

Adaptive Dark Mode for Improved OLED Viewing Comfort for Web Applications

Sang Suh^{1,*} and Tommy P. Hoang²⁾

^{1,2)}Department of Computer Science, Texas A&M University - Commerce,
Commerce, Texas

Abstract. OLED screens require more energy than most other components of a device. Since OLED screens use less energy when displaying dark displays (i.e., dark images, videos, backgrounds), dark mode options found in web applications are helpful in reducing energy consumption from OLED displays. Dark mode options are difficult forms of optimization as the option may provide a better feel, but the images and other static icons show more difficult contrasts. Previous studies have tackled this user experience problem with experimental dark mode styles; however, hardware and visual appeal limits how well dark mode functions for certain tasks. In this paper, we tackle the problems in dark mode options that cause contrast issues, which will, consequently, improve battery longevity in OLED devices.

Keywords. OLED, dark mode, battery

1. Introduction

Organic Light-Emitting Diode (OLED) screens have been made popular in portable devices as OLED screens are best used in applications that do not display static images for long periods of time. Thanks to this, attractive features like high efficiency, low power consumption, fast switching, wide viewing angle, lightweight and flexibility of these OLED panels make excellent screens or lighting panels [2]. In devices, there are normally functions that will darken the brightness and turn off the screen during idle use. This is known as Wake-locks and it is to prevent a variety of problems from regular use, such as burn-in and battery drain. Although the screen may turn off, it does not render the screen unusable as a touch will bring the screen back to life. OLED screens behave differently in power consumption because of the way they operate as compared to normal Liquid Crystal Display (LCD) screens which require backlighting. OLED

* Corresponding author: Sang.Suh@tamuc.edu

Received: Sep 11, 2023; Accepted: Oct 19, 2023; Published: Dec 31, 2023

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (<http://creativecommons.org/licenses/by-nc/3.0/>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

screens show the standard three colors of red, green, and blue per pixel [1]. Depending on the image that needs to be displayed, the diodes will light up to adjust the color, and the allure of OLED is that dark colors require less energy to operate because the diodes do not have to light up to display darker colors [10].

The popularity of dark mode options has been made apparent as more web applications have been implementing them into their settings. The benefits of dark mode on web applications appeal to OLED screen users as darker screens pertain to less strain to the eyes and keep users engaged in the application they are using. Inherently, dark mode applications will consume less power as OLED screens do not have to light up as brightly in comparison to lighter or a variety of light colors which require the diodes to match the colors displayed. However, dark mode applications bring a challenge of finding appealing contrast to colors in dark backgrounds that appear in the same aesthetics of the original web page. Mobile devices have technologically developed exponentially with application development platforms combined with hardware components such as a camera and GPS. Unfortunately, the benefits of a smartphone are heavily limited by its battery longevity [9]. The size of the battery in mobile devices at most takes up to 90% of a device's interior [3]. In connection to OLED screens' power consumption with the inefficiency of battery life, the power consumed could be placed in more purposeful activities such as app usage. However, power monitoring applications don't normally self-monitor its own energy consumption and could disrupt a phone's performance. In the beginning, it is mentioned that web applications (websites or mobile applications) will consume more battery power to allow OLED screens to display light colors, such as white. White is also the normal background color for many websites or applications to emphasize clarity and enhance spatial use. While white is an excellent design choice, it is also the most power-hungry color, which is optionally solved with dark mode settings that companies implement for better health preservation.

Energy optimization is a problem in Search-Based Software Engineering (SBSE) that lacks work on non-functional properties while the importance of non-functional properties continues to increase. Testing for execution time, quality of service, and energy consumption are problems that branch from inherent issues of device optimization and cover different branches of these problems. Thus, the use of dark mode applications for mobile devices could increase battery life thanks to the low powering effects of dark colors from OLEDs. Granularity is a term used to describe the measurement of energy consumption based on the execution time of lines of code. There are three types of granularities: coarse-grained, mid-grained, and fine-grained. Course granularity is the measurement of the whole program's execution time. Mid granularity measures the time it takes to execute a method or function. Fine-grained is the time execution of a per-line of code basis [3].

2. Related Works

Nyx is a Java-based solution to convert web applications from light or native color to its absolute contrast, and the most renowned piece of software developed in 2014. The developers tackled problems based on their methods:

- How much time does the software take to generate the Color Transformation Scheme (CTS)?
- How much energy is saved by the transformed web pages?
- What is the runtime overhead introduced into the modified web applications by the software?
- To what degree do users accept the appearance of the transformed web pages?

A recent study of color contrasts (see Figure 1) shows the viability of color use in web design. In Li's demonstration of Comparison in Questionnaires (Li et al. Figure 5 2014), yellow was used as a foreground color for a black background with green as the CSS label color. While effective, the inherent issue with using bright colors aside from white is shown to be visually unappealing in comparison to the original webpage. The results of the questionnaire showed that users will only use the program mostly when their phone is low on battery, thus conserving battery power only when absolutely needed. Contrast when applied to color selection must stay mindful of brand standards to websites that must stay consistent to an organization's brand guidelines [4]. While Nyx shows to successfully implement a dark mode that is fully optimized to reduce battery drain, it is also not appealing to company or user standards.

		Background								
		Red	Orange	Yellow	Green	Blue	Violet	Black	White	Gray
Foreground	Red	Red	Poor	Good	Poor	Poor	Poor	Good	Good	Poor
	Orange	Poor	Orange	Poor	Poor	Poor	Poor	Good	Poor	Poor
	Yellow	Good	Good	Yellow	Poor	Good	Poor	Good	Poor	Good
	Green	Poor	Poor	Poor	Green	Good	Poor	Good	Poor	Good
	Blue	Poor	Poor	Good	Good	Blue	Poor	Poor	Good	Poor
	Violet	Poor	Poor	Good	Poor	Poor	Violet	Good	Good	Poor
	Black	Poor	Good	Good	Good	Poor	Good	Black	Good	Poor
	White	Good	Good	Good	Poor	Good	Good	Good	White	Good
	Gray	Poor	Poor	Good	Good	Poor	Poor	Poor	Good	Gray

Figure 1. Jeremy Girard / Livewires 2021

A web browser, Opera GX, implemented “force dark pages” that can be accessed through the settings. This toggles major websites to convert gray or any high-scaled grays or whites to be converted into black or pure black colors and clarifies symbols and

JITA

icons more sharply to enhance icon/symbol recognition. In most cases, buttons and other CSS elements disappear, only showing clickable text, reducing clutter while keeping to the website's format or structure. While this is an excellent method of conversion, it is in beta to this date, and mainly works on websites with monochromatic backgrounds.

Chameleon, a color-adaptive web browser, follows the same principle as both Nyx and Opera GX. Instead of a toggle, Chameleon monitors and adapts the browser's color and web pages based on the user's mobile battery percentage [7].

Using a color selection algorithm, Multi-Object GUI Applications in OLED-Mobile Devices devised a program that alters the application's appearance based on the user's preference of color. Their aim is to minimize the color difference between other elements in the graphical user interface (GUI)[11].

A social platform, Discord, has an experimental feature on Android devices called "AMOLED mode." This experimental feature is a hidden option in settings that can be accessed by pressing the Dark Mode option ten times. This feature replaces the default dark Discord mode with a pure black background wherever possible to conserve battery consumption. Unfortunately, due to the experimental nature of this research, Discord and other sources do not have any sort of official documentation for their dark mode feature. flux, a renowned software, is a program that decreases the amount of blue light being received from monitors by adjusting screen color. Unlike dark mode options, flux covers all programs by changing the screen's color temperature based on location time. The research provided and gaining popularity of blue-light defense [5] enhances the appeal of utilizing software to reduce eye strain from long hours of screen time. The issue with flux is the incandescent color covers the entire screen. While that may be overall beneficial, images and videos that users would like to see in proper color may prefer to opt out for a better viewing experience. Aside from the beneficial nature of using warm colors to prevent disruption of a person's circadian rhythm [5], the use only applies when staring at a screen for extended periods of time.

3. Enhanced Dark Mode Methodology

The purpose of the experiment is to create a dark mode program, Dark Eyes JS, that will work on all websites. More specifically, the program will enhance a dark mode that is lightweight, efficient, will appeal to company brand standards, and enhance user experience. In many cases, websites use a monochromatic background color, but most options do not work on colored backgrounds.

In simplistic cases, changing from a light or default theme to a dark theme is trivial. As web applications need to rewrite CSS color tags from light, high-valued Red-Green-

Blue (RGB) colors to dark, low-valued RGB colors. The problem lies in utilizing brand colors. Since many elements of a website contain much more than a handful of colors than meets the eye, the program must extract all colors from the webpage that the user is viewing and convert every color into a darker version of it or swap the contrast while staying in the same hue.

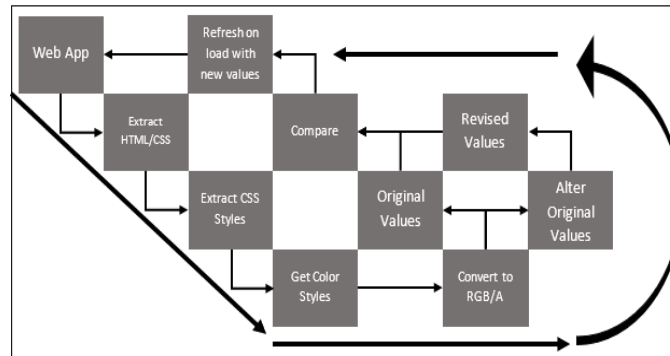


Figure 2. Program Function Methodology

In Figure 2, the methodology of this program will follow a round-trip cycle to implement a quick and efficient method of replacing website colors with revised colors.

Dark mode options provided by web applications are known to be more power-efficient in OLED devices, as OLED does not require lighting up brightly for dark colors. The problem is that most dark mode applications have clashing contrasts that may ruin picture quality or wash out icons. Some devices also contain a night mode, but this option forces the device itself to show monochromatic color schemes. This program will create an adaptive form of dark mode that changes the same color to a dark variant to all elements within the website. Starting with the methods, the approach will take three phases:

- First phase implements extraction of all HTML/CSS elements, then extracting down to each tag's styles, then collecting every color and adding them to an array.
- Second phase converting all colors to one color type. Since JavaScript utilizes multiple forms of color naming conventions, it would be best practice to convert them into one color type. Since RGB is most renowned in most programming languages, the conversion will result in RGB.
- Third phase is the revision of the original colors to create an array of dark mode colors.

- Fourth phase will then compare the original array with each element in the web page, and replace it with the revised color.
- Fifth phase this final phase is to reload the page with all element colors replaced with its dark mode variant.

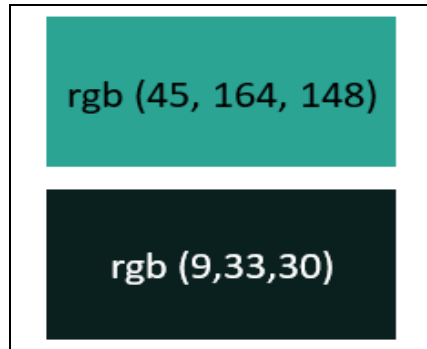


Figure 3. Comparison in Colors

The expected output of the project is a website that uses much darker colors than the original website without compromising on brand recognition, and eye strain from high-contrasting colors, plus the added benefit of readability through high saturation difference in background color, text, and other elements.

By modifying CSS tags to replace certain areas of web page colors with dark colors for battery conservation, the focus is user satisfaction using color contrasts, like in Li's research. My method utilizes the original website's colors as a form of numerical basis for RGB values. For example, a white background would have an RGB value of (255, 255, 255). For a dark mode, the value would be reduced by factors based on the percentage of what each value represents. For a dark mode that is not purely black, the 255 value should reach near zero, but not exactly zero. Figure 3 is an example of how the reduction of RGB values will show a drastic change in color brightness as the values reduce closer to zero. The value of the top rectangle is reduced by 80% to provide the dark-green color beneath it. This shows an excellent choice of color vibrancy in comparison the monotonic colors of black and white background and color. While numbers can be tweaked, the most noticeable appeal benefits from a 70% - 80% value reduction. When checking these color contrasts between the example in Figure 3 and previous works, this method does reduce contrast by a significant amount, but not to extreme contrast. The top rectangle of Figure 3 scored a 6.84 on the contrast scale, and the bottom half scored an astounding 16.82. While color contrast is mainly based on design choice, difference in contrast values will provide enhanced readability when colors are paired in higher differences in values.

4. Implementation

In previous studies, Java is the main choice language for implementation. While Java is a highly versatile programming language, its best use is in back-end development whereas the focus of this experiment will be front-end development or design. JavaScript is the choice for the software as it already contains the standard library of tools such as a color database native to JavaScript.

4.1 HTML/CSS Extraction

```
1· for i in elements{
2·   styles = style of elements[i];
3·   for j in styles{
4·     CSSName = styles[j];
5·     CSSValue = styles[CSSName];
6·     if(!CSSValue) continue;
7·     //Convert to string to avoid mismatch exception error
8·     CSSValue +="";
9·     ValueColor = CSSValue.match(Regex);
10·    if(!ValueColor) continue;
11·  }
12· }
```

Figure 4. HTML/CSS Extraction Pseudocode

The first part of the process is the retrieve CSS data from a web application. Using JavaScript to extract HTML and CSS elements, the colors of every element could be extracted through the computed styles of every element within the web page. By manipulating the DOM of a webpage, one can retrieve all information of its HTML code, like how inspecting the element from a browser's inspection tool can retrieve significant data from a website. While the HTML is gathered, a loop is required for all nested descriptions of a tag. From HTML, to CSS, to CSS styles, the color tag and its information can be retrieved and added to an array. In Figure 4, the method of extraction requires a form of nested loops, which does hinder on execution time. Due to JavaScript's versatility in color types, a regular expression is needed to identify if a color value is legitimate. JavaScript colors can vary from a list of:

- Color names
- RGB values
- HSL
- Hexadecimal

- Alpha values

The regular expression identifies what color type the website utilizes to match with whether the value of the CSS is a color.

4.2 *Converting All Types to RGB(A)*

The next problem in the program is color conversion. Since colors can come from a list of types, the value *RGB* and *RGBA* seems to be the most viable in this circumstance, as it is the better-known color type in programming, as it is also versatile in other programming languages.

The difference between *RGB* and *RGBA* is that the “A”, standing for “Alpha”, is the modifier for color transparency. This will not be an issue to the process as the element requiring the color’s transparency is an intended design to the website, much like choice of color; to change the alpha of the color can also affect the visibility of any contained elements.

For determining the value of each red, green, and blue; all color types do contain specific numerical values that dictate the hue of the element. Hexadecimal is a six-digit hex code from 0 to f in which two digits can range between 0 to 255 of an RGB color scale.

Hue, Saturation, Lightness (HSL); acts as a color wheel with hue values from 0 to 360 and saturation and lightness having percentage values. Conversion between HSL to RGB requires a formula where chroma (color intensity) is calculated by finding a percent difference. The next component is the saturation and lightness calculation based from a modifier calculated by the chroma and hue. While one factor will stay at a constant value of 0, the last portion is to multiply each rgb value (plus the modifier) by 255 and rounding the values. Since each color in the HSL color wheel is representative of red, green, blue, yellow, magenta, and cyan; each segment is divided into 60-degree angles of the color wheel.

$$chroma = (1 - \text{Math.abs}(2 * lightness - 1)) * saturation \quad (1)$$

$$temp = chroma * (1 - \text{Math.abs}((hue/60)\%2 - 1)) \quad (2)$$

$$modifier = lightness - chroma/2 \quad (3)$$

Color names contain an innate hex value for each name, so converting color names would first start from the name, to hex, then just converting hex to RGB.

4.3 Calculating to Dark Mode Values

Once the program receives all HTML/CSS elements, searches for all color values, then converts all color types into RGB/RGBA, the next step is to convert the values into a darker or lighter color depending on the threshold to flip the color brightness. The method requires passing each element of the RGB array into the color conversion with a percentage factor that will modify each RGB value based on the percentage value. The value best chosen is between 70% to 80% to maintain the hue of the color, but darkening/brightening based on the positive or negative value of the percentages.

To determine the threshold used to flip the colors, the program adds each value of RGB and depending on whether the value is greater or smaller than the threshold value will allow which color gets flipped. For example, RGB(255, 0, 0): Red, will flip to a dark red of RGB(51, 0, 0) if the threshold is equal to or higher than 255.

4.4 Comparison of Colors for Injection

The updated colors are stored into an array, which are automatically sorted to the same indexes as the original color array. The program will search through the HTML file and CSS elements again, then check each value of those original values.

```
1 for all index in originalColorArray{
2   if searched value === originalColorArray[index]
3     searchedValue = revisedColor[index];
4 }
```

Figure 5. Color Replacement Algorithm

Figure 5 shows the concept as the program will search through each CSS color value. The simple issue of refreshing edited local files is that refreshing web pages will revert edited files to the original server files. To combat this, the program is set to execute on load of the web page.

4.5 Loading Webpage with New Values

After all is done, the last portion is to replace all original values of a website with the revised values. At the beginning, the program added the original values to an array, which is automatically sorted based on order of insert, the revised values follow the same

sorted principle. Thus, an additional nested loop is required to search out and match with the original colors and set the page's original color with the revised color.

To minimize execution time, an event listener is used to create an on-load effect that executes the script the moment the page refreshes. The importance of execution time is emphasized by Google research, "53% of visits are abandoned of a mobile site takes longer than 3 seconds to load." While there are several factors that could play into website load times such as unoptimized images or network complications, any longer load times no matter the appeal, a website will lose its viewability if it does not load within those 3 seconds [6].

5. Results

This experiment is conducted on 1 LG Wing, with a timer on a separate device. Using a static image of both the original web page application and edited color of the output, a timer is set to count down until the phone battery drops by 1% starting at full charge (100%). To calculate the efficiency of the experiment, a static image of the original webpage is displayed until the phone battery drops to 99%, then the time is recorded. Next the phone is charged again until full and the image of the revised webpage is displayed. The timer starts immediately after the charger is unplugged from the device for both instances.

Table 1. Results of the Experiment

Name	Execution Time	Display Power Consumption
Nyx	90 - 180 seconds	25% - 40%
Chameleon	17 - 66 ms	37% - 67%
Multi-Object GUI	N/A	27.3%
DarkEyesJS - Original	N/A	~29 mins / 1%
DarkEyesJS - Revised	30 - 45 ms	~32 mins / 1%

To combat against Wake-lock, an automatic (often optional) mobile device feature that turns off the display to conserve battery [8], the screen of the LG Wing is tapped once every 5 to 10 minutes.

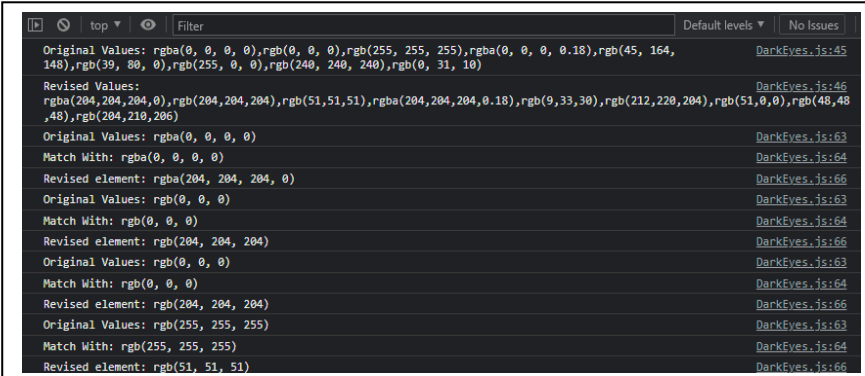
JITA

The results are shown in Table 1. In prior research of similar topics, it was mentioned that they achieved X amount of display power consumption, but never mentioned the rate in which the battery drained. During execution time, JavaScript used an event listener which executes the script as soon as the page is in the process of refreshing, thus achieving an average of 37ms for execution time. In comparison to the other two, Chameleon had an execution time between 17 and 66 milliseconds. Thus, the mean of execution time for Chameleon is 41.5 milliseconds. In the display power consumption, the battery life improved by 3 minutes per percent that is reduced. This is originally miniscule at first, but with its linear progression, that would be a rate of +3 minutes for every percentage. 10% of battery used would equal 30 minutes of saved energy.

6. Conclusion & Future Works

With the popularity of OLED screens, the problem arises on the issue of battery consumption on these beautiful displays that are fast and efficient. To tackle this problem several research on OLED screen energy optimization, using a variety of conducted experiments and software designs, many have achieved a great deal of reduction in battery usage. With very few sources relating to web application color optimization, front-end design on energy saving has ingrained the idea of maximum contrast for energy savings. Instead of furthering the idea of maximum contrast, this research is conducted with the focus on user appeal and branding standards by converting values that keep the same hue.

Figure 6. Console Output of Dar key esJS



```

Original Values: rgba(0, 0, 0, 0),rgb(0, 0, 0),rgb(255, 255, 255),rgba(0, 0, 0, 0.18),rgb(45, 164, 148),rgb(39, 80, 0),rgb(255, 0, 0),rgb(240, 240, 240),rgb(0, 31, 10) DarkEyes.js:45
Revised Values: DarkEyes.js:46
rgba(204, 204, 204, 0),rgb(204, 204, 204),rgb(51, 51, 51),rgba(204, 204, 204, 0.18),rgb(9, 33, 30),rgb(212, 220, 204),rgb(51, 0, 0),rgb(48, 48, 48),rgb(204, 210, 206)
Original Values: rgba(0, 0, 0, 0) DarkEyes.js:63
Match With: rgba(0, 0, 0, 0) DarkEyes.js:64
Revised element: rgba(204, 204, 204, 0) DarkEyes.js:66
Original Values: rgb(0, 0, 0) DarkEyes.js:63
Match With: rgb(0, 0, 0) DarkEyes.js:64
Revised element: rgb(204, 204, 204) DarkEyes.js:66
Original Values: rgb(0, 0, 0) DarkEyes.js:63
Match With: rgb(0, 0, 0) DarkEyes.js:64
Revised element: rgb(204, 204, 204) DarkEyes.js:66
Original Values: rgb(255, 255, 255) DarkEyes.js:63
Match With: rgb(255, 255, 255) DarkEyes.js:64
Revised element: rgb(51, 51, 51) DarkEyes.js:66

```

Dark Eyes JS is a script that can extract any website's color into an array of RGB values and revise those values to create an array of edited values of light-to-dark colors or vice versa. The program is currently unable to replace background colors due to limitations in the JavaScript language. While `getComputedStyle()` is excellent at

collecting all CSS descriptions, it is a read-only output. In consequence, the program was able to collect all color values, but it fails to differentiate between CSS tags of color and background-color. Due to this limitation, the background color of the test webpage did not change, and the experiment required a theoretical output instead while the colors of all other tags changed successfully. Based off the color array and the revised color array, values are calculated and printed in the console log. In Figure 6, the indexes for both arrays are sorted in a way that each index corresponds to the updated color.

As is, the program can collect all colors of a web page and output as an array, like `alwane.io`; a website that takes a website URL and outputs color palates in like-color order. `DarkEyesJS` can also send the revised array as output as dark mode color suggestions. In future works, if it was capable to tag the background color in the array, to be able to specifically convert to the revised background color, the next thing left would be to create this program to not require embedding in HTML files to execute properly, but rather as a standalone browser plugin. Which can be injected into any webpage.

References

- [1] Li, Ding, et al. "Making Web Applications More Energy Efficient for OLED Smartphones ..." *Making Web Applications More Energy Efficient for OLED Smartphones*, University of Southern California, 2014.
- [2] Sudheendran Swayamprabha, Sujith, et al. "Approaches for Long Lifetime Organic Light Emitting Diodes." *Advanced Science (Weinheim, Baden-Wurtemberg, Germany)*, John Wiley and Sons Inc., 12 Nov. 2020, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7788592/>.
- [3] Harman, Mark, et al. "Achievements, Open Problems and Challenges for Search Based Software Testing." *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, 2015, <https://doi.org/10.1109/icst.2015.7102580>.
- [4] Girard, Jeremy. "How to Choose Contrasting Colors for More Readable Websites." *ThoughtCo*, ThoughtCo, 12 Jan. 2021, <https://www.thoughtco.com/contrasting-foreground-background-colors-4061363>.
- [5] Corning, Anne. "Rhapsody in Blue: Understanding Blue Light & Blue LEDs." *Radiant Vision Systems*, 15 July 2019. <https://www.radiantvisionsystems.com/blog/rhapsody-blue-understanding-blue-light-blue-leds>.
- [6] "Mobile Site Abandonment after Delayed Load Time." *Google*, Google, <https://www.thinkwithgoogle.com/consumer-insights/consumer-trends/mobile-site-load-time-statistics/>.

- [7] Dong, Mian, and Lin Zhong. "Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays." *IEEE Transactions on Mobile Computing*, vol. 11, no. 5, 2012, pp. 724–738., <https://doi.org/10.1109/tmc.2012.40>.
- [8] Pathak, Abhinav, et al. "What Is Keeping My Phone Awake?" *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services - MobiSys '12*, 2012, <https://doi.org/10.1145/2307636.2307661>.
- [9] Pathak, Abhinav, et al. "Where Is the Energy Spent inside My App?" *Proceedings of the 7th ACM European Conference on Computer Systems - EuroSys '12*, 2012, <https://doi.org/10.1145/2168836.2168841>.
- [10] Andersen, Michael. "Dark-Mode Lowers Power Consumption on OLED Devices." *SustainableWWW*, 19 Aug. 2021, <https://sustainablewww.org/principles/dark-mode-lowers-power-consumption-on-oled-devices>.
- [11] Lee, Yeongju, and Minseok Song. "Adaptive Color Selection to Limit Power Consumption for Multi-Object GUI Applications in OLED-Based Mobile Devices." *Energies*, vol. 13, no. 10, MDPI AG, May 2020, p. 2425. <https://doi.org/10.3390/en13102425>.