT Creative Consilience Program grant funded by the Korea government (MSIT)(IITP-2026-RS-2020-II201819), the National Research Foundation of Korea (NRF) grant funded

by the Korea government (MSIT)(RS-2025-02313526, RS-2026-25489826, and RS-2021-NR060080), and a Korea University Research Grant.

References

- [1] 2026. *AdBlock*. <https://getadblock.com/>
- [2] Anthropic 2025. *API*. Anthropic. <https://docs.anthropic.com/en/release-notes/api>
- [3] 2025. *Axe DevTools for Web Accessibility Testing Tools*. <https://chromewebstore.google.com/detail/axe-devtools-web-accessib/lhdoppojpmngadmndnejejpokejbdd>
- [4] 2025. *The color system*. <https://m2.material.io/design/color/the-color-system.html>
- [5] 2025. *Dark theme*. <https://m2.material.io/design/color/dark-theme.html>
- [6] 2025. *Does Dark Mode Actually Save Battery Life and Is More Eco-Friendly?* <https://bejamas.com/blog/does-dark-mode-save-battery>
- [7] 2025. *Google AI Studio*. <https://aistudio.google.com/>
- [8] 2025. *Models - OpenAI API*. <https://platform.openai.com>
- [9] 2025. *Models and Pricing | xAI Docs*. <https://docs.x.ai/docs/models>
- [10] 2025. *React*. <https://react.dev/>
- [11] 2025. *A Research-Oriented Top Sites Ranking Hardened against Manipulation - Tranco*. <https://tranco-list.eu/>
- [12] 2025. *Selenium*. <https://www.selenium.dev/>
- [13] 2025. *Understanding Intersection over Union for Model Accuracy*. <https://viso.ai/computer-vision/intersection-over-union-iou/>
- [14] 2021. *How Terra improved user engagement thanks to Dark Mode*. <https://web.dev/case-studies/terra-dark-mode?hl=en#:~:text=At%20the%20time%20of%20writing%20%28December%202021%29%2C%20Chrome%20Platform%20Status%20determines%20that%20approximately%2022%25%20of%20the%20web%20traffic%20comes%20from%20users%20with%20the%2022prefer%20dark%22%20setting> [Online; accessed 11. Mar. 2025].
- [15] 2025. *35+ Dark mode statistics you need to know (2025)*. <https://forms.app/en/blog/dark-mode-statistics> [Online; accessed 11. Mar. 2025].
- [16] 2025. *Alexa Top 1 Million Sites*. <https://www.kaggle.com/datasets/cheedcheed/top1m>
- [17] 2025. *How Many People Use Dark Mode in 2025? (Key Statistics)*. <https://thesmallbusinessblog.com/dark-mode-users/> [Online; accessed 11. Mar. 2025].
- [18] 2025. *Implement dark theme*. <https://developer.android.com/develop/ui/views/theming/darktheme> [Online; accessed 11. Mar. 2025].
- [19] 2025. *Top 1000 Websites By Ranking Keywords – DataForSEO*. <https://dataforseo.com/free-seo-stats/top-1000-websites>
- [20] Moz 2025. *Top 500 Most Popular Websites*. Moz. <https://moz.com/top500>
- [21] 2025-03-13. *OpenCV: Canny Edge Detection*. https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
- [22] 2025-03-13. *Upstage Console*. <https://console.upstage.ai>
- [23] 2026. *Replication package of ChromaEyes*. <https://github.com/PALM-Group-Korea-University/ChromaEyes-ISSTA-2026>
- [24] Raj Adhav. 2024. *Does Dark Mode Improve The User Experience?* <https://designial.com/blogs/does-dark-mode-improve-the-user-experience/>
- [25] Ali S. Alotaibi, Paul T. Chiou, Fazle M. Tawsif, and William G. J. Halfond. 2023-10. *ScaleFix: An Automated Repair of UI Scaling Accessibility Issues in Android Applications*. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (Bogotá, Colombia). 147–159. doi:10.1109/ICSME58846.2023.00025
- [26] Sarah Andrew, Chelsea Bishop, and Garreth W. Tigwell. 2024. *Light and Dark Mode: A Comparison Between Android and iOS App UI Modes and Interviews with App Designers and Developers*. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 8, 1 (March 2024), 1–23. doi:10.1145/3643539
- [27] Apple Inc. 2025. *Get Ready for Dark Mode - Latest News*. <https://developer.apple.com/news/?id=0829019a> [Online; accessed 2025-01-28].
- [28] Apple Newsroom. 2025. *Apple Introduces macOS Mojave*. <https://www.apple.com/newsroom/2018/06/apple-introduces-macos-mojave> [Online; accessed 2025-01-28].
- [29] Florian Bordes, Richard Yuanzhe Pang, Anurag Ajay, Alexander C. Li, Adrien Bardes, Suzanne Petryk, Oscar Mañas, Zhiqui Lin, Anas Mahmoud, Bargav Jayaraman, Mark Ibrahim, Melissa Hall, Yunyang Xiong, Jonathan Lebensold, Candace Ross, Srihari Jayakumar, Chuan Guo, Diane Bouchacourt, Haider Al-Tahan, Karthik Padthe, Vasu Sharma, Hu Xu, Xiaoqing Ellen Tan, Megan Richards, Samuel Lavoie, Pietro Astolfi, Reyhane Askari Hemmat, Jun Chen, Kushal Tirumala, Rim Assouel, Mazda Moayeri, Arjang Talattof, Kamalika Chaudhuri, Zechun Liu, Xilun Chen, Quentin Garrido, Karen Ullrich, Aishwarya Agrawal, Kate Saenko, Asli Celikyilmaz, and Vikas Chandra. 2024. *An Introduction to Vision-Language Modeling*. doi:10.48550/arXiv.2405.17247 arXiv:2405.17247 [cs]

- [30] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models Are Few-Shot Learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, 1877–1901.
- [31] Aaron Carroll and Gernot Heiser. 2010. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference (USA)*. USENIX Association.
- [32] Junxiang Chen, Pengyu Zhu, Jing Yun, Baixuan Tian, Yalan Yang, and Na Liu. 2023-07-23. Text Readability of Smartphone in Dark Mode: Effects of Font Type, Font Weight and Color. In *Human Aspects of IT for the Aged Population: 9th International Conference, ITAP 2023, Held as Part of the 25th HCI International Conference, HCII 2023, Copenhagen, Denmark, July 23–28, 2023, Proceedings, Part I* (Berlin, Heidelberg). Springer-Verlag. doi:10.1007/978-3-031-34866-2_2
- [33] Sen Chen, Chunyang Chen, Lingling Fan, Mingming Fan, Xian Zhan, and Yang Liu. 2022. Accessible or Not? An Empirical Investigation of Android App Accessibility. *IEEE Transactions on Software Engineering* 48, 10 (2022), 3954–3968. doi:10.1109/TSE.2021.3108162
- [34] Contributors to Wikimedia projects. 2025. Android 10 - Wikipedia. https://en.wikipedia.org/w/index.php?title=Android_10&oldid=1259097175
- [35] Pranab Dash and Y. Charlie Hu. 2025. How Much Battery Does Dark Mode Save? An Accurate OLED Display Power Profiler for Modern Smartphones. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2021-06-24). Association for Computing Machinery. doi:10.1145/3458864.3467682
- [36] Austin Erickson, Kangsoo Kim, Alexis Lambert, Gerd Bruder, Michael P. Browne, and Gregory F. Welch. 2021-05-20. An Extended Analysis on the Benefits of Dark Mode User Interfaces in Optical See-Through Head-Mounted Displays. 18 (2021-05-20), 12:1–12:22. doi:10.1145/3456874
- [37] Jindong Gu, Zhen Han, Shuo Chen, Ahmad Beirami, Bailan He, Gengyuan Zhang, Ruotong Liao, Yao Qin, Volker Tresp, and Philip Torr. 2023. A Systematic Survey of Prompt Engineering on Vision-Language Foundation Models. doi:10.48550/arXiv.2307.12980 arXiv:2307.12980 [cs]
- [38] Kangsoo Kim, Austin Erickson, Alexis Lambert, Gerd Bruder, and Greg Welch. 2019-10-19. Effects of Dark Mode on Visual Fatigue and Acuity in Optical See-Through Head-Mounted Displays. In *Symposium on Spatial User Interaction* (New York, NY, USA). Association for Computing Machinery. doi:10.1145/3357251.3357584
- [39] Luka Koning and Marianne Junger. 2021. Dark user interface, dark behavior? The effect of ‘dark mode’ on honesty. *Computers in Human Behavior Reports* 4 (Aug. 2021), 100107. doi:10.1016/j.chbr.2021.100107
- [40] Zhe Liu, Chunyang Chen, Junjie Wang, Yuekai Huang, Jun Hu, and Qing Wang. 2021. Owl eyes: spotting UI display issues via visual understanding (ASE ’20). Association for Computing Machinery, New York, NY, USA, 398–409. doi:10.1145/3324884.3416547
- [41] Zhe Liu, Chunyang Chen, Junjie Wang, Yuekai Huang, Jun Hu, and Qing Wang. 2023. Nighthawk: Fully Automated Localizing UI Display Issues via Visual Understanding. *IEEE Transactions on Software Engineering* 49, 1 (2023), 403–418. doi:10.1109/TSE.2022.3150876
- [42] Suyu Ma, Chunyang Chen, Hourieh Khalajzadeh, and John Grundy. 2023. A First Look at Dark Mode in Real-world Android Apps. *ACM Trans. Software Eng. Method.* 33, 1 (Nov. 2023), 1–26. doi:10.1145/3604607
- [43] Quinn McNemar. 1947. Note on the Sampling Error of the Difference between Correlated Proportions or Percentages. *Psychometrika* 12, 2 (June 1947), 153–157. doi:10.1007/BF02295996
- [44] Hilary Palmén, Michael Gilbert, and David Crossland. 2023. How bold can we be? The impact of adjusting font grade on readability in light and dark polarities. In *ACM Conferences*. Association for Computing Machinery, New York, NY, USA, 1–11. doi:10.1145/3544548.3581552
- [45] Lasse Apalnes Pedersen, Svavar Skuli Einarsson, Fredrik Arne Rikheim, and Frode Eika Sandnes. 2020-07-19. User Interfaces in Dark Mode During Daytime – Improved Productivity or Just Cool-Looking?. In *Universal Access in Human-Computer Interaction. Design Approaches and Supporting Technologies: 14th International Conference, UAHCI 2020, Held as Part of the 22nd HCI International Conference, HCII 2020, Copenhagen, Denmark, July 19–24, 2020, Proceedings, Part I* (Berlin, Heidelberg). Springer-Verlag, 178–187. doi:10.1007/978-3-030-49282-3_13
- [46] Laria Reynolds and Kyle McDonell. 2021. Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems (CHI EA '21)*. Association for Computing Machinery, New York, NY, USA, 1–7. doi:10.1145/3411763.3451760
- [47] Yeonhee Ryou and Sukyoung Ryu. 2018. Automatic Detection of Visibility Faults by Layout Changes in HTML5 Web Pages. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)* (Västerås, Sweden). 182–192. doi:10.1109/ICST.2018.00027

- [48] Arooba Shahoor, Askar Yeltayuli Khamit, Jooyong Yi, and Dongsun Kim. 2023. LeakPair: Proactive Repairing of Memory Leaks in Single Page Web Applications. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1175–1187. doi:10.1109/ASE56229.2023.00097
- [49] Arooba Shahoor, Jooyong Yi, and Dongsun Kim. 2024. Preserving Reactiveness: Understanding and Improving the Debugging Practice of Blocking–Call Bugs. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 768–780. doi:10.1145/3650212.3680319
- [50] Thomas Steiner. 2019. *Let There Be Darkness! Maybe...* <https://medium.com/dev-channel/let-there-be-darkness-maybe-9facd9c3023d>
- [51] Embracing the Dark Side: Detecting, Repairing Inconsistencies between Light, and Dark Modes of Web Applications. 2026. *Code and data*. <https://doi.org/10.5281/zenodo.21715790>
- [52] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. doi:10.48550/arXiv.2302.13971 arXiv:2302.13971 [cs.CL]
- [53] John Vilk and Emery D. Berger. 2018. BLeak: Automatically Debugging Memory Leaks in Web Applications. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2018)*. Association for Computing Machinery, New York, NY, USA, 15–29. doi:10.1145/3192366.3192376
- [54] Mindaugas Vinciūnas and Kęstutis Kapočius. 2023. Impact of Dark and Light Graphical User Interface Modes on System Usability: Preliminary Findings of an Experimental Study /. 3575 (2023), 141–151. <https://epubl.ktu.edu/object/elaba:184888793/index.html>
- [55] Julius Virtanen. 2023-11-29. Dark Mode Preferences: Exploring User Motivations in Interface Theme Selection. (2023-11-29). <https://www.utupub.fi/handle/10024/176173>
- [56] Chris Welch. 2018. *Google Confirms Dark Mode Is a Huge Help for Battery Life on Android*. <https://www.theverge.com/2018/11/8/18076502/google-dark-mode-android-battery-life>
- [57] Terry Winograd. 1995-06-01. From Programming Environments to Environments for Designing. 38, 6 (1995-06-01), 65–74. doi:10.1145/203241.203259
- [58] Mulong Xie, Sidong Feng, Zhenchang Xing, Jieshan Chen, and Chunyang Chen. 2020. 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* (New York, NY, USA). Association for Computing Machinery. doi:10.1145/3368089.3417940
- [59] Bo Yang, Zhenchang Xing, Xin Xia, Chunyang Chen, Deheng Ye, and Shanping Li. 2021. Don't Do That! Hunting Down Visual Design Smells in Complex UIs Against Design Guidelines. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 761–772. doi:10.1109/ICSE43902.2021.00075
- [60] Amy Zhang and Tanner Kohler. 2025. Dark Mode: How Users Think About It and Issues to Avoid. *Nielsen Norman Group* (2025). <https://www.nngroup.com/articles/dark-mode-users-issues>
- [61] Mengxi Zhang, Huaxiao Liu, Chunyang Chen, Guangyong Gao, Han Li, and Jian Zhao. 2024-02. AccessFixer: Enhancing GUI Accessibility for Low Vision Users With R-GCN Model. 50 (2024-02), 173–189. doi:10.1109/TSE.2023.3337421
- [62] Yuxin Zhang, Sen Chen, Lingling Fan, Chunyang Chen, and Xiaohong Li. 2023. Automated and Context-Aware Repair of Color-Related Accessibility Issues for Android Apps. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (New York, NY, USA) (ESEC/FSE 2023). Association for Computing Machinery. doi:10.1145/3611643.3616329

Received 2026-01-30; accepted 2026-06-25