Detecting Display Energy Hotspots in Android Apps

Mian Wan, Yuchen Jin, Ding Li and William G. J. Halfond
University of Southern California
Los Angeles, California, USA
Email: {mianwan, yuchenji, dingli, halfond}@usc.edu

***Abstract*—Energy consumption of mobile apps has become an important consideration as the underlying devices are constrained by battery capacity. Display represents a significant portion of an app's energy consumption. However, developers lack techniques to identify the user interfaces in their apps for which energy needs to be improved. In this paper, we present a technique for detecting display energy hotspots – user interfaces of a mobile app whose energy consumption is greater than optimal. Our technique leverages display power modeling and automated display transformation techniques to detect these hotspots and prioritize them for developers. In an evaluation on a set of popular Android apps, our technique was very accurate in both predicting energy consumption and ranking the display energy hotspots. Our approach was also able to detect display energy hotspots in 398 Android market apps, showing its effectiveness and the pervasiveness of the problem. These results indicate that our approach represents a potentially useful technique for helping developers to detect energy related problems and reduce the energy consumption of their mobile apps.**

***Keywords*—Mobile applications, energy consumption**

I. Introduction

In less than six years, mobile apps have gone from zero downloads to over 35 billion downloads in 2013 [1], [2]. Simultaneously, smartphones have achieved a nearly 31% penetration rate [3]. Smartphones and apps have become so popular, in part, because they combine sensors and data access to provide many useful services and a rich user experience. However, the usability of these devices is inherently limited by their battery power, and the use of popular features, such as the camera and network, can quickly deplete a device's limited battery power. Therefore, energy consumption has become an important concern. For the most part, major reductions in energy consumption have come about through a focus on developing better batteries, more efficient hardware, and better operating system level resource management. However, software engineers have become increasingly aware of the way an app's implementation can impact its energy consumption [4], [5], [6], [7]. This realization has motivated the development of software-level techniques that can identify energy bugs and provide more insights into the energy related behaviors of an application.

One important insight is that the display component of a smartphone consumes a significant portion of the device's battery power [4]. This problem has only grown as smartphone display sizes have increased from an average of 2.9 inches in 2007 to 4.8 inches in 2014 [8]. Studies show that display can now consume on average 60% of the total energy expended by a mobile app [9]. Traditionally, optimizing display power has been seen as outside of the control of software developers. This is true for LCD screens, for which energy consumption is based on the display's brightness. This brightness, in turn, is controlled by either the end user or by the OS performing opportunistic dimming of the display. However, most modern smartphones, such as the Samsung Galaxy SII, are powered by a new generation of screen technology, the OLED. For this type of screen, brightness is still important [10]; however, the colors that are displayed also become important. Due to the underlying technology, this type of screen consumes less energy when displaying darker colors (e.g., black) than lighter ones (e.g., white). The use of these screens means there are enormous energy savings to be realized at the software level by optimizing the colors and layouts of the user interfaces displayed by the smartphone. In fact, prior studies have shown that savings of over 40% can be achieved by this method [6], [9].

Despite the high impact of focusing on display energy, developers lack techniques that can help them identify where in their apps such savings can be realized. For example, the well-known Android battery monitor only provides device level display energy consumption and cannot isolate the display energy per app or per user interface screen. Other energy related techniques have focused on surveys to identify patterns of energy consumption [5], design refactoring techniques that improve energy consumption [7], [11], programming language level constructs to make implementation more energy aware [12], energy visualization techniques [13], or energy prediction techniques [14]. Although helpful, the mentioned techniques do not account for display energy nor are they able to isolate display related energy. Existing work on display energy has focused on techniques that can transform the colors in a user interface (e.g., Nyx [6] and Chameleon [9]). But these techniques do not guide developers as to where they should be applied, therefore they must be (1) used automatically for the entire app, which means that although colors will be transformed automatically into more energy efficient equivalents, the color transformation may be less aesthetically pleasing than a developer guided one; or (2) applied based solely on developers' intuition as to where they would be most effective, which means that some energy inefficient user interfaces may be missed.

In this paper, we present a novel approach to assist developers in identifying the user interfaces of their apps that can be improved with respect to energy consumption. To do this, our approach combines display energy modeling and color transformation techniques to identify a *display energy hotspot (DEH)* — a user interface of a mobile app whose energy consumption is higher than an energy-optimized but functionally equivalent user interface. Our approach is fully automated and does not require software developers to use power monitoring equipment to isolate display energy, which, as we explain in Section III, requires extensive infrastructure

and technical expertise. Our approach to identify DEHs is comprised of four steps. First, our approach traverses the user interfaces of an app and takes screenshots of the app's user interfaces when they change in response to different user actions. Second, our approach transforms the screenshots to a more energy-efficient version by applying a new energy-efficient color transformation technique [6]. Third, using a display power model, our approach estimates the power consumption for each screenshot and its energy-efficient version. Fourth, the approach calculates the power and energy consumption differences between each user interface and its display-energy-optimized alternative and ranks the screenshots based on the magnitude of these differences. The approach reports these results, along with detailed power and energy information, to the developer, who can target the most impactful user interfaces for energy-optimizing transformations.

We performed an empirical evaluation of our approach on a collection of real-world and popular mobile apps. We found that our approach was able to accurately estimate display power consumption to within 12% of the measured ground truth and identify the most impactful DEHs. Furthermore, we found that the results generated by our approach can be generalized from one hardware platform to others. We also used our approach to investigate 962 Android market apps and found that 41% of these apps have DEHs. Some of them consume over 101% more display energy than a display-energy-optimized version! Overall, these results indicate that our approach can accurately identify DEHs and assist developers in reducing display energy of their mobile applications.

The rest of this paper is organized as follows: In Section II we describe our approach for detecting DEHs. Section III describes how we built the display power model for a device. We present the results of our evaluation in Section IV. Related work is discussed in Section V. Finally we conclude in Section VI.

II. Approach

The goal of our approach is to assist developers in identifying user interfaces (UIs) that can be improved with respect to energy consumption. More specifically, our approach detects *DEHs*, which are UIs that consume more display energy than their energy optimized versions would. To detect these, the approach automatically scans each UI of a mobile app and then determines if a more energy efficient version could be designed. It is important to note that DEHs are not necessarily energy bugs, as the DEHs may not be caused by a fault in the traditional sense. Instead, DEHs represent points where code is energy-inefficient with respect to an optimized alternative. After detecting the DEHs, the UIs are ranked in order of the potential energy improvement that could be realized via energy optimization, and then reported to the developers.

To achieve complete automation and not require developers to have power monitoring equipment, there are two significant challenges we have to address. The first is to determine how much display energy will be consumed by an app at runtime without physical measurements. Our insight is that power can be estimated by a display power model that takes UI screenshots as input. The second challenge is to determine whether a more energy-efficient version of a UI exists and to quantify the difference between these two versions. Our insight is that we can use automated energy-oriented color transformation techniques to recolor the screenshots and then calculate the gap between the original and the more efficient version. We have leveraged these two insights to define an approach that can automatically detect DEHs without requiring power monitoring equipment.

An overview of our approach is shown in Figure 1. Our approach requires three inputs: (1) a description of the workload for which the developers wants to evaluate the UIs, (2) a Display Energy Profile (DEP) that describes the hardware characteristics of the target mobile device platform, and (3) the mobile app to be analyzed. Using these inputs, our approach performs the detection in four steps. The first step is to run the app based on the workload description and capture screenshots of the different UIs displayed. In the second step, the approach processes these screenshots and generates energy-efficient versions via a color transformation technique. Next, in the third step, a hardware model based on the DEP is used to predict the display energy that would be consumed by each of the screenshots and their energy-optimized versions. Finally, the fourth step compares the energy consumption of each UI with that of its optimized version and gives the developers a list of UIs ranked according to the potential energy impact of transforming each UI. We now explain each of these steps in more detail.

A. Step 1: Workload Execution and Screenshot Capture

The goal of the first step is to convert the workload description into a set of screenshots that can drive the display energy analysis in the subsequent steps. Strictly speaking, a workload description is not a necessary input to the approach since an automated UI crawler (e.g., PUMA [15]) could navigate an app and execute a fixed or random distribution of actions over the UI elements. However, the use of a workload description allows our approach to analyze the app using realistic user behaviors or a particular workload of interest to the developers. For example, a developers could collect execution traces of real users interacting with their app and use this to define a workload for the energy evaluation.

The inputs to this step are a target app and its workload description. The app A is the APK file that can be executed on a mobile device. The workload W is represented as a sequence of event tuples in which each tuple is of the form $\langle e, t \rangle$, where e is a representation of the event (e.g., "OK button pressed") and t is a timestamp of when the event occurs relative to the first event (i.e., $t = 0$ for the first event e_1). Our approach does not impose a specific format or syntax on W except that it must be reproducible. In other words, it must be specified in a way that allows for some mechanism to replay the workload. For example, our implementation (Section IV-B) uses the ReRan tool [16] to record and then replay a workload description, so the exact syntax and format of W is dictated by that tool.

Given W and A, the approach captures screenshots of the different UIs displayed on a device's screen during the execution of W on A. The general process is as follows. The replay mechanism executes each event at its specified time. A monitor mechanism executes in the background of the device and captures a screenshot of the display every time it changes.

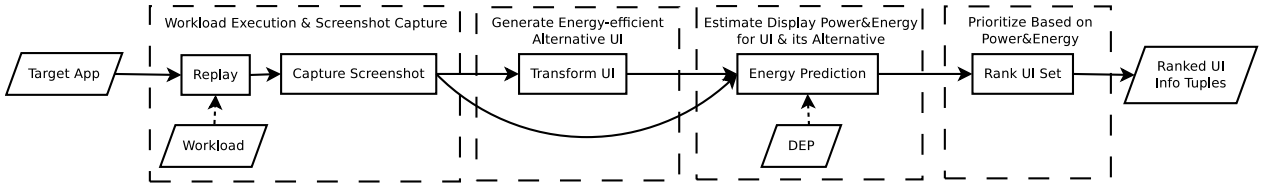

Fig. 1: Overview of our approach.

This is done by hooking into the refresh and repaint events of the underlying device. The execution of the workload continues until all event tuples have been executed. The output of the first step is a sequence of tuples S, in which each tuple is of the form $\langle s, t \rangle$, where s is a screenshot and t is the time at which the screenshot was taken (i.e., when the display changed). Note that a necessary condition of both the replay and monitor mechanisms is that their use does not alter the functionality of the app or the UI's appearance when it is rendered on the device. These conditions were met by all monitor and replay mechanisms that we investigated.

B. Step 2: Generate Energy-Efficient Alternative UI

The goal of the second step is to generate an optimized version of each screenshot tuple in S so that the following step can obtain estimates of the energy consumption for each screenshot and its optimized version. However, this optimized alternative does not exist, so the approach must first generate a reasonable approximation of what such an alternative would look like. A guiding insight is that related work has shown that darker colors on OLED screens are more energy efficient than lighter colors [17]. To take advantage of this insight, our approach could invert colors of UIs with a white-colored background or systematically shift colors to make them darker, and then use this transformation as the optimized version. However, these approaches neglect the fact that both color inversion and linear color shifts do not maintain color difference, which is the visual relationship that humans perceive when they look at a colored display [6]. Therefore, although the color adjusted UIs would be more energy efficient, they would not represent a reasonable approximation of optimized UIs as the resulting UIs would not be aesthetically pleasing.

To address this challenge, we leverage a color transformation technique, Nyx, that we developed in prior work [6]. A key aspect of Nyx is that the Color Transformation Scheme (CTS) represents a reasonably aesthetically pleasing new color scheme. Nyx statically analyzes the structure of the HTML pages of a web application and generates a CTS that represents a more energy efficient color scheme for the web application. Nyx does this by first creating a Color Conflict Graph (CCG), where each node in the graph is a color that appears in a web page and each edge represents the type of visual relationship (e.g., "next to", "enclosing", or "not touching") that any two colors in the CCG have. The edges in the CCG are weighted by the type of visual relationship, with higher weights given to edges so that "enclosing" $>$ "next to" $>$ "not touching". Then, Nyx solves for a recoloring of the CCG that is energy efficient, and also maintains, as much as possible, the color distances between colors in the original page that have a visual relationship. The weighting allows Nyx to prioritize maintaining certain types of color distances over others. Empirical studies show that the resulting color schemes can reduce display power consumption of web apps by over 40% while maintaining a high degree of user acceptance of the resulting pages' aesthetics.

For our approach, we adapt the CTS generation process of Nyx. There are two primary challenges that we have to address to carry out this adaptation — generation of the CCG and scalability. In the Nyx approach, the CCG was built by statically analyzing server-side code that dynamically generated web pages. Our approach uses screenshots, therefore, the CCG models the color relationships between adjacent pixels, which we determine by analyzing each pixel of each screenshot and identifying its color and the colors of the surrounding pixels. This method of constructing the CCG leads to the second challenge, scalability. The recoloring of the CCG is an NP-hard problem with respect to the number of colors. The rendering kits of mobile devices use anti-aliasing and color shading to smooth lines and curves. This process means that a simple black circle over a white background may actually be rendered with many shades of gray to smooth out the black circle's outline. In the presence of this, many extra colors, the time needed to generate each CTS would make the approach impractical for developers to use.

To address the second challenge, we first cluster the colors of the screenshot according to the color differences [18] between each color and 140 predefined colors [19]. After that, we map each color to its clustered color so that we reduce the number of colors for building the CCG. Then we create the CCG based on this reduced set of colors. We maintain a lookup table between the original colors and the clustered colors. We then convert the color with the largest cluster to black, which is the most energy efficient color, and solve the CCG recoloring problem using an approximation algorithm. Since the approach uses an approximation algorithm, the generated CTS may not reflect the most optimal recoloring. However, this means that the recolored UI only represents a lower bound on the optimization. With the newly generated CTS, we recolor the original screenshot at a pixel level so that every color in the cluster is replaced with its corresponding color in the CTS. The use of clustering means the approach performs its analysis on a simpler version of the screenshots with fewer colors. This can also introduce inaccuracy into the power estimation of the color-optimized screenshot. However, unless the screenshots differ significantly in the amount of anti-aliasing used, this inaccuracy is small. To confirm this, we compared the power consumption between a set of screenshots and the versions of the screenshots using the results of the clustered colors and found it below 2%. Thus, we can treat them as equivalent.

This process is repeated for every screenshot tuple $\langle s, t \rangle \in$

S. For each such s, the approach generates an s', which is the alternate version of the screenshot recolored as described above. The output of this step is a function O that maps each s to its corresponding s'.

C. Step 3: Display Energy Prediction

The third step of the approach computes the display power and energy of the screenshots and their energy-efficient alternatives. The approach does this by analyzing each screenshot obtained in the first step and its optimized version generated in the second step with cost functions that estimate the energy consumption based on the colors used in the screenshot. The inputs to this step are the screenshot tuples S generated by the first step, O generated by the second step, and the cost function C provided by the DEP. (The development of the cost function provided by the DEP is explained in Section III and an evaluation of its accuracy is in Section IV-C.) The outputs of this step are two functions that map each screenshot tuple in S or O to its power (P) and energy (E).

$$P(s) = \sum_{k \in |s|} C(R_k, G_k, B_k) \tag{1}$$

$$E(s, t_s, t_e) = P(s) * (t_e - t_s) \tag{2}$$

The formulas for calculating the output functions are shown in Equations (1) and (2). Here, s can be replaced with $O(s)$. To calculate $P(s)$ for all $\langle s, t \rangle \in S$, the approach iterates over each pixel in s and calls the cost function C with the values associated with the red (R), green (G), and blue (B) values of the pixel's color. The value returned by P is in Watts. Recall that energy is equal to power multiplied by time. Therefore, E is equal to the power associated with the screenshot ($P(s)$) times the amount of time the screenshot is displayed. The display time is calculated by subtracting the time the screenshot is displayed (t_s) from the time the next screenshot is displayed (t_e), or in other words, subtracting the timestamp associated with screenshot s_i from the timestamp associated with screenshot s_{i+1}, which would be of the form $E(s_i, t_i, t_{i+1})$. The value returned by E is in Joules.

D. Step 4: Prioritizing the User Interfaces

The goal of the fourth step is to rank the UIs in order of their potential power and energy reduction. To do this, the approach calculates the power and energy of each color-transformed screenshot and compares it to the power and energy of the original screenshot. The inputs to the fourth step are S, P, E, and O.

$$D_P(s) = P(s) - P(O(s)) \tag{3}$$

$$D_E(s, t_s, t_e) = E(s, t_s, t_e) - E(O(s), t_s, t_e) \tag{4}$$

Given these inputs, the approach calculates the power and energy difference according to the formulas shown in Equations (3) and (4). For the difference in power (D_P), the approach subtracts the power of the corresponding $O(s_i)$ from that associated with each $s_i \in S$. The resulting number is in Watts and represents the power that could be saved by using s'_i, the color-optimized version of s. For the difference in energy (D_E), the approach subtracts the energy of the corresponding $O(s)$ from that associated with each $s_i \in S$. The resulting numbers is in Joules and represents the energy that could be saved by using s'_i instead of s_i.

The output of the fourth step is two sequences, R_P and R_E. Each sequence is comprised of the tuples $\langle s, D \rangle$ where s is the screenshot and D is either the difference in power (D_P) or difference in energy (D_E). By choosing the metric D_P or D_E, the users could choose whether or not to take the time spent by each screenshot in to consideration. The sequences are ordered by each tuple's D value from highest to lowest. This ranking is the output of the approach and represents a prioritization of the screenshots that appear during the workload's execution in order of their potential power and energy reduction if they were to be color optimized.

III. The Display Energy Profile

The DEP provides pixel-based power cost function for a target mobile device. The use of the DEP allows our approach to analyze display power for multiple devices by simply providing different DEPs as input to the approach. We expect that, in the future, a DEP will be developed and provided as part of a device's Software Development Kit (SDK). However, this is currently not common in practice, so in this section we discuss the steps required to develop a DEP.

At a high-level, the DEP provides a cost function that can predict how much power will be consumed by an OLED screen. Prior research work has shown that the power consumption of a pixel in an OLED screen is based on its color [20]. Therefore, we define the input to the cost function in the DEP as the RGB value that defines a pixel's color. The output of the cost function is the amount of Watts that will be consumed by the display of the pixel on the target device.

$$C(R, G, B) = rR + gG + bB + c \tag{5}$$

The general form of the cost function is shown in Equation (5). R, G, and B represent the red, green, and blue components of a pixel's color, respectively. The coefficients r, g, b, and c represent empirically determined constants. The value for each constant varies by mobile devices. Note that our power model does not account for screen brightness. This is generally controlled by the user or OS, not the software developer. Furthermore, savings incurred by adjusting brightness would apply uniformly across all UIs. We constructed DEPs for three mobile devices, a 2.83" μOLED-32028-P1T display (μOLED) from 4D Systems, a Samsung Galaxy SII (S2), and a Samsung Galaxy Nexus (Nexus). For all of these displays, we measured power consumption using the Monsoon Power Monitor (MPM) from Monsoon Solutions Inc. [21]. The MPM allows the voltage to be held constant while supplying a current that may be varied from its positive and negative terminals. The MPM samples the voltage and the current supplied and outputs the power consumption with a frequency of 5kHz. This measurement resolution is sufficient for our purposes since the average duration of screenshots is in the order of seconds.

We built each power model roughly following the process outlined by Dong and colleagues [20]. We first measured the power consumption of a completely black screen to define a baseline power usage for an active screen. To determine the parameters for each RGB component, we measured the power consumption of the screen while displaying solid-colored pages. We varied the intensity of each color component while holding the other two components at zero, and collected data points for 16 intensities for each component (R, G and B). In total, we had 48 data points for each device.

After taking measurements for each color component, we subtracted the black screen power usage from these measurements to produce the power consumption of each R, G, and B component. The relationship between the power consumption and the RGB value is non-linear due to the gamma encoding of the screen. Gamma encoding is a digital image editing process that defines the relationship between pixel values and the colors' luminance. It allows for our eyes to correctly perceive the shades of color of images that are captured digitally and displayed on monitors. To account for this encoding, we raised the RGB values to the 2.2 power to decode the image. While the gamma can vary between 1.8 to 2.6, 2.2 is the standard image gamma of screens adopted by the industry. After gamma correction of the RGB values, we used linear regression to determine the coefficients for Equation (5). The linear relationship between the new RGB values and the power consumption was very strong. The average R^2 value for the three models was 0.99111. The detailed coefficients for each device can be found in our project webpage [22].

One particular problem that we faced with the S2 and Nexus was that the measured energy also included the energy consumed by background processes and other hardware components of the smartphone. To properly isolate the display energy, we took two measurements. For the first, we disconnected the flex cable that provides power, as well as data and signals, between the display and the CPU. This provided us with a baseline measurement of the power consumption of the phone without the display. We chose this process because by disconnecting the cable, the phone still maintains its background processes instead of suspending them for sleep mode. We subtracted this baseline value from our second measurement, the power of the phone with the display cable attached, to calculate the display power for each of the colored pages.

IV. Evaluation

In this section, we present the results of an evaluation of our approach. We implemented our approach in a tool called dLens and used it to answer the following research questions:

RQ 1: How accurate is the dLens analysis?

RQ 2: How generalizable are the dLens results across devices?

RQ 3: How long does it take to perform the dLens analysis?

RQ 4: What is the potential impact of the dLens analysis?

A. Subject Applications

We selected the top ten most popular free Android market apps in the United States as of August 2014 for our subject applications. In general, our subject applications are from different developers with different features. For each of the subjects, we manually generated a workload by exercising the primary features and functions of each app. The average duration of the workloads was 340 seconds and each workload resulted in an average of 83 captured screenshots. Information about the apps is listed in Table I. For each app, we report the size of its APK file, the number of screenshots captured as part of its workload, and the time duration (in seconds) of the recorded workload.

TABLE I: Subject Application Information

Name	Size (MB)	Screenshots	Time (s)
Facebook	23.7	116	554
Facebook Messenger	12.9	55	268
FaceQ	17.9	96	470
Instagram	9.7	93	429
Pandora internet radio	8.0	75	278
Skype	19.9	65	254
Snapchat	8.8	142	465
Super-Bright LED Flashlight	5.1	20	51
Twitter	13.7	101	388
WhatsApp Messenger	15.3	65	242

B. Implementation

We implemented our approach in a prototype tool called dLens. The implementation of dLens leveraged several other libraries and tools. Workloads were recorded and replayed using the Reran tool [16]. We used the Ashot [23] tool to record screenshots of the different UIs displayed. We also modified Ashot to make it associate a timestamp with each generated screenshot. As described in Section II-B, the color transformation scheme generation was based on the code developed in the Nyx project [6]. We adapted Nyx to build CCGs using screenshots instead of static analysis. We measured the energy consumption of each screenshot on a Monsoon Power Monitor (MPM). Finally, we performed our experiments on three different platforms: a 2.83" μOLED-32028-P1T display (μOLED) from 4D Systems, a Samsung Galaxy SII smartphone (S2), and a Samsung Galaxy Nexus smartphone (Nexus).

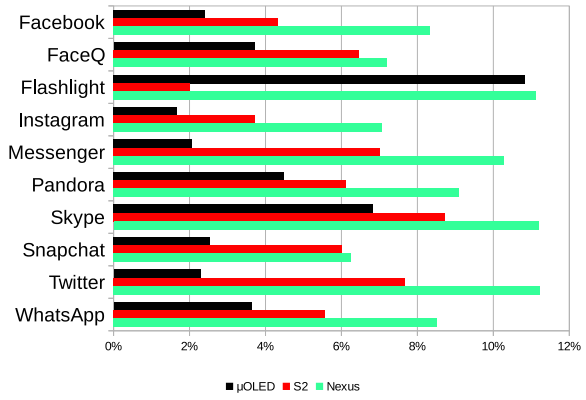

Fig. 2: The EER of the power model

C. RQ1: Accuracy

The first research question deals with the accuracy of the dLens approach. We evaluated the accuracy of dLens in two ways. First, we calculated the Error Estimation Rate (EER), which is the accuracy of the power estimate produced by dLens for a given screenshot. Note that the EER is different from the accuracy we reported in Section III, which is the Pearson coefficient, and expresses the closeness of the fit between the power measurements of the solid-color screenshots and the regression-based model. In contrast, for the EER, we evaluated the closeness of the ground truth of the screenshots from the subject applications with dLens's estimates. The second kind of accuracy is Device Ranking Accuracy (DRA), which is the accuracy of the UI screenshot ranking provided by dLens compared to the UI screenshot ranking provided by the ground truth power measurements. Note that if the EER was 0, the ranking of the screenshots would be 100% accurate. However, since we are dealing with physical systems, a certain amount of estimation error is to be expected. Therefore, the measurement of the DRA reflects the ability of the dLens approach to rank the UI screenshots accurately despite the underlying EER of the approach. We only reported these metrics for power since they were equal to those of energy. This is because the corresponding energy measurements would simply be obtained by multiplying the power estimates by the length of time the screenshot was displayed.

For calculating the EER, we performed the following process. First, we ran dLens on each of the subject applications and obtained the list of screenshots that had been ranked by power. To compute the EER, we randomly selected five screenshots from each app and their corresponding color-transformed version, and measured their ground truths using the MPM on each of the three devices. We chose to sample the screenshots instead of complete measurements of all the screenshots because the process for isolating a screenshot's power and energy (see Section III) was manual and, therefore, time-intensive. Then we compared the power values estimated by dLens to the ground truths for the selected screenshots. In Figure 2, we show, for each of the subject applications and for each of the three devices, the average EER for the five screenshots. Across all of the applications, the average EER for the μOLED, S2, and Nexus, was 5%, 8%, and 8%, respectively. We repeated this experiment for the color-transformed versions and had corresponding accuracy numbers of 6%, 8%, and 9%. We omitted the graph of these results for space considerations. Overall, we noted that the μOLED has a lower EER than the other two devices. We hypothesize that this is due to the fact that the μOLED is only a display device and therefore does not have any distortion of its power model by background components or processes that would be experienced by the two smartphones.

To compute the DRA, we followed a process similar to that of calculating the EER. However, instead of a random sample, we chose the top-five ranked screenshots. We chose to use the top five because this represented a group of screenshots that would likely be similar in terms of power consumption and therefore their relative ranking accuracy would be more sensitive to the underlying EER than a random subset. Also, the top five represented the set most likely to be used to guide the developers, and therefore is the most representative of the accuracy that an end user of dLens would experience in real usage. For these top five, we calculated the ground truths of each screenshot and its color-transformed version, and ranked these by their power consumption. Then we compared this ranking against the ranking of those screenshots computed by the dLens tool. To compare the rankings, we treated each ranking of the five screenshots as a vector and used Spearman's rank correlation coefficient to compute the closeness of these vectors. The coefficient gives -1/1 when the two rankings are perfectly negatively or positively correlated, and is 0 when the two rankings are not correlated. Across all of the applications, the average DRA for the μOLED, S2 and Nexus was 0.76, 0.77, and 0.57, respectively. These results indicate that the rankings were not an exact match. We investigated the cases where the ranking was not an exact match. We found that this was caused by the closeness of the power consumption of the screenshots in the top five. For example, it was typical to see the top five separated by about a 2% difference in their overall power consumption. This was well within the possible error range that we measured for the EER for each device and was likely the reason for this ranking variation.

Overall, the results for RQ1 show that the dLens tool is very accurate. For the EER, the estimated power was within 12% of the ground truth for all devices. Even with a non-zero EER, dLens was able to accurately rank the UIs as verified by the ground truth measurements. The minor variations seen in the rankings could be attributed to the size of th EER. Both of these accuracy measurements are important for developers as they indicate that their design changes can be made with a high degree of confidence in both the actual estimates and the relative ranking of the UI screenshots.

D. RQ2: Generalizability

The second research question evaluates the generalizability of the results computed by dLens. This research question addresses a potential limitation to the usage of our approach. Namely, the screenshots captured by the approach and the corresponding DEP reflect the power and energy usage of UIs displayed on a particular set of devices. In practice, this set would be the device(s) the developer has available for testing purposes. However, other devices are likely to vary in terms of screen resolution and power consumption characteristics. This research question evaluates how well the rankings for one device match the rankings that would be computed for other devices using their own DEPs. If the results of the dLens approach are generalizable across mobile devices, developers can use the result from their limited set of devices to infer the result of other or similar devices.

To answer this research question, we compared the similarity of the rankings generated by dLens for each of the mobile devices. For this experiment, we ran dLens for each device (i.e., using its own DEP) on the set of screenshots for each app. For each app, we compared its screenshot rankings against those computed for other platforms. To compare the rankings, we used the Spearman's rank correlation coefficient based technique explained in Section IV-C. We performed a pair-wise comparison of the rankings generated for each of the three devices. The average across all apps for each of the devices is shown in Table II.

TABLE II: Average Spearman's correlation coefficient of rankings between devices

Base Device	μOLED	S2	Nexus
μOLED	-	0.9914	0.9892
S2	0.9914	-	0.9981
Nexus	0.9892	0.9981	-

TABLE III: Analysis time of the dLens approach

Name	T_C(s)	T_E(s)	Overall(s)	Per UI (s)
Facebook	1,470	7	1,477	12
Facebook Messenger	997	3	1,001	18
FaceQ	1,145	5	1,151	12
Instagram	2,799	6	2,806	30
Pandora internet radio	1,418	4	1,423	19
Skype	871	3	875	13
Snapchat	1,444	8	1,453	10
Super-Bright LED Flashlight	863	1	865	43
Twitter	1,316	6	1,323	13
WhatApp	897	3	901	13

The results show that the rankings generated by dLens for each of the mobile devices was, in fact, highly similar. The implications of this finding are that the results of running dLens on one device are similar to that for other devices. More broadly, this indicates that energy-reducing redesigns undertaken by developers based on results from one device are likely to also reduce energy on other devices. However, we note this finding with a caveat that we were only able to test this on three devices and a larger and more varied pool of devices may reveal higher amounts of variation.

E. RQ3: Analysis time

The third research question addresses the time needed to analyze an app using the dLens approach. We measured the overall time needed to run the dLens approach and then isolated the time for two of the steps, power estimation and color transformation. Note that we have excluded the time to replay the workload, which is shown as column "Time" in Table I.

We only reported the analysis time for the S2 DEP because the results were very similar for all three devices. The results are shown in Table III. The time for the color transformation is shown as "T_C" and the time for power estimation is shown as "T_E". The total time is shown as "Overall", which also includes time for file processing, ranking, etc. Since each app varies in terms of the size and duration of the workload, we also normalized the timing measurement by dividing the overall time by the number of screenshots captured in each app's workload and list this value in the column labeled "Per UI." All time measurements are shown in seconds.

The average time for dLens to analyze an app was 22 minutes. However, the time ranged from 14 minutes to 46 minutes. Although this amount can be considered high, it was directly dependent on the overall size of the set of screenshots, which ranged from 20 to 142. Therefore, the per screenshot number, which ranged from 12 to 43 seconds, is more informative. For each screenshot, we observed that most of the time was taken by the generation of the CTS. The runtime of the CTS algorithm is based on the number of colors presented in a screenshot. This is due to the fact that the recoloring problem is an NP-Hard problem. The Nyx approach uses an approximation algorithm to solve this problem. Therefore, this aspect of the approach can be sped up, if needed, by accepting lower quality approximations. However, this may have a tradeoff in terms of accuracy of the computed results and rankings.

TABLE IV: The ten apps with the largest DEHs.

App Package Name	Potential Power Savings (%)
biblereader.olivetree	101
com.amirnaor.happybday	98
com.adp.run.mobile	97
com.airbnb.android	97
com.darkdroiddevs.blondJokes	96
com.chapslife.nyc.traffic.free	94
com.al.braces	94
appinventor.ai_vishnuelectric.notextwhiledriving2_checkpoint1	92
appinventor.ai_freebies_freesamples_coupons.StoreCoupons	92
bazinga.emoticon	91

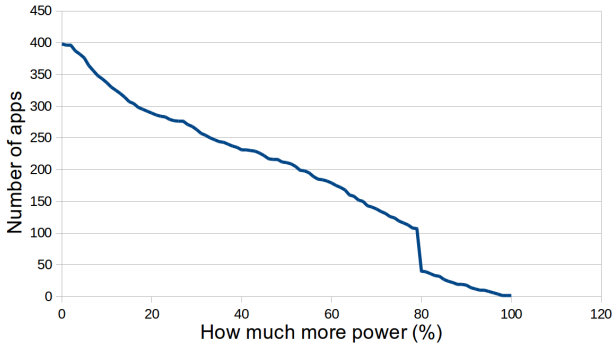

Fig. 3: The number of apps with DEHs

F. RQ 4: Potential Impact

In this research question, we investigated the potential impact of dLens by determining how many market apps contain DEHs. To do this, we used dLens to analyze 1,082 Android market apps. In our study, we looked at the initial page of an app. Due to the large number of test cases, we automated the process of executing the apps and taking a screenshot with the adb [24] tool from the Android SDK. One problem in this process is how to automatically detect when the page of an app is valid. We solved this problem with the heuristic that we take the screenshot of an app after it had been started for five seconds. To ensure all the apps have been executed successfully, we manually checked all screenshots and removed the screenshots that are not valid, such as crash pages. In total, we collected screenshots of 962 apps.

We found 398 apps containing DEHs. That means for 41% of the examined apps, their main page consumes more display power than the optimized version. On average, these apps consume 49% more energy than the optimized version. For some apps, this number could be as large as 101%. For all of the apps that have DEHs, we plotted how much more energy they consume than the optimized version in Figure 3. In this

(a) Original screenshot (b) Transformed screenshot

Fig. 4: Two version screenshots of the top energy-inefficient app

figure, a point (X,Y) on the line means that there are Y apps that consume at least X% more energy than their optimized version. The one with the largest potential energy savings was Bible Study. We show its original design and optimized design in Figure 4a and Figure 4b, respectively. In Bible Study, the original design uses a significant amount of white as the background color, which consumes more energy than other colors. However, by using black as the background color as we have done in Figure 4b , we could save much more energy while still maintaining the usefulness of the app. We also show the information of the top 10 energy-inefficient apps in Table IV.

From the result above, we conclude that many apps in the Android market are not optimized in terms of display energy efficiency. In fact, it is possible that an Android market app may consume more than double the amount of display power that an optimized version would.

We also had a further look at the UI designs of all of the apps. For those apps that contain no DEHs, we found that they all use black (#000000) as the main color. Here we say a color is the main color of a screenshot if it covers the most pixels in the screenshot. On average, black covers 79.3% of the pixels on the screen in these apps. Besides black, dimgray (#696969), darkslategray (#2F4F4F), darkgray (#A9A9A9) and gray (#808080) are also commonly used as the secondary colors in the apps without DEHs. They cover 8%, 5%, 3%, and 2% of the pixels, respectively, in the screenshots. Further, we found that the color combination black:darkgray:gray:white:dimgray with the ratio 794:32:13:10:6 is the most commonly used combination in apps without DEHs. It is used in 23% of apps without DEHs and on average covers 86% of the screenshots.

In the apps with DEHs, white, dimgray, and whitesmoke (#F5F5F5) are the three most popular main colors. They are used as the main colors in 42%, 12%, and 10% of apps with DEHs, respectively. On average, each of these colors respectively covers 65%, 56%, and 55% of the pixels on the screen.

Our extended study reveals some basic statistics and UI design patterns in both apps with and without DEHs. These results show that dLens can generate useful and actionable information that can have a large potential impact in terms of helping developers optimize the display energy for their apps above and beyond the detection of DEHs.

G. Threats to Validity

In this section, we discuss several threats to the validity of our results. The first is a general threat to external validity by our selection of only the most popular apps and our workload generation. For app selection, although the most popular apps may not be representative of all apps, choosing subjects in this way eliminated the threat of selection bias for the subjects and made it possible to argue that our approach is useful for even well-engineered apps. Furthermore, the results of RQ4 show the general applicability of our approach to a wide range of other apps. Even though we generated the workload ourselves, this does not affect the external validity of our results. Our workloads used the primary features of the apps, which was easy to identify in all cases, and the dLens results do not depend on accurately identifying typical or representative workloads, they can be used for any workload. Another general threat is to the internal validity of our results if the mechanism of establishing ground truth is inaccurate or the approximation of the CTS generated by the Nyx-based method is inaccurate. To ensure the accuracy of the ground truth measurements, we developed and tested the protocols used to ensure they reliably and accurately captured power measurements. Our protocols are also based on existing approaches by Dong and colleagues [20], [25] and prior work by our group [6]. The accuracy of the Nyx-based CTS has been established in prior work [6] and represents an approximation of the optimized version. In practice, users may not optimize their color designs by strictly following our generated version. However, by using Nyx, we can still provide a reasonable estimation of the energy consumption for the optimized version. Another aspect of the internal validity that we did not control for is that the UIs may contain dynamic elements (e.g., images) whose color should not be changed for optimization purposes. Since we did not eliminate these dynamic elements from our captured screenshots, the CTS may also change the color of these elements, overestimating the display power difference between a screenshot and its optimized version. Although this is a valid threat, its impact is likely low because the amount of such content was generally consistent across all captured screenshots. Finally, we assume that the CTSs provided by Nyx represent an aesthetically acceptable version of the UIs. Our prior user studies have shown that Nyx creates a slightly less aesthetically pleasing version of the colors, but that this version meets with high user acceptance when the energy savings are known.

For RQ1, a threat to validity is that we only measured the accuracy of the top-five ranked screenshots and a random sample of the screenshots. We used sampling since the screenshot isolation process we described in Section III is very time consuming and it was not feasible to measure the power and energy of every screenshot. Nonetheless, our experiment evaluation included over 1,800 such measurements to evaluate

the accuracy of the three devices for all of the experiments required by the RQs. We do not have a reason to believe that the results would differ with a larger subset. We chose the subset at random to eliminate any potential selection bias. Our raw data is available via the dLens project page [26].

For RQ2, there are two threats to validity. The first of these is a threat to criterion validity for our selection of ranking as the measurement we compared across devices. An alternative would have been to use power measurements. However, we argue that ranking is the more appropriate metric to be used since we expect developers will prioritize their work based on rank instead of actual power differences. Second, a threat to external validity for RQ2 is that we only used three power models. It is very likely that some phones have DEP cost functions that will result in different rankings due to different underlying hardware mechanisms. Therefore, we do not think our results will generalize for all devices. Future investigations into what enables more reliable result generalizability will enable us to make stronger conclusions on this aspect in the future.

V. Related Work

The closest work to dLens is that of Dong and colleagues [20], [25]. In their work, they constructed a power model for a commercial QVGA OLED display module at different granularities. In their power model, they demonstrated the linear relationship between power consumption and sRGB value, and achieved 90% accuracy in display power estimation. Other power models for μOLED screens have also been proposed [10], [27], [28]. Unlike dLens, their work only built models for μOLED screens. They did not detect DEHs. Thus, they could not provide information about the location of the potential points of display energy optimization of apps.

Sampson and colleagues [29] developed a tool called WebChar to evaluate browser performance and the energy consumption of different code features in HTML and CSS so that this tool can guide developers to optimize browsers or web applications. This work focused on the impact of HTML and CSS code structure on energy consumption, whereas dLens focuses on UI color design.

Other work that is related to dLens mainly focuses on reducing the display energy of OLED screens. However, this work does not provide guidance to help developers find the DEHs in their apps. One common way to reduce display energy is to change the content by manipulating the pixels [30] or colors [20], [31], [32], [33]. Specifically for web applications, there are some approaches to do color transformation on both the client [9], [17] and server[6] sides. Other ways to save display energy are darkening the unimportant parts of a screen [34], [35], [36], eliminating undesired details [37], and distorting image regions [38]. For LCD displays, energy optimization mainly reduces external brightness and adapts the content to the change [39], [40], [41], [42].

Many approaches [43], [44], [45] have also been proposed to model the power consumption of other components in mobile devices. Several approaches [10], [46], [47] acquired power models automatically based on the battery behavior of mobile devices. Other techniques [48], [49], [13], [14], [7], [50], [51] estimate energy consumption based on different models. These approaches only build power models for some or all the components of a mobile device, while our work not only builds power models for screens but also detects DEHs to help developers optimize their apps' display energy consumption.

VI. Conclusions

In this paper, we presented a new technique for detecting DEHs in mobile apps. The approach first executes a developer specified workload for a target app on a device and captures screenshots of the display as it changes. Then it generates a CTS for each captured screenshot and produces its energy-efficient version. After that, it analyzes each screenshot and its alternative version to determine their display energy. The approach calculates the power and energy difference between the two versions of screenshots and provides developers with a list of screenshots ordered by power or energy difference. We implemented our approach in a prototype tool, dLens, and evaluated it on real-world popular mobile apps. We found that it was able to accurately estimate display power consumption and rank screenshots with DEHs. Furthermore, we were able to use our technique to detect DEHs in 398 Android market apps. Overall, the results are very promising and indicate that our technique is a viable and effective technique for helping developers to reduce the energy consumption of their mobile apps.

VII. Acknowledgments

This work was supported by National Science Foundation under Grant No. CCF-1321141.

References

[1] A. Inc. Press Releases. http://www.apple.com/pr/library.

[2] C. Warren. Google Play Hits 1 Million Apps. Mashable. Retrieved 4 June 2014.

[3] T. Ahonen. Lets Do 2014 Numbers for the Mobile Industry: Now we are at 100% Mobile Subscription Penetration Rate Per Capita Globally. Retrieved May 12, 2014.

[4] D. Li, S. Hao, J. Gui, and W. G. Halfond, "An Empirical Study of the Energy Consumption of Android Applications," in *Proceeding of 30th International Conference on Software Maintenance and Evolution*, ser. ICSME 2014, 2014.

[5] D. Li and W. G. Halfond, "An Investigation Into Energy-Saving Programming Practices for Android Smartphone App Development," in *Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS)*, 2014.

[6] D. Li, A. H. Tran, and W. G. J. Halfond, "Making Web Applications More Energy Efficient for OLED Smartphones," in *Proceedings of the 36th International Conference on Software Engineering*, 2014.

[7] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff, "What is Keeping My Phone Awake?: Characterizing and Detecting No-sleep Energy Bugs in Smartphone Apps," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, 2012.

[8] A. Barredo. A comprehensive look at smartphone screen size statistics and trends. Medium.

[9] M. Dong and L. Zhong, "Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays," *Mobile Computing, IEEE Transactions on*, 2012.

[10] L. Zhang, "Power, Performance Modeling and Optimization for Mobile System and Applications," Ph.D. dissertation, University of Michigan, 2013.

[11] C. Sahin, F. Cayci, I. Gutierrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Initial explorations on design pattern energy usage," in *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, 2012.

[12] I. Manotas, L. Pollock, and J. Clause, "SEEDS: A Software Engineer's Energy-optimization Decision Support Framework," in *Proceedings of the 36th International Conference on Software Engineering*, 2014.

[13] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan, "Calculating Source Line Level Energy Information for Android Applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 2013.

[14] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating Mobile Application Energy Consumption Using Program Analysis," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013.

[15] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "PUMA: Programmable UI-automation for Large-scale Dynamic Analysis of Mobile Apps," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, 2014.

[16] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "RERAN: Timing- and touch-sensitive record and replay for Android," in *Software Engineering (ICSE), 2013 35th International Conference on*, 2013.

[17] M. Dong and L. Zhong, "Chameleon: A Color-adaptive Web Browser for Mobile OLED Displays," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, 2011.

[18] G. Sharma, W. Wu, and E. N. Dalal, "The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations," *Color Research & Application*, 2005.

[19] "Recognized color keyword names," http://www.w3.org/TR/SVG/types.html.

[20] M. Dong and L. Zhong, "Power Modeling and Optimization for OLED Displays," *Mobile Computing, IEEE Transactions on*, 2012.

[21] I. Monsoon Solutions, "Power Monitor," https://www.msoon.com/LabEquipment/PowerMonitor/.

[22] "DEP coefficients," https://sites.google.com/site/dlensproject/dep-coefficients.

[23] "Android Screenshots and Screen Capture," http://sourceforge.net/projects/ashot/.

[24] G. Inc., "Android Debug Bridge," http://developer.android.com/tools/help/adb.html.

[25] M. Dong, Y.-S. K. Choi, and L. Zhong, "Power Modeling of Graphical User Interfaces on OLED Displays," in *Proceedings of the 46th Annual Design Automation Conference*, 2009.

[26] "dLens," https://sites.google.com/site/dlensproject/.

[27] D. Kim, W. Jung, and H. Cha, "Runtime Power Estimation of Mobile AMOLED Displays," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013.

[28] R. Mittal, A. Kansal, and R. Chandra, "Empowering Developers to Estimate App Energy Consumption," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, 2012.

[29] A. Sampson, C. Cascaval, L. Ceze, P. Montesinos, and D. Suarez Gracia, "Automatic discovery of performance and energy pitfalls in HTML and CSS," in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*, 2012.

[30] N. Kamijoh, T. Inoue, C. M. Olsen, M. T. Raghunath, and C. Narayanaswami, "Energy Trade-offs in the IBM Wristwatch Computer," in *Proceedings of the 5th IEEE International Symposium on Wearable Computers*, 2001.

[31] J. Chuang, D. Weiskopf, and T. Möller, "Energy Aware Color Sets," in *Computer Graphics Forum*, 2009.

[32] M. Dong, Y.-S. K. Choi, and L. Zhong, "Power-saving Color Transformation of Mobile Graphical User Interfaces on OLED-based Displays," in *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design*, 2009.

[33] J. Wang, X. Lin, and C. North, "GreenVis: Energy-Saving Color Schemes for Sequential Data Visualization on OLED Displays," 2012.

[34] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan, "Energy-Adaptive Display System Designs for Future Mobile Environments," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, 2003.

[35] T. K. Wee and R. K. Balan, "Adaptive Display Power Management for OLED Displays," in *Proceedings of the First ACM International Workshop on Mobile Gaming*, 2012.

[36] K. W. Tan, T. Okoshi, A. Misra, and R. K. Balan, "FOCUS: A Usable & Effective Approach to OLED Display Power Management," in *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2013.

[37] H. Chen, J. Wang, W. Chen, H. Qu, and W. Chen, "An image-space energy-saving visualization scheme for OLED displays," *Computers & Graphics*, 2014.

[38] C.-H. Lin, C.-K. Kang, and P.-C. Hsiu, "Catch Your Attention: Quality-retaining Power Saving on Mobile OLED Displays," in *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, 2014.

[39] H. Shim, N. Chang, and M. Pedram, "A backlight power management framework for battery-operated multimedia systems," *Design Test of Computers, IEEE*, 2004.

[40] A. Iranli and M. Pedram, "DTM: dynamic tone mapping for backlight scaling," in *Design Automation Conference, 2005. Proceedings. 42nd*, 2005.

[41] A. Iranli, H. Fatemi, and M. Pedram, "HEBS: histogram equalization for backlight scaling," in *Design, Automation and Test in Europe, 2005. Proceedings*, 2005.

[42] A. K. Bhowmik and R. Brennan, "System-Level Display Power Reduction Technologies for Portable Computing and Communications Devices," in *Portable Information Devices, 2007. PORTABLE07. IEEE International Conference on*, 2007.

[43] A. Shye, B. Scholbrock, and G. Memik, "Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

[44] L. Negri, D. Barretta, and W. Fornaciari, "Application-level Power Management in Pervasive Computing Systems: A Case Study," in *Proceedings of the 1st Conference on Computing Frontiers*, 2004.

[45] Y. Li, H. Chen, and W. Shi, "Power Behavior Analysis of Mobile Applications Using Bugu," *Sustainable Computing: Informatics and Systems*, 2014.

[46] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2010.

[47] M. Dong and L. Zhong, "Self-constructive High-rate System Energy Modeling for Battery-powered Mobile Systems," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, 2011.

[48] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 1994.

[49] V. Tiwari, S. Malik, A. Wolfe, and M.-C. Lee, "Instruction level power analysis and optimization of software," in *VLSI Design, 1996. Proceedings., Ninth International Conference on*, 1996.

[50] C. Wang, F. Yan, Y. Guo, and X. Chen, "Power estimation for mobile applications with profile-driven battery traces," in *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, 2013.

[51] S.-L. Tsao, C.-K. Yu, and Y.-H. Chang, "Profiling Energy Consumption of I/O Functions in Embedded Applications," in *Architecture of Computing Systems ARCS 2013*. Springer Berlin Heidelberg, 2013.